Guidewire Contact Management Guide

Release 6.0.8

Product Name: Guidewire ClaimCenter
Product Release: 6.0.8
Document Name: Guidewire Contact Management Guide
Document Revision: 05-February-2013

Contents

About This Document

This document provides guidance for managing contacts with ClaimCenter and ContactCenter. It describes how to integrate the two applications, extend the contact data model, configure contacts, and control contact security.

This topic includes:

Intended Audience

This document is intended to help system administrators and developers manage contacts and configure and extend contact functionality in ClaimCenter.

Assumed Knowledge

For administrators, this document assumes that you are familiar with system administration for your database server, application server, and operating system. Additionally, this document assumes that you understand administration of the ClaimCenter application and can apply this knowledge to ContactCenter.

For developers, this document assumes that you are familiar with running the server and performing basic system administration, using ClaimCenter screens and functions, and using Gosu and the Studio editors.

Related Documents

See the following Guidewire documents for further information:

ClaimCenter Installation Guide – Describes how to install a new copy of ClaimCenter into Windows or UNIX environments. This guide is intended for system administrators and developers who need to install ClaimCenter.

ClaimCenter System Administration Guide – Provides guidance for the ongoing management of a ClaimCenter system. This document is intended to help system administrators monitor ClaimCenter, manage its security, and take care of routine tasks such as system backups, logging, and importing files.

ClaimCenter Configuration Guide – Describes how to configure ClaimCenter and includes basic steps and examples for implementing such configurations. This guide is intended for IT staff and system integrators who configure ClaimCenter for an initial implementation or create custom enhancements. This guide is intended as a reference, not to be read cover-to-cover.

Gosu Reference Guide – Describes the syntax of expressions and statements within ClaimCenter. This document also provides examples of how the syntax is used when creating rules. This document is intended for rule writers who create and maintain rules in Guidewire Studio.

ClaimCenter Integration Guide – Provides an architectural overview and examples of how to integrate ClaimCenter with external systems and custom code. This document is a learning tool for explanations and examples with links to the *Java API Reference Javadoc* and *SOAP API Javadoc* for further details. This document is written for integration programmers and consultants.

Guidewire Glossary – Defines terms used in the documentation.

Conventions In This Document

Text style	Meaning	Examples
italic	Emphasis, special terminology, or a book title.	A *destination* sends messages to an external system.
bold	Strong emphasis within standard text or table text.	You **must** define this property.
narrow bold	The name of a user interface element, such as a button name, a menu item name, or a tab name.	Next, click **Submit**.
`monospaced`	Literal text that you can type into code, computer output, class names, URLs, code examples, parameter names, string literals, and other objects that might appear in programming code.	Get the field from the `Address` object.
`monospaced italic`	Parameter names or other variable placeholder text within URLs or other code snippets.	Use `getName(`*`first`*`, `*`last`*`)`. `http://`*`SERVERNAME`*`/a.html`.

Support

For assistance with this software release, contact Guidewire Customer Support:

- At the Guidewire Resource Center – `http://guidewire.custhelp.com`
- By email – `support@guidewire.com`
- By phone – +1-650-356-4955

chapter 1

Key Concepts in Contact Management

This topic introduces you to key concepts for managing and working with your contacts in ClaimCenter.

This topic includes:

- "Components of Contact Management" on page 7
- "ClaimCenter Integration with ContactCenter" on page 8

Components of Contact Management

This topic introduces contact management and contains the following topics.

- What is a Contact?
- Using and Configuring Contact Management

What is a Contact?

A contact is an entity that your company interacts with and needs to communicate with. In ClaimCenter, a contact can be a specific person, such as a claimant on a claim, a policy's policy holder, or a vendor like a doctor or attorney. Additionally, a contact can be a business, such as a repair shop, a bank, or a hospital. Finally, a contact can be a specific place or venue for which your company frequently references the geographical location, such as a legal venue like a court house.

Using and Configuring Contact Management

Contact management is typically considered to be a set of tasks related to adding, modifying, searching for, and deleting contacts in an address book. ClaimCenter provides various opportunities for you to create, edit, or search for contacts. For example, when entering a new claim, you can create a new contact or search for a contact, such as a claimant, in the claim creation wizard. You can also create and search for contacts in other areas of the ClaimCenter interface where you need to specify a contact for a claim.

You can configure screens to match company requirements, such as creating and searching for new kinds of contacts, and you can create entire new screens if needed. Or you can perform configurations like validating that an entered postal code is the correct format.

Aside from edits and screen configuration, there can be other configurations you might want to make, such as standardizing the defined set of approved vendors for the company. For example, it is unlikely that a company would want each user entering or editing contact information for the automobile rental company Avis. A more efficient method would be to have one user with special permissions create and maintain this vendor, while other users simply reference the Avis contact when needed.

Standardizing contacts can assist your business in other ways. By entering `Avis` only one time, you save time and help prevent fraud. You also ensure that the information for Avis is always correct, like the location where payments are sent for Avis services. Additionally, you can associate together all the claims that have been assigned to that vendor.

See also

- "ClaimCenter Integration with ContactCenter" on page 8

ClaimCenter Integration with ContactCenter

ContactCenter is a separate, stand-alone Guidewire application. You can use ContactCenter as a central repository for standardizing and managing the key vendor and service provider data of your company. Typically, a company uses ContactCenter in conjunction with ClaimCenter to manage contacts that can be used in more than one claim, like vendors. A claim can also have contacts that are associated only with that one claim, such as a claimant. These contacts can be tracked directly in ClaimCenter itself.

This section covers the following basic concepts required to integrate ClaimCenter with ContactCenter:

- Centralized Contact Management
- Which Contacts to Store in ContactCenter
- Locally and Centrally Managed Contacts
- Deciding Whether to Use ContactCenter

For detailed steps showing how to integrate ClaimCenter with ContactCenter, see "Integrating ClaimCenter with ContactCenter" on page 11.

Centralized Contact Management

As the sole location for key contacts, ContactCenter enables you to manage and serve contact data centrally. ClaimCenter supplies this functionality on the **Address Book** tab, where you can create and search for centrally-maintained contacts. Once located, you can edit the contact data as needed or delete the contact if it is not used in the system.

After you integrate ContactCenter and ClaimCenter, users who are searching for contacts will automatically have access to centralized contacts. These contacts appear both in the **Address Book** and in other locations in ClaimCenter where contacts normally appear. For example, when searching in ClaimCenter for an auto shop for a repair, an adjuster can access a list of approved service providers in ContactCenter.

You can define different users for each Guidewire application. For example, you can have a group of ContactCenter users whose key function is to manage contact data and ensure that it is accurate. ContactCenter users deal primarily with entering and managing contact information for general business accounting and management reasons. These users need not have authorization for ClaimCenter.

A synchronizing facility between the two applications ensures that when a contact changes—regardless of whether the change occurred in ContactCenter or ClaimCenter—the change appears in both applications.

Which Contacts to Store in ContactCenter

You need not store all company contacts in ContactCenter. In some cases, it makes sense to centrally manage a contact, and in others it does not. ClaimCenter enables you to manage contacts both centrally in ContactCenter and locally in ClaimCenter.

Typically, you store in ContactCenter either contacts that you expect to appear across multiple claims or policies or contacts for which you want to have a single, definitive information source. For example, you would store service providers such as doctors, auto repair shops, and inspectors in ContactCenter. These contacts require correct tax ID data for reporting and accurate addresses for payments.

Contacts that are specific to an application object are best managed locally in ClaimCenter. These are contacts that appear only in specific instances and are unlikely to reappear. Additionally, they have no impact on your company’s business processes, nor do they have any regulatory requirements. Two examples are an accident witness on a claim and a bank employee who requests a policy verification.

Locally and Centrally Managed Contacts

When you create a new contact, depending on where in the ClaimCenter interface you are, the contact is created either locally in ClaimCenter or centrally in ContactCenter. If you are on the **Address Book** tab when the contact is created, the system stores the record centrally in the ContactCenter database. If you create a new contact directly on a claim, the system stores it locally on the claim, and it is not visible from the **Address Book** tab. Such contacts are associated only with the claim in which they are created and are called *claim contacts*.

By default, claim contacts are not associated with ContactCenter in any way. ClaimCenter does not attempt to determine if a contact associated with one claim appears elsewhere on another claim. Thus, any claim contact can be a duplicate of one or more other claim contacts associated with different claims. When such duplicates exist, changes to one will not propagate to another because they are not related to one another.

When you consider this behavior from a contact management perspective it makes sense. A claimant typically is not a contact that other adjusters will reuse. Even so, when you add a contact, you can choose to **Add an Existing Contact**, which enables you to search the centralized ContactCenter database before creating a contact.

Conversely, ContactCenter contacts, like vendors and service providers, are not claim dependent. From ClaimCenter, you can associate a single centrally-managed contact with any number of claims. Alternatively, you can add a new contact and then **Link** the contact to make it centrally managed and available for use with other claims.

To link between ClaimCenter and ContactCenter, you define a connection between a specific record in the ClaimCenter database and another record in the ContactCenter database. In effect, you are saying that the two contacts, though in two different databases, are the same and are to show the same information. Users of ClaimCenter can link contacts from ClaimCenter to ContactCenter. From ContactCenter, users cannot link contacts to ClaimCenter.

After a contact is linked, the contact can be updated from either application, and both applications will have the update. To keep contacts in ClaimCenter up-to-date, you can copy a linked contact from ContactCenter. Copying causes ClaimCenter to copy the latest information from ContactCenter’s central repository.

You can unlink a contact. An unlinked contact in ClaimCenter becomes a claim-specific contact again. However, even though it is no longer linked to ClaimCenter, the contact created originally in the **Address Book** with the claim is not deleted and remains in the ContactCenter database.

A claim adjuster who does not know that a contact already exists in the **Address Book** can manually add the contact as a claim contact. As a result, your ClaimCenter database might have many duplicate claim contacts, each associated with a separate claim. These duplicates can have slightly different contact information or even outdated or inaccurate information. Adjusters can avoid this issue by using the **Link** button to link contacts between the two applications. Additionally, an adjuster who is uncertain that a contact exists can search the **Address Book** before adding a new contact.

Deciding Whether to Use ContactCenter

ClaimCenter comes with a preconfigured address book plugin that is useful only for demonstration purposes. Guidewire also supplies a fully functional Gosu plugin that works with ContactCenter and is suitable for production. Do not go into a production environment with the demo enabled. If you decide not to use the integration with ContactCenter but instead want to integrate with another, third-party contact system, you must implement a replacement plugin. For information on how to create your own plugin, see "Address Book and Contact Search Plugins" on page 301 in the *Integration Guide*.

You are not required to use either ContactCenter or your own contact plugin in ClaimCenter. Without a contact plugin, ClaimCenter continues to maintain its own local set of contacts in its own database, but it no longer has the following **Address Book** functionality:

- The **Address Book** tab will be present, but attempts to use it for searching will result in an error.
- You will not be able to link and then copy contacts from the Address Book.
- The claim **Parties Involved** interface will have reduced functionality. You will be unable to view the **Address Book** from the screen or copy a contact or add an existing contact from the **Address Book**.
- **Related Claims, Matters**, **Activities**, and so on in the **Address Book** pop-up and search screen will no longer work.

The **Address Book** tab, buttons, and other related options will still be present. To remove them, you must edit the related PCF files.

> **Note:** Even without ContactCenter or a custom plugin, you will still be able to use the **Search** tab to search for local contacts. However, you will no longer be able to check for duplicate contacts.

See also

- "Components of Contact Management" on page 7
- "Integrating ClaimCenter with ContactCenter" on page 11

chapter 2

Integrating ClaimCenter with ContactCenter

This topic describes how to integrate ClaimCenter and ContactCenter. Before you work through this topic, you must be familiar with the installation process defined in the *ClaimCenter Installation Guide*.

This topic includes:

This topic discusses only integrating ClaimCenter with ContactCenter. For information on integrating Guidewire applications with third-party applications, see the *ClaimCenter Integration Guide*.

Integration Concepts and Prerequisites

You can integrate ClaimCenter and ContactCenter by using the Guidewire QuickStart installation set up, or you can use any supported application and database server. Use QuickStart only in a demo or development environment. For production, you will need a dedicated JVM for each Guidewire application. Running both ClaimCenter and ContactCenter in the same JVM can invite memory conflicts and other problems. If you do not use a QuickStart set up, install the products in separate JVMs as described later in Enhancing Integration Performance.

ClaimCenter and ContactCenter communicate through SOAP APIs. ClaimCenter provides the `abintegration` web service that communicates with ContactCenter through its published `IContactAPI`. ClaimCenter also defines a Gosu plugin, `IAddressBookAdapter`, that invokes the integration layer. The ClaimCenter Gosu module `ccabintegration` captures all operations required for integration. This ClaimCenter module can be modified in Studio.

Guidewire simplifies the integration process with ContactCenter and ClaimCenter by using Gosu and enabling you to do the integration in Studio.

Enhancing Integration Performance

Configure the applications to run in separate Java Virtual Machines (JVMs), and set the `XXMaxPermSize` value on each application server instance to at least 128MB. The following table provides tips for configuring the most common JVMs:

Tomcat	Install in separate Tomcat instances with separate installation directories. You can install the Tomcat binaries once and share them between both installation directories by using Tomcat's `CATALINA_BASE` system property to support this type of behavior. Refer to your Tomcat documentation for more information. You must make sure that each Tomcat instance is running on its own unique port.
WebLogic	Install each application in a separate domain.
WebSphere	Install each application in separate application server instances.

Also, use separate database instances for ContactCenter and ClaimCenter, especially in larger installations. Finally, make sure that ClaimCenter and ContactCenter run on different ports. The default port number for ClaimCenter is 8080 and for ContactCenter it is 8280.

IMPORTANT Failing to follow these guidelines can result in poor performance.

Considerations for Running Oracle

If you are running ContactCenter on an Oracle database, you need to create additional tablespaces in your existing database. Similarly to ClaimCenter, ContactCenter maps its own logical tablespaces to the database's physical tablespaces. Provide a separate physical tablespace for each of the following logical tablespaces.

Name	Usage
`ADMIN`	Stores system parameters
`OP`	Stores the main ContactCenter data tables
`TYPELIST`	Stores system code tables
`INDEX`	Stores system indexes
`STAGING`	Stores inbound staging data tables

Because ContactCenter does not directly use ClaimCenter objects and has its own set of objects, you must use different database user names. For example, the `jdbcURL` parameter in the ContactCenter `config.xml` file sets a different database user from the `jdbcURL` parameter in the ClaimCenter `config.xml` file.

See also

- "Installing and Integrating ContactCenter with QuickStart" on page 12
- "Installing and Integrating ContactCenter in a Development Environment" on page 15

Installing and Integrating ContactCenter with QuickStart

A QuickStart installation enables you to immediately use and test the product. You can integrate ClaimCenter and ContactCenter in a QuickStart installation by doing the following:

- Step 1: Install ContactCenter
- Step 2: Load Sample Data
- Step 3: Integrate ContactCenter with ClaimCenter
- Step 4: Test the Integration

- Advanced Authentication with ContactCenter

Step 1: Install ContactCenter

Install ClaimCenter according to the instructions in the *ClaimCenter Installation Guide*, and then install ContactCenter as follows:

1. If you have not already done so, create an installation directory for ContactCenter. This guide uses ContactCenter as the directory name.
2. Unzip the ContactCenter ZIP file into the `ContactCenter` directory.
3. If you are not using a version control system, make a read-only copy of the ContactCenter directory. This copy enables you to recover quickly from accidental changes that can prevent ContactCenter from starting.
4. If you are reinstalling ContactCenter, drop the QuickStart database created by the previous installation. At a command prompt, navigate to `ContactCenter\bin` and enter the following command:
   ```
   gwab dev-dropdb
   ```
   Note: This command also drops any defined archive databases.
5. At a command prompt, navigate to `ContactCenter\bin` and enter the following command:
   ```
   gwab copy-starter-resources
   ```
   This command copies configuration resources, including `config.xml`, to a configuration module at `ContactCenter\modules\configuration`.
6. At the command prompt, enter the following command:
   ```
   gwab dev-start
   ```
   When the server has started, you see `*****ContactCenter ready*****` in the command window.
7. Open a browser, enter the URL `http://localhost:8280/ab`, and log in as the default superuser:
   - User name `su`
   - Password `gw`

Step 2: Load Sample Data

The installation contains sample data that you can use to learn application functionality. To import the data:

1. If you have not already done so, follow the instructions in the previous topic to start ContactCenter (`gwab dev-start`). Then navigate to `http://localhost:8280/ab`, and log in as the superuser (`su/gw`).
2. Press **ALT+SHIFT+T**.
3. Click the **Internal Tools** tab.
4. Choose **AB Sample Data**, and then click the **Load Sample Data** button.
5. Wait to see the completion messages above the button confirming that the sample data was imported.
6. To verify that the data loaded correctly, click **Return to ContactCenter** in the upper right heading area, then do the following:
   a. Click the **Administration** tab.
   b. Under the **Actions** button on the left, you see `Default Root Group`. If necessary, click `Default Root Group` to show its contents, and then click `Enigma Fire and Casualty` to see the list of users that was just loaded as sample data.
7. To use ContactCenter as a regular user, log out, then log in with user name `aapplegate` and password `gw`.

Step 3: Integrate ContactCenter with ClaimCenter

At this point, you can use both applications separately, but they are not integrated. To integrate the two applications you must configure services in the ClaimCenter version of Studio. If you are running ClaimCenter, stop the server and then do the following:

1. At a command prompt, navigate to the ClaimCenter `bin` directory, and then start Studio by entering the following command:

   ```
   gwcc studio
   ```

2. In Studio in the **Resources** pane on the left, expand **configuration** → **Plugins**.
3. Navigate to **gw** → **plugin**→ **addressbook** → **IAddressBookAdapter** so you can change the address book adapter used by the system.

   Note: By default, the system uses a demonstration address book adapter, `com.guidewire.cc.plugin.addressbook.internal.AddressBookDemoAdapter`. This adapter is a Java plugin that is not designed to work with ContactCenter.

4. Click **Remove**.

   The system prompts you to confirm a module copy. Click **Yes** to continue and **Yes** again to confirm the removal.

5. After the tab page refreshes, click **Add** → **Gosu** to create a new plugin.
6. In the **Class** field enter the following value:

   ```
   gw.plugin.ccabintegration.impl.CCAddressBookPlugin
   ```

7. In the **Parameters** section, click **Add** to add each of the following two parameters:

Name	Value
`password`	`gw`
`username`	`ClientAppCC`

   Note: Do not modify `IAddressBookAdapter.xml` directly. Doing so can invalidate your installation.

8. In Studio in the **Resources** pane on the left, expand **configuration** → **Web Services**.
9. Click **abintegration** to open this web service in the editor.
10. Verify that the port number in the **URL** field is `8280`:

    ```
    http://localhost:8280/ab/soap/IContactAPI?wsdl
    ```

11. If the port number is not correct, click **Edit** to the right of the **URL** field and change it to `8280`.
12. Ensure that ContactCenter is running, and then click **Refresh** to refresh the web service.
13. Save your changes and close Studio.
14. At this point, stop and restart ClaimCenter.

    a. Stop ClaimCenter.

       At the command prompt in which ClaimCenter is running, press **CTR L+C** to stop the batch process, and then enter the following commands:

       ```
       y
       gwcc dev-stop
       ```

    b. Start the ClaimCenter application:

       ```
       gwcc dev-start
       ```

Step 4: Test the Integration

Ensure that you have started both ContactCenter and ClaimCenter. You can verify that the two applications are integrated as follows:

1. When ClaimCenter is ready, open a browser window and enter the following ClaimCenter URL:
   `http://localhost:8080/cc/ClaimCenter.do`
2. Log in as a user who can create a contact in ClaimCenter, such as the sample user `ssmith` with password `gw`.
3. Click the **Address Book** tab, and then create a new contact.
4. Open a browser window and enter the following ContactCenter URL:
   `http://localhost:8280/ab/ContactCenter.do`
5. Log in as a user who can view or edit a contact in ContactCenter, such as the sample user `aapplegate` with password `gw`.
6. In ContactCenter, verify that you can search for and locate the contact you just created in ClaimCenter.

Advanced Authentication with ContactCenter

Typical authentication from ClaimCenter to ContactCenter uses the name and password defined in the `IAddressBookAdapter` plugin in Studio for the ClaimCenter application. If you want to hide authentication information in a separate file, you can implement a special plugin interface specifically for this purpose, called `ABAuthenticationPlugin`. Refer to “ABAuthenticationPlugin for ContactCenter Authentication” on page 246 in the *Integration Guide* for details.

See also

- “Integration Concepts and Prerequisites” on page 11
- “Installing and Integrating ContactCenter in a Development Environment” on page 15

Installing and Integrating ContactCenter in a Development Environment

This topic explains how to configure ClaimCenter to run with ContactCenter in a development environment in which you are using your own application and database server rather than QuickStart. You can use any supported combination of application and database server.

This example uses Tomcat and SQL Server. Because you need a separate JVM for each application, you must install and configure a second instance of Tomcat. The *ClaimCenter Installation Guide* has information on installing with other servers and databases that you can apply to a ContactCenter installation.

To integrate ClaimCenter and ContactCenter using Tomcat and SQL Server, do the following:

- Step1: Install ContactCenter
- Step 2: Load Sample Data
- Step 3: Integrate Both Applications
- Step 4: Deploy a New ClaimCenter WAR File
- Step 5: Test the Integration

Additionally, see Advanced Authentication with ContactCenter.

Step1: Install ContactCenter

These directions assume you have already installed ClaimCenter in a development environment according to the directions in the *ClaimCenter Installation Guide*. To install ContactCenter, do the following:

1. Install and unpack the contents of `ContactCenter.zip` into a directory on your system.
2. Set up the Java Virtual Machine (JVM) as described in “Configuring the Application Server” on page 13 in the *Installation Guide*.
3. Set up Apache Tomcat as described at “Configuring Apache Tomcat” under “Configuring the Application Server” on page 13 in the *Installation Guide*.
4. Modify the Tomcat installation for ContactCenter by editing the Tomcat `conf\server.xml` file, finding the definition for the HTTP connector, and setting the connector port to `8280`:

```
<Connector port="8280" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"/>
```

5. Create your ContactCenter SQL database.

   Use the same options for the ContactCenter database as those you set for ClaimCenter. For information about how to set up your database, see “Configuring the Database” on page 20 in the *Installation Guide*.

6. Modify the database connection in the ContactCenter file `modules\configuration\config\config.xml` to point to your new SQL database. Your file will look similar to the following:

```
<!-- SQL Server -->
<database name="ContactCenterDatabase" driver="dbcp" dbtype="sqlserver"
          autoupgrade="false" checker="false">
  <param name="jdbcURL"
         value="jdbc:sqlserver://HOSTNAME:1433;
                DatabaseName=ContactCenter;User=sa;Password=123"/>
  <param name="stmtPool.enabled" value="false"/>
  <param name="maxWait" value="30000"/>
  <param name="checker.threads" value="1" />
  <upgrade collectstorageinstrumentation="false">
    <versiontriggers dbmsperfinfothreshold="600" />
  </upgrade>
</database>
```

7. Deploy to Tomcat using the instructions at “Deploying to Tomcat” under “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*. In that topic, there are references to *cc* and *ClaimCenter* that you will need to translate to *ab* and *ContactCenter*. For example, the following steps found in that topic have been changed to use *ab* and *ContactCenter*:

   a. From the `ContactCenter/bin` directory run the following command to build the `war` file:

   ```
   gwab build-war
   ```

   This command creates the `ab.war` file in the `ContactCenter/dist/war` directory.

   b. Deploy the package to Tomcat by copying the `ab.war` file to the `webapps` directory in your Tomcat server.

   When Tomcat starts up, it automatically recognizes this new application and unpacks the `ab.war` file into a directory structure in `Tomcat\webapps`, such as `Tomcat\webapps\ab`. Each time you deploy a new copy of an `ab.war` file, delete the pre-existing directory structure created by the old `ab.war` file.

8. Start your application server as described in the “Deploying to Tomcat” topic under “Deploying ClaimCenter to the Application Server” on page 56 in the *Installation Guide*. For example:

```
startup -Dgw.server.mode=dev -Dcatalina.base="c:\tomcat"
```

9. Open a browser and navigate to `http://localhost:8280/ab`, and then log in as the superuser with:
   - User name `su`
   - Password `gw`

Step 2: Load Sample Data

The steps for loading sample data are the same as those described previously under "Step 2: Load Sample Data" on page 13.

Step 3: Integrate Both Applications

While you can view both applications separately, they are not integrated yet. You use ClaimCenter Studio to integrate these applications. Before starting, make sure that ContactManager is running so you can refresh its web services API in ClaimCenter Studio.

1. At a command prompt, navigate to the ClaimCenter `bin` directory, and then start Studio by entering the following command:

   ```
   gwcc studio
   ```

2. In Studio in the **Resources** pane on the left, expand **configuration** → **Web Services**.
3. Click **abintegration** to open this web service in the editor.
4. Verify that the port number in the **URL** field is correct. By default, with the base application setup, the port is `8280`:

   ```
   http://localhost:8280/ab/soap/IContactAPI?wsdl
   ```

5. If the port number is not correct, click **Edit** to the right of the **URL** field and change it.
6. Ensure that ContactCenter is running, and then click **Refresh** to refresh the web service.
7. In Studio in the **Resources** pane on the left, expand **configuration** → **Plugins**.
8. Navigate to **gw** → **plugin**→ **addressbook** → **IAddressBookAdapter** so you can change the address book adapter used by the system.

   Note: By default, the system uses a demonstration address book adapter, `com.guidewire.cc.plugin.addressbook.internal.AddressBookDemoAdapter`. This demo adapter is a Java plugin that is not designed to work with ContactCenter.

9. Click **Remove**.

   The system prompts you to confirm a module copy. Click **Yes** to continue and **Yes** again to confirm the removal.

10. After the tab page refreshes, press **Add** → **Gosu** to create a new plugin.
11. In the **Class** field enter the following value:

    ```
    gw.plugin.ccabintegration.impl.CCAddressBookPlugin
    ```

12. In the **Parameters** section, click **Add** to add each of the following two parameters:

Name	Value
`password`	`gw`
`username`	`ClientAppCC`

13. Save your changes and close Studio.

Step 4: Deploy a New ClaimCenter WAR File

At this point, you must generate a new ClaimCenter `.war` file and deploy it to your server. Start ContactCenter, then redeploy and start ClaimCenter as follows:

1. Deploy to Tomcat using the instructions at "Deploying to Tomcat" under "Deploying ClaimCenter to the Application Server" on page 56 in the *Installation Guide*. Unpack the `war` file into the `docbase` specified in your Tomcat installation.

   You must use a different Tomcat installation for ClaimCenter than for ContactCenter. For example, you might use `c:\tomcat` for ContactCenter and `c:\tomcat2` for ClaimCenter.

2. Start your application server as described in the "Deploying to Tomcat" topic under "Deploying ClaimCenter to the Application Server" on page 56 in the *Installation Guide*. For example:

   ```
   startup -Dgw.server.mode=dev -Dcatalina.base="c:\tomcat2"
   ```

3. Open a browser and navigate to `http://localhost:8080/cc`, then log in as the superuser with:
   - User name: `su`
   - Password: `gw`

4. When ClaimCenter is ready, open a browser window and enter the following URL:

   ```
   http://localhost:8080/cc/ClaimCenter.do
   ```

   Note: Your ports might be different.

Step 5: Test the Integration

Ensure that you have started both ContactCenter and ClaimCenter. You can verify that the two applications are integrated as follows:

1. When ClaimCenter is ready, open a browser window and enter the following ClaimCenter URL:

   ```
   http://localhost:8080/cc/ClaimCenter.do
   ```

2. Log in as a user who can create a contact in ClaimCenter, such as the sample user `ssmith` with password `gw`.
3. Click the **Address Book** tab, and then create a new contact.
4. Open a browser window and enter the following ContactCenter URL:

   ```
   http://localhost:8280/ab/ContactCenter.do
   ```

5. Log in as a user who can view or edit a contact in ContactCenter, such as the sample user `aapplegate` with password `gw`.
6. In ContactCenter, verify that you can search for and locate the contact you just created in ClaimCenter.

Advanced Authentication with ContactCenter

See "Advanced Authentication with ContactCenter" on page 15.

See also

- "Integration Concepts and Prerequisites" on page 11
- "Installing and Integrating ContactCenter with QuickStart" on page 12

chapter 3

Management and Configuration Resources

This topic provides information about managing Address Book data and ContactCenter itself, and it discusses configuration files and resources available to you for contact management.

This topic includes:

- "Managing ContactCenter Information" on page 20
- "Configuring ClaimCenter Contact Management Features" on page 22
- "Configuring an Integrated ClaimCenter and ContactCenter" on page 22

See the *ClaimCenter System Administration Guide* for detailed administration information, much of which is applicable to ContactCenter. For example:

- For information on directory structure and key directories like `module` and `configuration`, see "The ClaimCenter Installation Directory" on page 11 in the *System Administration Guide*.
- For a list of command-line build tools like `regen-java-api` and `regen-dictionary`, see "Build Tools" on page 13 in the *System Administration Guide*.
- For information on configuration settings, see "The Application Server config.xml" on page 17 in the *System Administration Guide*.
- For information on defining the server environment, see "Defining the Application Server Environment" on page 18 in the *System Administration Guide*.
- For information on setting up logging, see "Logging Overview" on page 35 in the *System Administration Guide*.
- For detailed reference information on administrative commands like `table_import` and `import_tools`, see "ClaimCenter Administrative Commands" on page 169 in the *System Administration Guide*.

