

# Guidewire ClaimCenter®

## Guidewire Contact Management Guide

ClaimCenter RELEASE 8.0.2

**Guidewire®**  
Deliver insurance your way™

Copyright © 2001-2014 Guidewire Software, Inc. All rights reserved.

Guidewire, Guidewire Software, Guidewire ClaimCenter, Guidewire PolicyCenter, Guidewire BillingCenter, Guidewire Reinsurance Management, Guidewire ContactManager, Guidewire Vendor Data Management, Guidewire Client Data Management, Guidewire Rating Management, Guidewire InsuranceSuite, Guidewire ContactCenter, Guidewire Studio, Guidewire Product Designer, Guidewire Live, Guidewire DataHub, Guidewire InfoCenter, Guidewire Standard Reporting, Guidewire ExampleCenter, Guidewire Account Manager Portal, Guidewire Claim Portal, Guidewire Policyholder Portal, ClaimCenter, BillingCenter, PolicyCenter, InsuranceSuite, Gosu, Deliver Insurance Your Way, and the Guidewire logo are trademarks, service marks, or registered trademarks of Guidewire Software, Inc. in the United States and/or other countries.

All other trademarks are the property of their respective owners.

**This material is confidential and proprietary to Guidewire and subject to the confidentiality terms in the applicable license agreement and/or separate nondisclosure agreement.**

Guidewire products are protected by one or more United States patents.

Product Name: Guidewire ContactManager

Product Release: 8.0.2

Document Name: Guidewire Contact Management Guide

Document Revision: 20-May-2014

# Contents

<b>About ClaimCenter Documentation .....</b>	<b>5</b>
Conventions in This Document .....	6
Support .....	6
<b>1 New and Changed Features in ContactManager .....</b>	<b>7</b>
New And Changed Features in ContactManager 8.0.2 .....	7
New And Changed Features in ContactManager 8.0.1 .....	8
New And Changed Features in ContactManager 8.0.0 .....	11
New and Changed Features in ContactManager 7.0 .....	20
<b>2 Managing Integrated Contacts .....</b>	<b>33</b>
Client Data Management .....	33
Vendor Data Management .....	35
<b>3 Installing ContactManager .....</b>	<b>39</b>
Installing ContactManager with QuickStart for Development .....	39
Installing ContactManager with Tomcat and SQL Server for Development .....	41
<b>4 Integrating ContactManager with Guidewire Core Applications .....</b>	<b>45</b>
Integrating ContactManager Using QuickStart .....	46
Integrating ContactManager Using Tomcat and SQL Server .....	65
Configuring Core Application Authentication with ContactManager .....	69
Configuring ContactManager Authentication with Core Applications .....	73
<b>5 Searching for Contacts .....</b>	<b>83</b>
Overview of Contact Search .....	83
ContactManager Support for Contact Searches .....	84
ClaimCenter Support for Contact Searches .....	87
PolicyCenter Support for Contact Searches .....	106
BillingCenter Support for Contact Searches .....	107
<b>6 Securing Access to Contact Information .....</b>	<b>109</b>
Overview of Contact Security .....	109
ContactManager Contact Security .....	110
PolicyCenter Contact Security .....	120
BillingCenter Contact Security .....	120
ClaimCenter Contact Security .....	122
<b>7 Extending the Contact Data Model .....</b>	<b>133</b>
Overview of Contact Entities .....	133
Extending the Client Data Model .....	141
Extending the Vendor Contact Data Model .....	142
Changing the Subtype of a Contact Instance .....	172
<b>8 Contact Tags .....</b>	<b>177</b>
Contact Tag Overview .....	177
Contact Tag-based Security .....	178
<b>9 Vendor Services .....</b>	<b>181</b>
Vendor Services Overview .....	181
Working with Vendor Services in ContactManager .....	182
Vendor Services Data Model .....	184

<b>10 Linking and Synchronizing Contacts .....</b>	<b>191</b>
Linking a Contact .....	191
Synchronizing with ContactManager .....	196
<b>11 ContactManager Rules .....</b>	<b>207</b>
ContactManager Rule Sets .....	207
About Preupdate and Validation .....	212
Overview of ContactManager Validation .....	212
The Validation Graph in Guidewire ContactManager .....	213
Top-level ContactManager Entities That Trigger Validation .....	213
<b>12 ClaimCenter Service Provider Performance Reviews .....</b>	<b>215</b>
Service Provider Performance Reviews .....	216
Using Service Provider Performance Reviews .....	217
Data Model for Service Provider Performance Reviews .....	222
Configuring Service Provider Performance Reviews .....	224
Service Provider Review Plugin Reference .....	235
<b>13 Working Directly in ContactManager .....</b>	<b>237</b>
Logging in to ContactManager .....	238
Changing Your Password in ContactManager .....	239
Changing Your User Preferences in ContactManager .....	239
Selecting International Settings in ContactManager .....	239
ContactManager User Interface .....	240
Importing and Exporting Administrative Data .....	243
Managing Contact Data .....	245
Vendor Services Onboarding .....	254
<b>14 ContactManager Integration Reference .....</b>	<b>269</b>
ContactManager Integration Overview .....	269
ContactManager Entities .....	270
ContactManager Web Services .....	272
ContactManager Messaging Events by Entity .....	280
ABCClientAPI Interface .....	282
ContactMapper Class .....	285
ContactManager Plugins .....	293
IFindDuplicatesPlugin Plugin Interface .....	298
ValidateABCContactCreationPlugin Plugin Interface .....	306
Testing Clock Plugin Interface—Only For Non-Production Servers .....	308
<b>15 Release Notes Archive .....</b>	<b>311</b>
Guidewire ContactManager 7.0.1 Release Notes .....	311
Guidewire ContactManager 7.0.2 Release Notes .....	326
Guidewire ContactManager 7.0.3 Release Notes .....	337
Guidewire ContactManager 7.0.4 Release Notes .....	344
Guidewire ContactManager 7.0.5 Release Notes .....	351
Guidewire ContactManager 7.0.6 Release Notes .....	358
Guidewire ContactManager 8.0.0 Release Notes .....	363
Guidewire ContactManager 8.0.1 Release Notes .....	369

# About ClaimCenter Documentation

The following table lists the documents in ClaimCenter documentation.

Document	Purpose
<i>InsuranceSuite Guide</i>	If you are new to Guidewire InsuranceSuite applications, read the <i>InsuranceSuite Guide</i> for information on the architecture of Guidewire InsuranceSuite and application integrations. The intended readers are everyone who works with Guidewire applications.
<i>Application Guide</i>	If you are new to ClaimCenter or want to understand a feature, read the <i>Application Guide</i> . This guide describes features from a business perspective and provides links to other books as needed. The intended readers are everyone who works with ClaimCenter.
<i>Upgrade Guide</i>	Describes how to upgrade ClaimCenter from a previous major version. The intended readers are system administrators and implementation engineers who must merge base application changes into existing ClaimCenter application extensions and integrations.
<i>New and Changed Guide</i>	Describes new features and changes from prior ClaimCenter versions. Intended readers are business users and system administrators who want an overview of new features and changes to features. Consult the "Release Notes Archive" part of this document for changes in prior maintenance releases.
<i>Installation Guide</i>	Describes how to install ClaimCenter. The intended readers are everyone who installs the application for development or for production.
<i>System Administration Guide</i>	Describes how to manage a ClaimCenter system. The intended readers are system administrators responsible for managing security, backups, logging, importing user data, or application monitoring.
<i>Configuration Guide</i>	The primary reference for configuring initial implementation, data model extensions, and user interface (PCF) files. The intended readers are all IT staff and configuration engineers.
<i>Globalization Guide</i>	Describes how to configure ClaimCenter for a global environment. Covers globalization topics such as global locales, languages, date and number formats, names, currencies, addresses, and phone numbers. The intended readers are configuration engineers who work with locales and languages.
<i>Rules Guide</i>	Describes business rule methodology and the rule sets in ClaimCenter Studio. The intended readers are business analysts who define business processes, as well as programmers who write business rules in Gosu.
<i>Contact Management Guide</i>	Describes how to configure Guidewire InsuranceSuite applications to integrate with ContactManager and how to manage client and vendor contacts in a single system of record. The intended readers are ClaimCenter implementation engineers and ContactManager administrators.
<i>Best Practices Guide</i>	A reference of recommended design patterns for data model extensions, user interface, business rules, and Gosu programming. The intended readers are configuration engineers.
<i>Integration Guide</i>	Describes the integration architecture, concepts, and procedures for integrating ClaimCenter with external systems and extending application behavior with custom programming code. The intended readers are system architects and the integration programmers who write web services code or plugin code in Gosu or Java.
<i>Gosu Reference Guide</i>	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
<i>Glossary</i>	Defines industry terminology and technical terms in Guidewire documentation. The intended readers are everyone who works with Guidewire applications.