Managing ContactCenter Information

Typically, you use ClaimCenter screens to manage the vendors in the ContactCenter Address Book. For example, you can use the ClaimCenter **Address Book** tab to add a new contact to the Address Book. Additionally, you can access Address Book data from ClaimCenter by using screens for managing or adding claim contacts, such as the **New Claim Wizard**. To work with the Address Book from ClaimCenter, you must install both ContactCenter and ClaimCenter and integrate them as described in “Integrating ClaimCenter with ContactCenter”, on page 11.

It is also possible to manage the Address Book in ContactCenter by logging in directly to ContactCenter, if you have set up users in ContactCenter with the appropriate roles. Other reasons to log in to ContactCenter are to manage its users or to manage the server. For example, you might want to assign a user the User Admin role to enable that user to add and delete users in ContactCenter.

ContactCenter has a similar structure to the ClaimCenter user interface, with tabs at the top of the screen and an **Action** button and a menu on the left for navigation.

Contacts Tab

The **Contacts** tab enables you to:

- Create a new person, company, or place to store in the Address Book.
- Edit and delete any of these entities.
- Search for any of these entities by using search criteria, such as contact subtype and location.

Administration Tab

By using the ContactCenter **Administration** tab, you can manage roles, permissions, login name and password, and group membership for your ContactCenter users. These users log in directly to ContactCenter and are not necessarily the same as the ClaimCenter users who you might have authorized to perform Address Book changes.

> **Note:** To see this tab, you must be logged in as an administrator, such as the default super user with login name `su` and password `gw`.

- The **Actions** button on the left enables you to create new ContactCenter users and groups.
- Also on the left is a hierarchical list of all your groups and users.
- Additionally, you can choose the following items from the menu on the left:
  - **Search for Users:** Search for users by role, location, username, and first and last name.
  - **Search for Groups:** Search for groups by name and type.
  - **Roles:** Add and delete roles, add permissions to and remove permissions from roles, and assign roles to ContactCenter users. For more information, see “Overview of Contact Security” on page 101.
  - **Regions:** Add, delete, and edit the current regions for ContactCenter contacts. For more information on regions, see “Regions” on page 407 in the *System Administration Guide*.
  - **Event messages:** Suspend and resume event messages sent by ContactCenter and restart the messaging engine. For more information, see “Monitoring and Managing Event Messages” on page 67 in the *System Administration Guide*.
  - **Script parameters**: View existing ContactCenter script parameters. To create new script parameters, you must use ContactCenter Studio. For more information, see “Script Parameters” on page 111 in the *Configuration Guide*.

Internal Tools and Server Tools Tabs

As described in “Step 2: Load Sample Data”, on page 13, you can log in to ContactCenter as the super user. You can then press **ALT+SHIFT+T** to access the **Internal Tools** tab and load sample data.

You can also click the **Server Tools** tab and perform administrative tasks like viewing logs, running batch processes, viewing work queue information, and so on.

Of particular interest is **Batch Process Info**. This screen enables you to run batch processes manually, without having to wait for the scheduled processes to run. You can start and view information about ContactCenter batch processes, including writer processes for work queues. This information includes the batch process **Name**, **Description**, **Status**, **Last Run** time, **Next Scheduled Run** time, and scheduling information. To start a batch process, click **Run** in the **Action** column for the batch process.

There are several batch processes you might find useful to run from this screen:

Batch Process	Description
AB Contact Score Aggregator	Processes service provider review scores sent by ClaimCenter. For more information, see "Service Provider Performance Reviews" on page 121 in the *Application Guide*.
AB Geocode Writer	Geocodes addresses. For more information, see "Proximity Search Using Geocoding" on page 89.
Update MatchSetKey	If incorrect values have been loaded into the production database or `definitive-match-config.xml` has been changed, running this batch process updates the `MatchSetKey` of every contact in the database. **Note: Update MatchSetKey** processes all contact records in the system and can impact performance. Guidewire recommends that you run this process during off hours and that you run multiple threads of the worker process. For more information on loading database values, **MatchSetKey**, and `definitive-match-config.xml`, see "Configuring Definitive Match in ContactCenter" on page 44.

These batch process jobs have entries in the file `scheduler-config.xml` that you can uncomment and set to run at specific times. For an example, see "Step 4: Schedule Geocoding" on page 98.

Running Large Batch Jobs Can Impact Query Performance

If you run a large batch job on more than ten percent of the addresses in the ContactManager database, query performance can be impacted. To improve query performance, after running the batch job, your database administrator must run a series of database statistics statements.

To run these statements

1. At a command prompt, navigate to `ContactManager/admin/bin`.
2. Run the following command:

```
maintenance_tools -password password -getdbstatisticsstatements | grep -i ab_abaddress > file-name
```

   In this command, *`password`* is your administration password and *`file-name`* is the file where you want the output of the command to be saved. The `grep` command is available on UNIX or Linux systems. On Windows systems, you can run `grep` in a UNIX or Linux emulator like Cygwin.
3. Open the output file and run the database statistics statements in it.

For more information, see "Configuring Database Statistics" on page 53 in the *System Administration Guide*.

See also

- "Using Server and Internal Tools" on page 151 in the *System Administration Guide*
- "Managing ContactCenter Information" on page 20
- "Configuring ClaimCenter Contact Management Features" on page 22
- "Configuring an Integrated ClaimCenter and ContactCenter" on page 22

Configuring ClaimCenter Contact Management Features

ClaimCenter uses the `Contact` data model to store information about claim contacts and does not require integration with ContactCenter to do so. As with any entities in the ClaimCenter application, you can extend the `Contact` data model to add fields or subtypes. You can also associate contact roles with entities, such as the claimant on a claim or a judge on a legal matter. Additionally, you can create relationships between contacts, such as an employer who is related to an employee. The following table lists the configuration resources associated with the ClaimCenter contact management features:

Resource	Description
`ContactBidiRel`	A typelist containing typecodes that define both sides of contact relationships.
`ContactRel`	A typelist containing typecodes that name contact relationships.
`contact-relationship-config.xml`	A file that sets up relationships using `ContactBidiRel` and `ContactRel` typecodes.
`ContactRole`	A typelist specifying contact roles such as agent, underwriter, claimant, and so on.
`ContactRoleCategory`	A typelist that defines groups of related contacts, such as all vendor roles. The system uses this typelist to filter the parties involved in a claim.
`entityroleconstraints-config.xml`	A file that defines the associations between contacts and roles.
`security-config.xml`	A file that defines security configurations for centralized contacts in the Address Book. The `ContactPermissions` block is used together with the `typecode` definitions in the file.
`SystemPermissionType`	A typelist that defines custom permissions.
`config.xml`	A configuration file that contains a number of parameters related to configuring contacts.

Some entities have a **ContactRoleMessage** value specified for the **Entity Name** property. This value specifies how the entity name appears in role validation error messages. The **ContactRoleMessage** value is accessible only by the application's internal application code. You can configure the value with the Studio editor.

See also

- "Managing ContactCenter Information" on page 20
- "Configuring an Integrated ClaimCenter and ContactCenter" on page 22

Configuring an Integrated ClaimCenter and ContactCenter

If you integrate ClaimCenter and ContactCenter, you can use ContactCenter to centrally manage your contacts. You configure the integration between ClaimCenter and ContactCenter in Studio, as described in "Integrating ClaimCenter with ContactCenter" on page 11.

By default, the ClaimCenter **IAddressBookAdapter** uses a demonstration address book adapter, `com.guidewire.cc.plugin.addressbook.internal.AddressBookDemoAdapter`. This adapter is a Java plugin used for demonstration purposes and does not work with ContactCenter.

To enable ClaimCenter to work with ContactCenter, Guidewire provides the Gosu plugin `ccabintegration.CCAddressBookPlugin`. To use this plugin, you disable the Java demo plugin and replace it with the Gosu plugin. See "Step 3: Integrate ContactCenter with ClaimCenter" on page 14, for information on how to set up the plugin.

Alternatively, you can replace the demo plugin with your own custom contact manager plugin. For information on implementing your own custom plugin, see "Address Book Integration" on page 299 in the *Integration Guide*.

When you configure a contact management option in ClaimCenter, you must consider your ContactCenter integration as well. For example, if you create a new contact subtype in ClaimCenter that you want to centrally manage, you must create a corresponding subtype in ContactCenter.

You can use the following resources to configure your ContactCenter installation:

Resource	Description
`ContactBidiRel`	A typelist containing typecodes that define both sides of contact relationships.
`ContactRel`	A typelist containing typecodes that name contact relationships.
`abcontact-relationship-config.xml`	A file that sets up relationships using `ContactBidiRel` and `ContactRel` typecodes.
`ContactRole`	A typelist specifying roles for contacts.
`ContactRoleCategory`	A typelist that defines groups of related contacts, such as all vendor roles.
`config.xml`	A file in which you update the `AdditionalPrimaryABContactSearchCriteria` parameter with the fields you want to search for.
`search-config.xml`	A file in which you add a new `Criterion` element to the `ABContactSearchCriteria` definition.
`ContactSearchDV.pcf`	A user interface file to which you add any input fields necessary to capture a contact search value.
`definitive-match-config.xml`	A file that defines exact matches used for searching. By default, your subtype inherits these values from its parent type.
`potential-match-config.xml`	A file used when definitive matching fails. The system uses these fields to find potential matches during a search.

After extending the subtype in both ClaimCenter and ContactCenter, you edit the following ClaimCenter configuration files to map the new type to ContactCenter:

`ab-to-cc-data-mapping.xml`	Maps ContactCenter entities to ClaimCenter entities. Add new subtypes to this mapping file and exclude or remap mismatched fields on existing subtypes.The `cc-to-ab-data-mapping.xml` file is for the reverse mapping.
`cc-to-ab-data-mapping.xml`	Maps ClaimCenter entities to ContactCenter entities. Add new subtypes to this mapping file and exclude or remap mismatched fields on existing subtypes.

ContactCenter does not require these mapping files.

You can extend your contacts in a number of ways:

- Add a new field to an entity, as described in “Extending Contacts and Search with an Array” on page 76.
- Add a new `Contact` subtype, as described in “Example of Adding a Contact Subtype” on page 31.
- Add an array and extend search. For an example that shows both how to extend a contact and how to search by your extension, see “Extending Contacts and Search with an Array” on page 76.

You can also use the Geocoding feature with the addresses returned by ContactCenter and with local ClaimCenter addresses. See “Proximity Search Using Geocoding” on page 89 to learn more about geocoding and proximity searches.

See also

- “Managing ContactCenter Information” on page 20
- “Configuring an Integrated ClaimCenter and ContactCenter” on page 22
- “Extending the Contact Data Model” on page 25

chapter 4

Extending the Contact Data Model

This topic describes how to extend the contact data model to add your own subtypes or attributes. It supplies some common configuration examples you can use to configure your own installation.

This topic includes:

Understanding the Contact Data Model

A `Contact` is the ClaimCenter data model entity used in contact management. This entity has fields for first name, last name, phone number, email address, and so forth. For many vendor contacts it is necessary to maintain an ID for tax reporting. The `Contact` entity tracks this data as well.

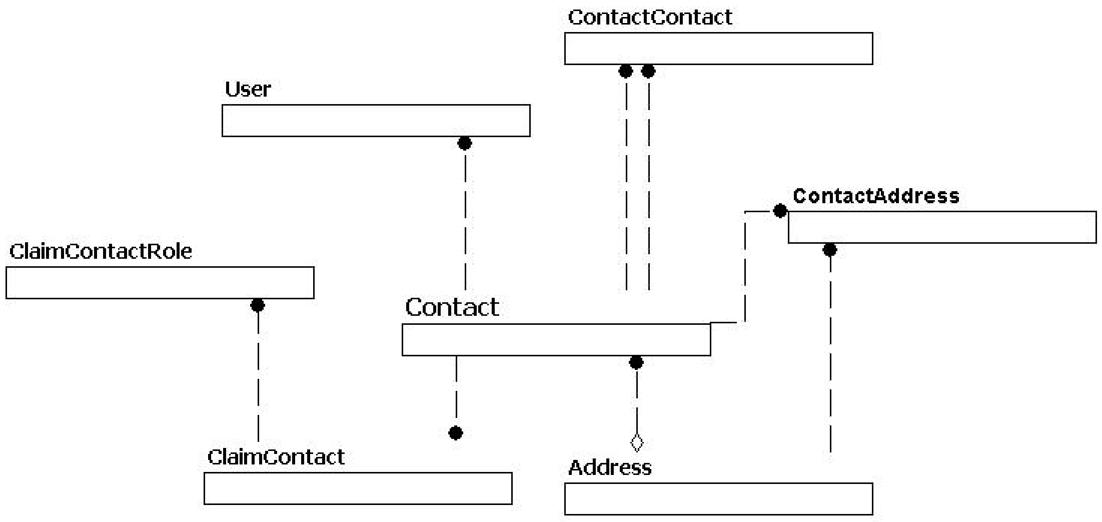
The `Contact` entity has subtypes for various types of contacts, like `Person` and `Company`. These subtypes in turn have additional subtypes, like `Adjudicator`, `CompanyVendor`, and so on, as described later in this topic.

The physical location, the address, of a `Contact` is not maintained in the `Contact` entity. Instead, the `Contact` entity references another entity, the `Address` entity, which maintains the contact's street or mailing address. A `Contact` can reference multiple `Address` entities: a primary address and other secondary addresses.

Contacts have relationships with other contacts. For example, a `PersonVendor`, such as a doctor or attorney, might be employed by a particular `Company`. The `ContactContact` entity maintains data about the relationships a contact can have with other contacts.

When extending or manipulating the contact management data model, you often need to consider entities other than the `Contact` entity. For example, to extend functionality related to contact addressing, you work with the

Address entity. The following entity-relationship diagram depicts the primary entities involved in managing contacts:



Each Contact has, optionally, one or more related ContactAddress entities. The ContactAddress is an edge table that represents the relationship between contacts and Address entities. The Address entity stores the physical address for a contact. The ContactContact entity models a hierarchical relationship between two contacts.

The ClaimContact entity represents the link between a contact and a claim or an exposure. A number of other entities in the system, such as Evaluation, Note, Negotiation, Check, and Activity, also reference this entity.

> **Note:** As you can see from the previous figure, each User entity is connected to the Contact entity. Specifically, the User entity has a foreign key to the UserContact subtype. The purpose of this foreign key is to provide a simple way to store useful data for each ClaimCenter user, like name, email address, phone number, and address data.

The Contact subtypes are Person, Company, and Place, each of which has its own subtypes. The base ClaimCenter configuration provides the following Contact entity hierarchy:

```
Contact
  Person
    Adjudicator
    PersonVendor
      Attorney
      Doctor
    UserContact
  Company
    CompanyVendor
      AutoRepairShop
      AutoTowingAgcy
      LawFirm
      MedicalCareOrg
  Place
      LegalVenue
```

Just as with other ClaimCenter entities, you can extend and customize this hierarchy to match your business needs.

ClaimCenter and ContactCenter Contact Entity Hierarchy

Both the ClaimCenter and ContactCenter data models group and classify contact data as hierarchies. By default, ClaimCenter and ContactCenter have nearly identical contact entity hierarchies. Typically, all the ClaimCenter contact types appear in ContactCenter except that the ContactCenter entities have the prefix AB (for **Address Book**).

ContactCenter has more subtypes to support multiple Guidewire applications. The following table shows the entire ClaimCenter hierarchy and the supporting hierarchy in ContactCenter:

ClaimCenter	ContactCenter
`Contact`	`ABContact`
`Person`	`ABPerson`
`Adjudicator`	`ABAdjudicator`
`PersonVendor`	`ABPersonVendor`
`Attorney`	`ABAttorney`
`Doctor`	`ABDoctor`
`UserContact`	`ABUserContact`
`Company`	`ABCompany`
`CompanyVendor`	`ABCompanyVendor`
`AutoRepairShop`	`ABAutoRepairShop`
`AutoTowingAgcy`	`ABAutoTowingAgcy`
`LawFirm`	`ABLawFirm`
`MedicalCareOrg`	`ABMedicalCareOrg`
`Place`	`ABPlace`
`LegalVenue`	`ABLegalVenue`

Each ClaimCenter contact entity has an `AddressBookUID` field. This field uniquely identifies the corresponding ContactCenter entity. When a ClaimCenter contact is linked to a ContactCenter contact, the field has a value. Otherwise, the field is `null`.

Guidewire designed the data models for both ClaimCenter and ContactCenter to be compatible. If you make a contact-related extension to the ClaimCenter data model, you typically extend the ContactCenter data model as well to reflect the change. You must extend both applications if you want contact information that is captured, stored, and used in one application to be available to the other.

Mapping entities ensures only that contact records are stored as the correct entity or subtype in both ClaimCenter and ContactCenter. The mappings do *not* link records between the two application. Linking must be done explicitly by the user, as described in “Synchronizing Contacts between ContactCenter and ClaimCenter”, on page 39.

Note: For ContactCenter only, do not modify the `PublicID` column of an entity after it has been created.

See also

- “Extending Contacts” on page 27
- “Working with Contact Mapping Files” on page 29
- “Example of Adding a Contact Subtype” on page 31

Extending Contacts

The following topics provide information to be aware of before extending the contact hierarchy. As discussed in the topic ClaimCenter and ContactCenter Contact Entity Hierarchy, when you extend ClaimCenter, you typically need to extend ContactCenter as well.

This topic covers the following information:

- General Guidelines for Extending Contacts
- Deciding Whether to Create a Subtype
- Limitations on Extension Configuration

General Guidelines for Extending Contacts

With some restrictions, you can customize the contact entity hierarchy by adding subtypes or custom attributes. You cannot make any of the following hierarchy modifications:

- Delete or move the root entity `Contact`
- Delete or move the three subtypes: `Person`, `Company`, and `Place`
- Create a contact entity that is a peer to `Contact`
- Create a contact entity that is a parent to `Person`, `Company`, or `Place`

You can create a peer to `Person`, `Company`, or `Place`, and you can modify these three entities. A peer entity to one of these three entities is a subtype of `Contact`. You can also create a new subtype and modify other subtypes, such as `Adjudicator`, `PersonVendor`, `Attorney`, `AutoRepairShop`, `MedicalCareOrg`, and so on.

If you have integrated ClaimCenter with ContactCenter, you typically extend both the ClaimCenter and the ContactCenter hierarchies so that they mirror each other. Only if you have a contact subtype that does not require central management would you create that type just in ClaimCenter and not in ContactCenter.

ClaimCenter contains two mapping resources for the purpose of extending the hierarchy—the `ab-to-cc-data-mapping.xml` and `cc-to-ab-data-mapping.xml` files. See "Synchronizing Contacts between ContactCenter and ClaimCenter", on page 39 for information on the functions associated with these files.

The matching and searching functions for contacts require each subtype to have a collection of fields that makes it unique. See "Synchronizing Contacts between ContactCenter and ClaimCenter", on page 39 for more information about matching and synchronizing.

Deciding Whether to Create a Subtype

Consider carefully before manipulating your contact hierarchy. A new subtype inherits the attributes and matching behavior of its supertype. Create a new subtype only if you need to treat one set of contacts differently from another. As much as possible, restrict the number of subtypes you need to accomplish a task.

Each time you create a new subtype, you need to modify PCF files to support both creating it and searching for it. Additionally, you might need to make supporting modifications to claim screens or other screens that reference contacts.

For example, suppose you want to distinguish among different service providers, such as inspectors, call centers, document storage, physicians, chiropractors, and dentists. You can create a single subtype `Provider` that has a code attribute to distinguish among the types. Alternatively you can create a `Provider` type and subtype that with different subtypes: `Physician`, `Chiropractor`, `Dentist`, `Inspector`, `CallCenter`, `DocStorage`, and so on. Using subtypes in this case would require more work in many more files.

Note: After making any data model modification, do the following:

1. Stop the application in which you have made the data model modifications.
2. Optionally regenerate the data dictionary for the application by running `regen-dictionary` from the command line.

   This step ensures that your changes work before rebuilding the application.
3. Regenerate the SOAP and Java APIs for the application that has changed.
4. If you have made a data model change to ContactCenter, refresh the ContactCenter plugin in ClaimCenter.
5. Start the application and confirm your changes.

For a series of steps that walk you through stopping, regenerating files for, and restarting ClaimCenter and ContactCenter, see "Step 3: Edit the Mapping Files" on page 32.

For examples of how to make modifications to the class hierarchy, see the following topics:

- “Example of Adding a Contact Subtype” on page 31
- “Extending Contacts and Search with an Array” on page 76

Limitations on Extension Configuration

There are limitations on the kinds of things you can do with contact type extensions. Your subtype can inherit from only a single parent type—multiple inheritance is not supported. For example, you could represent an adjudication practice with a single owner-proprietor either as a `Company` or as a `Adjudicator`. You cannot create a subtype that inherits the attributes of both types. To create the subtype you want, you select one type or the other and subtype it as something like `Practice`. Then, you add the attributes that are lacking from the other type because of single inheritance.

Unless you restrict the visibility of some subtypes through configuration, they can still appear in search results because searches return all subtypes. This behavior has special implications if your installation is integrated with ContactCenter. For example, suppose you configure ClaimCenter to make `Adjudicator` not a choice in the user interface. However, in the ContactCenter database there are a number of `ABAdjudicator` contact types. Searching the **Address Book** for a `Person` will return `ABAdjudicator` matches if they exist in ContactCenter. For information about restricting access to contact subtypes, see “Securing Contact Information”, on page 101.

See also

- “ClaimCenter and ContactCenter Contact Entity Hierarchy” on page 26
- “Working with Contact Mapping Files” on page 29
- “Example of Adding a Contact Subtype” on page 31

Working with Contact Mapping Files

If your ClaimCenter application is integrated with ContactCenter, you must use the mapping files to connect extensions. The mapping files match ClaimCenter entity types to ContactCenter entity types so that both applications recognize entity types the same way. If you extend your ClaimCenter data model, the system does not automatically extend the ContactCenter data model. To maintain data consistency, you make the extension to ContactCenter as well and then map the entities to each other. In the ClaimCenter installation, you use the files `ab-to-cc-data-mapping.xml` and `cc-to-ab-data-mapping.xml` to map ClaimCenter and ContactCenter entities to each other.

> **Note:** There might be situations in which you want to add a contact subtype to one application and not the other. If you have a contact subtype that does not require central management, you can create that type in ClaimCenter only and not in ContactCenter.

Each entity mapping entry maps a source entity to a target entity. For example, the following is an extract from `ab-to-cc-data-mapping.xml`—the file that defines how ContactCenter entities map to ClaimCenter:

```
<EntityMapping source="ABAdjudicator" target="Adjudicator"/>
```

In the example, the source is a ContactCenter entity mapped to a ClaimCenter entity. This mapping specifies that a sync from **Address Book** to ClaimCenter causes the record stored for this instance of `ABAdjudicator` in ContactCenter to overwrite the analogous `Adjudicator` record in ClaimCenter.

The file `cc-to-ab-data-mapping.xml` contains a mirror of this definition with the source and target reversed:

```
<EntityMapping source="Adjudicator" target="ABAdjudicator"/>
```

In the base product, these files contain default mappings for each of the base contact entities and subtypes.

Entities with Different Field Names

Typically, when you add contact entities or extend an existing entity, you use the same entity and field names for both sides. However, you might find it necessary to add an entity with one field name in ContactCenter and a different name in ClaimCenter. Alternatively, you might want to exclude an entity's field from the mapping. To accomplish these results, you use the regular `EntityMapping` element, but you add the `FieldMapping` and `MapperProperty` subelements to it.

The following example illustrates the use of these elements.

In `cc-to-ab-mapping.xml`, you specify the following mapping:

```
<EntityMapping source="Contact" target="ABContact">
  <FieldMapping source="PublicID"
                mapperClassName="gw.api.util.mapping.NullFieldMapper"/>
  <FieldMapping source="AddressBookUID"
                mapperClassName="gw.api.util.mapping.NameTranslatingFieldMapper">
    <MapperProperty name="newFieldName" value="LinkID"/>
  </FieldMapping>
  <FieldMapping source="AutoSync"
                mapperClassName="gw.api.util.mapping.NullFieldMapper"/>
</EntityMapping>
```

In the opposite `ab-to-cc-mapping.xml` file, you specify the following mapping:

```
<EntityMapping source="ABContact" target="Contact">
  <FieldMapping source="PublicID"
                mapperClassName="gw.api.util.mapping.NullFieldMapper"/>
  <FieldMapping source="LinkID"
                mapperClassName="gw.api.util.mapping.NameTranslatingFieldMapper">
    <MapperProperty name="newFieldName" value="AddressBookUID"/>
  </FieldMapping>
  <FieldMapping source="InitiateAutoSync"
                mapperClassName="gw.api.util.mapping.NullFieldMapper"/>
</EntityMapping>
```

In the example above, the `PublicID` field is excluded in the mapping from `Contact` to `ABContact` and from `ABContact` to `Contact`. The `AddressBookUID` on the `Contact` entity maps to the `LinkID` on the `ABContact` entity. Conversely, the `LinkID` on the `ABContact` entity maps to the `AddressBookUID` on the `Contact` entity.

The exclusions and field name mappings are accomplished by setting the `mapperClassName` to one of two classes.

- `gw.api.util.mapping.NullFieldMapper`—To exclude a field from mapping, set `mapperClassName` to this class.
- `gw.api.util.mapping.NameTranslatingFieldMapper`—If you want to map entities with different names, set set `mapperClass` name to this class and supply a `MapperProperty` subelement to indicate the field name on the target entity. `MapperProperty` requires a `name` attribute set to `newFieldName` and a `value` attribute specifying the field name on the target entity.

  Note: If you are extending a Contact subtype, you use the subtype in the `fkentity` field rather than the `Contact` supertype.

Subtypes inherit the field mappings defined on types higher up in the hierarchy. So, for example, excluding the `LinkID` field on `Contact` has the effect of excluding this mapping on all the `Contact` subtypes.

See also

- "ClaimCenter and ContactCenter Contact Entity Hierarchy" on page 26
- "Extending Contacts" on page 27
- "Example of Adding a Contact Subtype" on page 31

Example of Adding a Contact Subtype

This topic describes how to extend contacts with a new subtype.

> **Note:** Before beginning this example, if you have not done so already, follow the instructions for installing and integrating ContactCenter in "Integrating ClaimCenter with ContactCenter", on page 11. In particular, after installing ContactCenter, you need to import sample data as described in "Step 2: Load Sample Data" on page 13. Additionally, you must install the Gosu plugin as described in "Step 3: Integrate ContactCenter with ClaimCenter" on page 14.

The following topics walk you through adding a new subtype:

- Step 1: Add the Subtype to ClaimCenter
- Step2: Add the Subtype to ContactCenter
- Step 3: Edit the Mapping Files
- Step 4: Modify the ClaimCenter Address Book User Interface
- Step 5: Modify the ContactCenter User Interface
- Step 6: Restart Both Applications and Test

Step 1: Add the Subtype to ClaimCenter

Extend the entity in ClaimCenter:

1. If necessary, start ClaimCenter Studio.

   Open a command window, navigate to the `bin` directory of ClaimCenter, and enter the following command:

   ```
   gwcc studio
   ```

2. In the **Resources** tree on the left, navigate to **Data Model Extensions** → **extensions**, then right-click, and choose **New** → **Other file**.

3. Type the following file name and press Enter:

   ```
   Interpreter_Ext.eti
   ```

4. In the editor, enter the following code:

   ```
   <?xml version="1.0"?>
   <subtype
     entity="Interpreter_Ext"
     supertype="PersonVendor"
     desc="Interpreter"
     displayName="Interpreter">
     <column
       name="InterpreterSpecialty"
       desc="Interpreter's specialty"
       type="varchar">
       <columnParam
         name="size"
         value="30"/>
     </column>
   </subtype>
   ```

5. Save the file.

Step2: Add the Subtype to ContactCenter

Extend the entity from ContactCenter:

1. If necessary, start ContactCenter Studio.

   Open a command window, navigate to `ContactCenter/bin`, and enter the following command:

   ```
   gwab studio
   ```

2. In the **Resources** tree on the left, navigate to **Data Model Extensions** → **extensions**, then right-click, and choose **New** → **Other file**.

3. Type the following file name and press Enter:
   ```
   ABInterpreter_Ext.eti
   ```
4. In the editor, enter the following code:
   ```
   <?xml version="1.0"?>
   <subtype
     entity="ABInterpreter_Ext"
     supertype="ABPersonVendor"
     desc="Interpreter"
     displayName="Interpreter">
     <column
       name="InterpreterSpecialty"
       desc="Interpreter's specialty"
       type="varchar" >
       <columnParam
         name=”size”
         value="30"/>
     </column>
   </subtype>
   ```
5. Save the file.

Step 3: Edit the Mapping Files

Map the ClaimCenter `Interpreter_Ext` entity to ContactCenter's `ABInterpreter_Ext` entity.

1. In ClaimCenter Studio, in the **Resources** tree on the left, open the **Other Resources** folder and click `cc-to-ab-data-mapping.xml` to open it in an editor.
2. After the line `<EntityMapping source="UserContact" target="ABUserContact"/>` add the following entity mapping:
   ```
   <EntityMapping source="Interpreter_Ext" target="ABInterpreter_Ext"/>
   ```
3. Save your work.
4. Edit the `ab-to-cc-data-mapping.xml` resource and, after the line `<EntityMapping source="ABPolicyCompany" target="Company"/>`, add the following entity mapping:
   ```
   <EntityMapping source="ABInterpreter_Ext" target="Interpreter_Ext"/>
   ```
5. Save your work.
6. To pick up the data model changes, stop both applications, regenerate the APIs and data dictionaries, refresh the plugin, and restart, as follows:
   a. If necessary, stop ClaimCenter.

      At the command prompt in which ClaimCenter is running, press **CTRL+C** to stop the batch process, and then enter the following commands:
      ```
      y
      gwcc dev-stop
      ```
   b. Regenerate the ClaimCenter APIs:
      ```
      gwcc regen-java-api
      gwcc regen-soap-api
      ```
   c. Optionally regenerate the ClaimCenter data dictionary:
      ```
      gwcc regen-dictionary
      ```
   d. At the command prompt in which ContactCenter is running, press **CTRL+C** to stop the batch process, and then enter the following commands:
      ```
      y
      gwab dev-stop
      ```
   e. Regenerate the ContactCenter APIs:
      ```
      gwab regen-java-api
      gwab regen-soap-api
      ```

f. Optionally regenerate the ContactCenter data dictionary:

```
gwab regen-dictionary
```

g. Start the ContactCenter application:

```
gwab dev-start
```

h. In ClaimCenter Studio, refresh the ContactCenter plugin as follows:

In the **Resources** tree on the left, expand the **Web Services** node and click **abintegration** to open an editor for this web service. Next, click the **Refresh** button near the top of the editor, to the right of the **URL** field. If prompted to make a copy of the resource for editing, click `OK` and then click **Refresh** again.

i. Start the ClaimCenter application:

```
gwcc dev-start
```

Step 4: Modify the ClaimCenter Address Book User Interface

In this topic you modify the ClaimCenter user interface so you can create contacts with the new subtype. The PCF files you must edit depend on the subtype. For example, `Person` subtypes use a different set of PCF files from `Company` subtypes. You can discover the specific PCF files you need by examining those used by other subtypes.

In most instances, you can use the `mode` attribute of a PCF file to include your subtype. Using `mode` enables your subtype to use the preexisting PCF file. In other cases, you might need to create a new PCF file, particularly if your subtype has unique fields that causes the screen layout to be significantly different.

Note: This topic shows how to add the new subtype only to the **Address Book** interface. There are also screens and menus for claimcontacts that enable addition of a contact of this subtype directly to a claim. For more information, see "Example of a Constrained Picker" on page 72.

Edit the Actions Menu on the Address Book Tab

To add support to the Actions menu for the new Interpreter_Ext subtype

1. In ClaimCenter Studio, add a display key for the new contact type.

   a. Click **Display Keys** in the **Resources** tree on the left.

   b. Navigate to the **Web** → **NewContactMenu** folder, right-click it, and choose **Add**.

   c. Name the key `Web.NewContactMenu.Interpreter_Ext`, for the default value enter `Interpreter`, and click **OK** to save it.

2. In the **Resources** tree, expand **Page Configuration (PCF)** → **addressbook** and click **AddressBookMenuActions** to edit this PCF file.

3. Select the `New Vendor` submenu and use the following steps to add an `Interpreter` menu item to it:

   a. Drag a **MenuItem** widget from the **Toolbox** on the right to the `New Vendor` submenu and drop it below the **Medical Care Organization** item.

   b. Enter the following values for the **Basic properties**:

action	`NewAddressBookContact.go(entity.Interpreter_Ext)`
id	`AddressBookMenuActions_Interpreter_Ext`
label	`displaykey.Web.NewContactMenu.Interpreter_Ext`

Edit the Mode of the Detail View

1. In Studio, under **Page Configuration (PCF)** → **addressbook**, right-click the `AddressBookContactBasicsDV.Person` PCF file to display the context menu, and choose **Change mode**.

2. In the **Change Mode** dialog box, add `Interpreter_Ext` to the list of modes:

   ```
   Person|PersonVendor|Adjudicator|UserContact|Doctor|Attorney|Interpreter_Ext
   ```

   Adding this mode updates the Shared Section mode of the `InputSetRef` so the detail view will work with the new subtype.

Add the Subtype to the Contact Picker Menus

Take the following steps to ensure that your new subtype appears on the contact picker menus:

1. In Studio, expand **Page Configuration (PCF)** → **shared** → **contacts** and click **NewContactPickerMenuItemSet** so you can edit it.
2. Select the `NewContactPickerMenuItemSet_NewVendor` submenu.
3. The **Advanced** property **visible** contains a set of tests for contact type. Scroll all the way to the right and add the following test for the `Interpreter_Ext` type:

   ```
    or requiredContactType == entity.Interpreter_Ext
   ```

   After you have added the code, the **visible** property will have the following contents:

   ```
   requiredContactType == entity.Contact or requiredContactType == entity.Company or
   requiredContactType == entity.Person or requiredContactType == entity.PersonVendor or
   requiredContactType == entity.CompanyVendor or requiredContactType == entity.AutoRepairShop or
   requiredContactType == entity.AutoTowingAgcy or requiredContactType == entity.Doctor or
   requiredContactType == entity.MedicalCareOrg or
   requiredContactType == entity.Interpreter_Ext
   ```

4. Drag a **MenuItem** widget from the **Toolbox** on the right and drop it under the **Medical Care Organization** menu item.

   This action puts the new menu item in the **New Vendor** submenu, which has **id** `NewContactPickerMenuItemSet_NewVendor`.
5. Click the new menu item and set the following properties to make it an `Interpreter` item:

action	`NewAddressBookContactPopup.push(entity.Interpreter_Ext, parentContact)`
id	`NewContactPickerMenuItemSet_Interpreter_Ext`
label	`displaykey.Web.NewContactMenu.Interpreter_Ext`
visible	`requiredContactType.isAssignableFrom(entity.Interpreter_Ext)`

Edit the Input Sets

Update the input sets used by the detail view so you can create an `Interpreter` contact on the new contact page and see the new `Interpreter Specialty` field:

1. In Studio, click **Display Keys** in the **Resources** tree, then navigate to **Web** → **ContactDetail**.
2. Right-click **ContactDetail** and click **Add** to add a key named `Web.ContactDetail.Interpreter` with a default value of `Interpreter`.
3. Right-click the new **Interpreter** node and click **Add** to add a key named `Web.ContactDetail.Interpreter.InterpreterSpecialty` with a default value of `Interpreter Specialty`.
4. In the **Resources** tree, expand **Page Configuration (PCF)** → **addressbook**.
5. Right-click **addressbook**, then choose **New** → **PCF File**.
6. For **File type** choose **Input Set**, then enter the name `AddressBookInterpreterAdditionalInfo`.
7. For **Mode** enter `Interpreter_Ext`, then click **OK**.

   This modal PCF file supports the addition of the **Specialty** field to the basic `PersonVendor` input set. If you were not adding a new field, you could skip this step.

8. Select the newly created input set, click the **Required Variables** tab, click **+** to add a variable, and enter the following values:

name	`interpreter`
type	`entity.Interpreter_Ext`

9. Add an `Input` widget to the new resource and set the following properties:

editable	`true`
id	`InterpreterSpecialty`
label	`displaykey.Web.ContactDetail.Interpreter.InterpreterSpecialty`
required	`false`
value	`interpreter.InterpreterSpecialty`

10. Also under **Page Configuration (PCF)** → **addressbook**, right-click **AddressBookAdditionalInfoInputSet.PersonVendor** and choose **Change mode**.

11. In the **Change Mode** dialog, add `Interpreter_Ext` to the list of modes:

    `PersonVendor|Attorney|Doctor|`**`Interpreter_Ext`**

    Adding the `Interpreter_Ext` mode updates the Shared Section mode of the **InputSetRef** so the detail view will display the **InputSetRef** you add in the next steps.

12. Click the **AddressBookAdditionalInfoInputSet.PersonVendor** resource to edit it, and then drag an **InputSetRef** from the **Toolbox** and drop it under the **InputSetRef** for **AddressBookDoctorAdditionalInfoInputSet**.

13. Set the properties to the following values:

def	`AddressBookInterpreterAdditionalInfoInputSet(contact as entity.Interpreter_Ext)`
mode	`contact typeis Interpreter_Ext ? "Interpreter_Ext" : null`

    This statement enables the **Specialty** field to display in the **Additional Info** section when the contact type is `Interpreter_Ext`.

Step 5: Modify the ContactCenter User Interface

This topic describes how to modify the ContactCenter user interface to enable users to create contacts with the new subtype.

Edit the Actions Menu on the ContactCenter Contacts Tab

To add support to the Actions menu for the new ABInterpreter_Ext subtype

1. In ContactCenter Studio, add a display key for the new contact type.
   a. Click **Display Keys** in the **Resources** tree on the left.
   b. Navigate to the **Web** → **NewContactMenu** folder, right-click it, and choose **Add**.
   c. Name the key `Web.NewContactMenu.ABInterpreter_Ext`, for the default value enter `Interpreter`, and click **OK** to save it.
2. In the **Resources** tree, expand **Page Configuration (PCF)** → **contacts** and click **ContactsMenuActions** to edit this PCF file.
3. Drag a **MenuItem** widget from the **Toolbox** on the right and drop it under the **Doctor** menu item.

   This action puts the new menu item in the **Vendor** submenu, which has **id** `ContactsMenuActions_PersonVendorMenuItem`.

4. Click the new **MenuItem** and enter the following values for the **Basic properties** of this `Interpreter` menu item:

action	`NewContact.go(entity.ABInterpreter_Ext)`
id	`ContactsMenuActions_ABInterpreter_ExtMenuItem`
label	`displaykey.Web.NewContactMenu.ABInterpreter_Ext`

Edit the Mode of the ContactCenter Detail View

1. In Studio, under **Page Configuration (PCF)** → **contacts** → **basics**, right-click the `ContactBasicsDV.ABPerson` PCF file and choose **Change mode**.
2. In the **Change Mode** dialog, add `ABInterpreter_Ext` to the list of modes, as follows:

```
ABPerson|ABPersonVendor|ABAdjudicator|ABUserContact|ABDoctor|
ABAttorney|ABPolicyPerson|ABInterpreter_Ext
```

Adding this mode updates the Shared Section mode of the `InputSetRef` so the detail view will work with the new subtype.

Add the Subtype to the ContactCenter Contact Picker Menus

Take the following steps to ensure that your new subtype appears on the contact picker menus:

1. In Studio, expand **Page Configuration (PCF)** → **contacts** and click **NewContactPickerMenuItemSet** so you can edit it.
2. Select the submenu labeled `Vendor`, which has the following **id** property:

```
NewContactPickerMenuItemSet_PersonVendorMenuItem
```

3. Drag a new **MenuItem** widget from the **Toolbox** and drop it under the `Doctor` menu item.
4. Set the following properties for this new `Interpreter` menu item:

action	`NewContactPopup.push(entity.ABInterpreter_Ext, parentContact)`
id	`NewContactPickerMenuItemSet_ABInterpreter_ExtMenuItem`
label	`displaykey.Web.NewContactMenu.ABInterpreter_Ext`
visible	`requiredContactType.isAssignableFrom(entity.ABInterpreter_Ext)`

Add an Input Set for the Interpreter Specialty Field

In this topic, you add a new modal input set for the `Interpreter Specialty` field so this field can be used by the `ContactBasicsDV.ABPerson` detail view

To add the new input set

1. In Studio, click **Display Keys** in the **Resources** tree, and then navigate to **Web** → **ContactDetail**.
2. Right-click **ContactDetail** and click **Add**, and then add a key with the name `Web.ContactDetail.ABInterpreter_Ext` and a default value of `Interpreter`.
3. Right-click the new **ABInterpreter_Ext** node, click **Add**, and add the key `Web.ContactDetail.ABInterpreter_Ext.InterpreterSpecialty` with a default value of `Interpreter Specialty`.
4. In the **Resources** tree, expand **Page Configuration (PCF)** → **contacts** → **basics**.
5. Right-click **basics** and choose **New** → **PCF file**.
6. For **File type** choose **Input Set**, then enter the name `ABPersonVendorSpecialty`.
7. For **Mode** enter `ABInterpreter_Ext`, and then click **OK**.

This modal PCF file supports the addition of the **Specialty** field to the basic `PersonVendor` input set. If you were not adding a new field, you could skip this step.

8. Select the newly created input set, click the **Required Variables** tab, click **+** to add a variable, and enter the following values:

name	`person`
type	`entity.ABPerson`

9. Drag an **Input** widget from the **Toolbox** on the right to the new resource and set the following properties:

editable	`true`
id	`InterpreterSpecialty`
label	`displaykey.Web.ContactDetail.ABInterpreter_Ext.InterpreterSpecialty`
required	`false`
value	`(person as ABInterpreter_Ext).InterpreterSpecialty`

Add a New Mode to ABPersonVendorInputSet

The **ContactBasicsDV.ABPerson** detail view uses the modal input set **ABPersonVendor.ABPersonVendor|ABAttorney|ABDoctor** to add some standard vendor information for an `ABPersonVendor`, an `ABAttorney`, or an `ABDoctor`. Adding the `ABInterpreter_Ext` mode to this input set will make these fields available when creating a new Interpreter. However, this input set is not a separate modal PCF file, but is coded inside `ABPersonVendorInputSet.pcf`. For this reason, it appears in the **Resources** tree as a subnode of **ABPersonVendorInputSet**, and it is not possible to change the mode of the PCF file in Studio.

You have two options for making this set of fields available to the detail view:

- Edit the XML source for the `ABPersonVendorInputSet.pcf` file in an external editor to add the new `ABInterpreter_Ext` mode.
- In Studio under **Page Configuration (PCF)** → **contacts**, right-click the **basics** node and click **New** → **PCF File**. Choose the file type **InputSet** and name it `ABPersonVendor`, and then specify the mode `ABInterpreter_Ext` and click **OK** to create a new file at the same level as **ABPersonVendorInputSet**. Then open **ABPersonVendor.ABPerson-Vendor|ABAttorney|ABDoctor** in Studio and copy its widgets to the new input set, updating properties as necessary to make the new modal file work.

This topic describes how to use the first technique.

To edit and update ABPersonVendorInputSet.pcf

1. Under **Page Configuration (PCF)** → **contacts** → **basics**, right-click **ABPersonVendorInputSet** and choose **Find in Explorer**.

   The Explorer opens the following path:

   ```
   ContactCenter\modules\ab\config\web\pcf\contacts\basics\ABPersonVendorInputSet.pcf
   ```

2. Copy `ABPersonVendorInputSet.pcf` from this location to the following path:

   ```
   ContactCenter\modules\configuration\config\web\pcf\contacts\basics
   ```

   Note: If necessary, create the subfolders needed for this path.

3. Open the new copy of `ABPersonVendorInputSet.pcf` in an external editor.

4. To the `InputSet` with `id="ABPersonVendorInputSet",` add the mode `ABInterpreter_Ext` as follows:

   ```
   <InputSet
     id="ABPersonVendorInputSet"
     mode="ABPersonVendor|ABAttorney|ABDoctor|ABInterpreter_Ext">
   ```

5. Save the file.

6. In Studio, check to see that the modal file name under **ABPersonVendorInputSet** in the **Resources** tree has been updated to include the new mode, as follows:
   ABPersonVendor.ABPersonVendor|ABAttorney|ABDoctor|ABInterpreter_Ext

Step 6: Restart Both Applications and Test

1. Stop ClaimCenter and ContactCenter, then rebuild and deploy both applications.
2. Log in to ClaimCenter as a user who can create new contacts. For example, log in as user `su` with password `gw`.
3. Choose the **Address Book** tab, then choose **Actions** → **New Vendor** → **Interpreter** to see the **New Interpreter** dialog. Note that **Interpreter Specialty** is listed in the **Additional Info** section.
4. Enter enough information to create an interpreter contact, and then click **Update Address Book** to save the new contact.
5. Click **Search** in the **Sidebar** on the left, then on the **Search** page pick **Interpreter** from the dropdown and search for the new contact.
6. Select the new contact found by **Search** and verify that the entry is correct and that you can edit and update it.
7. Log in to ContactCenter as a user who can create new contacts. For example, log in as user `su` with password `gw`.
8. From the **Sidebar** on the left, choose **Actions** → **New Person** → **Vendor** → **Interpreter** to see the **New Interpreter** dialog. Note that **Interpreter Specialty** is listed in the **Additional Info** section.
9. Enter enough information to create an `Interpreter` contact, and then save the new contact.
10. Click **Search** in the **Sidebar** on the left, then on the **Search** page pick **Interpreter** from the dropdown and search for the new contact.
11. Select the new contact found by **Search** and verify that the entry is correct and that you can edit and update it.

See also

- “Understanding the Contact Data Model” on page 25
- “Extending Contacts” on page 27
- “Working with Contact Mapping Files” on page 29

chapter 5

Synchronizing Contacts between ContactCenter and ClaimCenter

This topic covers information you need to know regarding synchronizing contact information between ClaimCenter and the ContactCenter Address Book. This topic applies only if you are managing contacts in an environment where ContactCenter is integrated with ClaimCenter.

This topic includes:

Configuring Automatic Synchronization

You must have the correct permissions to be able to create new, centrally-managed contacts in the **Address Book**. Creating a contact in the ClaimCenter **Address Book** automatically places the contact in ContactCenter. After you create a centrally managed contact, other ClaimCenter users can use it for contact fields throughout the application. Additionally, both ClaimCenter users and ContactCenter users with the proper permissions can edit the contact.

The synchronizing mechanism ensures data consistency between ClaimCenter and ContactCenter. This mechanism updates centrally-managed contacts appropriately when a contact changes in either application. The synchronizing mechanism has both ContactCenter-specific controls and ClaimCenter-specific controls.

> **Note:** Automatic synchronization of contacts is an optional feature that you can enable in your environment.

ContactCenter Synchronizing Controls

The ContactCenter entity `ABContact` has a single boolean attribute, `InitiateAutoSync`, that sets the synchronizing behavior on a contact instance. When this value is set, changing a contact's data causes the system to send an `ABContactChanged` event. There is an **Event Fired** rule for **ContactAutoSync** that creates a message when this event fires.

> **Note:** The attribute `InitiateAutoSync` is not available in the user interface. You can configure the user interface to expose this attribute if you want your users to be able to control automatic synchronization of specific contacts.

The expectation is that you want to broadcast changes of contact data to a listening application—in this case, ClaimCenter. The **Messaging** resource in the Studio **Resource** pane contains a message destination that corresponds to the `ABContactChanged` event. This destination has an **ID** of `80` and a **Name** of `ABContact AutoSync Broadcast`, and it uses the **Transport Plugin** `autoSyncBroadcastTransport`. This destination has the following additional settings:

Poll Interval	`10000`
Initial Retry Interval	`100`
Max Retries	`3`
Retry Backoff Multiplier	`2`
Number Sender Threads	`1`
Chunk Size	`100000`
Shutdown Timeout	`30000`
Enabled	`Yes`
Events	`ABContactChanged`

The broadcast is done through the **Plugin AutoSyncBroadcastTransport**, which uses the Gosu class `gw.plugin.autosync.impl.AutoSyncBroadcastTransport`. This plugin class calls the ClaimCenter built-in web service `IContactAutoSyncAPI` to notify ClaimCenter of the change.

ClaimCenter Synchronizing Controls

Your users can change data for centrally managed contacts in either ContactCenter or ClaimCenter. Like the ContactCenter entity `ABContact`, the ClaimCenter `Contact` entity has an attribute called `AutoSync` that controls whether contacts are automatically synchronized. For any `Contact` instance, the `AutoSync` value can be one of three codes:

`Allow`	Allow the contact to be synchronized automatically.
`Disallow`	Do not allow the contact to be synchronized.
`Suspended`	The contact was synchronized in the past, but synchronizing is not currently allowed.

> **Note:** The attribute `AutoSync` is not available in the user interface. You can configure the user interface to expose this attribute if you want your users to be able to control automatic synchronization of specific contacts.

To process each change, ClaimCenter uses a work queue named `ContactAutoSyncWorkQueue`. You can configure work queues in `work-queue.xml`.

Open this file from ClaimCenter Studio as follows:

1. Start Studio. At a command prompt, navigate to `ClaimCenter\bin` and enter the following command:

   ```
   gwcc studio
   ```

2. In the **Resources** tree on the left, click **Other Resources**, and then click **work-queue.xml** to open it in an external editor.

   Note: After you edit any of the configuration files described in this topic, you must rebuild and redeploy ClaimCenter for the changes to take effect.

The following code shows the default settings for this queue:

```
<work-queue
    workQueueClass="com.guidewire.pl.system.contactautosync.ContactAutoSyncWorkQueue"
    progressinterval="600000">
  <worker instances="1" maxpollinterval="0"/>
</work-queue>
```

You can change the attributes for a work queue and add additional worker instances in `work-queue.xml`.

The comments at the beginning of the `work-queue.xml` file document the attributes for a work queue and provide some guidelines for adding worker instances. For general information on configuring work queues, see “Configuring Distributed Work Queues” on page 133 in the *System Administration Guide*.

You can have your work queues process changes on a schedule or in real time as they happen. The `InstantaneousContactAutoSync` attribute in the `config.xml` file controls this behavior. To open this file in an external editor, start Studio, and in the **Resources** tree, expand **Other Resources** and click **config.xml**. You must rebuild and redeploy ClaimCenter for changes to take effect.

By default, the `InstantaneousContactAutoSync` attribute is set to `true` and causes the work queue to process changes as they are received:

```
<param name="InstantaneousContactAutoSync" value="true"/>
```

You can configure a scheduled update by setting `InstantaneousContactAutoSync` to `false`. You can then set a schedule for `ContactAutoSync` batch processing in the `scheduler-config.xml` file. To open this file in an external editor, start Studio, and in the **Resources** tree, expand **Other Resources** and click **scheduler-config.xml**. You must rebuild and redeploy ClaimCenter for changes to take effect.

By default, `ContactAutoSync` is commented out in this file, and you need to uncomment it to schedule the process. The default, commented-out setting looks like this:

```
<!-- Contact Auto Sync batch process should run every day at 3 am -->
<!--
  <ProcessSchedule process="ContactAutoSync">
    <CronSchedule hours="3"/>
  </ProcessSchedule>
-->
```

If you set `InstantaneousContactAutoSync` to `false` and do not schedule a `ContactAutoSync` batch process, the work queue does not run and the system does not process changes. If you set `InstantaneousContactAutoSync` to `false` and you have a schedule set, changes process immediately, and then the scheduled `ContactAutoSync` process also runs at the appointed time.

For general information on administering and configuring scheduled batch processes, see “Scheduling Batch Processes and Distributed Work Queues” on page 143 in the *System Administration Guide*.

Not All Contacts Synchronize

In the default configurations of ClaimCenter and ContactCenter, not all contacts synchronize. You must determine which contacts you want your users to be able to synchronize. For example, you would not necessarily want to synchronize `Person` and `ABPerson` contacts, but you might want to synchronize all `Vendor` and `ABVendor` contacts. Or, you might want to synchronize only contacts that meet particular criteria.

You can use the Preupdate rules in Studio for both ContactCenter and ClaimCenter to set the auto synchronize values.

For example, there are two predefined **ContactPreupdate** rules in ClaimCenter. To see them:

1. Start Studio. At a command prompt, navigate to `ClaimCenter\bin` and enter the following command:
   ```
   gwcc studio
   ```
2. In the **Resource** pane on the left, navigate to **Rule Sets** → **ContactPreupdate**.
3. The two predefined rule sets are:
   - **COP01000 - Update Check Address**
   - **COP02000 - All Vendors should sync**

Initiating Manual Synchronization from ClaimCenter

You can initiate synchronization manually in ClaimCenter. Centrally managed contacts have a status message that informs you when they are not synchronized. You can then choose if you want to synchronize the Address Book data, which copies the contact data from ContactCenter to ClaimCenter.

See also
- "Synchronizing Contact Attributes" on page 42
- "Configuring Link Functionality" on page 43

Synchronizing Contact Attributes

Synchronizing from ContactCenter to ClaimCenter updates the fields in a ClaimCenter contact with the centralized ContactCenter data. By default, the system overwrites any exportable field and ignores all relationships. You can instruct the system not to overwrite specific fields or to include or not include specific relationships by editing the ClaimCenter `contact-sync-config.xml` file from ClaimCenter Studio.

IMPORTANT Do not configure the `contact-sync-config.xml` file in ContactCenter.

Open this file from ClaimCenter Studio as follows:

1. Start Studio. At a command prompt, navigate to `ClaimCenter\bin` and enter the following command:
   ```
   gwcc studio
   ```
2. In the **Resources** tree on the left, click **Other Resources**, and then click **contact-sync-config.xml** to open it in an external editor.

The `contact-sync-config.xml` file contains one or more `ContactSyncEntityConfig` elements. Each element refers to a ClaimCenter entity. The entity attribute must be either a valid subtype of `Contact` or `Contact` itself.

A `ContactSyncEntityConfig` element can contain one or more `IgnoreProperty` subelements. These subelements define the `name` of a field to ignore when synchronizing data from ContactCenter. The `appliesToSubtypes` attribute determines whether the system uses this setting with subtypes of the `entity`. This attribute has a default value of `true`. Therefore, if you omit `appliesToSubtypes` in an `IgnoreProperty` element, the setting applies to all subtypes of the entity. The resulting behavior is that the subtypes do not copy the property, and they cannot override the setting.

A `ContactSyncEntityConfig` element can contain one or more `IncludeRelationship` or `ExcludeRelationship` elements, which cause the system to include or exclude specific contact relationships during a copy. For example, you might include a relationship on one contact type but exclude it on its child subtypes. The default behavior for an entity's relationships is that the system ignores them. You need to specify `ExcludeRelationship` only if you want it to apply to the element's subtypes or you want to override a setting of `IncludeRelationship` in a parent type.

The relationship elements take the `contactBidiRelCode` and `appliesToSubtypes` attributes. The `contactBidiRelCode` attribute is required and must be a valid typecode in the `ContactRel` typelist. The `appliesToSubtypes` attribute determines whether the system uses this setting with subtypes of the `entity`. This attribute has a default value of `true`. Therefore, if you omit `appliesToSubtypes` in an `IncludeRelationship` or `ExcludeRelationship` element, the relationship defined in the current type applies to all its subtypes.

The following example illustrates some uses of the `ContactSyncEntityConfig` element:

```
<ContactSyncEntityConfig entity="Contact">
  <IgnoreProperty name="PrimaryLanguage"/>
  <IgnoreProperty name="AddressBookFingerprint"/>
  <IgnoreProperty name="AddressBookUID"/>
  <IgnoreProperty name="OrganizationType"/>
  <IgnoreProperty name="SpecialtyType"/>
  <IncludeRelationship contactBidiRelCode="primarycontact"/>
</ContactSyncEntityConfig>

<ContactSyncEntityConfig entity="Person">
  <IncludeRelationship contactBidiRelCode="guardian" appliesToSubtypes="false"/>
  <IncludeRelationship contactBidiRelCode="employer" appliesToSubtypes="false"/>
  <ExcludeRelationship contactBidiRelCode="primarycontact" appliesToSubtypes="false"/>
</ContactSyncEntityConfig>

<ContactSyncEntityConfig entity="Adjudicator">
  <IncludeRelationship contactBidiRelCode="employer"/>
</ContactSyncEntityConfig>
```

When synchronizing `Contact` and its subtypes, ClaimCenter ignores the five fields listed and does *not* copy them from ContactCenter data. The system includes the `primarycontact` relationship for `Contact` and all subtypes of `Contact` unless the subtype explicitly overrides the setting with an `ExcludeRelationships` setting.

As described under “ClaimCenter and ContactCenter Contact Entity Hierarchy” on page 26, `Person` is a subtype of `Contact`. When synchronizing `Person` and its subtypes, ClaimCenter ignores the five properties set in the `ignoreProperty` attributes of `Contact`. Additionally, the system includes the relationships `guardian` and `employer` for the `Person` type only, and not for its subtypes. Finally, the system overrides the `Contact` setting for `primarycontact` and excludes this relationship only for `Person`, but not for its subtypes.

`Adjudicater` is a subtype of `Person`. When synchronizing `Adjudicator`, ClaimCenter ignores the five properties set in the `ignoreProperty` attributes of `Contact`. Addtionally, the system includes both the `primarycontact` relationship defined in `Contact` and the `employer` relationship defined in `Adjudicator`.

IMPORTANT Typically, you include only relationships that appear on the **Contact Basics** panel of the ClaimCenter **Parties Involved** page. In particular, avoid zero-or-more relationship types. These types can grow quickly and can result in performance issues as the system pulls down many contacts from the ContactCenter Address Book.

See also

- “Configuring Automatic Synchronization” on page 39
- “Configuring Link Functionality” on page 43

Configuring Link Functionality

In ClaimCenter, you can choose to link a local contact to the Address Book. For example, on the **Parties Involved** screen of a claim, you click the name of a contact and, below it in the **Basics** tab, click **Link**. After you click **Link**, ClaimCenter queries the ContactCenter database for a matching contact. Depending on the type of match it finds, ClaimCenter attempts to link the contact from ContactCenter. A successful link causes ClaimCenter to copy the fields from the ContactCenter entry into the local contact in ClaimCenter, overwriting any mapped data previously defined on that contact. After a successful link, the contact becomes subject to synchronizing rules.