## Conventions in This Document

Text style	Meaning	Examples
<i>italic</i>	Emphasis, special terminology, or a book title.	A <i>destination</i> sends messages to an external system.
<b>bold</b>	Strong emphasis within standard text or table text.	You <b>must</b> define this property.
<b>narrow bold</b>	The name of a user interface element, such as a button name, a menu item name, or a tab name.	Next, click <b>Submit</b> .
<code>monospaced</code>	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. In code blocks, bold formatting highlights relevant sections to notice or to configure.	Get the field from the <code>Address</code> object.
<code>monospaced italic</code>	Parameter names or other variable placeholder text within URLs or other code snippets.	Use <code>getName(first, last)</code> . <code>http://SERVERNAME/a.html</code> .

## Support

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

- At the Guidewire Resource Portal – <http://guidewire.custhelp.com>
- By email – [support@guidewire.com](mailto:support@guidewire.com)
- By phone – +1-650-356-4955

# New and Changed Features in ContactManager

This topic presents information on what is new and changed in ContactManager.

This topic is intended for system administrators and business users who want to upgrade to ContactManager 8.0 and need an understanding of the new features and changes in the releases. The assumption is that you are familiar with using previous releases of ContactManager.

This topic includes:

- “New And Changed Features in ContactManager 8.0.2” on page 7
- “New And Changed Features in ContactManager 8.0.1” on page 8
- “New And Changed Features in ContactManager 8.0.0” on page 11
- “New and Changed Features in ContactManager 7.0” on page 20

## New And Changed Features in ContactManager 8.0.2

### New Configuration Parameter to Limit Number of Services Specified in a Search

There is a new ContactManager configuration parameter that you can set in `config.xml`, `MaxNumberServicesInSearchQuery`. This configuration parameter, which defaults to a value of 20, limits the number of services that can be specified in a single contact search. See “Limiting the Number of Service Elements Specified in a Contact Search” on page 87.

### Change to Order of Proximity Based Search Results

Proximity search in ContactManager has been changed for webservice searches to always return the search results ordered by distance, smallest to largest. Any other type of sorting is discarded for proximity searches. See “Geocoding and Proximity Searches for Vendor Contacts” on page 99.

## Vendor Services Onboarding

In ContactManager, after you create your set of vendor services, you can add services to each contact by editing the contact. However, if you have a large number of contacts that need services added to them, adding services to each contact, one at a time, might not be practical. You can use vendor services onboarding to process all your vendor contacts and add services to them.

See “Vendor Services Onboarding” on page 254.

## New And Changed Features in ContactManager 8.0.1

This topic includes:

- “Core Applications Can Specify a Unique ID when Creating a Contact” on page 8
- “Changes to Contact Linking and Synchronizing APIs in ClaimCenter” on page 8
- “Approving and Editing a Pending Update or Create” on page 9
- “Finding Duplicate Contacts for a Pending Create” on page 9
- “Merging and Editing Potential Duplicate Contacts” on page 9
- “Changing the Subtype of a Contact Instance” on page 9
- “Using an Address Field to Search for Contacts” on page 10
- “Moved Changed ab800 Web Services, Helper Classes, and WSDL files to ab801” on page 10
- “Changes to Core Application Integration with ContactManager” on page 10

### Core Applications Can Specify a Unique ID when Creating a Contact

Previous to this release, ContactManager was the only application that could create unique IDs for new contacts that it created. Now it is possible for a core application to send ContactManager a unique ID with a `createContact` request. If ContactManager detects that an external unique ID is specified, it uses that ID as the `LinkID` for the new contact. Otherwise, ContactManager creates a `LinkID` for the new contact, as it did previously.

If ContactManager detects that the unique ID specified by a core application is already in use, ContactManager throws an exception and does not create the contact. The calling application must recover from this state, either by providing a new unique ID or allowing ContactManager to create a unique ID.

PolicyCenter is the only core application that implements this functionality in its base configuration. PolicyCenter sends new contacts both to ContactManager and to BillingCenter with messages that are asynchronous. To ensure that both BillingCenter and ContactManager reference the same contact, PolicyCenter must specify the unique ID itself, rather than letting ContactManager create one.

#### See also

- “Creating and Linking a Contact” on page 192
- “ContactManager Link IDs and Comparison to Other IDs” on page 271

### Changes to Contact Linking and Synchronizing APIs in ClaimCenter

Significant changes to contact APIs resulted from the change in contact system plugin interfaces from `IAddressBookAdapter` to `ContactSystemPlugin`. See “ClaimCenter File Name, Method, and Package Changes for ContactManager Integration” on page 15. Included in these changes were deprecation of the methods `Contact.sync`, `Contact.linked`, and `Contact.synced`.

Instead of calling these methods on the contact entity itself, use the following methods from `gw.api.contact.ContactSystemUtil`:

- `ContactSystemUtil.INSTANCE.generateLinkStatus(aContact)`
- `ContactSystemUtil.INSTANCE.syncToContactSystem(aContact)`
- `ContactSystemUtil.INSTANCE.contactIsLinked (aContact)` – See following note about improving performance.
- `ContactSystemUtil.INSTANCE.contactIsSynced(aContact)` – See following note about improving performance.

#### Note

For better performance, first call `ContactSystemUtil.INSTANCE.generateLinkStatus(aContact)` and then query the returned `ContactSystemLinkStatus` object for the status you are interested in, such as linked or synced. For example:

```
var linkStatus = ContactSystemUtil.INSTANCE.generateLinkStatus(theContact)
if (!linkStatus.isLinked() && linkStatus.isSynced()) {...
```

It is more expensive to call `ContactSystemUtil.INSTANCE.contactIsLinked(aContact)` and then call `ContactSystemsUtil.INSTANCE.contactIsSynced(aContact)`. Calling those methods on `aContact` results in two round trips between ClaimCenter and ContactManager.

## Approving and Editing a Pending Update or Create

There is a new button on the screens for pending updates and pending creates, **Approve Then Edit**. Click this button if you want to make further changes to contact data after you approve the create or update. See “Reviewing Pending Changes to Contacts” on page 252.

## Finding Duplicate Contacts for a Pending Create

There is a new button on the screens for pending creates, **Find Duplicates**. Click this button if you want to see if a contact in the pending create list is already in ContactManager. See “Reviewing Pending Changes to Contacts” on page 252.

## Merging and Editing Potential Duplicate Contacts

There is a new button on the screens for merging potential duplicate contacts, **Merge Then Edit**. Click this button if you want to make further changes to contact data after you merge two contacts. See “Merging Duplicate Contacts” on page 249.

## Changing the Subtype of a Contact Instance

---

**WARNING** This feature must be used with caution by experienced database and configuration professionals. It requires that ContactManager and ClaimCenter be in maintenance mode. The change of subtype directly affects the database and makes it impossible to synchronize the contact with ClaimCenter until you made the same change in both ClaimCenter and ContactManager. Additionally, the change can prevent users from editing the contact until you delete fields that are not compatible with the contact subtype.

---

You can change the subtype of a contact who was created with the wrong Contact subtype without having to delete and re-create the contact. For example, you created a contact with subtype **Doctor** when you intended the contact to be an **Attorney**. It can be less work, and safer, to delete the contact and re-create it with the correct subtype. However, deleting and re-creating might not be practical if the contact has history that you must preserve, such as being the payee on a check.

This feature is available through a new command-line utility. See “Changing the Subtype of a Contact Instance” on page 172.

## Using an Address Field to Search for Contacts

You can now add Address fields to ContactManager and core applications searches. See “Adding an Address Field to Contact Search” on page 93.