Linking causes the system to build a query that uses the fields defined in the ContactCenter `definitive-match-config.xml` and `potential-match-config.xml` files. The system first searches for a definitive match. If it cannot locate a definitive match, the system searches for a potential match. ClaimCenter responds based on the type of match found. Following are the possible match types ClaimCenter can find after a user clicks the **Link** button:

Match Type	Description
`plausible`	A plausible match is a record in ContactCenter that, according to the current match configuration, exactly matches the definitive fields and at least some of the potential match fields. The system links the contact automatically. If the system is an exact match on the definitive and potential fields, the system also sets the contact's status to synced—synchronized with the Address Book.
`implausible`	The query returns one or more potential matches, meaning that the potential fields match but not the definitive fields. Then, ClaimCenter displays a list of matches to the user. The user can select one contact or **Cancel** the link operation. If the user attempting the link has the appropriate permissions, the system adds the contact to the ContactCenter Address Book and then both links the contact and copies it.
`incompatible`	The query cannot locate a contact with a contact type that matches. For example, a `Company` must match an `ABCompany`. ClaimCenter presumes that the contact does not exist in ContactCenter.

While querying for a match, the system downcasts through the contact hierarchy. For example, if you are linking a `PersonVendor`, the system returns `PersonVendor` and the subtypes `Attorney`, `Doctor`, and `UserContact`.

Configuring Definitive Match in ContactCenter

When attempting to link contacts, ClaimCenter first uses a set of definitive fields to search the Address Book. This definitive match field set is a collection of fields that, taken together, constitute a unique ID for Address Book entries, the `MatchSetKey` property.

IMPORTANT The `MatchSetKey` property of `ABContact` defines a unique index and match criteria. It is calculated automatically before every insert and update to ABContact. When you use staging tables to load a set of data, the `MatchSetKey` values are calculated automatically. However, as defined in `definitive-match-config.xml`, if one of the fields used for the keys is encrypted, but the `MatchSetKey` column is not encrypted, incorrect values are stored.

To prevent this problem from occurring, do not run the encryption process on the staging tables. If you want to encrypt any fields that store `MatchSetKey` values, set the fields and the `MatchSetKey` to be encrypted. When the server starts, it encrypts the fields automatically. For more information on setting a field to be encrypted, see "Overriding Data Type Attributes" on page 240 in the *Configuration Guide*.

Additionally, you can run the Update MatchSetKey batch process to fix incorrect values in the ContactCenter database, as described at "Internal Tools and Server Tools Tabs" on page 20.

The ContactCenter file `definitive-match-config.xml` contains one or more `DefinitiveMatchSet` elements that define the match fields. You can find this file in Studio under **Other Resources** in the **Resource** pane.

Each element has the following format:

```
<DefinitiveMatchSet name="setname">
  <AppliesToSubtype subtype="address_book_subtype" />
  <ExactMatchField path="fieldname"/>
</DefinitiveMatchSet>
```

Each `DefinitiveMatchSet` must have a unique `name`. You associate a set with one or more ContactCenter contact types through the `AppliesToSubtype` subelement. A contact type can appear in only one `DefinitiveMatchSet` element. Each `DefinitiveMatchSet` includes one or more `ExactMatchField` elements representing the contact field to match between the contacts.

In a query for a match, the system uses the nearest branch on the contact type hierarchy. For example, if you are linking an attorney contact to ContactCenter, then the candidate entity type is `Attorney`. If there is a `DefinitiveMatchSet` for both `ABPerson` and its subtype, `ABAttorney`, the system uses the `ABAttorney` set.

Note: All ContactCenter data types have the prefix `AB`.

The following `DefinitiveMatchConfig` elements are part of the base configuration:

```
<DefinitiveMatchConfig
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="match-config.xsd">

  <DefinitiveMatchSet name="TaxID">
    <AppliesToSubtype subtype="ABCompany" />
    <AppliesToSubtype subtype="ABPerson" />
    <ExactMatchField path="taxid"/>
  </DefinitiveMatchSet>

  <DefinitiveMatchSet name="FullName" >
    <AppliesToSubtype subtype="ABAdjudicator" />
    <ExactMatchField path="FirstName"/>
    <ExactMatchField path="LastName"/>
  </DefinitiveMatchSet>

</DefinitiveMatchConfig>
```

After a user requests to link a claim contact to an Address Book contact, the system interprets these elements as follows:

- For any claim contact of subtype `Company` or `Person`, search the Address Book for contacts of subtype `ABCompany` and `ABPerson` with a matching tax ID. The `Adjudicator` subtype is excluded from this search because it has a separately defined `DefinitiveMatchSet`.
- For any claim contact of subtype `Adjudicator`, search the **Address Book** for contacts of subtype `ABAdjudicator` whose `FirstName` and `LastName` fields match those of the claim contact.

You can customize the properties used for definitive matching by editing `definitive-match-config.xml`. To open this file in an external editor, start ContactCenter Studio and click the file's node under **Other Resources** in the **Resource** pane.

It is important that you select the appropriate fields for definitive matches. When querying for a contact, the system can fail due to an anomalous type match. Anomalous type matches occur when the `ExactMatchField` elements are a match, but the records have a different contact type (for example a query for a `Person` finds a `Company`). If the query returns an anomalous match, ClaimCenter cannot link the contacts. To make incompatible matches less likely, consider contact type when you specify a definitive match.

Configuring Potential Matches in ContactCenter

When definitive matching fails, the system uses potential matching to identify the Address Book contacts that might be matches. The fields the system uses to determine a potential match are configured in ContactCenter's `potential-match-config.xml` file. You open this file from ContactCenter Studio by clicking its node under **Other Resources** in the **Resource** pane.

This file contains one or more `PotentialMatchSet` elements. These elements have the following format:

```
<PotentialMatchSet subtype="abtype" >
  <PartialMatchField path="field"/>
  <PartialMatchField path="stringfield" matchType="type" />
</PotentialMatchSet>
```

Unlike `DefinitiveMatchSets`, you do not name `PotentialMatchSet` elements. Instead, the `PotentialMatchSet` element takes a `subtype` attribute that identifies the contact type to match. The value of the `subtype` attribute must be either `ABContact` or a subtype of `ABContact`.

When attempting to find a match for a contact, the system evaluates each `PotentialMatchSet` against the contact in the order in which the element is defined. The system uses the first matching `PotentialMatchSet` whose subtype matches the candidate's subtype.

If the system cannot find a `PotentialMatchSet` for the specific subtype, the system uses the nearest branch on the contact hierarchy to do the search. For example, if you are linking an attorney contact to ContactCenter, then the candidate entity type is `Attorney`. If there is a `PotentialMatchSet` for `ABPerson` but not for its subtype, `ABAttorney`, the system uses the `ABPerson` set.

Each `PotentialMatchSet` includes one or more `PartialMatchField` elements that identify the field to match. For string fields, you can use an optional `matchType` attribute. Valid values for `matchType` are `equals` and `startsWith`. The default value is `equals`.

Following are two `PotentialMatchConfig` element configurations:

```
<PotentialMatchSet subtype="ABContact" >
  <PartialMatchField path="name"/>
</PotentialMatchSet>
<PotentialMatchSet subtype="ABPerson">
  <PartialMatchField path="firstname" matchType="startsWith"/>
  <PartialMatchField path="lastname"/>
</PotentialMatchSet>
```

If definitive matching fails to link a claim contact to the Address Book, the system can use these definitions to find potential matches, as follows:

- For any claim contact of subtype `Contact`, search the Address Book for contacts of subtype `ABContact` to find a contact record that has a `name` that matches.
- For any claim contact of subtype `Person`, search the Address Book for `ABPerson` contacts with the following criteria:
  - The same `lastname` as the claim contact
  - A `firstname` string that matches the beginning of the claim contact's first name

You can accept the default configuration for potential matching, or you can customize the properties used as needed.

See also

- "Configuring Automatic Synchronization" on page 39
- "Synchronizing Contact Attributes" on page 42

chapter 6

Configuring Roles and Relationships

This topic describes how to customize and configure contact roles and relationships in a Guidewire application. It supplies some common examples you can use to configure your own installation.

This topic includes:

The roles discussed in this topic apply only to contacts. Another type of role is the user role, which is a collection of permissions that enable a user to view or edit various ClaimCenter or ContactCenter objects.

- For information on user roles and security related to accessing contacts in the Address Book, see "Securing Contact Information" on page 101.
- For information on roles in general and on roles used in ClaimCenter, see "Roles" on page 406 in the *Application Guide*.

Adding Contact Roles

The contacts in the system have associations with your business entities that depend on the functions they perform for the business entities. For example, a `Person` can be associated with a claim as the insured party. In ClaimCenter, these associations are defined as roles that contacts perform for claims, evaluations, incidents, exposures, negotiations, matters, and policies. For example, a defense attorney performs a role in a matter that involves litigation on a claim.

The ClaimCenter base configuration has a number of predefined contact roles. Your business might require unique associations that would necessitate adding your own roles.

Note: Do not add a contact role to a custom entity. ClaimCenter does not support this configuration.

To add a contact role, you define an association between a role and a contact type, and then associate that role with an entity

1. Add the role to the `ContactRole` typelist as described in "Defining Contact Roles" on page 48.
2. Define the following two types of constraints for the role in the `entityroleconstraints-config.xml` file, as described in "Defining Role Constraints" on page 49:
   - A *contact role constraint* indicates the contact subtype, such as `Person` or `Company`, to which the role applies. See "Defining Contact Role Constraints" on page 49.
   - An *entity role constraint* defines which roles an entity can use. It can also optionally specify whether a role is required, exclusive, or prohibited, and conditions that apply to use of the role by the entity. See "Defining Entity Role Constraints" on page 50.
3. After completing work on the typelist and the XML file, you must rebuild and redeploy ClaimCenter. Regenerating the Data Dictionary is also recommended, and is required for production deployments.

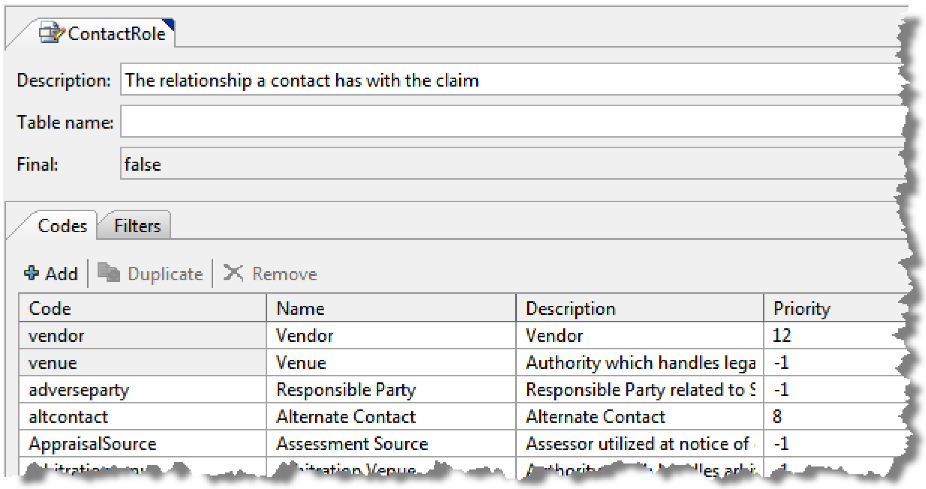
   For a series of steps that walk you through stopping, regenerating files for, and restarting ClaimCenter, see step 8 of "Example of Adding a New Contact Role" on page 54.

Defining Contact Roles

The `ContactRole` typelist defines the available roles. For example, the following contact role definition creates a `mattermanager` role:

Code	Name	Description
`mattermanager`	`Legal Case Manager`	`Legal Case Manager`

Many of the codes in the `ContactRole` typelist, such as `activityowner`, `claimant`, `insured`, `vendor`, and `venue`, are internal to the application and must not be removed. The internal codes are listed in the typelist editor with a **Code** field that has a gray background. The editor does not let you remove these typecodes.



Description:	The relationship a contact has with the claim
Table name:	
Final:	false

Code	Name	Description	Priority
vendor	Vendor	Vendor	12
venue	Venue	Authority which handles lega	-1
adverseparty	Responsible Party	Responsible Party related to S	-1
altcontact	Alternate Contact	Alternate Contact	8
AppraisalSource	Assessment Source	Assessor utilized at notice of	-1

You can also look up the `ContactRole` typelist in the Data Dictionary. The **Internal** column is marked **true** for the codes that defined as internal.

> **IMPORTANT** In general, when removing typecodes from any typelist, first ensure that the code is not in use in the system in a PCF file or referenced by another typelist.

Contact roles exist in relation to entities. You define which entities use a role by referencing the role from the `entityroleconstraints-config.xml` file, as described later in "Defining Role Constraints" on page 49.

To view or edit the typelist, open it from ClaimCenter Studio

1. At a command prompt, navigate to `ClaimCenter\bin` and enter the following command:

   ```
   gwcc studio
   ```

2. In the **Resources** tree on the left, expand **Typelists** and click **ContactRole**.

Defining Role Constraints

You configure the relationships between entities and roles in the `entityroleconstraints-config.xml` file. In this file, you define the `Contact` subtypes that can use a role. Additionally, you configure which contact roles are available to which entities, such as a `Claim` or an `Exposure` entity. As described previously in "Defining Contact Roles" on page 48, the contact role codes you use in `entityroleconstraints-config.xml` must be defined in the `ContactRole` typelist.

> **Note:** Only the entities supplied in the base configuration and configured on `ClaimContactRole` can use roles. These entities are `Claim`, `Evaluation`, `Exposure`, `Incident`, `Matter`, `Negotiation`, and `Policy`.

To view or edit the file, in ClaimCenter Studio, expand **Other Resources** and click **entityroleconstraints-config.xml** in the **Resources** tree on the left.

Defining Contact Role Constraints

The `ContactroleTypeConstraint` element defines the `Contact` subtype to which the role belongs. For example, a `mattermanager` is of type `Person`, as defined by the following code:

```
<ContactRoleTypeConstraint contactRoleCode="mattermanager" contactSubtype="Person"/>
```

- The `contactRoleCode` specifies the role's code name, `mattermanager`.
- The `contactSubtype` identifies the subtype to which the role belongs, `Person`.

If each of your contact roles uses only one `Contact` subtype, the system can ensure that the role is assigned to the correct contact type. Additionally, your PCF configuration is relatively simple. If a role uses more than one subtype, you must do extra configuration to ensure that selection of this role does not result in assignment of the wrong contact type. Even with extra configuration, it is possible to have the wrong type assigned to the role, as explained in the following `claimant` role description.

The base configuration roles are all set up to use one subtype, with a single exception, the `claimant` role, which can be both a `Person` and a `Company`. To specify the additional subtype, this configuration uses the `ExceptionConstraint` tag, as shown in the following example:

```
<ContactRoleTypeConstraint contactRoleCode="claimant" contactSubtype="Person">
    <ExceptionConstraint contactSubtype="Company" entityRef="Exposure"/>
</ContactRoleTypeConstraint>
```

The `ExceptionConstraint` tag ensures that the `claimant` role is always performed by a `Person` except when using an `Exposure` entity, when a `Company` can also perform this role. When the system starts up, it calculates the nearest supertype of the two subtypes and effectively assigns the `claimant` role to that supertype. For the `claimant` role, the nearest supertype is `Contact`.

> **Note:** Calculating the nearest supertype can result in assignment of an incorrect contact type unless you account for this possibility in your configuration. The possibility of incorrect type assignment is one reason you are encouraged to associate a contact role with a single subtype.

Defining Entity Role Constraints

The `EntityRoleConstraint` block defines the associations between roles and entities. The following example from the base configuration for `entityroleconstraints-config.xml` shows the uses of the key configuration tags:

```
<EntityRoleConstraint>
  <EntityRef entityType="Claim">
    <RoleRef contactRoleCode="FirstIntakeDoctor">
        <RoleConstraint constraintType="Exclusive"/>
    </RoleRef>
    <RoleRef contactRoleCode="InsuredRep"/>
    <RoleRef contactRoleCode="LawEnfcAgcy"/>
    <RoleRef contactRoleCode="OccTherapist"/>
    <RoleRef contactRoleCode="PhysTherapist"/>
    <RoleRef contactRoleCode="PrimaryDoctor">
      <RoleConstraint constraintType="Exclusive"/>
    </RoleRef>
    <!-- some definitions skipped for brevity -->
    <RoleRef contactRoleCode="claimant">
        <RoleConstraint constraintType="Exclusive"/>
        <RoleConstraint constraintType="Required">
            <AdditionalInfo propertyName="LossType" value="WC"/>
        </RoleConstraint>
        <RoleConstraint constraintType="Prohibited">
            <AdditionalInfo propertyName="LossType" value="AUTO"/>
            <AdditionalInfo propertyName="LossType" value="PR"/>
            <AdditionalInfo propertyName="LossType" value="GL"/>
        </RoleConstraint>
    </RoleRef>
    <RoleRef contactRoleCode="claimantdep"/>
    <RoleRef contactRoleCode="codefendant"/>
    ...
  </EntityRef>
</EntityRoleConstraint>
```

An `EntityRef` tag designates an `entityType`, an entity that can use contact roles, and contains one or more `RoleRef` tags that define contact roles for the entity. By default, there are no constraints on the entity’s use of the role. In the previous example, a `Claim` can use or not use the `InsuredRep`, `LawEnfcAgcy`, `OccTherapist`, `PhysTherapist`, `claimantdep`, and `codefendant` roles without restriction.

Adding Constraints to a Contact Role

You can constrain the relationship between an entity and a role by putting a `RoleConstraint` tag in the `RoleRef` block. As shown in the previous `<EntityRoleConstraint>` example, the `FirstIntakeDoctor`, `PrimaryDoctor`, and `claimant` contact role codes use this tag. The following `constraintType` values are possible:

`Exclusive`	At most one contact associated with the entity can fulfill this role. The entity may have no one in this role.
`Required`	This entity must have at least one contact in this role.
`Prohibited`	None of the contacts for this entity can have this role. Often qualified with an `AddtionaInfo` tag to limit the application of this constraint.
`ZeroToMore`	This entity can have no contact, one contact, or many contacts that use this role, which is the default behavior for a role that has no constraint types defined. Do not use this `constraintType` with `Exclusive`.

If you want to ensure that an entity has exactly one contact that fills a particular role, add two `RoleConstraint` tags to the `RoleRef`—an `Exclusive` and a `Required` constraint.

In the previous `<EntityRoleConstraint>` example, the `FirstIntakeDoctor` role has an `Exclusive` constraint. This constraint means that a `FirstIntakeDoctor` is not required for a `Claim`, but if a `FirstIntakeDoctor` is needed for the `Claim`, only one can be assigned. The same is true for a `PrimaryDoctor`.

Putting Limitations on Constraints

A constraint can be further refined with `AdditionalInfo` tags. An `AdditonalInfo` tag specifies a `propertyName` and `value` that must exist for the system to apply the constraint.

In the previous `<EntityRoleConstraint>` example, a `claimant` has three constraints, `Exclusive`, `Required`, and `Prohibited`. The `Exclusive` constraint applies to all allowable types of claims, limiting the number of claimants on these claims to no more than one. The `Required` and `Prohibited` constraints have `AdditionalInfo` tags that restrict their application. When the `LossType` for a claim is `WC` (workers' comp), the `claimant` role is `Required` and must be filled. If the `LossType` for a claim is `AUTO`, `PR` (property), or `GL` (general liability), the `claimant` role is `Prohibited` and cannot be filled at all.

> **Note:** For auto, property, and general liability claims, the claimants do not exist at the claim level. For these types of claims, you cannot add a claimant until you have exposures. The `claimant` contacts are owned by the exposures, not by the claim itself, because each exposure can have a separate claimant. For workers' comp claims, there is always a single claim, regardless of the number of exposures. Therefore, a workers' comp claim can (and must) own a contact with the role of `claimant`.

Resolving Conflicts in Role Constraint Configuration

It is possible to specify role constraints that can result in conflicts between constraints. For example, take the following fictional configuration:

```
<EntityRef entityType="Claim">
  <RoleRef roleCode="claimant">
    <RoleConstraint constraintType="Required">
      <AdditionalInfo propertyName="LossType" value="AUTO"/>
    </RoleConstraint>
    <RoleConstraint constraintType="Prohibited">
      <AdditionalInfo propertyName="State" value="CA"/>
    </RoleConstraint>
  </RoleRef>
</EntityRef>
```

If at run time a `Claim` has a `LossType` of `AUTO` and a `State` value of `CA`, there is a conflict. Auto claims require a `claimant`, but according to this constraint definition, in California, a `claimant` is prohibited on a claim. The system resolves this conflict by calculating constraint precedence and using the constraint with the highest precedence. The following list shows the order of constraint precedence from highest to lowest:

- `Prohibited`
- `Exclusive & Required`
- `Exclusive`
- `Required`
- `ZeroToMore`

In the previous example, `Prohibited` has higher precedence than `Required`, so the `claimant` role cannot be assigned on this auto insurance claim in California.

See also

- "How Configuring Roles Impacts Entity Data and Types" on page 52
- "Example of Adding a New Contact Role" on page 54
- "Relationships Between Contacts" on page 57

How Configuring Roles Impacts Entity Data and Types

Creating or changing a role configuration affects the data definition of the associated entity. When the system generates the entity's type information, it must include information for the role. Depending on how you configure a contact role, the application generates either a simple property or an array property on the entity. The system also provides methods on the entity for manipulating roles in general and for specific properties.

Note: If a role is prohibited for an entity, the system does not generate a property or any methods for the role.

Generated Role Methods

The system generates a number of methods on the entities related to a role. These methods enable you to manipulate and gather information about roles and contacts. The following table shows some of the methods available on the `Claim` entity.

Similar methods are also available on the entities `Exposure`, `Incident`, and `Matter`.

Note: In addition to the `get` methods for `Contact` entities, there are `get` methods for `ClaimContact` entities that are similar, except that they return `ClaimContact` entities rather than `Contact` entities. To see all the methods associated with these entities, you can use the Gosu API Reference, which is available in Studio on the **Help** menu. You can also run the Gosu Tester, which is available in Studio on the **Tools** menu. In the Gosu Tester, you can declare variables of the various entity types and use code completion pop-ups (**CTRL+SPACEBAR**) to show you what is available for an entity.

Method	Description
`getClaimant(): Contact`	Returns the claimant for a claim or, in the case of an exposure that has a claimant, for the exposure.
`getClaimants(): Contact[]`	Returns all claimants for a claim, such as an Auto claim that has multiple exposures.
`getClaimantNames(): String[]`	Returns names of all claimants on a claim as well as those on related exposures.
`getClaimContactsForAllRoles(): ClaimContact[]`	Gets all ClaimContact entities for all roles.
`getContactByRole( role: ContactRole): Contact`	Get the contact serving in an exclusive role. If you call this method on a role that is not exclusive, the system throws an exception.
`getContactsByRole( role: ContactRole): Contact[]`	Get the directly related Contact objects in the given role. This method returns only contacts attached directly to the entity. Contacts attached to the entity's subobjects are not returned.
`getContactsByRoles( roles: ContactRole[]): Contact[]`	Get the directly related Contacts with at least one of the given roles. Contacts related to subobjects are not returned.
`getContactsExcludeRole( role: ContactRole): Contact[]`	Get the directly related Contact objects that are not in the given role. This method returns only contacts attached directly to the entity. Contacts attached to the entity's subobjects are not returned.
`getContactsExcludeRoles( roles: ContactRole[]): Contact[]`	Get directly related Contacts that are not in any of the given roles. Contacts related to subobjects are not returned.
`getContactType( role: ContactRole): IType`	Get the type of the specified contact.
`setContactByRole( role: ContactRole, contact : Contact): void`	Set the contact for a specified role if the role is `Exclusive` or both `Exclusive` and `Required`.
`addRole(role: ContactRole, contact: Contact): ClaimContactRole`	Add a role with the specified contact to the entity. You can use this method only with `Required` or `ZeroToMore` roles. For `Exclusive` roles or for `Exclusive` and `Required` roles, use the explicit `setContactByRole` method for the role. If either the role or contact does not exist, this method does nothing.

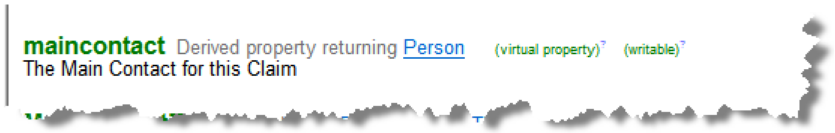
Method	Description
`removeRole( claimContactRole: ClaimContactRole): void`	Remove a role from the entity. If the role is the only role for the associated contact, it attempts to remove the contact as well. If either the role or contact does not exist on the entity, this method does nothing.
`removeRole(role: ContactRole, contact: Contact): void`	Remove a role from the entity for this specific contact. You can use this method only with `Required` or `ZeroToMore` roles. For `Exclusive` roles or the `Exclusive` and `Required` roles, use the other `removeRole` method. If either the role or contact does not exist on the entity, this method does nothing.

Entity Property for Exclusive or Exclusive and Required Role

If you add an `Exclusive` role or an `Exclusive` and `Required` role to an entity, the system adds a single property to the entity for the role. You can view this property in the Data Dictionary. For example, the role `maincontact` is defined in the `ContactRole` typelist and configured in `entityroleconstraints-config.xml` as follows:

```
...
<ContactRoleTypeConstraint contactRoleCode="maincontact" contactSubtype="Person"/>
...
<EntityRef entityType="Claim">
...
    <RoleRef contactRoleCode="maincontact">
        <RoleConstraint constraintType="Exclusive"/>
    </RoleRef>
</EntityRef>
...
```

When you regenerate the Data Dictionary, you see the following `maincontact` property on the `Claim` that has a foreign key to the `Person` contact type.


maincontact Derived property returning Person (virtual property) (writable)
The Main Contact for this Claim

The system also generates getter and setter methods on the `Claim` plugin class for this property—in this example, the `getMaincontact` and `setMaincontact` methods. You can also get and set this property using Gosu, for example, by adding Gosu code to a rule.

Entity Array Key for Required or ZeroToMore Role

If you add a `Required` or `ZeroToMore` role to an entity, the system adds an array key to the entity. If there is no role constraint specified, the default is `ZeroToMore`. For example, the `arbitrator` role is configured as follows:

```
...
<ContactRoleTypeConstraint contactRoleCode="arbitrator" contactSubtype="Adjudicator"/>
...
<EntityRef entityType="Claim">
...
  <RoleRef contactRoleCode="arbitrationvenue"/>
  <RoleRef contactRoleCode="arbitrator"/>
  <RoleRef contactRoleCode="assessor"/>
...
</EntityRef>
...
```

The system generates the array key `arbitrator` on the `Claim` entity. The array key points to an array of `Adjudicator` contacts.

The system also generates a getter on the entity that enables you to get a list of the contacts in the `arbitrator` role. The system does not generate a setter for array properties. To manipulate the array's contents, use the `addRole` and `removeRole` methods.

Avoiding Errors with Contact Properties

For every subtype you create, your Guidewire application creates a property for that subtype on its parent. For example, if you extend `CompanyVendor` with a `Dentist` subtype, the system creates a `Dentist` property. In this case, do not create additional database entities with the name `Dentist`. For example, if you create a custom relationship, do not name the relationship accessor `Dentist`. If you do use the property name for the name of a database entity, you will receive an error similar to the following:

```
[java] java.lang.ClassCastException: com.guidewire.commons.metadata.internal.loader.ArrayData
  [java] at com.guidewire.tools.datadictionary.support.TableWriter.writeColumns(TableWriter.java:239)
  [java] at com.guidewire.tools.datadictionary.support.TableWriter.writeSubtype(TableWriter.java:504)
  [java] at com.guidewire.tools.datadictionary.support.TableWriter.writeSubtypes(TableWriter.java:476)
  [java] at com.guidewire.tools.datadictionary.support.TableWriter.writeTable(TableWriter.java:155)
  [java] at
    com.guidewire.tools.datadictionary.support.DictionaryWriter.writeTable(DictionaryWriter.java:409)
  [java] at
  com.guidewire.tools.datadictionary.support.
         DictionaryWriter.outputDictionary(DictionaryWriter.java:360)
  [java] at com.guidewire.tools.datadictionary.Main.main(Main.java:132)
  [java] Exception in thread "main"
```

See also

- "Adding Contact Roles" on page 47
- "Example of Adding a New Contact Role" on page 54
- "Relationships Between Contacts" on page 57

Example of Adding a New Contact Role

In this example, you add a new contact role.