## Moved Changed ab800 Web Services, Helper Classes, and WSDL files to ab801

The following web services, APIs, and helper classes changed between ContactManager 8.0.0 and 8.0.1. The ab800 node in the original packages was changed to ab801, as shown in the following list:

- gsrc/gw/webservice/ab/ab801/MaintenanceToolsAPI.gs
- gsrc/gw/webservice/ab/ab801/MessagingToolsAPI.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPI.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPIAddressSearch.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPIFindDuplicatesResult.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPIFindDuplicatesResultContainer.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPIPendingContactChange.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPIProximitySearchParameters.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPIRelatedContact.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPISearchCriteria.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPISearchResult.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPISearchResultContainer.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPISearchSortColumn.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPISearchSpec.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPISpecialistService.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPISubtypeFilter.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPITagMatcher.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPIUtil.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ABContactAPIValidateCreateContactResult.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/AddressBookUIDContainer.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/AddressBookUIDTuple.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/AddressInfo.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/ExceptionHandler.gs
- gsrc/gw/webservice/ab/ab801/abcontactapi/RelatedContactInfoContainer.gs
- gsrc/wsi/local/gw/webservice/ab/ab801/MaintenanceToolsAPI.wsdl
- gsrc/wsi/local/gw/webservice/ab/ab801/MessagingToolsAPI.wsdl
- gsrc/wsi/local/gw/webservice/ab/ab801/abcontactapi/ABContactAPI.wsdl

## Changes to Core Application Integration with ContactManager

The following core application web services that are used to integrate with ContactManager have new xx801 files or references:

- ClaimCenter now uses `wsi.remote.gw.webservice.ab.ab801.wsc` to access the ContactManager web service  `${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl`. See “Step 1: Integrate ContactManager with ClaimCenter” on page 47.
- PolicyCenter now uses `wsi.remote.gw.webservice.ab.ab801.wsc` to access the ContactManager web service  `${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl`. See “Step 1: Integrate ContactManager with PolicyCenter” on page 53.
- BillingCenter now uses `wsi.remote.gw.webservice.ab.ab801.wsc` to access the ContactManager web service  `${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl`. See “Step 1: Integrate ContactManager with BillingCenter” on page 60.
- BillingCenter has moved all its 8.0.1 web services to directories with a `bc801` folder. For example,
  - The BillingCenter implementation of `ABCClientAPI`, `ContactAPI.gs`, is now in the package `gw.webservice.bc.bc801.contact`. See “Synchronizing BillingCenter and ContactManager Contacts”

on page 198.

- The `ContactAPI.wsdl` file is accessible in the BillingCenter studio Project window in configuration → gsrc at `wsi.local.gw.webservice.bc.bc801.contact`.

Therefore, the ContactManager web services collection `bc800.wsc`, which is in `wsi.remote.gw.webservice.bc`, now uses the following path to access the WSDL file for BillingCenter:

`${bc}/ws/gw/webservice/bc/bc801/contact/ContactAPI?wsdl`

See:

- “Step 1: Integrate ContactManager with BillingCenter” on page 60
- “Configuring ContactManager-to-BillingCenter Authentication” on page 79

## New And Changed Features in ContactManager 8.0.0

This topic includes:

- “New Features in ContactManager 8.0.0” on page 11
- “Changes in ContactManager 8.0.0” on page 12
- “ContactManager 8.0 Compatibility with Core Applications” on page 20

### New Features in ContactManager 8.0.0

This topic, which describes new features in ContactManager 8.0, includes:

- “Pending Creates and Pending Changes to Contacts” on page 11
- “Vendor Services” on page 11

#### Pending Creates and Pending Changes to Contacts

As part of the changes in contact management implemented in ClaimCenter, there are new APIs and a new Pending Changes screen in the ContactManager Contacts tab.

- A pending create is created when a ClaimCenter user without permission to create a ContactManager contact creates a vendor contact.

**Note:** When ContactManager receives a pending contact creation request for a Vendor contact, it creates a new contact entity that it marks as Pending. If that new contact creation is disapproved, ContactManager must delete that entity. Vendor contact edits, however, are just not applied until approved.

- A pending change is created when a ClaimCenter user without contact edit permissions makes a change to a vendor contact.

Either action is pending until a contact administrator logs into ContactManager and either confirms or denies the pending change or create.

#### See also

- For information on using the new Pending Changes screen, see “Reviewing Pending Changes to Contacts” on page 252.
- For information on the new APIs, see:
  - “ABContactAPI Web Service” on page 275
  - “ABClientAPI Interface” on page 282

#### Vendor Services

Vendor services in ContactManager support the Services feature in ClaimCenter. You can create the services that you expect vendors to provide, like carpentry or auto towing, and connect them to the vendor contacts who

provide them. ContactManager maintains the connection between each vendor and the vendor's services, and enables ClaimCenter to search for services and get a list of vendors who provide them.

**See also**

- For information on ContactManager support, see “Vendor Services” on page 181.
- For information on ClaimCenter vendor services, see “Services” on page 375 in the *Application Guide*.

## Changes in ContactManager 8.0.0

This topic describes what has changed for ContactManager 8.0.0. This topic includes:

- “Changes to Contact Integration Plugins, Web Services, and Mapper Classes” on page 12
- “Changes to Search Classes” on page 18
- “Contact Entity Mapping Classes and XML Files” on page 19
- “Changes to ClaimCenter Contact Synchronization” on page 19

### Changes to Contact Integration Plugins, Web Services, and Mapper Classes

There are changes to the names and packages of the plugins, web services, and entity mapping classes that ContactManager and the core applications use for integration and communicating contact information.

This topic includes:

- “ContactManager File and Package Changes for Core Application Integration” on page 13
- “ClaimCenter File Name, Method, and Package Changes for ContactManager Integration” on page 15
- “PolicyCenter File Name and Package Changes for ContactManager Integration” on page 16
- “BillingCenter File Name and Package Changes for ContactManager Integration” on page 17

## ContactManager File and Package Changes for Core Application Integration

**Note:** The following table does not include the changes to the search web services. See “Changes to Search Classes” on page 18.

ContactManager 7.0	ContactManager 8.0.0
<p>ABContactAPI.gs</p> <p>The web service available to core applications to make contact related calls into ContactManager, such as create, retrieve, update, and delete contacts.</p> <p>Package name – gw.webservice.ab.ab700.abcontactapi.</p>	<p>ABContactAPI.gs</p> <p>This web service has been updated to support services and pending contact changes.</p> <p>Package name – gw.webservice.ab.ab800.abcontactapi</p> <p>To find and open the class in Studio, use the <b>Class Search</b> dialog in non-project mode. <b>Note:</b> Press Ctrl+N twice to turn on search that includes non-project classes.</p> <p>See “ABContactAPI Web Service” on page 275.</p>
<p>ABCClientAPI and ABContactAPI methods have TransactionID parameter.</p> <p>The following methods used a transactionID parameter to identify the method call to the web service:</p> <ul style="list-style-type: none"> <li>• ABCClientAPI           <ul style="list-style-type: none"> <li>.mergeContacts</li> <li>.removeContact</li> <li>.updateContact</li> </ul> </li> <li>• ABContactAPI           <ul style="list-style-type: none"> <li>.createContact</li> <li>.removeContact</li> <li>.updateContact</li> </ul> </li> </ul>	<p>ABCClientAPI and ABContactAPI methods no longer use TransactionID parameter.</p> <p>The transaction ID is now set for the SOAP header in gw.webservice.contactapi.ContactAPIUtil.setTransactionId</p> <p>Instead of passing the transaction ID as part of the contact method call, it is set in a separate method. For example:</p> <pre>ContactAPIUtil. setTransactionId(     ABContactAPI.Config,     transactionId)  ABContactAPI. updateContact(xml)</pre> <p><b>See also</b></p> <ul style="list-style-type: none"> <li>• “Setting Guidewire Transaction IDs” on page 81 in the <i>Integration Guide</i></li> <li>• “ABCClientAPI Interface” on page 282</li> </ul> <p>“ABContactAPI Web Service” on page 275</p>
<p>IReviewSummaryAPI.gs</p> <p>An RPC-E web service available to core applications for creating and deleting review summaries corresponding to vendor service provider reviews in ClaimCenter.</p> <p>Package name – gw.webservice.ab.ab700.reviewsummary</p>	<p>ABVendorEvaluationAPI.gs</p> <p>A WS-I compliant web service. ClaimCenter uses this service to communicate information on vendor service provider reviews with ContactManager.</p> <p>Package name – gw.webservice.ab.ab800.abvendorevaluationapi</p> <p>To find and open the class in Studio, use the <b>Class Search</b> dialog in non-project mode. <b>Note:</b> Press Ctrl+N twice to turn on search that includes non-project classes.</p> <p>See “ABVendorEvaluationAPI Web Service” on page 279</p>
<p>ContactIntegrationXMLMapper.gs</p> <p>Name of the class that maps contact data as XML between ContactManager and the core applications.</p> <p>Package name – gw.webservice.ab.ab700.abcontactapi</p>	<p>ContactMapper</p> <p>This XML mapping class has been completely changed in ContactManager 8.0.</p> <p>Package name – gw.contactmapper.ab800</p> <p>See “ContactMapper Class” on page 285.</p>
	<p>ContactIntegrationXMLMapper.gs</p> <p>This XML mapping class supports integration with version 7.0 core applications.</p> <p>Package name – gw.contactmapper.ab700</p>

ContactManager 7.0	ContactManager 8.0.0
<b>ClientSystemPlugin</b> Name of the plugin interface: ClientSystemPlugin.java. Package name: gw.plugin.	<b>ClientSystemPlugin</b> No name change. Read-only file is now visible in Studio if you do a Ctrl+N search for ClientSystemPlugin.
<b>ClaimSystemPlugin.xml</b> Name of plugin registry. Navigate in Resource pane to configuration → Plugins → gw → plugin → ClientSystemPlugin	<b>ClaimSystemPlugin.gwp</b> Name of plugin registry. Navigate in Project window to configuration → config → Plugins → Registry
<b>CCClaimSystemPlugin</b> Plugin class to register when ClaimCenter 7.0 is installed. Extends AbstractClientSystemPlugin.gs. Package name – gw.plugin.claim.cc700.	<b>CCClaimSystemPlugin (cc800 version)</b> Plugin class to register when ClaimCenter 8.0 is installed. Extends ClientSystemPlugin800.gs. Package name – gw.plugin.claim.cc800. When ClaimCenter 8.0 is installed with ContactManager 8.0, register this plugin implementation.
	<b>CCClaimSystemPlugin (cc700 version)</b> Plugin class to register when ClaimCenter 7.0 is installed. Extends ClientSystemPlugin700.gs. Package name – gw.plugin.claim.cc700. When ClaimCenter 7.0 is installed with ContactManager 8.0, register this plugin implementation.
<b>PolicySystemPlugin.xml</b> Name of plugin registry. Navigate in Resource pane to configuration → Plugins → gw → plugin → ClientSystemPlugin	<b>PolicySystemPlugin.gwp</b> Name of plugin registry. Navigate in Project window to configuration → config → Plugins → Registry
<b>PCPolicySystemPlugin</b> Plugin class that extends AbstractClientSystemPlugin.gs. Package name – gw.plugin.policy.pc700.	<b>PCPolicySystemPlugin (pc800 version)</b> Plugin class that extends ClientSystemPlugin800.gs. Package name – gw.plugin.policy.pc800. When PolicyCenter 8.0 is installed with ContactManager 8.0, register this plugin implementation.
	<b>PCPolicySystemPlugin (pc700 version)</b> Plugin class that extends ClientSystemPlugin700.gs. Package name – gw.plugin.policy.pc700. When PolicyCenter 7.0 is installed with ContactManager 8.0, register this plugin implementation.
<b>BillingSystemPlugin.xml</b> Name of plugin registry. Navigate in Resource pane to configuration → Plugins → gw → plugin → ClientSystemPlugin	<b>BillingSystemPlugin.gwp</b> Name of plugin registry. Navigate in Project window to configuration → config → Plugins → Registry
<b>BCBillingSystemPlugin</b> Plugin class that extends AbstractClientSystemPlugin.gs. Package name – gw.plugin.billing.bc700	<b>BCBillingSystemPlugin (bc800 version)</b> Plugin class that extends ClientSystemPlugin800.gs. Package name – gw.plugin.billing.bc800. When BillingCenter 8.0 is installed with ContactManager 8.0, register this plugin implementation.

ContactManager 7.0	ContactManager 8.0.0
	<p>BCBillingSystemPlugin (bc700 version)            Plugin class that extends ClientSystemPlugin700.gs.            Package name –  <code>gw.plugin.billing.bc700</code>.            When BillingCenter 7.0 is installed with ContactManager 8.0, register this plugin implementation.</p>
<b>ABCClientAPI.gs</b> The interface implemented by core applications to provide a way for ContactManager to call into those applications. Package name – <code>gw.webservice.ab.ab700abcontactapi</code> .	<p>ABCClientAPI.gs (ab800 version)            This interface has been updated to support pending contact changes.            Package name –  <code>gw.webservice.contactapi.ab800</code>            To find and open the class in Studio, use the <b>Class Search</b> dialog in non-project mode. <b>Note</b> Press <b>Ctrl+N</b> twice to turn on search that includes non-project classes.            See “ABCClientAPI Interface” on page 282.</p>
	<p>ABCClientAPI.gs (ab700 version)            This interface supports ContactManager 7.0 and is in a new package.            Package name –  <code>gw.webservice.contactapi.ab700</code></p>

#### ClaimCenter File Name, Method, and Package Changes for ContactManager Integration

ClaimCenter 7.0	ClaimCenter 8.0.0
<b>IAddressBookAdapter</b> <b>IContactSearchAdapter</b> Both 7.0 plugin interfaces are deprecated in ClaimCenter 8.0 and have been replaced by <b>ContactSystemPlugin</b> .	<b>ContactSystemPlugin</b> Name of both the plugin registry, <b>ContactSystemPlugin.gwp</b> , and the plugin interface, <b>ContactSystemPlugin.java</b> . See “Integrating ContactManager with ClaimCenter in Quick-Start” on page 46.
<b>ABContactPlugin</b> Plugin class that implements <b>IAddressBookAdapter.java</b> Package name – <code>gw.plugin.addressbook.ab700</code> .	<b>ABContactSystemPlugin (ab800 version)</b> Plugin class that implements <b>ContactSystemPlugin.java</b> Package name – <code>gw.plugin.contact.ab800</code> . When ContactManager 8.0 is installed with ClaimCenter 8.0, register this plugin implementation.
<b>ContactIntegrationXMLMapper.gs</b> Name of the class that maps contact data as XML between ClaimCenter and ContactManager. Package name – <code>gw.plugin.addressbook.ab700.mapper</code> .	<b>ContactMapper.gs</b> This XML mapping class has been completely changed in ClaimCenter 8.0. Package name – <code>gw.contactmapper.ab800</code> See “ContactMapper Class” on page 285.
<b>cc-to-cm-type-mapping.xml</b> and <b>cm-to-cc-type-mapping.xml</b> Name of the files used to specify how to map differing entity names and typecodes between ClaimCenter and ContactManager.	<b>CCNameMapper.gs</b> This Gosu class replaces the two XML mapping files in ClaimCenter 8.0. Package name – <code>gw.contactmapper.ab800</code> See “Core Application Mapping” on page 141.

<b>ClaimCenter 7.0</b>	<b>ClaimCenter 8.0.0</b>
<p>ContactAPI.gs</p> <p>The web service that implements ABClientAPI to provide a way for ContactManager to call into ClaimCenter.</p> <p>Package name – gw.webservice.cc.cc700.contact</p>	<p>ContactAPI.gs</p> <p>This web service has been updated to support services and pending contact changes.</p> <p>Package name – gw.webservice.cc.cc800.contact</p> <p>See “ABClientAPI Interface” on page 282.</p>
<p>contact-sync-config.xml</p> <p>When determining if a ClaimCenter contact is synchronized with its ContactManager counterpart, this file defines:</p> <ul style="list-style-type: none"> <li>• The contact properties to exclude from the comparison</li> <li>• The contact relationships to include in or exclude from the comparison</li> </ul> <p>You can specify relationships to include or exclude based on Contact subtype.</p>	<ul style="list-style-type: none"> <li>• <code>withAffectsSync(false)</code> Use this method, available in the ClaimCenter ContactMapper class, to exclude fields of Contact entities and subentities from the synchronization check.</li> <li>• <code>RelationshipSyncConfig.gs</code> Replaces the XML file, which was visible in PolicyCenter and BillingCenter even though those core applications did not use it. Now available only for ClaimCenter, this Gosu class provides the same functionality as the XML file through its <code>includeRelationship</code> and <code>excludeRelationship</code> methods.</li> </ul> <p>Package name – gw.plugin.contact.ab800</p> <p>See “Synchronizing ClaimCenter Contact Fields” on page 203.</p>