1. If necessary, start ClaimCenter Studio.

   At a command prompt, navigate to `ClaimCenter\bin` and enter the following command:

   ```
   gwcc studio
   ```

2. In the **Resources** tree on the left, expand **Typelists** and click **ContactRole**.
3. Click **Add** and add a role with the following values:

Code	Name	Description
`negotiator`	`Negotiator`	`Person who handles a negotiation.`

   Adding this line causes the database upgrader to add a `negotiator` value to the `ContactRole` typelist.
4. In the **Resources** tree under **Typelists**, click the **ContactRoleCategory** typelist to open it in the editor.
5. Select the `Vendors` category and, below on the **Categories** tab, click **Add** to add the following `ContactRole`:

Typelist	Code
`ContactRole`	`negotiator`

   This step sets up filtering by this particular role for parties involved.
6. In the `entityroleconstraints-config.xml` file, add the `negotiator` role, and then add `negotiator` as a role reference to the list of `Negotiation` roles.

   a. In the **Resources** tree, expand **Other Resources** and click **entityroleconstraints-config.xml** to edit it.

b. Add the following `ContactRoleTypeConstraint` to the section for role constraints at the top of the file:

```
<ContactRoleTypeConstraint
  contactRoleCode="negotiator"
  contactSubtype="Person"/>
```

c. Add the `negotiator` role with the `Exclusive` (zero or one interpreter) constraint to the `Negotiation` entity, `<EntityRef entityType="Negotiation">`:

```
<RoleRef contactRoleCode="negotiator">
  <RoleConstraint constraintType="Exclusive"/>
</RoleRef>
```

7. Save the file.
8. Stop ClaimCenter, regenerate the APIs, optionally regenerate the data dictionary, and restart ClaimCenter, as described in the steps that follow.

   Note: Because the role changes and additions are data model changes, you must at least stop and restart ClaimCenter.

   a. If necessary, stop ClaimCenter.

   At the command prompt in which ClaimCenter is running, press **CTR L+ C** to stop the batch process and then enter the following commands:

```
y
gwcc dev-stop
```

   b. Regenerate the APIs:

```
gwcc regen-java-api
gwcc regen-soap-api
```

   c. Optionally regenerate the data dictionary:

```
gwcc regen-dictionary
```

   d. Start the ClaimCenter application:

```
gwcc dev-start
```

9. You also need to quit ClaimCenter Studio and restart it to enable use of the `Negotiation.negotiator` virtual property. To restart Studio, enter:

```
gwcc studio
```

10. In Studio, click **Display Keys** and navigate to **NVV** → **Matter** → **SubView** → **MatterNegotiationDetail** → **General**.
11. Right-click **General** and click **Add**, and then add a display key with name `NVV.Matter.SubView.MatterNegotiationDetail.General.Negotiator` and value `Negotiator`.
12. In the **Resources** tree on the left, navigate to **Page Configuration (PCF)** → **claim** → **newother** and click **NewNegotiationDV** to open this PCF file in the editor.
13. Drag a **ClaimContactInput** widget from the **Toolbox** on the right and drop it after the **ClaimContactInput** with **id** `General_NegContact`, and then select it and set the following properties:

claim	`Negotiation.Claim`
valueRange	`Negotiation.Claim.RelatedContacts as Person[]`
editable	`true`
id	`NegotiatorContact`
label	`displaykey.NVV.Matter.SubView.MatterNegotiationDetail.General.Negotiator`
required	`false`
value	`Negotiation.negotiator`

Because the `Negotiation.negotiator` value is a `Person`, but the **valueRange** for this field is all `Claim` contacts, you must cast the **valueRange** to be an array of `Person`. If you regenerated the Data Dictionary, you can use it to view the `Negotiation` entity and the new `negotiator` field.

14. In the **Resources** tree on the left, navigate to **Page Configuration (PCF)** → **claim** → **planofaction** and click **ClaimNegotiationDetailsDV** to open this PCF file in the editor.

15. Drag a **ClaimContactInput** widget from the **Toolbox** on the right and drop it after the **ClaimContactInput** with **id** `General_NegContact`, and then select it and set the following properties:

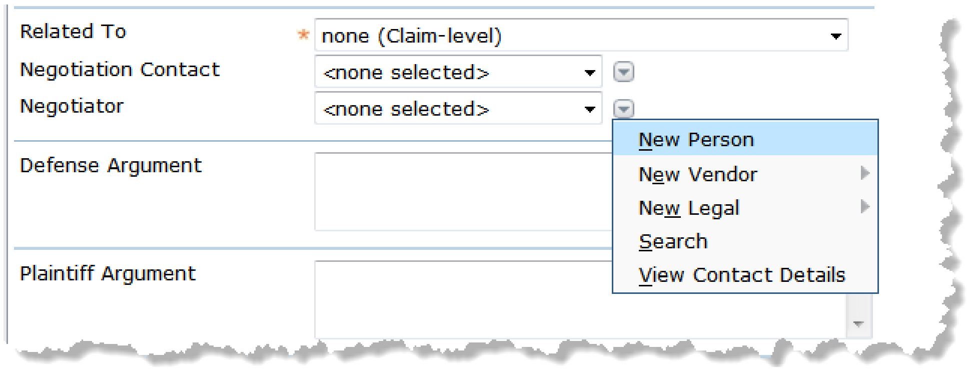
claim	`Negotiation.Claim`
valueRange	`Negotiation.Claim.RelatedContacts as Person[]`
editable	`true`
id	`NegotiatorContact`
label	`displaykey.NVV.Matter.SubView.MatterNegotiationDetail.General.Negotiator`
required	`false`
value	`Negotiation.negotiator`

Because the `Negotiation.negotiator` value is a `Person`, but the **valueRange** for this field is all `Claim` contacts, you must cast the **valueRange** to be an array of `Person`. You can use the Data Dictionary to view the `Negotiation` entity and the new `negotiator` field.

16. If you are running ClaimCenter in development mode, log in and click **ALT+SHIFT+L** to reload your user interface changes. An alternative is to stop and restart ClaimCenter, and then log in.

17. Open a claim, and then choose **Actions** → **New** → **Negotiation**.

18. When you view the **Negotiator** field in the **New Negotiation** window, the system automatically generates a picker for it:


Related To * none (Claim-level)
Negotiation Contact <none selected>
Negotiator <none selected>
Defense Argument
Plaintiff Argument

19. Add a negotiator and enter some information about the negotiation and then save the new negotiation.

20. Click **Plan of Action** on the left, and then click **Negotiations** in the blue bar at the top to show the current negotiations.

21. Select the negotiation you created and look for the **Negotiator** field, which shows the negotiator you added. You can click **Edit** to change the settings for the negotiation, including the **Negotiator**.

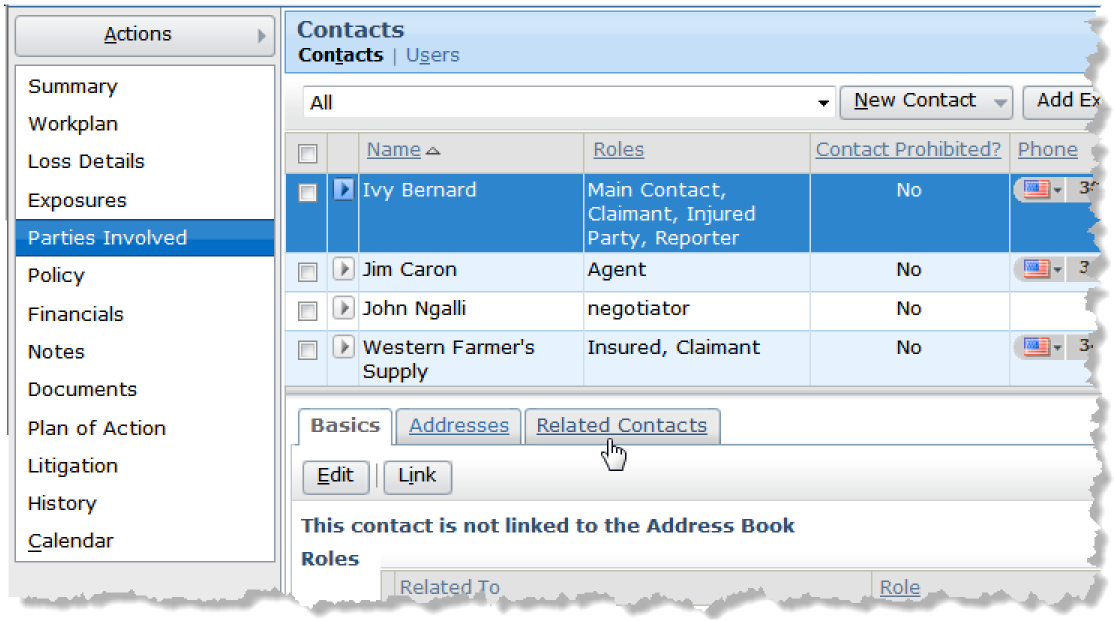
See also

- "Adding Contact Roles" on page 47
- "How Configuring Roles Impacts Entity Data and Types" on page 52
- "Relationships Between Contacts" on page 57

Relationships Between Contacts

Relationships enable you to associate two contacts by applying an association to them. Examples of associations are employer and employee or attorney and law firm. You see a contact's relationships on the ClaimCenter **Related Contacts** subtab.

For example, if you open a claim and click **Parties Involved** on the left, you see a list of contacts for the claim. Below this list is an **Edit** window for the selected contact that has three tabs, one of which is **Related Contacts**.



Name	Roles	Contact Prohibited?	Phone
Ivy Bernard	Main Contact, Claimant, Injured Party, Reporter	No	3
Jim Caron	Agent	No	3
John Ngalli	negotiator	No	
Western Farmer's Supply	Insured, Claimant	No	3

This contact is not linked to the Address Book

Roles

Related To	Role

You configure contact relationships by opening Studio and editing the `contact-relationship-config.xml` file and the typelists **ContactBidiRel** and **ContactRel**.

Relationships are automatically bidirectional. If you add a `Person` contact to a `Company` contact as an `Employee`, the company appears on the person's **Related Contacts** tab as an `Employer`. Additionally, the person appears on the company's **Related Contacts** tab as an `Employee`.

At the top of the `contact-relationship-config.xml` file there are a number of internal contact relationships: `guardian`, `employer`, `primarycontact`, and so on. To add your own relationships, add them at the end of the file.

IMPORTANT The system relies on these internal relationships. Do not modify or remove the internal contact relationships. However, you can use a filter attribute to filter these values from a PCF drop-down.

Example of Adding a Relationship to ClaimCenter and ContactCenter

In this topic, you add a relationship between two contacts to indicate the primary and secondary insured parties on a claim.

To add a new relationship to ClaimCenter

1. If necessary, start `ClaimCenter` Studio.

   At a command prompt, navigate to `ClaimCenter\bin` and enter the following command:

   ```
   gwcc studio
   ```

2. In the **Resources** tree on the left, expand **Typelists** and click **ContactRel** to edit it.

3. Click **Add** and add a contact relationship typecode with the following values:

Code	Name	Description
primaryinsured	Primary Insured	Primary insured

4. In the **Resources** tree on the left, expand **Typelists** and click **ContactBidiRel** to edit it.
5. **Add** the following typecodes:

Code	Name	Description
primaryinsured	Primary Insured	Primary insured
secondaryinsured	Secondary Insured	Secondary insured

6. In the `contact-relationship-config.xml` file, add elements to combine the relationship `primaryinsured` with its inverse, `secondaryinsured`, as follows:
   a. In the **Resources** tree, expand **Other Resources** and click **contact-relationship-config.xml** to edit it.
   b. Add the following elements before the final `</ContactRelationshipConfigFile>` tag:
```
<ContactRelationshipPair contactRelCode="primaryinsured" >
  <Primary name="PrimaryInsured"
           cardinality="zeroorone"
           contactBidiRelCode="primaryinsured"
           entity="Contact" />
  <Inverse name="SecondaryInsured"
           cardinality="zeroormore"
           contactBidiRelCode="secondaryinsured"
           entity="Contact" />
</ContactRelationshipPair>
```
   c. Save the file.
7. Ensure that your sync configuration also has the primary relationship defined as a sync attribute. Add the relationship to `Contact` as follows:
   a. In the **Resources** tree on the left, click **Other Resources**, and then click **contact-sync-config.xml** to open it in an external editor.
   b. Under `<ContactSyncEntityConfig entity="Contact">` add the following element:
```
<IncludeRelationship contactBidiRelCode="primaryinsured"/>
```
8. Save the file, and then stop ClaimCenter, regenerate the APIs, optionally regenerate the data dictionary, and restart ClaimCenter, as described in the following steps. You need to do all this because the relationship additions cause data model changes to `Contact`.
   a. At a command prompt, navigate to the ClaimCenter `bin` directory.
   b. If necessary, stop the ClaimCenter application:

      At the command prompt in which ClaimCenter is running, press **CTR L+ C** to stop the batch process and then enter the following commands:
```
y
gwcc dev-stop
```
   c. Regenerate the APIs:
```
gwcc regen-java-api
gwcc regen-soap-api
```
   d. Optionally regenerate the data dictionary:
```
gwcc regen-dictionary
```
   e. Start the ClaimCenter application:
```
gwcc dev-start
```

9. To test that the changes work:
   a. Log in as a user that has claims, such as `aapplegate/gw`.
   b. Open an existing claim and open the **Parties Involved** window.
   c. If you need to add contacts to this screen, ensure that there are at least two.
   d. Click the name of one of the contacts to open the **Basics** screen below, then click the **Related Contacts** tab.
   e. Click **Edit**, then **Add**, and then click the drop-down list below **Name** to add one of the contacts on the list.
   f. Under **Relation to Contact**, click the drop-down list and choose `Primary Insured`.
   g. Click **Update** to save your changes.
   h. In the list of contacts above the **Related Contacts** tab, click the contact you just added as a related contact, then click the **Related Contacts** tab for that contact.

      That tab shows the first contact you picked and the inverse relationship `Secondary Insured`.

To add the same new relationship to ContactCenter

1. If necessary, start ContactCenter Studio.

   At a command prompt, navigate to `ContactCenter\bin` and enter the following command:

   ```
   gwab studio
   ```
2. In the **Resources** tree on the left, expand **Typelists** and click **ContactRel** to edit it.
3. Click **Add** and add a contact relationship typecode with the following values:

Code	Name	Description
primaryinsured	Primary Insured	Primary insured

4. In the **Resources** tree on the left, expand **Typelists** and click **ContactBidiRel** to edit it.
5. **Add** the following typecodes:

Code	Name	Description
primaryinsured	Primary Insured	Primary insured
secondaryinsured	Secondary Insured	Secondary insured

6. In the `abcontact-relationship-config.xml` file, add elements to combine the relationship `primaryinsured` with its inverse, `secondaryinsured`, as follows:
   a. In the **Resources** tree, expand **Other Resources** and click **abcontact-relationship-config** to edit it.
   b. Add the following elements before the final `</ContactRelationshipConfigFile>` tag:

      ```
      <ContactRelationshipPair contactRelCode="primaryinsured" >
        <Primary name="PrimaryInsured"
                 cardinality="zeroorone"
                 contactBidiRelCode="primaryinsured"
                 entity="ABContact" />
        <Inverse name="SecondaryInsured"
                 cardinality="zeroormore"
                 contactBidiRelCode="secondaryinsured"
                 entity="ABContact" />
      </ContactRelationshipPair>
      ```
   c. Save the file.
7. Stop ContactCenter, optionally regenerate the data dictionary, and restart ContactCenter, as described in the steps that follow.

You need to do all this because the relationship additions cause data model changes to `ABContact`.

a. If necessary, stop the ContactCenter application:

   At the command prompt in which ContactCenter is running, press **CTRL+C** to stop the batch process and then enter the following commands:

   ```
   y
   gwab dev-stop
   ```

b. Regenerate the APIs:

   ```
   gwab regen-java-api
   gwab regen-soap-api
   ```

c. Optionally regenerate the data dictionary:

   ```
   gwab regen-dictionary
   ```

d. Start the ContactCenter application:

   ```
   gwab dev-start
   ```

8. To test that the changes work:

   a. Log in as a user, such as `aapplegate/gw`.

   b. Search for an existing Address Book contact, such as the company `Whole Foods` from the imported sample data, and then click the **Name** link in **Search Results**.

   c. In the **Address Book** entry for your selection, click the **Related Contacts** tab, and then click **Edit** and then **Add** to add a new related contact.

   d. Click the drop-down under **Name** and choose `Search` from the list, and then search for an existing contact. For example, search for the `Person` with **First Name** of `Eric` and **Last Name** of `Andy` from the imported sample data.

   e. Click **Select** next to the contact to want to add, and then, under the **Related Contacts** tab's **Relationship** heading, click the drop-down list and choose `Primary Insured`.

   f. Click **Update** to save your changes.

   g. Click **Search** on the left, and then search for the contact you just added as a related contact.

      For example, choose `Person` as the **Contact Type** and enter `Andy` for the **Last Name**, and then click the **Search** button.

   h. Click the **Name** link for that contact in **Search Results**.

      For example, click `Eric Andy` under **Search Results**.

   i. On the **Address Book** entry for the contact, click the **Related Contacts** tab.

      That tab shows the first contact you picked, such as `Whole Foods`, and the inverse relationship `Secondary Insured`.

See also

- "Adding Contact Roles" on page 47
- "How Configuring Roles Impacts Entity Data and Types" on page 52
- "Example of Adding a New Contact Role" on page 54

chapter 7

Configuring Contact Address Interfaces

This topic discusses the terms and concepts relating to autofill and address zones and describes how to configure zones and addresses.

This topic includes:

Understanding Autofill and Zone Mapping

The *autofill* feature enables you to enter part of an address in a field in a Guidewire application and have the application fill in the rest automatically. For example, if you enter a US ZIP code and tab out of the field, ClaimCenter automatically fills in the city and state.

If the application cannot locate an exact match, it offers you a choice among the closest matches. This feature, called *autocomplete*, enables you to enter the first few characters of a field and see a drop-down with the various matches. For example, you enter `941` in a postal code field and ClaimCenter offers the ZIP codes `94100`, `94101`, and so forth.

This section describes the zone mapping feature that supports autofill and autocomplete. The section also covers how to use configuration resources to manage both autofill and autocomplete.

IMPORTANT **Default Country Setting and Base Zones.** In the base configuration, Guidewire defines hierarchies of zones, such as city, county, and state or province, for the United States and Canada. You can add additional country zone definitions to the file `zone-config.xml`, as described in the topics that follow.

Only one country zone at a time is active in ClaimCenter. You set the application default country by setting the configuration parameter `DefaultCountryCode` in the `config.xml` file. The default value is `US` for United States. You must set this value before you start the ClaimCenter database for the first time, and you cannot change it thereafter.

Zones and Links

A *zone* is a geographical area or region in a country, such as a city, a state in the United States, or a province in Canada. Guidewire applications use zones to perform assignments by location, to autofill addresses, and to enable holiday and business week definition by zone.

Typically, country zones are organized according to postal address. To load zones into your application, you can use Guidewire zone files or import a zone dataset from a file purchased from a vendor. For example, your company might have bought data from a company like GreatData. You then use this dataset to plan your zone configuration. Each installation of ClaimCenter ships with US and Canada address data. For more information, see “Importing Address Zone Data” on page 63.

In planning, consider what parts of an address you need to autofill and the format of the ZIP or postal codes. Also, note if any of the values are identical for a country. For example, does your country include two cities with the same name?

While most countries have a similar format for addressing, there can be significant differences. For example, the format for the Czech Republic is made up of a postal code, city name, and a one- to three-digit post office identifier. France uses a five-digit postal code plus a city. Additionally, France has a *small locality* format that includes more information.

You can define multiple zones for a country. You define *zone types* as typecodes in the `ZoneTypes` typelist and then use them in the `zone` element in the `zone-config.xml` file. For example, you can divide your country into county, state, and city. Most cities are in a single zone, such as a state in the United States or a province in Canada.

While planning your zones, consider that some zones can be referenced by another zone. Using zone mapping, you can specify these references by using *links*. ClaimCenter uses links to look up a zone based on another zone. An example is the relationship between the ZIP code and the state for US zones. Your configuration would link ZIP code to state. For example, given the ZIP code of 94404, the application can autofill the corresponding state, California.

You do not necessarily need to define all the `Zone` subelements for a zone. Instead, you can define a zone as the entire country if you want. For example, for Australia:

```
<Zones countryCode="AU"/>
```

For further information on zone and link concepts and setting their values, see

- “Adding Basic Zone Types” on page 63 for information on defining zone types for a country.
- “Working with the zone-config.xml File” on page 63 for information on using the `zone` element with zone types and defining links with the `link` element.

Adding Basic Zone Types

You can define basic zone types for your country configuration in the `ZoneType` typelist.

The base configuration comes with a set of typecodes for the United States and Canada. The base ClaimCenter configuration contains zones for:

- ZIP code
- city
- county
- state
- province
- postal code
- FSA (Forward Sortation Area, first three letters of a Canadian postal code)

To edit the typelist:

1. If necessary, start ClaimCenter Studio.

   At a command prompt, navigate to the `ClaimCenter\bin` directory and enter:

   ```
   gwcc studio
   ```
2. In the **Resources** tree, expand the **Typelists** node, and then click **ZoneType** to edit the typelist.

See also

- “Importing Address Zone Data” on page 63
- “Working with the zone-config.xml File” on page 63
- “Configuring Addressing” on page 65
- “Implementing Autofill and Autocomplete in a PCF File” on page 66

Importing Address Zone Data

Import address zone data on first installing the product and at infrequent intervals thereafter as you receive data updates. Guidewire ships a data file for the United States and Canada with each product release. If you are interested in data for a different country, speak to your Guidewire sales representative.

For instructions on how to import an address zone data file, see “zone_import Command” on page 178 in the *System Administration Guide*.

See also

- “Understanding Autofill and Zone Mapping” on page 61
- “Working with the zone-config.xml File” on page 63
- “Configuring Addressing” on page 65
- “Implementing Autofill and Autocomplete in a PCF File” on page 66

Working with the zone-config.xml File

You define the zones and their format in the `zone-config.xml` file. In this file, you define the links between zones, which sets up the zone hierarchy. Additionally, you define how ClaimCenter is to use the links to extract a zone’s value from address data. For information on the elements you configure in this file, such as `Zones`, `Zone`, `fileColumn`, and `AddressZoneValue`, see “Configuring Zone Information” on page 476 in the *Configuration Guide*.

In general, to configure zones for a country:

1. If necessary, add zone type typecodes to the `ZoneType` typelist, as described at "Adding Basic Zone Types" on page 63.
2. Create a `Zones` element for each country in which your company operates.
3. In the `Zones` element, define each `Zone` with a specific `code`. The code must correspond to a value in the `ZoneType` typelist, such as `state`, `county`, and `city` in the United States.
4. Optionally specify the `fileColumn`, the column in which the zone appears in the import data.

   Note: While this attribute is optional, you must specify the `fileColumn` for the `Zone` elements you want to import from the address data file.
5. Optionally, if your `Zone` data contains two or more identical values, define a `ZoneCode` that uses the codes from the `Zone` elements for the current `Zones` element.

   For example, the United States contains many counties with the name `Union`, but in each state, the name of a county is unique. To make the `county` code unique for the US, you use `ZoneCode` to prefix the `county` Zone code with the `state` Zone code, as follows:

   ```
   <ZoneCode>
     state + &quot;:&quot; + county
   </ZoneCode>
   ```

   For an example, see "Example of a Zone Configuration" on page 64.
6. Optionally specify a set of `Links` between one zone type and another. Each `Link` defines a connection between one zone type and another, such as between a county and a ZIP code in the United States. A link can also define a lookup order to speed searches.
7. Optionally specify an `AddressZoneValue` to define how to extract the zone code from an address. The values of this element use entity dot notation, such as `Address.County`, and you can concatenate them. The system matches the value that ClaimCenter extracts against the zone data in the database.

Example of a Zone Configuration

The example below shows the `county` zone code definition for the United States. In the United States, a county is a region inside a state. Counties contain one or more ZIP codes. In most cases, counties comprise multiple cities and ZIP codes. An exception is the county of San Francisco in California, which contains only the city of San Francisco.

```
<Zone code="county"
      fileColumn="4"
      regionMatchOrder="2"
      granularity="3">
  <ZoneCode>
    state + &quot;:&quot; + county
  </ZoneCode>
  <AddressZoneValue>
    Address.State.Code + &quot;:&quot; + Address.County
  </AddressZoneValue>
  <Links>
    <Link toZone="zip" lookupOrder="1"/>
    <Link toZone="city" lookupOrder="2"/>
  </Links>
</Zone>
```

This `Zone` defines the `county` zone in the United States `Zones` element. The county value appears in the fourth column of the data file and is the second in the matching order. It is also third in granularity, one size less than a state (granularity 4) and one size more than a city (granularity 2). Additionally, the `county` zone value by itself is not guaranteed to be unique across all states. For example, there is a county named *Union* in seventeen states, and a county named *Adams* in twelve states. Therefore, the configuration appends the state value to the county value to get a unique `ZoneCode`. The `AddressZoneValue` requires a similar configuration.

The county zone links both to the ZIP code and to the city. There are two `lookupOrder` values. Based on these values, ClaimCenter, given an address, attempts to extract the county first from the ZIP code and second from the city. In each instance, the application searches for a unique match.

See also

- "Understanding Autofill and Zone Mapping" on page 61
- "Importing Address Zone Data" on page 63
- "Configuring Addressing" on page 65
- "Implementing Autofill and Autocomplete in a PCF File" on page 66

Configuring Addressing

Use the `address-config.xml` file to configure the behavior of address autofill. You can also use this file to configure how ClaimCenter validates an address field for the postal code.

To edit the `address-config.xml` file, in ClaimCenter Studio go to the **Resources** tree on the left, expand **Other Resources**, and click **address-config.xml**.

The following default configuration for the United States shows all the elements you can use in this file:

```
<AddressDef name="US">
  <Match>
    <Field name="Country" value="US"/>
  </Match>
  <Fields>
    <Field name="City" zonecode="city">
      <AutoFillFromZone code="zip"/>
      <AutoFillFromZone code="state"/>
    </Field>
    <Field name="County" zonecode="county">
      <AutoFillFromZone code="zip"/>
      <AutoFillFromZone code="city"/>
    </Field>
    <Field name="State" zonecode="state">
      <AutoFillFromZone code="zip"/>
      <AutoFillFromZone code="city"/>
    </Field>
    <Field name="PostalCode" zonecode="zip">
      <AutoFillFromZone code="city"/>
      <ValidatorDef
         value="[0-9]{5}(-[0-9]{4})?"
         description="Validator.PostalCode"
         input-mask="#####-####"/>
    </Field>
  </Fields>
</AddressDef>
```

The following table describes the elements in the `address-config.xml` file.

`AddressDef`	The name of the address format. This element takes a `name` attribute and an optional `priority` attribute. The `name` is usually the country code. A country can have more than one address format — for example, it is possible that different regions have different formats. The `priority` attribute specifies which address definition has priority if several of them match.
`Match`	Each `AddressDef` must contain one `Match` element. ClaimCenter matches only on the country. The `Match` element contains a `Field` subelement that defines the `name` and `value` attributes that the application uses to determine which definition applies.
`Fields`	Each `AddressDef` contains a single `Fields` element that contains a list of the address `Field` elements.

`Field`	Specifies an address field. Each field takes a name that must match a property on the `Address` entity. The `Field` element can appear in the `Match` element or the `Fields` element. In the `Match` element, the `Field` element has a required `name` attribute and an optional `value` attribute, which is the `code` value from the `Country` typelist. It has no child elements. In the `Fields` element, the `Field` element has a required `name` attribute and optional `value`, `zonecode`, and `autoCompleteTriggerChars` attributes. It can also have the child elements `AutoFillFromZone` and `ValidatorDef`. • The `value` attribute is a valid value for the `Field`. This value is usually a code value from a typelist, such as a state typecode from the **State** typelist for a `Field` with the `name` set to `State`. • The `zonecode` attribute corresponds to a `Zone` code configured for the given country in `zone-config.xml`. This value links the `Address` configuration to the `Zone` configuration. For information on zone configuration, see “Working with the zone-config.xml File” on page 63. • The `autoCompleteTriggerChars` attribute specifies the number of characters to enter before the application triggers autocomplete. The default value is 0 (zero).
`AutoFillFromZone`	Specifies the zone `code` that ClaimCenter uses to look up the field. The lookup of this field from other fields is through this code and not the field name.
`ValidatorDef`	Contains information for validating the field in the optional attributes `value`, `description`, and `input-mask`. For more information on `ValidatorDef`, see “<ValidatorDef>” on page 295 in the *Configuration Guide*.

The previous code listing defines a `Match` on the country with the match value as `US`. Additionally, it defines four fields:

- `City`
- `County`
- `State`
- `PostalCode`

Each field defines one or more `AutoFillFromZone` elements. Looking at the `County`, you can see that the application fills the `County` either from the `zip` or the `city`. This information is already defined in the `zone-config.xml` file as described in “Example of a Zone Configuration” on page 64. That file specifies that the autofill first tries to look up the county by ZIP code and, if that fails, then by city. You repeat the information in `address-config.xml` to show explicitly how the application is autofilling.

As ClaimCenter loads `address-config.xml`, it validates the configuration. ClaimCenter verifies that every `Field` element, regardless of whether it is defined in `Match` or `Fields`, corresponds to a field on the `Address` entity. Then, ClaimCenter verifies that each `AutoFillFromZone` element references a zone in the `zone-config.xml` configuration.

See also

- “Understanding Autofill and Zone Mapping” on page 61
- “Importing Address Zone Data” on page 63
- “Working with the zone-config.xml File” on page 63
- “Implementing Autofill and Autocomplete in a PCF File” on page 66

Implementing Autofill and Autocomplete in a PCF File

You implement autofill and autocomplete after you have configured zone mapping and addressing and have imported your zone data. The main components used for address autofill and autocomplete in the user interface are the following classes:

- `gw.api.contact.AddressAutocompleteHandler` provides methods for getting matching values from the database.
- `gw.api.contact.AddressAutocompleteUtil` supplies methods for automatically completing an address field.