### PolicyCenter File Name and Package Changes for ContactManager Integration

<b>PolicyCenter 7.0</b>	<b>PolicyCenter 8.0.0</b>
<p>ContactSystemPlugin</p> <p>Name of both plugin registry, <code>ContactSystemPlugin.xml</code>, and plugin interface, <code>ContactSystemPlugin.java</code></p>	<p>ContactSystemPlugin</p> <p>Name of both the plugin registry, <code>ContactSystemPlugin.gwp</code>, and the plugin interface, <code>ContactSystemPlugin.java</code></p> <p>See “Integrating ContactManager with PolicyCenter in QuickStart” on page 52.</p>
<p>ABContactSystemPlugin</p> <p>Plugin class that implements <code>ContactSystemPlugin.java</code>.</p> <p>Package name – <code>gw.plugin.contact.ab700</code></p>	<p>ABContactSystemPlugin (ab800 version)</p> <p>Plugin class that implements <code>ContactSystemPlugin.java</code>.</p> <p>Package name – <code>gw.plugin.contact.ab800</code>.</p> <p>When ContactManager 8.0 is installed with PolicyCenter 8.0, register this plugin implementation.</p>
	<p>ABContactSystemPlugin (ab700 version)</p> <p>Plugin class that implements <code>ContactSystemPlugin.java</code>.</p> <p>Package name – <code>gw.plugin.contact.ab700</code>.</p> <p>When ContactManager 7.0 is installed with PolicyCenter 8.0, register this plugin implementation.</p>
<p>ContactIntegrationXMLMapper.gs</p> <p>Name of the class that maps contact data as XML between PolicyCenter and ContactManager.</p> <p>Package name – <code>gw.plugin.contact.ab700</code></p>	<p>ContactMapper.gs</p> <p>This XML mapping class has been completely changed in PolicyCenter 8.0. It supports integration with ContactManager 8.0.</p> <p>Package name – <code>gw.contactmapper.ab800</code></p> <p>See “ContactMapper Class” on page 285.</p>
	<p>ContactIntegrationXMLMapper.gs</p> <p>This XML mapping class supports integration with ContactManager 7.0.</p> <p>Package name – <code>gw.contactmapper.ab700</code></p>

PolicyCenter 7.0	PolicyCenter 8.0.0
<p><code>pc-to-cm-type-mapping.xml</code> and <code>cm-to-pc-type-mapping.xml</code></p> <p>Name of the files used to specify how to map differing entity names and typecodes between PolicyCenter and ContactManager.</p>	<p><code>PCNameMapper.gs</code></p> <p>This Gosu class replaces the two XML mapping files in PolicyCenter 8.0.</p> <p>Package name – <code>gw.contactmapper.ab800</code></p> <p>See “Core Application Mapping” on page 141.</p>
<p><code>ContactAPI.gs</code></p> <p>The web service that implements ABClientAPI to provide a way for ContactManager to call into PolicyCenter.</p> <p>Package name – <code>gw.webservice.pc.pc700.contact</code></p>	<p><code>ContactAPI.gs</code> (ab800 version)</p> <p>Package name – <code>gw.webservice.pc.pc800.contact</code></p> <p>See “ABClientAPI Interface” on page 282.</p>
	<p><code>ContactAPI.gs</code> (ab700 version)</p> <p>This web service supports integration with ContactManager 7.0.</p> <p>Package name – <code>gw.webservice.pc.pc700.contact</code></p>

### BillingCenter File Name and Package Changes for ContactManager Integration

BillingCenter 7.0	BillingCenter 8.0.0
<p><code>IContactSystemPlugin</code></p> <p>Name of both plugin registry, <code>IContactSystemPlugin.xml</code>, and plugin interface, <code>IContactSystemPlugin.java</code></p>	<p><code>ContactSystemPlugin</code></p> <p>Name of both the plugin registry, <code>ContactSystemPlugin.gwp</code>, and the plugin interface, <code>ContactSystemPlugin.java</code></p> <p>See “Integrating ContactManager with BillingCenter in QuickStart” on page 59.</p>
<p><code>ContactManagerSystemPlugin</code></p> <p>Plugin class that implements <code>IContactSystemPlugin.java</code>.</p> <p>Package name – <code>gw.plugin.contact.ab700</code></p>	<p><code>ABContactSystemPlugin</code> (ab800 version)</p> <p>Plugin class that implements <code>ContactSystemPlugin.java</code>.</p> <p>Package name – <code>gw.plugin.contact.ab800</code>.</p> <p>When ContactManager 8.0 is installed with BillingCenter 8.0, register this plugin implementation.</p>
	<p><code>ContactManagerSystemPlugin</code> (ab700 version)</p> <p>Plugin class that implements <code>IContactSystemPlugin.java</code>.</p> <p>Package name – <code>gw.plugin.contact.ab700</code>.</p> <p>When ContactManager 7.0 is installed with BillingCenter 8.0, register this plugin implementation in the registry <code>ContactSystemPlugin.gwp</code>.</p> <p><b>Note:</b> Plugin implementation class name and plugin interface it implements are different from ab800 version.</p>
<p><code>ContactIntegrationXMLMapper.gs</code></p> <p>Name of the class that maps contact data as XML between BillingCenter and ContactManager.</p> <p>Package name – <code>gw.plugin.addressbook.impl</code></p>	<p><code>ContactMapper.gs</code></p> <p>This XML mapping class has been completely changed in BillingCenter 8.0. It supports integration with ContactManager 8.0.</p> <p>Package name – <code>gw.contactmapper.ab800</code></p> <p>See “ContactMapper Class” on page 285.</p>

BillingCenter 7.0	BillingCenter 8.0.0
	<p><code>ContactIntegrationXMLMapper.gs</code>  This XML mapping class supports integration with ContactManager 7.0.</p> <p>Package name –  <code>gw.contactmapper.ab700</code></p>
<code>bc-to-cm-type-mapping.xml</code> and <code>cm-to-bc-type-mapping.xml</code> Name of the files used to specify how to map differing entity names and typecodes between BillingCenter and ContactManager.	<p><code>BCNameMapper.gs</code>  This Gosu class replaces the two XML mapping files in BillingCenter 8.0.</p> <p>Package name –  <code>gw.contactmapper.ab800</code></p> <p>See “Core Application Mapping” on page 141.</p>
<code>ContactAPI.gs</code> The web service that implements ABClientAPI to provide a way for ContactManager to call into BillingCenter.  Package name – <code>gw.webservice.bc.bc700.contact</code>	<p><code>ContactAPI.gs</code> (ab800 version)  Package name –  <code>gw.webservice.bc.bc800.contact</code></p> <p>See “ABClientAPI Interface” on page 282.</p>
	<p><code>ContactAPI.gs</code> (ab700 version)  This web service supports integration with ContactManager 7.0 and has been moved to a different package.</p> <p>Package name –  <code>gw.webservice.bc.bc700.contact</code></p>

### See also

- For instructions on how to register core Guidewire plugin implementations to integrate with ContactManager, see “Integrating ContactManager with Guidewire Core Applications” on page 45.
- For a reference description of the API, see “ABContactAPI Web Service” on page 275.

### Changes to Search Classes

There are changes to the names of search classes that ContactManager provides to core applications for searching for contacts. All the ContactManager 8.0.0 classes are in the class package `gw.webservice.ab.ab800.abcontactapi`.

ContactManager 7.0	ContactManager 8.0.0
<code>ABContactSearchCriteriaInfo.gs</code> A class that specifies the contact search criteria that the Guidewire core applications can use.	<code>ABContactAPISearchCriteria.gs</code> A class that specifies the contact search criteria that the Guidewire core applications can use. You can edit this class to add new search criteria for contacts. See “ContactManager Support for Core Application Searches” on page 86.
<code>ABContactAPISearchResult.gs</code> A class that specifies the fields included in the contact search results that ContactManager sends back to the Guidewire core application.	<code>ABContactAPISearchResult.gs</code> A class that specifies the fields included in the contact search results that ContactManager sends back to the Guidewire core application. You can edit this class to send additional search results to core applications. See “ContactManager Support for Core Application Searches” on page 86.
<code>ABContactAPISearchSpecInfo.gs</code> Provides the search criteria used by the web service method <code>ABContactAPI.findDuplicates</code> .	<code>ABContactAPISearchSpec.gs</code> Provides the search criteria used by the web service method <code>ABContactAPI.findDuplicates</code> .

ContactManager 7.0	ContactManager 8.0.0
AddressSearchInfo.gs	ABContactAPIAddressSearch.gs Used by ABContactAPISearchCriteria to declare the Address and ProximitySearchCenter variables.
ProximitySearchParametersInfo.gs	ABContactAPIProximitySearchParameters.gs Used by ABContactAPISearchCriteria to declare the ProximitySearchParameters variable.

## Changes to Contact Search Functionality

In addition to the changes in class name listed in the previous topic, there are other changes that affect how searches are performed across the core applications and ContactManager.

### Changes to Typekey Criteria in Contact Searches

ABContactAPI now exposes typekeys used in Contact search as strings.

In the previous release, ContactManager 7.0 exposed these typekeys as enum values that a core application could access from `ContactSearchMapper` as follows:

```
searchCriteriaInfo.VendorType =
    wsi.remote.gw.webservice.ab.ab700.abcontactapi.enums.VendorType.forGosuValue(
        searchCriteria.VendorType.Code)
```

In ContactManager 8.0, ABContactAPI exposes typekeys as strings, such as the `VendorType` typelist:

```
@WsiExposeEnumAsString(typekey.VendorType)
```

Now that these typekeys are simple strings, the core application method call is also much simpler. For example, the ClaimCenter `ContactSearchMapper` code for `VendorType` search criterion is now:

```
searchCriteriaInfo.VendorType = searchCriteria.VendorType.Code
```

## Contact Entity Mapping Classes and XML Files

As described previously, the mapping class names have changed and the XML mapping files used in core applications are now Gosu classes. Specifically:

- The `ContactIntegrationXMLMapper` class has been refactored in `ContactMapper`. The new class provides:
  - Simpler code for adding foreign keys and array references
  - Ability to specify fields that determine if a contact is in sync
  - Ability to specify fields for a contact that are persisted in the core application
- The XML configuration files `cc-to-cm-type-mapping.xml` and `cm-to-cc-type-mapping.xml` have been replaced by the single Gosu class `gw.contactmapper.ab800.CCNameMapper.gs`.

For the tables showing the name changes, see “Changes to Contact Integration Plugins, Web Services, and Mapper Classes” on page 12.

### See also

- “Working with Contact Mapping Files” on page 140
- “Core Application Mapping” on page 141
- “ContactMapper Class” on page 285

## Changes to ClaimCenter Contact Synchronization

Previously, you used the `IgnoreProperty` element of `contact-sync-config.xml` to exclude Contact fields from determining synchronization. In ContactManager 8.0, you no longer use this file at all.

- To exclude other fields of a contact, instead of using the `IgnoreProperty` element, you use the `ContactMapper` method `withAffectsSync`. By default, all fields that you add to `ContactMapper` are included

in the fields that are checked for synchronization status unless you specifically exclude them by using `withAffectsSync`. See “Setting Properties to Ignore in Determining a Contact’s Synchronization Status” on page 203.

- To exclude contact relationships from the set of fields ClaimCenter uses to determine if a contact is synchronized with ContactManager, use the Gosu class `RelationshipSyncConfig`. See “Setting Relationships to Be Included or Excluded” on page 204.

#### See also

- “Synchronizing ClaimCenter Contact Fields” on page 203

## ContactManager 8.0 Compatibility with Core Applications

ContactManager and the Guidewire core applications support the following compatibility scenarios:

- ContactManager 8 and a core application with version 8.0 or later.  
**Note:** ClaimCenter 8.0 supports only ContactManager 8.0. The 8.0 version of ClaimCenter does not support ContactManager 7.0.
- ContactManager 8 and ClaimCenter 7.0.0 or later. See “Integrating ClaimCenter 7 with ContactManager 8” on page 51.
- ContactManager 8 and PolicyCenter 7.0.1 or later. See “Integrating PolicyCenter 7 with ContactManager 8” on page 58.
- ContactManager 8 and BillingCenter 7.0.0 or later. See “Integrating BillingCenter 7 with ContactManager 8” on page 64.

## New and Changed Features in ContactManager 7.0

This topic describes new features in ContactManager 7.0. This topic includes:

- “ContactManager 7.0 Client Data Management Add-on Module for PolicyCenter” on page 20
- “ContactManager 7.0 Web Services API for Integration with Guidewire Core Applications” on page 21
- “ContactManager 7.0 Plugins for Communicating with Guidewire Core Applications” on page 21
- “ContactManager 7.0 Contact Tags” on page 21
- “ContactManager 7.0 Minimum Criteria for Creating an ABContact” on page 21
- “Finding Duplicate Contacts with ContactManager 7.0” on page 21
- “Duplicate Batch Detection and Merge in ContactManager 7.0” on page 22

## New Features in ContactManager 7.0

### ContactManager 7.0 Client Data Management Add-on Module for PolicyCenter

ContactManager now can support client contacts, available with PolicyCenter as an optional add-on module called Client Data Management. In PolicyCenter you can save your client data, such as the primary insured for an account or policy, to ContactManager. This centralized database for client contacts is the system of record for all Guidewire InsuranceSuite core applications.

Client Data Management provides client data integration and management across all Guidewire core applications by synchronizing client changes and managing a unique ID for each contact.

**Note:** ContactManager will continue to ship with ClaimCenter and provide an address book database for vendor contacts in ClaimCenter.

## ContactManager 7.0 Web Services API for Integration with Guidewire Core Applications

The web service that ContactManager provides for working with contacts, including creating, updating, retrieving, and searching for contacts, is `ABContactAPI`, a WS-I compliant SOAP service. This web service is now usable by PolicyCenter as well as ClaimCenter. For more information, see:

## ContactManager 7.0 Plugins for Communicating with Guidewire Core Applications

ContactManager has two plugins that it uses to broadcast changes in contacts to ClaimCenter and PolicyCenter.

- For communicating with ClaimCenter, ContactManager uses `ClaimSystemPlugin`.
- For communicating with PolicyCenter, ContactManager uses `PolicySystemPlugin`.
- For communicating with BillingCenter, ContactManager uses `BillingSystemPlugin`.

## ContactManager 7.0 Contact Tags

There is a new feature that enables you to classify contacts without having to add new subtypes, *contact tags*. Every contact stored by ContactManager must have at least one tag. The three tags provided in the base product are:

- **Client** – A policy or account contact in PolicyCenter. For example, the holder of an account, the primary named insured on a policy, or a driver of a vehicle insured by a policy.
- **Vendor** – A ClaimCenter vendor providing services that help resolve claim losses. For example, a body shop, assessor, attorney, or physical therapy clinic.
- **Claim Party** – A party to a claim in ClaimCenter, such as the insured or a claimant.

These tags indicate that a contact is a client, and therefore of interest to PolicyCenter, and potentially ClaimCenter and BillingCenter, or a vendor, of interest only to ClaimCenter. Additionally, the Claim Party tag indicates that a contact has been added as a party to a claim.

Any combination of tags can apply to a contact, such as a contact that is both a vendor and a client. You can also add additional tags and change how the applications use them.

As with contact subtypes, there are permissions associated with tags. The base application permissions are special permissions that enable a user to create, edit, delete, and view contacts that have any tag.

ContactManager uses a set of permission check expressions to control access to screens. Previously, to perform an action, users needed permission only for the contact's subtype. Now, they need permission for both the contact's subtype and the contact's tags. For example, suppose there is a Company contact. To edit that contact, the user needs permission to edit contacts of that subtype. Suppose the contact has the Client tag. The user still needs permission to edit Company contacts and now also needs permission to edit contacts with the Client tag.

## ContactManager 7.0 Minimum Criteria for Creating an ABContact

The Gosu plugin `ValidateABContactCreationPlugin` defines the minimum criteria required to create a contact. Whenever ContactManager processes a request to create a contact, it uses this plugin to determine if there is sufficient data to continue with the creation. For example, ContactManager does this validation when a Guidewire core application calls the ContactManager web services API method `ABContactAPI.createContact` to create a new contact. This method simply creates the contact. It does not check to see if one already exists.

## Finding Duplicate Contacts with ContactManager 7.0

Finding duplicate contacts in the ContactManager database is supported by the `FindDuplicatesPlugin` class. This class, which implements the interface `IFindDuplicatesPlugin`, uses a set of Duplicate Finder classes to define criteria for exact and potential matches. There is also a new ContactManager web service, `ABContactAPI.findDuplicates`, that Guidewire core applications can call to check for duplicates of a contact.

## Duplicate Batch Detection and Merge in ContactManager 7.0

As described in “ContactManager 7.0 Definitive Match and Minimum Creation Criteria” on page 24, creation of contacts can now result in duplicate contacts. Therefore, there are new features in ContactManager to detect duplicate contacts and enable merging them. A user with appropriate privileges in ContactManager can run a batch process to detect duplicate contacts and then merge each duplicate contact in the [Merge Contacts](#) screen.