You can integrate these components into a PCF file to:

- Complete a field automatically after you type in a few characters. For example, if you enter `941` for the ZIP code, the autocomplete list shows `94110`, `94111`, `94112`, and so forth.
- Fill other address fields after you complete a field and tab out of it. For example, after you enter the ZIP code `94115`, autofill can fill in the city, county, and state as `San Francisco`, `San Francisco`, and `California` respectively. This level of autofill happens only if there is a unique match between the fields.
- Force an override of already filled fields. By default, if a field already has a value, ClaimCenter does not overwrite it. You can force it to override the field value with zone data by using a method on `AddressAutocompleteUtil` to autofill the address.

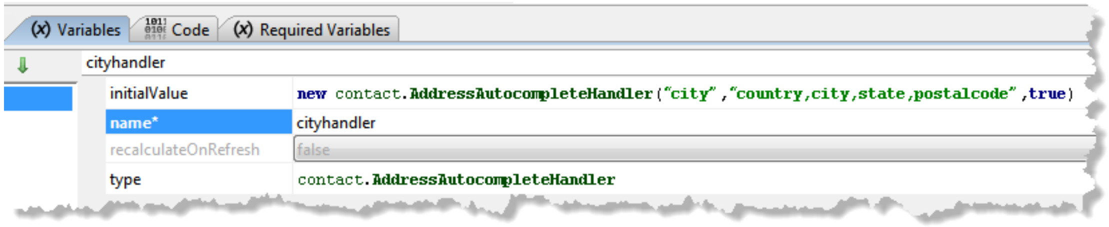
There are two PCF widgets that you can use to support autofill: the `AddressAutoFillRange` and the `AddressAutoFillInput` widgets. Both widgets provide a small icon for autofilling the fields on demand.

Adding Autocomplete

To use the address autocomplete feature, first create an `AddressAutocompleteHandler` instance in the appropriate PCF file. The constructor takes the following parameters:

- A String identifying the field to complete
- A comma-separated list of the reference fields that the application passes to the handler
- A Boolean value that specifies whether ClaimCenter waits for a key press before fetching suggestions

You edit the PCF file in Guidewire Studio to add the handler constructor to the `initialValue` variables. The following figure shows an example of constructing an `AddressAutocompleteHandler` called `cityhandler` in the `AddressInputSet.CA` PCF file:



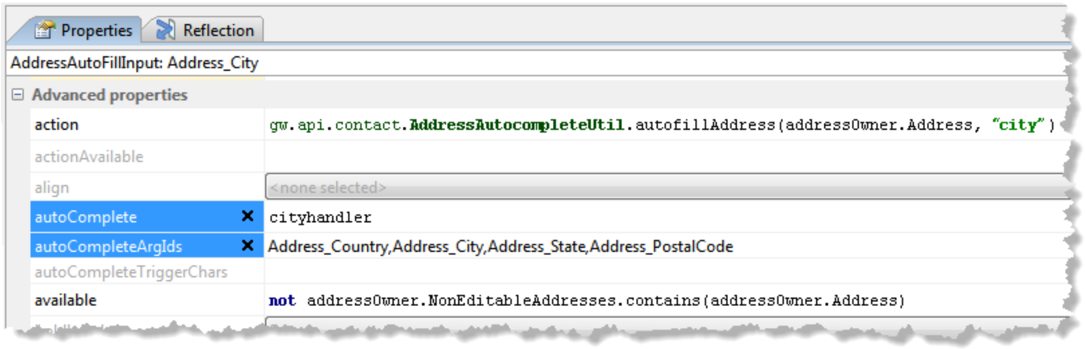
cityhandler

initialValue	`new contact.AddressAutocompleteHandler("city","country,city,state,postalcode",true)`
name*	cityhandler
recalculateOnRefresh	false
type	`contact.AddressAutocompleteHandler`

From the constructor, you can see that this handler operates on the `city` field. The handler expects the following set of reference fields: `Country`, `City`, `State`, and `PostalCode`.

Next, edit the widget that uses the handler. Specify which handler is responsible for autofill in the `autoComplete` field and the arguments the handler expects in the `autoCompleteArgIds` field.

> **Note:** Do not set `postOnChange` to `true` for the widget if you use the `autoComplete` property. The `postOnChange` property can interfere with the functioning of `autoComplete`.



AddressAutoFillInput: Address_City

Advanced properties

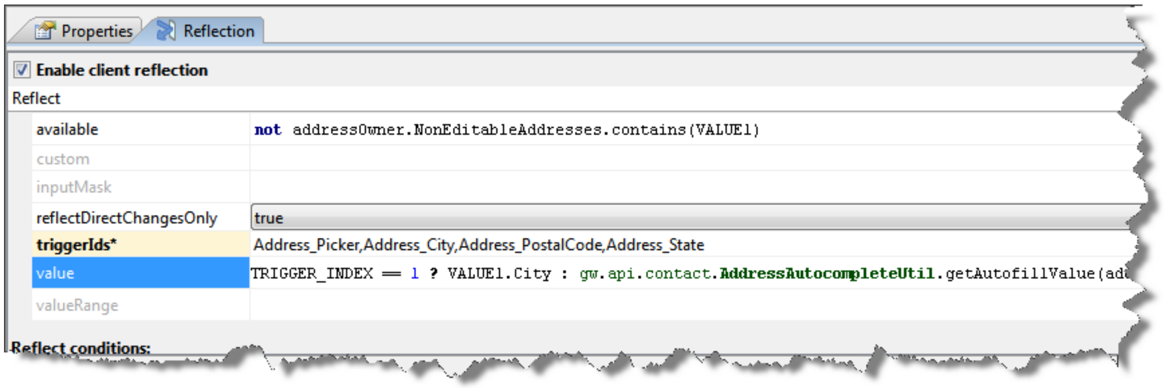
action	`gw.api.contact.AddressAutocompleteUtil.autofillAddress(addressOwner.Address, "city")`
actionAvailable	
align	<none selected>
autoComplete	`cityhandler`
autoCompleteArgIds	Address_Country,Address_City,Address_State,Address_PostalCode
autoCompleteTriggerChars	
available	`not addressOwner.NonEditableAddresses.contains(addressOwner.Address)`

The widget that contains the city is an `AddressAutoFillInput` widget, which has an autofill icon beside it that acts as a cue. You do not have to use this specific type of widget for this purpose.

Adding Autofill

Reflection enables you to change one field in a Guidewire user interface based on a change in another field without waiting for the user to submit the form. You configure reflection by adding a `Reflect` element to an input widget.

The following figure depicts the use of a `Reflect` element in the `City` input field:



☑ Enable client reflection

Reflect

available	not addressOwner.NonEditableAddresses.contains(VALUE1)
custom	
inputMask	
reflectDirectChangesOnly	true
triggerIds*	Address_Picker,Address_City,Address_PostalCode,Address_State
value	TRIGGER_INDEX == 1 ? VALUE1.City : gw.api.contact.AddressAutocompleteUtil.getAutofillValue(add
valueRange	

Reflect conditions:

The **value** property uses the `AddressAutocompleteUtil.getAutofillValue` method as follows:

```
TRIGGER_INDEX == 1 ? VALUE1.City :
    gw.api.contact.AddressAutocompleteUtil.getAutofillValue(
        addressOwner.InputSetMode, "city",
        {"city","postalcode","state"},
        {VALUE2,VALUE3,VALUE4}) as java.lang.String
```

The `getAutofillValue` method used in the example takes four arguments. There is an optional version of the method that takes a fifth argument, `doOverride`, and another optional version of the method that takes an additional, sixth argument, `triggerIndex`. Following are descriptions of the six arguments:

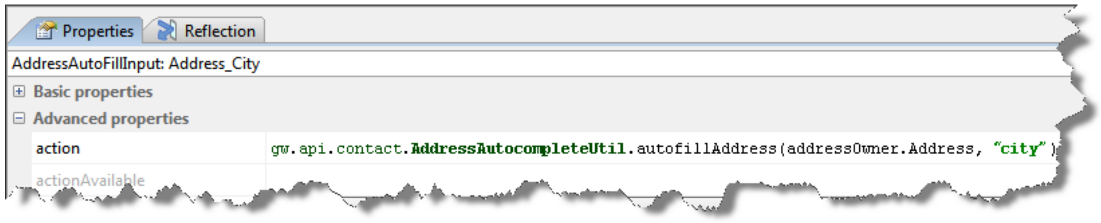
`country`	The country name. Make this correspond to a value in the zone mapping configuration.
`fieldName`	The field name for the field you want to fill.
`fieldNames`	An array of address field names. ClaimCenter uses this array to determine the autofill value.
`fieldValues`	An array of address field values. There must be a one-to-one correspondence between these values and the `fieldNames`.
`doOverride`	If true, ClaimCenter fills the `fieldName` regardless of whether the field is null or not.
`triggerIndex`	The index of the field that triggered the autofill.

ClaimCenter fills in the `fieldName` only if there is a unique match. If there is no unique match, the autocomplete configuration on the same field forces the application to show a narrowed list of possible matches.

If you are not sure what to do, you can use the `AddressAutocompleteUtil.autofillAddress` method along with the version of `getAutofillValue` that uses the `doOverride` argument to force an override. This method enables the user to explicitly overwrite a value.

For example, your enter San Mateo for the city and CA for the state. There are multiple ZIP codes for San Mateo, so the application does not autofill. You move to the ZIP field. Instead of taking values from the autocomplete drop-down, you enter `94` and tab out of the box. If `doOverride` is `true` on `getAutofillValue`, ClaimCenter changes San Mateo to San Francisco. You then have to reenter `San Mateo` in the city widget and reenter the ZIP code again to correct the values.

The `autofillAddress` method takes the address to autofill and a comma-separated list of address field names that the application uses to trigger the autofill.



action `gw.api.contact.AddressAutocompleteUtil.autofillAddress(addressOwner.Address, "city")`

You add this method to the `action` property of the `AddressAutoFillInput` widget.

See also

- “Understanding Autofill and Zone Mapping” on page 61
- “Importing Address Zone Data” on page 63
- “Working with the zone-config.xml File” on page 63
- “Configuring Addressing” on page 65

chapter 8

Contact Pickers and Search

This topic describes how to manage contact pickers and contact searching in your installation.

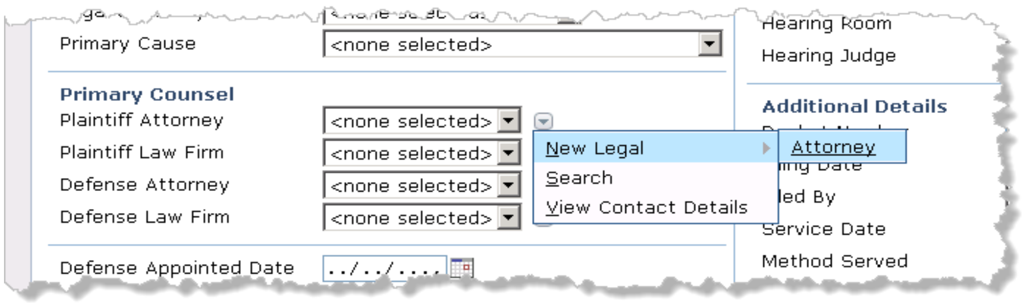
This topic includes:

- "How Pickers Work" on page 71
- "Example of a Constrained Picker" on page 72
- "Extending Contacts and Search with an Array" on page 76

How Pickers Work

In the application interface, you encounter *pickers* on screens where you enter contact information. A picker is a small widget beside a field indicating that you can search for a contact to enter into a field. The menu opened by the picker enables you to add a new contact, search for an existing contact, or view details about the specific contact referenced by the field.

When you use ClaimCenter Studio to add a `ClaimContactInput` field to a screen, the system automatically associates a picker with the field. You can configure a picker to show all available subtypes or a subset. For example, the attorney pickers in the litigation area show the following subset:



In the ClaimCenter Studio **Resources** tree, the **contacts** folder under the **Page Configuration (PCF)** → **shared** node contains the following PCF files that define the default picker menus:

- **ClaimNewCompanyOnlyPickerMenuItemSet**
- **ClaimNewContactPickerMenuItemSet**
- **ClaimNewDoctorOnlyPickerMenuItemSet**
- **ClaimNewPersonOnlyPickerMenuItemSet**
- **ClaimNewRepairShopOnlyPickerMenuItemSet**
- **NewContactPickerMenuItemSet**
- **NewPersonOnlyPickerMenuItemSet**

You can edit these PCF files to configure default menus associated with a picker. You can also create new PCF files that define custom menus and reference those as well. Additionally, you can prevent creation of new contacts with pickers and restrict the picker to just the **Search** choice and the **View Contact Details** choice.

See also

- "Example of a Constrained Picker" on page 72
- "Extending Contacts and Search with an Array" on page 76

Example of a Constrained Picker

When you add a new contact subtype, ClaimCenter automatically generates an array key for the subtype on the `Claim` entity. For example, adding the `Interpreter_Ext` subtype as described in "Example of Adding a Contact Subtype" on page 31 caused the system to create a `Claim.RelatedInterpreter_ExtArray` attribute. This attribute points to an array of `Interpreter_Ext` contacts. You can use the relationship between a subtype and its array to constrain a picker that enables adding only new contacts of a specific subtype.

In the following topics, you set up the picker menus and display views to enable designating an interpreter for each legal matter:

- Step1: Add the Subtype to the Claim Contact Picker Menus
- Step 2: Configure the Claim Contacts User Interface
- Step 3: Add a New Role
- Step 4: Configure the User Interface to Use the New Role

Step1: Add the Subtype to the Claim Contact Picker Menus

In ClaimCenter, take the following steps to ensure that your new subtype appears on the claim contact picker menus:

1. If necessary, start ClaimCenter Studio.

   At a command prompt, navigate to the `ClaimCenter\bin` directory and enter:

   ```
   gwcc studio
   ```

2. Expand **Page Configuration (PCF)** → **shared** → **contacts** and click **ClaimNewContactPickerMenuItemSet** so you can edit it.

3. Select the `NewContactPickerMenuItemSet_NewVendor` submenu.

4. The **Advanced** property **visible** contains a set of tests for contact type. Scroll all the way to the right and add the following test for the `Interpreter_Ext` type:

   ```
    or requiredContactType.isAssignableFrom(entity.Interpreter_Ext)
   ```

   When you are done, the **visible** property will have the following contents:

   ```
   requiredContactType == entity.Contact or requiredContactType == entity.Company or
   requiredContactType == entity.Person or requiredContactType == entity.PersonVendor or
   requiredContactType == entity.CompanyVendor or requiredContactType == entity.AutoRepairShop or
   requiredContactType == entity.AutoTowingAgcy or requiredContactType == entity.Doctor or
   requiredContactType == entity.MedicalCareOrg or
   requiredContactType.isAssignableFrom(entity.Interpreter_Ext)
   ```

5. Add a new `NewContactPickerMenuItemSet_Interpreter_Ext` menu item to the `NewContactPickerMenuItemSet_NewVendor` submenu and set the following properties:

action	`NewContactPopup.push(typekey.Contact.TC_INTERPRETER_EXT, parentContact, claim)`
id	`NewContactPickerMenuItemSet_Interpreter_Ext`
label	`displaykey.Web.NewContactMenu.Interpreter_Ext`
visible	`requiredContactType.isAssignableFrom(entity.Interpreter_Ext)`

Step 2: Configure the Claim Contacts User Interface

This part of the configuration is necessary to enable you to create a new `Interpreter` contact for a `Matter`.

1. In the **Resources** tree, expand **Page Configuration (PCF)** → **shared** → **contacts.**
2. Right-click **contacts**, then choose **New** → **PCF File**.
3. For **File type** choose **Input Set**, then enter the name `InterpreterAdditionalInfo`.
4. For **Mode** enter `Interpreter_Ext`, then click **OK**.

   This modal PCF file supports the addition of the **Specialty** field to the basic `PersonVendor InputSet`. If you were not adding a new field, you could skip this step.
5. Select the newly created **InputSet**, click the **Required Variables** tab, click **+** to add a variable, and enter the following values:

name	`contactHandle`
type	`contact.ContactHandle`

6. Click the **Code** tab and enter the following code:

```
property get Interpreter_Ext() :
  Interpreter_Ext {return contactHandle.Contact as Interpreter_Ext;}
```

7. Add an `Input` widget to the new resource and set the following properties:

editable	`true`
id	`InterpreterSpecialty`
label	`displaykey.Web.ContactDetail.Interpreter.InterpreterSpecialty`
required	`false`
value	`Interpreter_Ext.InterpreterSpecialty`

8. In the **Resources** tree under **Page Configuration (PCF)** → **shared** → **contacts**, right-click the `AdditionalInfoInputSet.PersonVendor` PCF file and choose **Change mode**.
9. Add `Interpreter_Ext` to the list of modes:

```
PersonVendor|Doctor|Attorney|Interpreter_Ext
```

   Adding the `Interpreter_Ext` mode updates the Shared Section mode of the `InputSetRef` so the `DetailView` will display the **InputSetRef** you add in the next steps.
10. Click the `AdditionalInfoInputSet.PersonVendor` PCF file to edit it, and add an **InputSetRef** to the view.
11. Set the properties to the following values:

def	`InterpreterAdditionalInfoInputSet(contactHandle)`
mode	`PersonVendor typeis Interpreter_Ext ? "Interpreter_Ext" : null`

This statement enables the **Specialty** field to display in the **Additional Info** section when the `Contact` subtype is `Interpreter_Ext`.

12. Navigate in the **Resources** tree to **Page Configuration (PCF)** → **shared** → **contacts** and right-click the **ContactBasicsDV.Person** PCF file, then click **Change mode**.

13. In the **Change Mode** dialog, add `Interpreter_Ext` to the list of modes:

```
Person|PersonVendor|Adjudicator|UserContact|Doctor|Attorney|Interpreter_Ext
```

Adding the `Interpreter_Ext` mode updates the Shared Section mode of the `InputSetRef` so the `DetailView` will display the **InputSetRef** you added in the previous steps.

Step 3: Add a New Role

You cannot directly access the `Interpreter_Ext` subtype from the `Matter` entity as a foreign key, because doing so causes cyclic graph errors with the entities. Instead, you use an `interpreter` role that is tied to the `Interpreter_Ext` subtype.

To add the new role

1. Ensure that you have created the `Interpreter_Ext` subtype as described in “Example of Adding a Contact Subtype” on page 31.
2. In Studio, add the `interpreter` role as a typecode to the `ContactRole` typelist.
   a. In the **Resources** tree, expand the **Typelists** node, then click **ContactRole** to edit this typelist.
   b. Click **Add**, then enter the following typecode:

Code	Name	Description
`interpreter`	`interpreter`	`Interpreter for a Matter`

3. Add the `interpreter` role to the `litigation` contact role category.
   a. In the **Resources** tree under **Typelists**, click **ContactRoleCategory** to edit this typelist.
   b. On the **Codes** tab, select the row for the `litigation` role.
   c. On the **Categories** tab, select a row in the **Categories** list, and then click **Add**.
   d. Enter the `interpreter` typecode:

Typelist	Code
`ContactRole`	`interpreter`

4. In the `entityroleconstraints-config.xml` file, add the `interpreter` role, and then add `interpreter` as a role reference to the list of `Matter` roles.
   a. In the **Resources** tree, expand **Other Resources** and click **entityroleconstraints-config.xml** to edit it.
   b. Add the following `ContactRoleTypeConstraint` to the section for role constraints at the top of the file:

```
<ContactRoleTypeConstraint contactRoleCode="interpreter"
                           contactSubtype="Interpreter_Ext"/>
```

   c. Add the `interpreter` role with the `Exclusive` (zero or one interpreter) constraint to the `Matter` entity, `<EntityRef entityType="Matter">`:

```
<RoleRef contactRoleCode="interpreter">
  <RoleConstraint constraintType="Exclusive"/>
</RoleRef>
```

5. Save the file, and then stop ClaimCenter, regenerate the APIs, optionally regenerate the data dictionary, and restart ClaimCenter, as described in the following steps.

You need to perform all these steps because the role changes and additions are data model changes.

a. At a command prompt, navigate to the ClaimCenter `bin` directory.

b. If necessary, stop ClaimCenter.

   At the command prompt in which ClaimCenter is running, press **CTR L+ C** to stop the batch process, and then enter the following commands:

```
y
gwcc dev-stop
```

c. Regenerate the APIs:

```
gwcc regen-java-api
gwcc regen-soap-api
```

d. Optionally regenerate the data dictionary:

```
gwcc regen-dictionary
```

e. Start the ClaimCenter application:

```
gwcc dev-start
```

6. Quit ClaimCenter Studio and restart it to enable use of the `Matter.interpreter` virtual property.

```
gwcc studio
```

Step 4: Configure the User Interface to Use the New Role

With the `interpreter` role set up, you can now configure the `Matter` menus and screens to use it.

1. In Studio, click **Display Keys** and navigate to **NVV** → **Matter** → **SubView** → **MatterGeneral** → **Counsel**.
2. Right-click **Counsel**, click **Add**, and enter the **Display Key Name** `NVV.Matter.SubView.MatterGeneral.Counsel.Interpreter` and the **Default Value** `Interpreter`.
3. Create a similar display key in **NVV** → **Matter** → **SubView** → **NewMatterGeneral** → **Counsel** with a new display key named `NVV.Matter.SubView.NewMatterGeneral.Counsel.Interpreter` and value `Interpreter`.
4. Navigate in the **Resources** tree to **Page Configuration (PCF)** → **claim** → **newother** and click the **NewMatterDV** PCF file to edit it.
5. Below the `Counsel_DefenseLawFirm` field and above the input divider, add a `ClaimContactInput` widget for the `Interpreter` with the following property settings:

editable	`true`
id	`Counsel_Interpreter`
label	`displaykey.NVV.Matter.SubView.NewMatterGeneral.Counsel.Interpreter`
required	`false`
value	`Matter.interpreter`
claim	`Matter.Claim`
valueRange	`Matter.Claim.RelatedInterpreter_ExtArray`
visible	`Matter.MatterType == "Lawsuit" || Matter.MatterType == "Arbitration" || Matter.MatterType == "Hearing" || Matter.MatterType == "General"`

   When you add a new contact field to a dialog, the system generates a picker for that field. The picker is constrained by the expected value `Matter.interpreter`. The **valueRange** attribute must point to an array of the expected values.

6. Navigate in the **Resources** tree to **Page Configuration (PCF)** → **claim** → **litigation** and click the **MatterDetailsDV** PCF file to edit it.

7. Find the `Counsel_DefenseLawFirm` field near the bottom of the left input column. Below this field and above the input divider add a `ClaimContactInput` widget with the following property settings:

editable	`true`
id	`Counsel_Interpreter`
label	`displaykey.NVV.Matter.SubView.MatterGeneral.Counsel.Interpreter`
required	`false`
value	`Matter.interpreter`
claim	`Matter.Claim`
valueRange	`Matter.Claim.RelatedInterpreter_ExtArray`
visible	`Matter.MatterType == "Lawsuit" || Matter.MatterType == "Arbitration" || Matter.MatterType == "Hearing" || Matter.MatterType == "General"`

8. Test your changes.
   a. Log in to ClaimCenter and select an existing claim.
   b. Click **Litigation** on the left and open a new matter.
   c. On the **New Matter** page, choose a matter type other than `Mediation` (for which `Interpreter` is not defined).
   d. Give the matter a name, and then add an interpreter and save the matter.

      When you select the **Interpreter** picker, the system presents a constrained picker.

Other Things to Try

If a user uses the new picker to search for a contact, the system restricts the search to **Interpreter** contacts. If you want to hide the **New Vendor** menu, specify a **BlankNewContactPickerMenuItemSet** in the **newContactMenu** field of the **ClaimContactInput** widget. The system will display **Search** and **View Contact Details** as the only options. You could also restrict the picker by specifying a limited menu such as **ClaimNewPersonOnlyPickerMenuItemSet** for the **newContactMenu** value. Finally, you could define your own menu in a PCF file and specify it in the **newContactMenu**.

See also

- "How Pickers Work" on page 71
- "Extending Contacts and Search with an Array" on page 76

Extending Contacts and Search with an Array

This topic describes how to extend the `ABContact` entity in ContactCenter and the `Contact` entity in ClaimCenter with an array of states that indicate the service area for the contact. Adding an array extension to an entity is fairly standard. This topic also shows you how to make the array entities work together across both products and then connect them to the Search feature in each product.

The following topics walk you through adding this functionality:

- Step 1: Create New Entities in ContactCenter and ClaimCenter
- Step 2: Add Extension Arrays to Parent Entities
- Step 3: Map Entities and Array Extensions
- Step 4: Extend the ClaimCenter User Interface
- Step 5: Extend the ContactCenter User Interface
- Step 6: Add a Preupdate Rule and a Gosu Class in ContactCenter
- Step 7: Test Your Changes
- Step 8: Extend the Contact Search Criteria Entities

- Step 9: Add Search Capability in ContactCenter
- Step 10: Identify Field as a Primary Search Key in ContactCenter
- Step 11: Configure the Address Book Search Interface in Both Applications
- Step 12: Test Your Changes

Step 1: Create New Entities in ContactCenter and ClaimCenter

In this topic, you add a service state entity to ContactCenter and to ClaimCenter. As with the contact subentity `Interpreter_Ext` added in "Example of Adding a Contact Subtype" on page 31, you need to add the entity to both products.

These entities have the following notable features:

- The entity names begin with a capital letter.
- The entity names and their corresponding element names are the same in both applications.
- The array entity and the corresponding elements are set to `exportable="true"`.
- The foreign key elements in both applications are set to `nullok="false"`.
- There are both forward and reverse indexes for all searchable elements.
- The `LinkID` column in the ContactCenter entity has a unique index defined for it.

  Note: The index name can have no more than 18 characters.
- Both entities must implement the correct delegates. The ClaimCenter object must implement `Extractable`, `OverlapTable`, and `AddressBookLinkable`. The ContactCenter object must implement `ABLinkable`.

Adding the Service State Entity to ContactCenter

1. If necessary, start ContactCenter Studio.

   At a command prompt, navigate to `ContactCenter\bin`, and enter the following command:

   ```
   gwab studio
   ```

2. In the **Resources** tree on the left, navigate to **Data Model Extensions** → **extensions**, and then right-click and choose **New** → **Other file**.

3. Type the following file name and press **ENTER**:

   ```
   ContactServiceState_Ext.eti
   ```

4. In the editor, enter the following code defining the new entity:

   ```
   <?xml version="1.0"?>
   <!-- Service State in ContactCenter -->
   <entity xmlns="http://guidewire.com/datamodel"
     desc="Represents a state where the contact provides services."
     entity="ContactServiceState_Ext"
     exportable="true"
     loadable="true"
     platform="false"
     table="ContactServiceState_Ext"
     type="retireable">
     <implementsEntity name="ABLinkable"/>
     <fulldescription>
       <![CDATA[Represents a state where the contact provides services.]]>
     </fulldescription>
     <foreignkey
       columnName="ContactID"
       desc="Contact that this Service State table relates to."
       fkentity="ABContact"
       name="Contact"
       nullok="false"/>
     <typekey
       desc="State serviced by the contact"
       name="ServiceState"
       typelist="State"
       exportable="true"
       loadable="true">
   ```

```
      </typekey>
      <index
        desc="Preserve uniqueness of LinkID"
        name="absrvstatelinkid"
        unique="true">
        <indexcol keyposition="1" name="LinkID"/>
        <indexcol keyposition="2" name="Retired"/>
      </index>
      <index name="ind1" unique="true">
        <indexcol keyposition="1" name="ServiceState"/>
        <indexcol keyposition="2" name="Retired"/>
        <indexcol keyposition="3" name="ContactID"/>
      </index>
      <index name="ind2" unique="true">
        <indexcol keyposition="1" name="ContactID"/>
        <indexcol keyposition="2" name="Retired"/>
        <indexcol keyposition="3" name="ServiceState"/>
      </index>
    </entity>
```

5. Save the file.

Adding the Service State Entity to ClaimCenter

1. In ClaimCenter Studio, go to the **Resources** tree on the left, navigate to **Data Model Extensions** → **extensions**, and then right-click and choose **New** → **Other file**.
2. Type the following file name and press **ENTER**:

```
ContactServiceState_Ext.eti
```

3. In the editor, enter the following code defining the new entity:

```
<?xml version="1.0"?>
<!-- Service State entity in ClaimCenter -->
<entity xmlns="http://guidewire.com/datamodel"
  desc="Represents a state where the contact provides services."
  entity="ContactServiceState_Ext"
  exportable="true"
  loadable="true"
  platform="false"
  table="ContactServiceState_Ext"
  type="retireable">
  <implementsEntity name="Extractable"/>
  <implementsEntity name="OverlapTable"/>
  <implementsEntity name="AddressBookLinkable"/>
  <fulldescription>
    <![CDATA[Represents a state where the contact provides services.]]>
  </fulldescription>
  <column
    desc="Represents ID of associated object in Address Book.
          Null if object not linked to Address Book."
    name="AddressBookUID"
    loadable="true"
    type="varchar">
    <columnParam name="size" value="30"/>
  </column>
  <foreignkey
    columnName="ContactID"
    desc="Contact that this Service State table relates to."
    fkentity="Contact"
    name="Contact"
    nullok="false"/>
  <typekey
    desc="State serviced by the contact"
    name="ServiceState"
    typelist="State"
    exportable="true"
    loadable="true">
  </typekey>
  <index name="ind1" unique="true">
    <indexcol keyposition="1" name="ServiceState"/>
    <indexcol keyposition="2" name="Retired"/>
    <indexcol keyposition="3" name="ContactID"/>
  </index>
  <index name="ind2" unique="true">
    <indexcol keyposition="1" name="ContactID"/>
    <indexcol keyposition="2" name="Retired"/>
    <indexcol keyposition="3" name="ServiceState"/>
    </index>
  </entity>
```

4. Save the file.

Step 2: Add Extension Arrays to Parent Entities

In this topic you extend the `Contact` entity in ClaimCenter and the `ABContact` entity in ContactCenter to enable each to use an array of service state entities.