## Changed Features in ContactManager 7.0

This topic describes what has changed for ContactManager 7.0. This topic includes:

- “ContactManager 7.0 Name Change from ContactCenter to ContactManager” on page 22
- “ContactManager 7.0 as System of Record” on page 22
- “ContactManager 7.0 Web Services API and Plugins for Integration with Guidewire Core Applications” on page 22
- “ContactManager 7.0 Entity Mapping Classes and XML Files” on page 23
- “ContactManager 7.0 Definitive Match and Minimum Creation Criteria” on page 24
- “Definitive and Potential Matching and ContactManager 7.0 Find Duplicates” on page 25
- “Setting Search Criteria and Search Results in ContactManager 7.0” on page 25
- “Setting Minimum Search Criteria in ContactManager 7.0” on page 25
- “Broadcasting Contact Changes from ContactManager 7.0 to Guidewire Core Applications” on page 26
- “ContactManager 7.0 Geocoding Changes” on page 27

### ContactManager 7.0 Name Change from ContactCenter to ContactManager

Previous releases of the product were called ContactCenter. Starting with the 7.0 version, the name is ContactManager.

### ContactManager 7.0 as System of Record

ContactManager now has the ability to be the system of record (SOR) for contacts for all Guidewire core applications. In previous versions of the products, any change to a contact sent by ClaimCenter to ContactManager was accepted unconditionally. ClaimCenter was the only application communicating with ContactManager, and it controlled the visibility of the [Save to Address Book](#) button to enforce when updates were allowed.

Now that it supports all Guidewire core applications, ContactManager cannot unconditionally accept contact updates. ContactManager now requires that a Guidewire core application sending an update include the original value of the contact’s information. This requirement ensures that the contact has not already been changed by another Guidewire core application.

These changes are supported by a new mapping class in ContactManager and in each Guidewire core application, [ContactIntegrationXMLMapper](#). In addition, each application has new XML mapping configuration files that map differing entity names and typecode values for Contact entities and subentities.

One change to the user interface that supports this SOR role for ContactManager is an [Update Address Book](#) button in ClaimCenter that explicitly sends contact changes to ContactManager. This button is available only if the contact has been synchronized with ContactManager. You can use the [Update Address Book](#) button when you are editing a contact in ClaimCenter. Additionally, if you are working with claim party, you might see the [Save to Claim and Address Book](#) button when editing or adding the contact.

### ContactManager 7.0 Web Services API and Plugins for Integration with Guidewire Core Applications

In previous releases of ContactManager, the web services API that ContactManager provided for integration with Guidewire core applications was [IContactAPI](#). That API is now deprecated. The replacement is [ABContactAPI](#),

a WS-I compliant SOAP service. The new API provides methods that allow more flexibility in supporting different configurations.

ClaimCenter continues to use a plugin implementation of `IAddressBookAdapter` to integrate with ContactManager, but the implementation has changed.

- The old plugin implementation was `gw.plugin.ccabintegration.impl.CCAddressBookPlugin`. This implementation called the ContactCenter web service `IContactAPI`.
- The new plugin implementation is `gw.plugin.addressbook.ab700.ABContactPlugin`. This implementation calls the ContactManager web service `ABContactAPI`.

## ContactManager 7.0 Entity Mapping Classes and XML Files

There are changes to the `Contact` entity mapping process in ClaimCenter that now require that you configure ContactManager as well.

In the past, integration between ClaimCenter and ContactManager involved passing entities between the two applications. The ClaimCenter file `cc-ab-data-mapping.xml` enabled mapping ClaimCenter entities to ContactCenter entities before sending the entities. The ClaimCenter file `ab-to-cc-data-mapping.xml` mapped incoming ContactCenter entities to ClaimCenter entities.

With the change to WS-I web services in the current release of ContactManager, web services no longer pass entities between applications. Instead, they pass XML representations of those entities. To support mapping between XML and entities, all Guidewire applications, including ContactManager, now use `ContactIntegrationXMLMapper` mapping classes.

The changed process requires that you explicitly map all fields of your `Contact` extensions both in the Guidewire core application and in ContactManager. The files in all the Guidewire applications have some differences.

Previous versions of ClaimCenter used `cc-to-ab-data-mapping.xml` and `ab-to-cc-data-mapping.xml` to map `Contact` entities and their fields in ClaimCenter to `ABContact` entities and their fields in ContactCenter.

ClaimCenter no longer uses those files. Instead, ClaimCenter now uses a different combination of files for this purpose:

- The class `gw.plugin.addressbook.ab700.mapper.ContactIntegrationXMLMapper` maps the fields, arrays, and foreign keys.
- The files `cc-to-cm-type-mapping.xml` and `cm-to-cc-type-mapping.xml` map differing entity names and typecodes.

In PolicyCenter, you use the following files:

- The class `gw.plugin.contact.ab700.ContactIntegrationXMLMapper` maps the fields of contact entities between PolicyCenter and ContactManager.
- The files `pc-to-cm-type-mapping.xml` and `cm-to-pc-type-mapping.xml` map differing entity names and typecodes.

In BillingCenter, you use the following files:

- The class `gw.plugin.contact.impl.ContactIntegrationXMLMapper` maps the fields of contact entities between PolicyCenter and ContactManager.
- The files `bc-to-cm-type-mapping.xml` and `cm-to-bc-type-mapping.xml` map differing entity names and typecodes.

In ContactManager, you use the class

`gw.webservice.ab.ab700.abcontactapi.ContactIntegrationXMLMapper` to map `ABContact` entity fields between ContactManager and ClaimCenter or one of the other Guidewire core applications. ContactManager does not require data-mapping XML files—all entity and typecode mapping is performed in the Guidewire core applications that integrate with ContactManager.

## Authentication Between ContactManager 7.0 and Core Applications

The way you set up user names and passwords to communicate between Guidewire Core Applications and ContactManager has changed. There are new classes in the core applications that define the user name and password to use with the ContactManager web services. There are equivalent classes in ContactManager that define the user names and passwords that ContactManager uses when it calls core application web services.

### ContactManager 7.0 Definitive Match and Minimum Creation Criteria

In previous releases, ContactManager used definitive matching to uniquely identify a contact. This matching was based on a set of configurable fields defined in the `definitive-match-config.xml` file. The set of fields for a contact constituted a unique ID for the contact, a `MatchSetKey` enforced at the database level with a unique index on the composite key. This definitive matching was used as the first step both in creating new contacts and in linking to contacts from Guidewire core applications.

This scheme had its limits. All criteria specified in the configuration file had to be applied all the time, and there could not be more than one definitive match definition for a single subtype. There are locales that do not require unique identifiers, like tax IDs, and it was not possible to evaluate the country of the contact and apply different criteria. It was also not possible to require certain criteria and make others optional. For example, you could not require first name and last name and either date of birth or address or drivers license.

Additionally, for PolicyCenter client contacts, the amount of information available at creation time can vary based on the collector of the information and the role of the client. Therefore, a rigid set of criteria does not work.

Therefore, ContactManager has relaxed the minimum creation criteria for contacts and has changed from using an XML configuration file to using a Gosu class, `ValidateABContactCreationPluginImpl`. This class gives you the flexibility of using the Gosu language.

**Note:** Relaxing the minimum contact creation criteria means that there is more potential for duplicates in contact data. To detect duplicates and manage them, ContactManager has added a new batch process and a new **Merge Contacts** screen. See “Duplicate Batch Detection and Merge in ContactManager 7.0” on page 22.

ContactManager has relaxed minimum creation criteria and has made the following changes in the process for creating a new contact:

- There is no database-level constraint that prevents duplicate contacts from being created. To support these changes:
  - The following uniqueness index on `MatchSetKey` is no longer in the `ABContact` data model.

```
<index desc="Unique index on {MatchSetKey, Retired}"  
       name="matchsetkey" unique="true" verifyInLoader="false">  
    <indexcol keyposition="1" name="MatchSetKey"/>  
    <indexcol keyposition="2" name="Retired"/>  
</index>
```
  - The `MatchSetKey` column in the `ABContact` data model is now deprecated.

```
<column  
      deprecated="true"  
      desc="Match set key for the contact. Used to find definitive matches."  
      exportable="false"  
      getterScriptability="hidden"  
      loadable="false"  
      name="MatchSetKey"  
      setterScriptability="hidden"  
      type="abcontactmatchsetkey"/>
```
- There is a new Gosu plugin, `ValidateABContactCreationPlugin`, which in the base product has more relaxed criteria for creating a new contact.
- The web services API method `ABContactAPI.createContact` calls the Gosu plugin to determine if there is sufficient information to create a new contact. This method simply creates the contact. It does not check to see if one already exists.