Key points to note with these extension arrays:

- The `triggersValidation` attribute is set to `true` for both entities. Setting this attribute enables validation rules to be triggered when an element of the array changes.
- The array names are the same in both applications.

Adding the Array Entity to ContactCenter

1. In ContactCenter Studio, in the **Resources** tree on the left, navigate to **Data Model Extensions** → **extensions**.
2. Click **ABContact.etx** to open the entity extension in an editor.
3. At the end of the file before the </extension> tag, insert the following code that defines the new array:

```
<array
  arrayentity="ContactServiceState_Ext"
  arrayfield="Contact"
  desc="Geographical area where the contact provides service."
  name="ContactServiceArea_Ext"
  triggersValidation="true"/>
```

4. Save the file.

Adding the Array Entity to ClaimCenter

1. In ClaimCenter Studio, in the **Resources** tree on the left, navigate to **Data Model Extensions** → **extensions**.
2. Click **Contact.etx** to open the entity extension in an editor.
3. At the end of the file before the </extension> tag, insert the following code that defines the new array:

```
<array
  arrayentity="ContactServiceState_Ext"
  arrayfield="Contact"
  desc="Geographical area where the contact provides service."
  name="ContactServiceArea_Ext"
  triggersValidation="true"/>
```

4. Save the file.

Step 3: Map Entities and Array Extensions

As described in "Working with Contact Mapping Files" on page 29, you need to map the new entities between ClaimCenter and ContactCenter. You do this mapping in the ClaimCenter files `ab-to-cc-data-mapping.xml` and `cc-to-ab-data-mapping.xml`.

1. In ClaimCenter Studio, in the **Resources** tree on the left, open the **Other Resources** folder and click `cc-to-ab-data-mapping.xml` to open it in an editor.
2. Enter the following entity mapping after the `Contact` to `ABContact` entity mapping:

```
<EntityMapping source="ContactServiceState_Ext" target="ContactServiceState_Ext">
  <FieldMapping source="PublicID"
                mapperClassName="gw.api.util.mapping.NullFieldMapper"/>
  <FieldMapping source="AddressBookUID"
                mapperClassName="gw.api.util.mapping.NameTranslatingFieldMapper">
    <MapperProperty name="newFieldName" value="LinkID"/>
  </FieldMapping>
</EntityMapping>
```

3. In the **Resources** tree, click `ab-to-cc-data-mapping.xml` to open it in an editor.
4. Enter the following entity mapping after the `ABContact` to `Contact` entity mapping

```
<EntityMapping source="ContactServiceState_Ext" target="ContactServiceState_Ext">
  <FieldMapping source="PublicID"
                mapperClassName="gw.api.util.mapping.NullFieldMapper"/>
  <FieldMapping source="LinkID"
                mapperClassName="gw.api.util.mapping.NameTranslatingFieldMapper">
    <MapperProperty name="newFieldName" value="AddressBookUID"/>
  </FieldMapping>
</EntityMapping>
```

Step 4: Extend the ClaimCenter User Interface

Using ClaimCenter Studio, make the following user interface changes.

Enable ClaimCenter Address Book to Add States to a Company

1. In the **Resources** tree, click **Display Keys.**
2. Navigate to **Web** → **ContactDetail**, right-click and choose **Add**, and create a new display key with name `Web.ContactDetail.ServiceStates` and default value `Service States`.
3. In the **Resources** tree, expand **Page Configuration (PCF)**, then right-click **addressbook** and click **New** → **PCF File**.
4. For **File type** select **List View**, then enter the **File name** `AddressBookServiceStates` and click `OK` to edit this PCF file.
5. Click the new list view panel to open its **Properties** window, then click the **Required Variables** tab.
6. Click the plus (+) button and add the following new required variable:

name	`contact`
type	`Contact`

7. Add a **RowIterator** with the following values:

editable	`true`
elementName	`currentServiceState`
toAdd	`contact.addToContactServiceArea_Ext(currentServiceState)`
toRemove	`contact.removeFromContactServiceArea_Ext(currentServiceState)`
value	`contact.ContactServiceArea_Ext`
canPick	`true`

8. Add a new **Row** to the **RowIterator**, and to the new row add a **Cell**, and then set the following cell properties:

editable	`true`
id	`ServiceState`
value	`currentServiceState.ServiceState`
align	`center`
unique	`true`
validationLabel	`displaykey.Web.ContactDetail.ServiceStates`

9. In the **Resources** tree, navigate to **Page Configuration (PCF)** → **addressbook** and click **AddressBookContactBasicsDV.Company** to edit this PCF file.

10. At the bottom of the input column containing the `TextAreaInput` called `Notes`, add a **ListViewInput** with the following attributes:

def	`AddressBookServiceStatesLV(contact)`
label	`displaykey.Web.ContactDetail.ServiceStates`
labelAbove	`true`
boldLabel	`true`

11. Add a **Toolbar** to the **ListViewInput** above the row iterator, and then add **Iterator Buttons** to the toolbar so you can add and remove states.

Enable Addition of States to a Company Associated with a Claim

1. In the **Resources** tree, navigate to **Page Configuration (PCF)** → **shared** → **contacts**, then right-click **contacts** and click **New** → **PCF File.**
2. For **File type** choose **List View**, then enter the **File name** `VendorServiceStates` and click `OK` to edit this PCF file.
3. Click the new list view panel to open its **Properties** window, then click the **Required Variables** tab.
4. Click the plus (+) button and add the following new required variable:

name	`contactHandle`
type	`contact.ContactHandle`

5. Click the Code tab and enter the following code:

```
property get contact() : Contact { return contactHandle.Contact; }
```

6. Add a **RowIterator** with the following values:

editable	`true`
elementName	`currentServiceState`
toAdd	`contact.addToContactServiceArea_Ext(currentServiceState)`
toRemove	`contact.removeFromContactServiceArea_Ext(currentServiceState)`
value	`contact.ContactServiceArea_Ext`
canPick	`true`

7. Add a new **Row** to the **RowIterator**, and to the new row add a **Cell**, and then set the following cell properties:

editable	`true`
id	`ServiceState`
value	`currentServiceState.ServiceState`
align	`center`
unique	`true`
validationLabel	`displaykey.Web.ContactDetail.ServiceStates`

8. In the **Resources** tree, navigate to **Page Configuration (PCF)** → **shared** → **contacts**, click **ContactBasicsDV.Company** to edit this PCF file.
9. At the bottom of the input column containing the `TextAreaInput` called `Notes`, add a **ListViewInput** with the following attributes:

def	`VendorServiceStatesLV(contactHandle)`

label	`displaykey.Web.ContactDetail.ServiceStates`
labelAbove	`true`
boldLabel	`true`

10. Add a **Toolbar** to the **ListViewInput** above the row iterator, and then add **Iterator Buttons** to the toolbar so you can add and remove states.

Step 5: Extend the ContactCenter User Interface

Using ContactCenter Studio, make the following user interface changes.

1. In the **Resources** tree, click **Display Keys.**
2. Navigate to **Web** → **ContactDetail**, right-click and choose **Add**, and create a new display key with name `Web.ContactDetail.ServiceStates` and default value `Service States`.
3. In the **Resources** tree, navigate to **Page Configuration (PCF)** → **contacts** → **basics**, then click **ContactBasicsDV.ABCompany** to edit this PCF file.
4. At the bottom of the input column containing the `TextAreaInput` called `Notes`, add a **ListViewInput** with the following attributes:

label	`displaykey.Web.ContactDetail.ServiceStates`
labelAbove	`true`
boldLabel	`true`

5. Add a **Toolbar**, and then add **Iterator Buttons** to the toolbar so you can add and remove states.
6. Add a **RowIterator** with the following values:

editable	`true`
elementName	`currentServiceState`
toAdd	`contact.addToContactServiceArea_Ext(currentServiceState)`
toRemove	`contact.removeFromContactServiceArea_Ext(currentServiceState)`
value	`contact.ContactServiceArea_Ext`
canPick	`true`

7. Add a new **Row** to the **RowIterator**, and to the new row add a **Cell**, and then set the following cell properties:

editable	`true`
id	`ServiceState`
value	`currentServiceState.ServiceState`
align	`center`
unique	`true`
validationLabel	`displaykey.Web.ContactDetail.ServiceStates`

Step 6: Add a Preupdate Rule and a Gosu Class in ContactCenter

In this topic, you add a Gosu class that ensures that the `LinkID` is set correctly for the new array. You then add a preupdate rule that calls the class. The purpose of this rule is to set the `LinkID` correctly on the new array entity that you added to `ABContact`. You need this rule only for custom array entity extensions that implement

AddressBookLinkable. ContactCenter sets the LinkID correctly for ABContact array entities supplied in the base configuration.

Add a Gosu Class That Updates the LinkID

1. Using ContactCenter Studio, in the **Resources** tree on the left expand **Classes** → **gw** and right-click **entity**, then click **New** → **class**.
2. For the class name, enter ABLinkableBundleTransactionCallback.
3. In the code editor, enter the following code:

```
/*
This class is called by the ABContact Preupdate rule
ABPU01000 - Update Service Areas LinkID.
The class updates the LinkID on custom array entities to ensure
that the unique index that contains the LinkID in ABContact can
be set correctly. The LinkID is set to the value of the PublicID.
*/
package gw.entity

uses gw.transaction.AbstractBundleTransactionCallback
uses java.lang.IllegalArgumentException

class ABLinkableBundleTransactionCallback extends
      AbstractBundleTransactionCallback {

  private var abLinkable : ABLinkable

  construct(_abLinkable : ABLinkable) {
    if (not (_abLinkable typeis KeyableBean)) {
      throw new IllegalArgumentException(
        "abLinkable must also be a KeyableBean")
    }
    this.abLinkable = _abLinkable

  }

  override public function afterSetIds() {
    abLinkable.LinkID = (abLinkable as KeyableBean).PublicID
  }
}
```

Setting the Preupdate Rule

1. Using ContactCenter Studio, in the **Resources** tree on the left expand **Rule Sets** → **Preupdate** and click **ABContact-Preupdate**.
2. When the rule set editor opens, right-click the **ABContact Pre-update** rule set node and choose **New Rule**, then enter the name ABPU01000 - Update Service Areas LinkID.
3. Enter the following comments in **Rule Conditions** before the condition true:

```
/*
This ABContact Preupdate Rule is required to correctly
set the LinkID on a custom array entity extension
that implements AddressBookLinkable in ContactCenter.
The LinkID is set to the value of the PublicID.
The rule calls the Gosu class ABLinkableBundleTransactionCallback.
*/
true
```

4. Enter the following code in **Rule Actions**:

```
for (record in aBContact.ContactServiceArea_Ext)
{
  if (record.New==true)
  {
    record.Bundle.addBundleTransactionCallback (new
      gw.entity.ABLinkableBundleTransactionCallback (record))
  }
}
```

Step 7: Test Your Changes

Test the changes you have made so far to ensure that they all work before continuing with the rest of the exercise. The data model extensions must be working before you can continue with setting up search for the new service states data.

Shut Down, Regenerate, and Restart Both Applications

These instructions assume that you are using QuickStart. If you are not, you must regenerate the WAR file for each application after regenerating the APIs. For example, at a command prompt navigate to `ClaimCenter\bin` and enter `gwcc build-war`.

1. At a command prompt, navigate to the `ClaimCenter\bin` directory.
2. If necessary, stop ClaimCenter.

   At the command prompt in which ClaimCenter is running, press **CTR L+ C** to stop the batch process, and then enter the following commands:

   ```
   y
   gwcc dev-stop
   ```
3. Regenerate the APIs:

   ```
   gwcc regen-java-api
   gwcc regen-soap-api
   ```
4. Optionally regenerate the data dictionary:

   ```
   gwcc regen-dictionary
   ```
5. At the command prompt in which ContactCenter is running, press **CTRL+C** to stop the batch process and then enter the following commands:

   ```
   y
   gwab dev-stop
   ```
6. Regenerate the APIs:

   ```
   gwab regen-java-api
   gwab regen-soap-api
   ```
7. Optionally regenerate the data dictionary:

   ```
   gwab regen-dictionary
   ```
8. Start the ContactCenter application:

   ```
   gwab dev-start
   ```
9. In ClaimCenter Studio, refresh the ContactCenter plugin.

   a. In the **Resources** tree on the left, expand the **Web Services** node and click **abintegration** to open an editor for this web service.

   b. Click the **Refresh** button near the top of the editor, to the right of the **URL** field.

   c. If prompted to make a copy of the resource for editing, click `OK` and then click **Refresh** again.
10. Start the ClaimCenter application:

    ```
    gwcc dev-start
    ```
11. Test both applications as described in the topics that follow.

Test the ClaimCenter Changes

1. Log in to ClaimCenter as a user who can create new contacts. For example, log in as user `su` with password gw.
2. Click the **Address Book** tab, then choose **Actions** → **New Company** to see the **New Company** screen. Notice that **Service States** is listed under the **Notes** section on the right.

3. Click **Add** to add a state, then select a state from the list.
4. Select the new row, and then click **Remove** to delete it.
5. Add several states, and then enter enough information to create the new `Company` and click **Update Address Book**.
6. Click the **Claim** tab and open an existing claim.
7. Click **Parties Involved** in the left **Sidebar**, and on that screen click **New Contact** → **Company**. Notice that **Service States** is listed under the **Notes** section on the right.
8. Click **Add** to add a state, then select a state from the list.
9. Select the new row, and then click **Delete** to delete it.
10. Add several states, and then enter enough information to create the new company.
11. Under **Roles** near the top of the screen, click **Add** and add a role for this company for the current policy.
12. Click **Update** to add the company to the **Parties Involved** list.

Test the ContactCenter Changes

1. Log in to ContactCenter as a user who can create new contacts. For example, log in as user `su` with password `gw`.
2. Search for the company added in ClaimCenter to see if it was correctly added to the database.
3. Click **Actions** → **New Company** and choose each kind of company to see that there is a **Service States** list for each type of company.
4. Choose a type of company to create, then add and delete a service state. Add several service states, then enter enough information to create the new `Company`.
5. Click **Update** to save the company.
6. Go back to ClaimCenter and search in the address book for the company that was added in ContactCenter.

Step 8: Extend the Contact Search Criteria Entities

In this topic, you take the first steps to make the new `ServiceState` column searchable in both ClaimCenter and ContactCenter. The column names must match in both applications.

1. Using ClaimCenter Studio, in the **Resources** tree on the left, navigate to **Data Model Extensions** → **extensions**.
2. Click **ContactSearchCriteria.etx** to open the entity extension in an editor.
3. Add the following typekey:

```
<typekey
  desc="Service State"
  name="ServiceState"
  typelist="State"/>
```

4. Using ContactCenter Studio, in the **Resources** tree on the left, navigate to **Data Model Extensions** → **extensions**.
5. Click **ABContactSearchCriteria.etx** to open the entity extension in an editor.
6. Add the following typekey:

```
<typekey
  desc="Service State"
  name="ServiceState"
  typelist="State"/>
```

7. Save the file.

Step 9: Add Search Capability in ContactCenter

Add an ArrayCriterion element to the ContactCenter search-config.xml file to specify where the ServiceState column is located. In this element:

- property is the name attribute given for the column in the ABContactSearchCriteria entity.
- targetProperty is the name of the array on the base entity ABContact.
- arrayMemberProperty is the name of the column on the extension array entity ContactServiceState_Ext.

To add the element to search-config.xml

1. Using ContactCenter Studio, in the **Resources** tree on the left, expand **Other Resources** and click **search-config.xml** to edit the file.
2. Increment the version attribute in the SearchConfig element at the top of the file. In the following example, the original version number was 1 and has been changed to 2.

```
<SearchConfig
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="search-config.xsd"
  version="2">
```

3. In the CriteriaDef element for the target entity ABContact, add a new ArrayCriterion element as follows:

```
<!-- Search by ABContact Fields -->
  <CriteriaDef entity="ABContactSearchCriteria" targetEntity="ABContact">
    <Criterion property="TaxID" matchType="eq"/>
    <Criterion property="VendorType" matchType="eq"/>
    <Criterion property="Keyword" matchType="startsWith"/>
    <Criterion property="Score" matchType="ge"/>
    <ArrayCriterion property="ServiceState"
                    targetProperty="ContactServiceArea_Ext"
                    arrayMemberProperty="ServiceState" />
    <!-- Some additional search criteria are defined in code -->
  </CriteriaDef>
```

4. Save the file.

Step 10: Identify Field as a Primary Search Key in ContactCenter

Make the new field in ABContactSearchCriteria a primary search key by adding it to the AdditionalPrimaryABContactSearchCriteria parameter in config.xml.

1. Using ContactCenter Studio, in the **Resources** tree on the left, expand **Other Resources** and click **config.xml** to edit the file.
2. Search for AdditionalPrimaryABContactSearchCriteria and add the ServiceState field from ABContactSearchCriteria as its value. If a value is already set for the param, add another AdditionalPrimaryABContactSearchCriteria element.

```
<!-- Names (comma separated) of fields on ABContactSearchCriteria that
     would be searchable even without specifying any other criterion.
     Remember to add the appropriate indexes to the ABContact table and
     to update the MinimumSearchCriteria display properties. -->
  <param name="AdditionalPrimaryABContactSearchCriteria" value="ServiceState"/>
```

   Note: The value is not case sensitive, but to maintain consistency, use the same capitalization used when specifying the field in ABContactSearchCriteria.

3. Save the file.

Step 11: Configure the Address Book Search Interface in Both Applications

In this step, you add the ServiceState search field to the PCF files used in searching for Address Book contacts in both ContactCenter and ClaimCenter.

Configure ContactCenter

1. In the **Resources** tree on the left, click **Display Keys** and navigate to **Web** → **ContactSearch**.
2. Right-click **ContactSearch** and choose **Add**, then create a new display key with name `Display Keys.Web.ContactSearch.ServiceState` and default value `Service State`.
3. In the **Resources** tree, navigate to **Page Configuration (PCF)** → **search** and click **ContactSearchDV** to edit this PCF file.
4. In the **InputColumn** on the left, locate the **InputSet** containing the **Organization Name** input, then add an **InputSet** widget under it.
5. Click the new **InputSet** and set the following property:

visible	`isCompany(SearchCriteria)`

6. Add an **Input** widget to the **InputSet** and set the following properties:

editable	`true`
id	`ServiceState`
label	`displaykey.Web.ContactSearch.ServiceStates`
value	`SearchCriteria.ServiceState`

Configure ClaimCenter

1. In the **Resources** tree on the left, click **Display Keys** and navigate to **Web** → **AddressBook** → **Search**.
2. Right-click **Search** and choose **Add**, then create a new display key with name `Web.AddressBook.Search.ServiceStates` and default value `Service States`.
3. In the **Resources** tree, navigate to **Page Configuration (PCF)** → **addressbook** and click **AddressBookSearchDV** to edit this PCF file.
4. In the **InputColumn** on the left, locate the **InputSet** containing the **Organization Name** input, then add an **InputSet** widget under it.
5. Click the new **InputSet** and set the following property:

visible	`searchCriteria.isSearchFor(entity.Company)`

6. Add an **Input** widget to the **InputSet** and set the following properties:

editable	`true`
id	`ServiceState`
label	`displaykey.Web.AddressBook.Search.ServiceState`
value	`searchCriteria.ServiceState`

Step 12: Test Your Changes

1. Shut down and restart both ClaimCenter and ContactCenter as described previously in “Shut Down, Regenerate, and Restart Both Applications” on page 84.
2. Log in to ContactCenter and click **Search** on the left, then choose **Company** in the **Contact Type** drop-down.
3. Search for one of the companies you created during your earlier tests.
   a. If you have not done so already, create a new company and specify a service state. Then do a search for that company with the state specified in the search to see if the search returns that company.

   b. Search for the same company with a state specified that is not in the list of states for that company to see if you get zero returns.
   c. Do a search that includes that company and some others that do not have the state defined for them and see if the search returns only the correct company.
4. Log in to ClaimCenter and use the Address Book to search for a company. Use the same search strategies as you did for ContactCenter.

See also

- "How Pickers Work" on page 71
- "Example of a Constrained Picker" on page 72

chapter 9

Proximity Search Using Geocoding

This topic introduces geocoding and discusses how you can use it in your environment. Additionally, it covers the proximity search feature, which uses the geocoding feature to pinpoint addresses geographically and then uses the result to calculate distances between addresses.

This topic includes:

Overview of the Geocoding Feature

This section discusses what geocoding is, how an application can use it, and aspects of how geocoding works in the application.

This topic covers:

- What is Geocoding?
- How the Application Uses Geocoding
- Synchronous and Asynchronous Geocoding
- The Data Model and Geocoding

What is Geocoding?

Guidewire supports *geocoding* in ClaimCenter and ContactCenter. Geocoding enables the assignment of latitude and longitude to an address. Latitude indicates how far north or south a location is from the Earth's equator. Longitude is how far east or west a point is relative to the prime meridian running through Greenwich, England.

By assigning latitude and longitude, the system is able to pinpoint an address as a location and specify its geographic coordinates. The application can then use these geographic coordinates to present data like the distance between two addresses.

Geocoding is optional in ClaimCenter. To implement geocoding in your installation, you must use a geocoding plugin. The geocoding plugin's tasks are:

- Finding candidate addresses based on a set of coordinates.
- Assigning a latitude, longitude, and geocode status to a given address synchronously (*geocoding*).
- Getting driving directions and maps between two already geocoded addresses (optional operation).
- Getting a map centered on a single geocoded address.
- Correcting the fields of a passed-in address.

The plugin uses a third-party mapping service that calculates and returns geographic coordinates for an address. Guidewire provides you with a fully functioning and supported geocoding plugin that makes use of Microsoft's Bing Maps Web Service. The plugin is written in Gosu, and you can use Studio to edit it.

> **Note:** Microsoft is discontinuing the MapPoint web service on November 18, 2011. If you currently use this service in your application, you will need to switch to Bing Maps. For more information on getting going with Bing Maps and using the Bing Maps plugin, see "Configuring Geocoding" on page 94.

For detailed information about implementing your own custom geocoding plugin, see "Geographic Data Integration", on page 315 in the *ClaimCenter Integration Guide*.

How the Application Uses Geocoding

Geocoding enables a Guidewire application to provide the following features in the user interface:

- **Assigning users to work based on proximity**. Assign work tasks based on the distance between two addresses, such as the insured and adjuster's addresses, or the adjuster's address and the claim loss address. This assignment is accomplished through assignment rules that assign work to users close to an address, such as the loss location.
- **Searching by proximity to an address.** Search for nearby service providers based on proximity to a given location, such as the loss location or a claimant's home address. You can restrict such searches by distance or number of results.
- **Providing driving directions and maps.** Return driving directions and times between two locations, such as between a potential client and an agent's office.

You can develop your own custom features that use geocoding. For example, you might write Gosu rules that send a claimant an email containing your recommended vendor, along with a map and driving directions. Or, you could write specialized assignment methods that assign an adjuster both by proximity to a claimant and by some other attribute, such as language.

Synchronous and Asynchronous Geocoding

The geocoding feature processes address data both synchronously and asynchronously.

Synchronous Processing

Synchronous processing occurs when the user requests a geocoding function on a new address, such as listing body shops close to a claim's loss location. Synchronous geocoding also takes place when a rule containing a Gosu request for geocoding runs.

During synchronous processing, the user interface is inaccessible until it receives the geocoded address from the mapping service. If the geocode application is unavailable or takes more than sixty seconds to reply, ClaimCenter stops the process and returns an error message. The wait time is coded into the plugin, and you can modify it in the web services definition in Studio.

Note: For installations with very large databases, a long search can also be caused by a heavy load on the ContactCenter server. There is no timeout in this case because this communication is internal, between two Guidewire systems.

Asynchronous Processing

The geocoding feature uses the Guidewire work queue infrastructure for asynchronous processing of addresses stored in the database. The work queue infrastructure supports both a daemon writer and a daemon worker. The writer daemon searches the database for addresses without geocodes and marks them for geocoding. The worker daemon submits the marked addresses to the plugin's geocoding service. After the geocoding plugin geocodes an address, the worker daemon retrieves its geocode and saves it to the database. If an address changes during the interval between submission and retrieval, the worker daemon resubmits it.

The work queue infrastructure is especially useful in a clustered environment as it enables parallel processing of addresses across the cluster members. In a clustered environment, the worker and writer daemon can run on any server in a cluster, not just the batch server.

For information on setting up the work queue daemons, see "Step 3: Configure the Work Queue" on page 97.

IMPORTANT When starting your application server for the first time, it might take a long time for the geocoding plugin to process new addresses. You can schedule geocoding to run when general access to your ClaimCenter server is restricted, as described at "Step 4: Schedule Geocoding" on page 98.

Synchronous and asynchronous processing work in combination to increase the performance of the application. ClaimCenter always synchronously converts a single new address needed for a search. However, Guidewire expects that as a result of asynchronous processing, the majority of existing addresses in the system already have geocode coordinates. Typically, a new proximity search requires at most a single conversion, and the existing addresses already have geocode coordinates in the database.

The Data Model and Geocoding

The following table shows the key objects used by the geocoding feature:

`Address`	An entity representing a street or mailing address. This entity has a number of fields that support geocoding: • The decimal fields `Latitude` and `Longitude`, which are degrees of latitude and longitude. • The decimal field `HTMID`, which stores a hierarchal triangular mesh identifier (`HTMID`), a single decimal value that represents the same location as that identified by the longitude and latitude. • The fields `FailureGeocoding`, `GeocodeStatus`, `Map`, `NeverGeocoded`, and `SuccessfullyGeocoded`.
`AddressCorrection`	An entity representing address data that, after geocoding, differs from the `Address` stored in the database. This entity contains a foreign key to the original address.
`GeocodeStatus`	A typelist that represents the possible status values returned by the geocoding process. You cannot extend this typelist. The status values are: • `city` — The geocoding service was able to locate only the city from the supplied address. • `exact` — The geocoding service was able to find an exact match for the supplied address. • `failure` — The geocoding service was unable to geocode the address. • `none` — No Address was submitted for geocoding. • `postalcode` — The geocoding service was able to locate only the postal code from the supplied address. • `street` — The geocoding service was able to locate only the street from the supplied address.

The entities and the typelist exist in both ClaimCenter and ContactCenter.

See also

- "The Mechanics of Geocoding" on page 92
- "Configuring Geocoding" on page 94
- "Proximity Search and Geocoding with Gosu" on page 99

The Mechanics of Geocoding

This topic covers some of the data model and processing behind the geocoding feature and describes how the system captures and stores geographic data.

This topic includes:

- Corrected Addresses
- Hierarchical Triangular Mesh ID (HTMID)
- Proximity Searches
- Search Pages

Corrected Addresses

The Bing Maps geocoding service can correct the original address it receives and return the address to the Guidewire application. For example, if ClaimCenter sends `123 Main`, the service might return `123 North Main Street`. ClaimCenter adds this corrected address to the `AddressCorrection` table, but otherwise it does nothing with it. The application does not automatically update user entered addresses with corrected addresses. They are available in the `AddressCorrection` table if you want to add functionality that makes use of these corrections.

Hierarchical Triangular Mesh ID (HTMID)

Guidewire applications use HTMID (Hierarchical Triangular Mesh ID) to compute proximity. HTM represents areas on the surface of the globe as a hierarchy of embedded triangles. Each triangle is called a *trixel* and has a unique identifying number, an HTMID. Each latitude and longitude corresponds to an HTMID. Every geocoded address in the database is accompanied by its HTMID.

A proximity search consists of synchronously geocoding the address at the search's center, and then using this result to search previously geocoded locations to create a list of closest locations. A proximity search uses the HTMID field in the database to perform fast searches. After a proximity search develops a list of destinations, a second call to the mapping service, using both the center and a destination, can deliver a map and driving directions.

Proximity Searches

Note: To try the example in this section, you must first configure geocoding as described at "Configuring Geocoding" on page 94. Additionally, the geocoding batch process must have run. If you are not on a production system, you could run the batch process from the command line as described at the end of that section.

You can request a distance-based proximity search or an order-based proximity search. A distance-based search finds, for example, the closest 10 body shops within 20 miles of San Francisco. An order-based proximity search finds, for example, the 10 closest addresses to the city San Mateo in California.

The system sorts the results of a proximity search, but does not display driving directions. To obtain driving directions, you must select a returned address and click the **Return Driving Directions** button. Once requested, additional columns for driving distances and times appear, as well as links to display the directions for the requested search results.

Note: You cannot sort these results.

Distanced-based Search

This example shows you how to request a distance-based proximity search from the ClaimCenter user interface. In this example, you request the closest auto repair shops within 25 miles of San Mateo, California, USA. It also discusses what the system does in response to the search request.

Note: To walk through this example in ClaimCenter, you must have done the following:

- Integrated ContactCenter with ClaimCenter.
- Set up geocoding for both applications.
- Run the `geocode` process for ClaimCenter and the `abgeocode` process for ContactCenter.
- Started up both applications.

For more information, see "Configuring Geocoding" on page 94.

1. Log in to ClaimCenter as a user who can search the Address Book. For example, enter user name `ssmith` and password `gw`.
2. Click the **Address Book** tab.
3. On the search screen, choose the following settings:

Field	Value
Type	`Auto Repair Shop`
Search Radius	`25 miles`
City (under Search Radius)	`San Mateo`
ZIP Code	`94404`

4. ClaimCenter determines the center of the proximity search by synchronously geocoding this one address.
5. The system calculates the HTMID range corresponding to the proximity search from the latitude and longitude of the center and the search parameters.
6. The system applies any filters that you set in the search. For example, if under **Location** you set the **City** to `Burlingame` and the **State** to `California`, the search eliminates every address that does not have a city set to `Burlingame`. The system also filters these values by their HTMID value. Any address without a latitude and longitude is not considered.
7. The system calculates the distance of the remaining results from the search center. If you limited the range as shown above, it discards any results that are too far away.
8. The system sorts the remaining results in ascending order of proximity, from closest to farthest away, and returns them in the table of **Search Results**.

Order-based Search

An order-based proximity search is typically significantly slower than a distance-based search. If you request an order-based proximity search, such as locating the 10 closest addresses to San Mateo, the proximity search first starts with a search radius of 1.4 kilometers. Then, it does the following:

1. The system performs the previous step 4 through step 8 of a distance-based search within the given radius.

2. If the system obtains fewer than 10 results it multiplies the radius by the square root of two and repeats the last step. This process continues until the system either obtains the number of results the user requested or it hits the maximum search distance as defined in your configuration (`ProximitySearchOrdinalMaxDistance`).
3. Finally, the system calculates the distance of the remaining results from the search center and returns the 10 closest results.

Search Pages

You can perform proximity searches in ClaimCenter or ContactCenter. Be default, the system limits these searches to particular types of objects as described in the following table:

Application	Searchable
ClaimCenter standalone	`User` objects
ContactCenter standalone	`ABContact` objects
ClaimCenter and ContactCenter integrated	`User` objects and `ABContact` objects

Proximity searches require that the object's related address has already been geocoded.

IMPORTANT The system never geocodes ClaimCenter claim-specific, local contacts. You can write integration code that enables you to Geocode these addresses. See "Geographic Data Integration", on page 315 in the *ClaimCenter Integration Guide* for more information.

The following user interface elements implement proximity search:

Address Book page	Click the **Address Book** tab in ClaimCenter or, in ContactCenter, use the search page to find a user. In ClaimCenter, open any page with a selectable contact (such as any claim page), then click **Edit** and then click the picker (down arrow) icon next to the contact drop-down.
Assignment search page	In ClaimCenter, click the checkbox on anything assignable on the desktop page and click the **Assign** button.
	In ClaimCenter, click the search icon 🔍 on the details page of an assignable item, such as the **New Activity** screen of a claim.
User search page	Log in as an administrator and on the **Administration** tab, select the **Search for User** menu item.

By default, all the search pages use the same search ranges. You can customize these ranges globally by editing the function `setDefaultSearchRange` in the Gosu class `gw.util.Geocode` in Studio.

IMPORTANT Large ranges and large values for the closest `n` contacts can have a negative performance impact. Large search ranges have the biggest impact.

See also

- "Overview of the Geocoding Feature" on page 89
- "Configuring Geocoding" on page 94
- "Proximity Search and Geocoding with Gosu" on page 99

Configuring Geocoding

Configuration involves activating the geocoding plugin, setting `config.xml` parameters, and enabling the work queue and scheduler to run batch geocoding of addresses.

To configure geocoding

- Step 1: Activate the Plugin
- Step 2: Configure Geocoding in config.xml
- Step 3: Configure the Work Queue
- Step 4: Schedule Geocoding
- Step 5: Log Geocode Information
- Step 6: Stop and Restart ClaimCenter and ContactCenter

IMPORTANT If your environment includes a ContactCenter integration, you must perform these steps in both ClaimCenter and ContactCenter to ensure proper functioning of the geocoding feature.

Step 1: Activate the Plugin

By default, the `GeocodePlugin` interface uses the Microsoft Bing Maps web service and is disabled. The `BingMapsPlugin` geocode plugin is written in Gosu. Its path is `gw.plugin.geocode.impl.BingMapsPlugin`.

Note: To use this plugin, your company must have its own account, login, and application key with Bing Maps. For more information, go to `http://www.bingmapsportal.com`, where you can set up a Bing Maps account and obtain an application key. There is also a video tutorial at that site describing how to get a key. When you create a key, the application name is arbitrary and no application URL is required.

To enable and use the plugin in a Guidewire application

1. Start ClaimCenter Studio.

   At a command prompt, navigate to `ClaimCenter\bin` and enter the following command:

   ```
   gwcc studio
   ```
2. From the **Resources** tree choose **Plugins** → **gw** → **plugin** → **geocode** → **GeocodePlugin**.
3. Click the **Enabled** check box to enable the plugin. If you see a message asking if you want to create a copy in the current module, click Yes.
4. Under **Parameters**, click the **Value** field for **applicationKey**, and then enter the application key you obtained from Bing Maps. For more information on getting a Bing Maps application key, see the note under “What is Geocoding?” on page 89.
5. Save your changes.

To enable and use the plugin in ContactCenter

1. Start ContactCenter Studio.

   At a command prompt, navigate to `ContactCenter\bin` and enter the following command:

   ```
   gwab studio
   ```
2. From the **Resources** tree choose **Plugins** → **gw** → **plugin** → **geocode** → **GeocodePlugin**.
3. Click the **Enabled** check box to enable the plugin. If you see a message asking if you want to create a copy in the current module, click Yes.
4. Under **Parameters**, click the **Value** field for **applicationKey**, and then enter the application key you obtained from Bing Maps. For more information on getting a Bing Maps application key, see the note under “What is Geocoding?” on page 89.
5. Save your changes.

Step 2: Configure Geocoding in config.xml

The `config.xml` file has parameters that you must set to enable geocoding features in the user interface. If you have ContactCenter integrated with ClaimCenter, change settings in the `config.xml` file in each application.

To open `config.xml`, if necessary, start ClaimCenter or ContactCenter Studio. Then go to the **Resources** tree on the left, expand **Other Resources**, and click **config.xml** to open the file in an editor.

The ClaimCenter `config.xml` file contains the following geocoding configuration parameters:

Parameter	Description
`UseGeocodingInPrimaryApp`	Set this parameter to `true` to enable geographical data and proximity search on application pages in `ClaimCenter`. The setting affects pages like Assignment and User search windows. This parameter must be `true` to perform assignment by proximity. The default setting is `false`.
`UseGeocodingInAddressBook`	Set this parameter to `true` to enable geocoding on ClaimCenter **Address Book** pages if ClaimCenter is integrated with ContactCenter. The default setting is `false`.
`UseMetricDistancesByDefault`	Use kilometers in the user interface. The default setting, `false`, is to use miles. The setting for this parameter must be the same in ContactCenter and ClaimCenter.
`ProximitySearchOrdinalMaxDistance`	Maximum distance to use when performing an ordinal (nearest `N`) proximity search. The default is `300`. The actual unit value of the distance is miles or kilometers depending on how you have set `UseMetricDistancesByDefault`. This parameter has no effect on radius (`n`-miles) searches or proximity assignment based on walking the group tree.
`ProximityRadiusSearchDefaultMaxResultCount`	Maximum number of results to return when performing a radius (`n` miles or kilometers) search from ClaimCenter. The default is `1000`. This parameter has no effect on ordinal (nearest `n`) proximity searches. This parameter does not have to match the value of the corresponding parameter in the ContactCenter `config.xml` file.

The ContactCenter `config.xml` file contains the following geocoding configuration parameters:

Parameter	Description
`UseGeocodingInAddressBook`	Set this parameter to `true` to enable geocoding for ContactCenter **Address Book** entries. The default is `false`. The setting for this parameter must be the same in ContactCenter and ClaimCenter.
`UseMetricDistancesByDefault`	Use kilometers in the user interface. The default, `false`, is to use miles. The setting for this parameter must be the same in ContactCenter and ClaimCenter.
`ProximitySearchOrdinalMaxDistance`	Maximum distance to use when performing an ordinal (nearest-n) proximity search. The default is `300`. The actual unit value of the distance is miles or kilometers depending on how you have set `UseMetricDistancesByDefault`. This parameter has no effect on radius (n-miles) searches or proximity assignment based on walking the group tree. The setting for this parameter must be the same in ContactCenter and ClaimCenter.
`ProximityRadiusSearchDefaultMaxResultCount`	Maximum number of results to return when performing a radius (`n` miles or kilometers) search from the ContactCenter search screen. The default is `1000`. This parameter has no effect on ordinal (nearest `n`) proximity searches. This parameter does not have to match the value of the corresponding parameter in the ClaimCenter `config.xml` file.

IMPORTANT If you have integrated ClaimCenter and ContactCenter, ensure that `UseGeocodingInAddressBook`, `UseMetricDistancesByDefault`, and `ProximitySearchOrdinalMaxDistance` have the same values in both ClaimCenter and ContactCenter.

Step 3: Configure the Work Queue

As described at “Asynchronous Processing” on page 91, the geocoding feature uses the Guidewire work queue infrastructure for asynchronous searches. For example, the ClaimCenter `geocode` process searches asynchronously for new addresses at the times specified in the `scheduler-config.xml` file, as described at “Step 4: Schedule Geocoding” on page 98. The `geocode` process geocodes all addresses in the database that have not yet been geocoded. ContactCenter has an `abgeocode` process that does the same thing.

The ClaimCenter `work-queue.xml` file configures one `geocode` worker to geocode addresses.

```
<work-queue
    workQueueClass="com.guidewire.pl.domain.geodata.geocode.GeocodeWorkQueue"
    progressinterval="600000">
  <worker instances="1" batchsize="100"/>
</work-queue>
```

The ContactCenter `work-queue.xml` file configures one `abgeocode` worker to geocode addresses.

```
<work-queue
    progressinterval="600000"
    workQueueClass="com.guidewire.ab.domain.geodata.geocode.ABGeocodeWorkQueue">
  <worker batchsize="100" instances="1"/>
</work-queue>
```

You can modify these configurations, but ensure that you have a good understanding of the work queue infrastructure first. For example, one area to understand is how `progressinterval` is used with `batchsize` by the system. The default setting of `progressinterval` is 600,000 milliseconds, or 10 minutes. This setting is the amount of time that ClaimCenter or ContactCenter allots for a worker to process `batchsize` geocode work items. If the time a worker has held a batch of items exceeds the `progressinterval`, the work items become *orphans*. ClaimCenter reassigns orphan work items to a new worker instance. Therefore, you need to set the `progressinterval` larger than the longest time required to process a work item, multiplied by the `batchsize`.

For more information on work queues, see:

- "Understanding Distributed Work Queues" on page 130 in the *System Administration Guide*
- "Configuring Distributed Work Queues" on page 133 in the *System Administration Guide*

If you have many new `User` or `Contact` objects in ClaimCenter or many new `ABContact` objects in ContactCenter, processing these objects can be a system intensive operation. In this case, run the geocode batch process when you anticipate that system use will be low, such as late at night. For information on scheduling the process, see "Step 4: Schedule Geocoding" on page 98.

Running the Geocoding Batch Process Manually

You can run the geocoding process manually as a system administrator on the **Server Tools** screen.

Running the Process Manually in ClaimCenter

1. Log in as an administrator, such as username `su` and password `gw`.
2. Press **ALT+SHIFT+T** to access the **Server Tools** screen.

   The screen opens with the **Server Tools** tab selected and the **Batch Process Info** item under **Actions** selected.
3. Locate the batch process **Geocode Writer** and click **Run** in the column on the right.

Running the Process Manually in ContactCenter

1. Log in as an administrator, such as username `su` and password `gw`.
2. Press **ALT+SHIFT+T** to access the **Server Tools**.
3. In the menu on the left under **Actions**, click **Batch Process Info**.
4. Locate the batch process **AB Geocode Writer** and click **Run** in the column on the right.

Step 4: Schedule Geocoding

If you enable the Geocode plugin in ClaimCenter, you must also uncomment the code that schedules the geocoding batch job in the `scheduler-config.xml` file:

```
<!--  New addresses searched for geocoding at 1:30 am  -->
<!--   <ProcessSchedule process="Geocode">
         <CronSchedule hours="1" minutes="30"/>
       </ProcessSchedule> -->
```

Note: Open `scheduler-config.xml` from Studio by going to the **Resources** tree, expanding **Other Resources**, and clicking **scheduler-config.xml**.

This line instructs the batch process to start searching for new addresses to geocode every night at 1:30 AM.

If ClaimCenter is integrated with ContactCenter, you must also uncomment the entry in the ContactCenter `scheduler-config.xml` file for `ABGeocode`:

```
<!--   <ProcessSchedule process="ABGeocode">
         <CronSchedule minutes="0"/>
       </ProcessSchedule> -->
```

This line instructs the batch process to start searching for new addresses to geocode every hour on the hour.

If you are adding many new contacts, especially into ContactCenter, tune this parameter to match your expected daily load of new addresses. For more information on scheduling batch processes see "Defining a Schedule Specification" on page 144 in the *System Administration Guide*.

IMPORTANT Geocoding a large number of addresses can require considerable application resources. Set up your implementation to do its batch geocoding of addresses at a time where general access to your server is restricted.

Step 5: Log Geocode Information

The `logging.properties` file controls logging for the application. This file is located at `modules\configuration\config\logging\logging.properties` in your ClaimCenter and ContactCenter installations.

The file includes plugin sections that you can configure to ensure that the log contains details of all plugin activity, including geocoding. See “Logging Overview” on page 35 in the *System Administration Guide* for details on logging configuration.

Step 6: Stop and Restart ClaimCenter and ContactCenter

After making all the configuration and logging file changes, you must stop the Guidewire applications that you configured if they are running, and then rebuild and redeploy them. Following are the steps for stopping and starting both ClaimCenter and ContactCenter when they are running in development mode:

1. Stop ClaimCenter.

   At the command prompt in which ClaimCenter is running, press **CTRL+C** to stop the batch process, and then enter the following commands:

   ```
   y
   gwcc dev-stop
   ```

2. Stop ContactCenter.

   At the command prompt in which ContactCenter is running, press **CTRL+C** to stop the batch process, and then enter the following commands:

   ```
   y
   gwab dev-stop
   ```

3. Start the ContactCenter application:

   ```
   gwab dev-start
   ```

4. Start the ClaimCenter application:

   ```
   gwcc dev-start
   ```

See also

- “Overview of the Geocoding Feature” on page 89
- “The Mechanics of Geocoding” on page 92
- “Proximity Search and Geocoding with Gosu” on page 99

Proximity Search and Geocoding with Gosu

Guidewire provides a number of Gosu methods that you can use to perform proximity searches either on contacts in ContactCenter or on users in the local application. Additionally, there are Gosu methods that you can use to geocode an address or to check to see if an address has already been geocoded.

- ClaimCenter provides Gosu support for proximity searches primarily through the following class:

  ```
  gw.api.geocode.CCGeocodeScriptHelper
  ```

- ContactCenter provides Gosu support for proximity searches primarily through the following entity method:

  ```
  entity.ABContactSearchCriteria.performSearch
  ```

  For more information, see “ContactCenter Proximity Search Gosu Example” on page 100.

- Both ClaimCenter and ContactCenter provide Gosu support for geocoding through the following class:

  ```
  entity.Address
  ```

ClaimCenter also provides Gosu methods like `assignUserByProximityWithAssignmentSearchCriteria` that are useful in assignment rules that do proximity-based assignment.

- For a description of these Gosu methods, see “Proximity-Based Assignment” on page 113 in the *Rules Guide*.

- For general information about assignment rules, see "Assignment in ClaimCenter" on page 97 in the *Rules Guide*.

ContactCenter Proximity Search Gosu Example

ContactCenter provides the method `gw.api.database.performSearch` to support database search with a number of entities. You can use `ABContactSearchCriteria.performSearch` to customize how search works with the Geocoding feature.

To use this method, you declare an object of type `ABContactSearchCriteria`, define proximity search parameters with `ABContactSearchCriteria.ProximitySearchParameters`, and call `ABContactSearchCriteria.performSearch(true)`. The parameter for `performSearch` is `isClearBundle:boolean`. You set it to `true` to clear the `ABContactSearchCriteria` bundle, which ensures that only the criteria you specify are left.

For example, the following method takes a contact subtype, a latitude, and a longitude, and returns an array of `ABContact` objects:

```
uses java.math.BigDecimal

...

public function getClosestVendorForLatLong(
    ContactSubtype: String,
    Latitude : BigDecimal,
    Longitude : BigDecimal) : ABContact[] {
  var myLatitude = Latitude.setScale(3,2)
  var myLongitude = Longitude.setScale(3,2)
  var mySearchCriteria = new ABContactSearchCriteria()
  mySearchCriteria.ProximitySearchParameters.DistanceBasedSearch = false
  mySearchCriteria.ProximitySearchParameters.Number = 5
  mySearchCriteria.ProximitySearchParameters.setGeocodeFieldsFromLatLong(
    myLatitude, myLongitude)
  mySearchCriteria.ContactSubtype = ContactSubtype
  var results =
    mySearchCriteria.performSearch(true).toTypedArray() as ABContact[]
  return results
}
```

You could expose this method as a web service and call it from an external application.

For more information on methods and properties of `ProximitySearchParameters` and `ABContactSearchCriteria`, see the **Gosu API Reference**, available from the **Help** menu in Guidewire Studio.

See also

- "Overview of the Geocoding Feature" on page 89
- "The Mechanics of Geocoding" on page 92
- "Configuring Geocoding" on page 94
- "Proximity-Based Assignment" on page 113 in the *Rules Guide*

chapter 10

Securing Contact Information

This topic describes how to grant secure access to the information associated with contacts.

This topic includes:

- "Overview of Contact Security" on page 101
- "Configuring Contact Permissions" on page 104

Overview of Contact Security

To get the most out of this discussion of contact security, you need to understand permissions, access control lists, and configuration values. See the following references;

- "Security: Roles, Permissions, and Access Controls" on page 379 in the *Application Guide*.
- "Securing Access to ClaimCenter Objects" on page 99 in the *System Administration Guide*

Contact Permissions

ClaimCenter and ContactCenter provide permissions that you can use to control access to contacts. These permissions make a distinction between local contacts and centralized, or address book, contacts. The `SystemPermissionType` typelist lists all the permissions in your system.

The following table lists the base permissions provided with ClaimCenter and ContactCenter contacts:

Code	Enables a User to
`abcreate`	Create a new contact in the address book.
`abcreatepref`	Create a new preferred vendor in the address book.
`abdelete`	Delete an existing contact from the address book.
`abdeletepref`	Delete an existing preferred vendor address book entry.
`abedit`	Edit an existing contact in the address book.
`abeditpref`	Edit an existing preferred vendor in the address book.

Code	Enables a User to
`abview`	View the details of contact entries in the address book.
`abviewsearch`	Search contact entries in the address book.
`ctccreate`	Create a new local contact. (ClaimCenter only)
`ctcedit`	Edit an existing local contact. (ClaimCenter only)
`ctcview`	View and search local contact entries. (ClaimCenter only)
`reviewcreate`	Create a new review on the **Reviews** tab. (ClaimCenter only)
`reviewdeletecompleted`	Delete a completed review on the **Reviews** tab. (ClaimCenter only)
`reviewdeleteincomplete`	Delete an incomplete review on the **Reviews** tab. (ClaimCenter only)
`reviewedit`	Edit the **Reviews** tab where the scores are entered for each claim. (ClaimCenter only)
`reviewviewdetail`	View the **Reviews** tab to see the answers entered for each claim. (ClaimCenter only)
`reviewviewlist`	See the **Reviews** tab on a contact and view the list of reviews. (ClaimCenter only)
`revsumviewdetail`	View the **Review Summary** page to see the category scores for each summarized review. (ContactCenter only)
`revsumviewlist`	View list of review summaries on the **Review Summary** page and view the reviews tab for an `ABContact`. (ContactCenter only)

The system uses role-based security for these permissions. To implement role-based security, a system administrator associates permissions with roles in the system and assigns roles to users. For each role assigned, the user acquires the permissions associated with that role. For example, a role associated with the `abcreate` permission enables the user who has this role to create any type of contact in the address book.

The base contact permissions apply across all contact subtypes. If you want to restrict permissions according to contact subtype, you can do so by redefining the ClaimCenter contact permissions. For example, you can enable a user with `abcreate` permission to create only `PersonVendor` contacts, but not `CompanyVendor` contacts. You configure contact permissions through `SystemPermissionType` typelist and the `security-config.xml` resource.

For an example, see “Configuring Contact Permissions” on page 104.

config.xml Parameters

There are two parameters you can set in the `config.xml` file to restrict how searching works with the **Address Book**, `RestrictSearchesToPermittedItems` and `RestrictContactPotentialMatchToPermittedItems`.

To edit config.xml

1. Start ClaimCenter Studio.

   At a command prompt, navigate to `ClaimCenter\bin` and enter the following command:
   ```
   gwcc studio
   ```
2. In the **Resources** tree, click **Other Resources** → **config.xml**.

You can use the `RestrictSearchesToPermittedItems` configuration parameter to control the interaction between the `abviewsearch` permission and the `abview` permission. The `abviewsearch` permission determines which users can use the Address Book search function. However, not all users with `abviewsearch` permission also have `abview` permission. The `abview` permission enables users to view the contact’s detailed information.

If `RestrictSearchesToPermittedItems` is `false`, in response to a search the system returns all contacts that match the search criteria. If this parameter is `true`, the system returns only contacts for which the user has view permissions. This setting also interacts with permission settings. For example, if a user can view the `Person` subtype and `RestrictSearchesToPermittedItems` is `true`, the system returns only contacts of the `Person` subtype.

Additionally, you can set the `RestrictContactPotentialMatchToPermittedItems` parameter. This parameter controls the security of potential search results. If the parameter is `true`, only the potential matches for which the user has view permissions are returned.

Permission Check Expressions

In a page configuration file (PCF), you can control permissions on specific widgets with Gosu expressions that determine if a user has the permission to perform an operation. For example, the setting for the `NewAddressBookContact` page's `canVisit` attribute is `perm.Contact.createab(contactType)`. This setting limits users of this page to creating only contact types for which they have the `abcreate` permission.

> **Note:** To see this file, in ClaimCenter Studio, type **CTRL+N** and enter `NewAddressBookContact`, and then click **NewAddressBookContact (pcf.NewAddressBookContact)** in the list of objects that the system finds. In the **Resource** pane, this file is in **Page Configuration (PCF)** → **addressbook**.

Some Gosu permission check expressions require an input parameter, and some do not. The following table lists the `Contact` permission check expressions:

Expression	Description
`perm.Contact.createab`	Takes a `Contact` type as an input parameter and checks that the user has the `abcreate` permission to create a new contact in the address book.
`perm.Contact.deleteab`	Takes a Contact instance as an input parameter and checks that the user has `abdelete` permission to delete a contact from the address book.
`perm.Contact.editab`	Requires a `Contact` instance as an input parameter and checks that the user has the `abedit` permission to edit a contact in the address book.
`perm.Contact.viewab`	Takes a `Contact` instance as an input parameter and checks that the user has the `abview` permission to view a contact in the address book.
`perm.Contact.viewsearchab`	Optionally takes a `Contact` instance as an input parameter and checks that the user has the `abviewsearch` permission to search for a contact in the address book.
`perm.Contact.createpreferredab`	Optionally takes a `Contact` type as an input parameter and checks that the user has the `abcreatepref` permission to add a preferred contact in the address book.
`perm.Contact.deletpreferredab`	Takes a Contact instance as an input parameter and checks that the user has `abdeletepref` permission to delete a preferred vendor from the address book.
`perm.Contact.editpreferredab`	Requires a `Contact` instance as an input parameter and checks that the user has the `abeditpref` permission to edit a preferred vendor in the address book.
`perm.Contact.createlocal`	Optionally takes a `Contact` type as an input parameter and checks that the user has the `ctccreate` permission to create a local contact.
`perm.Contact.editlocal`	Requires a `Contact` instance as an input parameter and checks that the user has the `useredit` permission to edit an existing user, except for roles, authority limits, or attributes.
`perm.Contact.viewlocal`	Optionally takes a `Contact` instance as an input parameter and checks that the user has the `ctcview` permission to view and search local contact entries.

You can use the ClaimCenter Security Dictionary to get information on the application permission keys. For example, the Security Dictionary entry in the left list pane for the Gosu expression `perm.Contact.editab` is the application permission key **Contact editab**. You can filter the list by application permission keys, pages, system permissions, and roles.

To build and view the Security Dictionary

1. At a command prompt, run the following command from `ClaimCenter\bin`:

   ```
   gwcc regen-dictionary
   ```

   This command builds the dictionary in the following location

   ```
   ClaimCenter\build\dictionary\security\index.html
   ```

2. Navigate in a web browser to the location of the dictionary. For example:

   ```
   file:///C:/ClaimCenter/build/dictionary/security/index.html
   ```

By using the dictionary, you can determine the following

- The system permission related to an application permission key
- The PCF files and widgets that use an application permission key
- The roles, application permission keys, PCF pages, and widgets that use a system permission
- A list of the Gosu application permission expressions called from each PCF page
- A list of the permissions assigned to each role

See also

- "Configuring Contact Permissions" on page 104

Configuring Contact Permissions

Use the **SystemPermissionType** typelist and the **security-config.xml** resource to define new ClaimCenter permissions and create more finely grained system permissions for specific subtypes.

> **Note:** You can redefine permissions in ContactCenter as well. The typelist and configuration file have the same names as in ClaimCenter. You can use the same entries as in the example that follows, except that you need to map the typecodes to the `ABPerson` subtype in ContactCenter.

To create Contact Permissions in ClaimCenter

1. If necessary, start ClaimCenter Studio.

   At a command prompt, navigate to `ClaimCenter/bin` and enter the following command:

   ```
   gwcc studio
   ```

2. In the **Resources** tree on the left, choose **Typelists** → **SystemPermissionType** to open this typelist in an editor.
3. Add one or more new typecodes to **SystemPermissionType**.

   For example, for each of the following contact permission typecodes, click **Add** and enter the information for the new typecode:

Code	Name	Description
personcreate	Create person	Permission to create a person subtype
personview	View person	Permission to view a person subtype
personedit	Edit person	Permission to edit a person subtype
persondelete	Delete person	Permission to delete a person subtype

4. In the **Resources** tree on the left, click **Other Resources** → **security-config.xml**.
5. Associate the new permissions with a `Contact` subtype in the `security-config.xml` file.

   For example, map the new typecodes to the `Person` contact subtype as follows:

   ```
   <ContactPermissions>
     <ContactSubtypeAccessProfile entity="Person">
       <ContactCreatePermission permission="personcreate"/>
   ```

```
    <ContactViewPermission permission="personview"/>
    <ContactEditPermission permission="personedit"/>
    <ContactDeletePermission permission="persondelete"/>
  </ContactSubtypeAccessProfile>
</ContactPermissions>
```

6. Stop the ClaimCenter server, regenerate the data and security dictionaries, and then restart ClaimCenter, as follows:
   a. If ClaimCenter is running, in the command window in which ClaimCenter is running, press **CTRL+C** to stop the batch process, and then enter the following commands:
      ```
      y
      gwcc dev-stop
      ```
   b. Optionally, at a command prompt navigate to `ClaimCenter\bin` and regenerate the data and security dictionaries:
      ```
      gwcc regen-dictionary
      ```
   c. Restart ClaimCenter:
      ```
      gwcc dev-start
      ```

   Note: Regenerating the security dictionary is a good way to catch mistakes you might have made in defining the new permissions.

7. Add one or more new permissions to a user role. For example:
   a. Log in to ClaimCenter as a user that has the User Admin role, such as user name `admin` with password `gw`.
   b. Click the **Administration** tab.
   c. Click **Roles** on the left and choose a role, such as **Claims Supervisor.**
   d. On the **Basics** tab, click **Edit**, and then click **Add** to add a new role.
   e. Click the drop down arrow in the new field and choose the **Create person** permission.
   f. Click **Update** to add the new permission to the role.

8. Assign the role to one or more users. For example:
   a. Click **Roles** on the left, and then click **Claims Supervisor.**
   b. Click the **Users** tab and click **Edit**, and then click **Add**.
   c. On the **Search Users** page, enter `A` in the **Last Name** field and click **Search.**
   d. If you have the sample data loaded, `Andy Applegate` is one of the **Search Results**. Click the checkbox next to this name and click Select.
   e. Click **Update** to add `Andy Applegate` to the **Claims Supervisor** role.

      Andy Applegate now has permission to create a new Person in the address book as a result of having the Claims Supervisor role.

See also

- "Overview of Contact Security" on page 101
- "Security: Roles, Permissions, and Access Controls" on page 379 in the *Application Guide*.
- "Securing Access to ClaimCenter Objects" on page 99 in the *System Administration Guide*