## Definitive and Potential Matching and ContactManager 7.0 Find Duplicates

In previous releases, ContactManager used definitive matching as the first stage in creating new contacts, as described at “ContactManager 7.0 Definitive Match and Minimum Creation Criteria” on page 24. Additionally, definitive matching was the first stage in returning matches to the Guidewire core applications. The definitive match criteria were defined in `definitive-match-config.xml`. If no definitive match was found, the second stage of matching was to find a potential match. The potential match search was based on elements defined in `potential-match-config.xml`. This search could expand to all contact subtypes if no potential match was found for the specific subtype with which the search started.

Also in previous releases, the web services that supported finding definitive and potential matches were `IContactAPI.findDefinitiveMatch` and `IContactAPI.findPotentialMatches`.

This scheme had its limits. All criteria specified in the configuration files had to be applied all the time, and there could not be more than one definitive or potential match definition for a subtype. There are locales that do not require unique identifiers, like tax IDs, and it was not possible to evaluate the country of the contact and apply different criteria. It was also not possible to require certain criteria and make others optional.

This behavior is now changed. Definitive matching is no longer the basis for creating a new contact, as described previously in “ContactManager 7.0 Definitive Match and Minimum Creation Criteria” on page 24. Additionally, the matching functionality is now called *Find Duplicates*, and the match types are called *exact match* and *potential match*.

Instead of the XML configuration files, there is a new `FindDuplicatesPlugin` plugin that implements the interface `IFindDuplicatesPlugin`, as well as a set of Duplicate Finder classes. There is also a new web service, `ABContactAPI.findDuplicates`.

## Setting Search Criteria and Search Results in ContactManager 7.0

In previous releases, entries in `ABContactSearchCriteria.etc` defined how ContactManager handled entities and fields in Search requests. Additionally, to specify that a field be shown in the search results for a Guidewire core application, you needed to add an entry to `search-filter.xml` in that application.

Instead of adding entries to `search-filter.xml` in the Guidewire core application, you now define fields for ContactManager in `ABContactAPISearchResult`. There are also a new classes in each Guidewire core application that handle sending search criteria and showing search results. For example, ClaimCenter now uses the classes `ContactSearchMapper` and `ContactSearchResultMapper` for these purposes.

ContactManager continues to use `ABContactSearchCriteria.etc`. There are now two additional Gosu files in ContactManager that provide web services supporting search by Guidewire core applications. These files are:

- `gw.webservice.ab.ab700.abcontactapi.ABContactSearchCriteriaInfo` – Provides search criteria as a web service to Guidewire core applications. A Guidewire core application uses this web service to determine which criteria to use in a contact Search.
- `gw.webservice.ab.ab700.abcontactapi.ABContactAPISearchResult` – Defines search results sent to Guidewire core applications as a web service. A Guidewire core application uses this web service to interpret Search results that ContactManager returns after a search request from the application.

**Note:** To improve search performance, Guidewire recommends that you add an index in the ContactManager entity definition for every field that you use in the search.

## Setting Minimum Search Criteria in ContactManager 7.0

In earlier versions of ContactManager, if you wanted to specify that a `Contact` entity field was required in a search, you added it to the `AdditionalPrimaryABContactSearchCriteria` element in `config.xml`. That approach had some limits. For example, you could not specify different fields for contacts located in different countries.

In ContactManager, this functionality has been moved to the plugin `ValidateABContactSearchCriteriaPlugin`. You can add new, required search fields by editing the class `gw.plugin.contact.ValidateABContactSearchCriteriaPluginImpl`. Since the class is written in Gosu, you have considerable flexibility in how you specify which fields to require for a search.

The following code snippet from this class shows the default search criteria defined for an `ABPerson` entity:

```
private class ABPersonLogic extends ABContactSubtypeLogic {
    override function execute(bean : Bean) : boolean {
        var searchCriteria = (bean as ABContactSearchCriteria)
        if (searchCriteria.Keyword == null
            and searchCriteria.FirstName == null
            and searchCriteria.TaxID == null
            and isInLocale(searchCriteria)
            and searchCriteria.Address.PostalCode == null
            and not searchCriteria.isValidProximitySearch()) {
            return false
        }
        return true
    }
}
```

### Backwards Compatibility

If you want to continue to use your `AdditionalPrimaryABContactSearchCriteria` settings in `search-config.xml`, you can register the plugin `BackwardsCompatibleValidateABContactSearchCriteriaPlugin` as follows:

1. At a command prompt, navigate to the `ContactManager bin` directory, and then start Studio by entering the following command:  
`gwab studio`
2. Navigate in the **Resources** pane to **configuration** → **Plugins** → **gw** → **plugin** → **Contact** → **ValidateABContactSearchCriteriaPlugin**.
3. In the **Plugin** editor, you can see that the default registered plugin is:  
`gw.plugin.contact.ValidateABContactSearchCriteriaPluginImpl`
4. Click **Remove**. If you see a message asking if you want create a copy of the plugin, click **Yes**.
5. Click **Add** → **Gosu** and enter the following class:  
`gw.plugin.contact.impl.BackwardsCompatibleValidateABContactSearchCriteriaPlugin`
6. At this point, you need to refresh the web services for ContactManager and your Guidewire core applications. To do that, stop and restart all the servers.

### Broadcasting Contact Changes from ContactManager 7.0 to Guidewire Core Applications

In previous releases, to synchronize changes in contact data with ClaimCenter, ContactManager used the ClaimCenter web services API `IContactAutoSyncAPI`. ContactManager no longer uses this web service.

ContactManager now supports synchronization of contact data with all Guidewire core applications. It manages all broadcasts of changes through the `ABCClientAPI` interface, which is implemented as a web service by each Guidewire core application.

ContactManager uses the following plugins to broadcast changes:

- To send changes to ClaimCenter, ContactManager uses a plugin registered in `ClaimSystemPlugin`. In ContactManager Studio, navigate in the **Resources** pane to **configuration** → **Plugins** → **gw** → **ClientSystemPlugin** → **ClaimSystemPlugin**. You must register the plugin `gw.plugin.claim.cc700.CCClaimSystemPlugin`. On the ClaimCenter side, calls from ContactManager are handled by methods of the `ContactAPI` web service., which implements the `ABCClientAPI` interface.
- To send changes to PolicyCenter, ContactManager uses a plugin registered with `PolicySystemPlugin`. In ContactManager Studio, navigate in the **Resources** pane to **configuration** → **Plugins** → **gw** → **ClientSystemPlugin** → **PolicySystemPlugin**. You must register the plugin `gw.plugin.policy.pc700.PCPolicySystemPlugin`.

On the PolicyCenter side, calls from ContactManager are handled by methods of the `ContactAPI` web service, which implements the `ABCClientAPI` interface.

- To send changes to BillingCenter, ContactManager uses a plugin registered with `BillingSystemPlugin`. In ContactManager Studio, navigate in the `Resources` pane to `configuration → Plugins → gw → ClientSystemPlugin → BillingSystemPlugin`. You must register the plugin `gw.plugin.billing.bc700.BCBillingSystemPlugin`.

On the BillingCenter side, calls from ContactManager are handled by methods of the `ContactAPI` web service, which implements the `ABCClientAPI` interface.

## ContactManager 7.0 Geocoding Changes

The changes to geocoding include a new Bing Maps plugin and a new `BatchGeocode` property on the `Address` entity.

### MapPoint Now Bing Maps

The geocoding plugin registry now uses a Bing Maps plugin instead of a MapPoint plugin. Microsoft is canceling MapPoint support and moving all geocoding API support to Bing Maps. In addition to the steps required to enable geocoding, using Bing Maps requires that you set up an account for the Microsoft service and get an application key.

### New `BatchGeocode` Property to Indicate an Address Needs Geocoding

If an address's `GeocodeStatus` is set to `none`, there is no geocode information for the address. By default, all new contacts have their primary address's `GeocodeStatus` set to `none`. In the past, this value also indicated that the address needed to be geocoded.

Because of the addition of client contact support to ContactManager, this single property was no longer sufficient. It makes sense to geocode all vendor contact addresses, but not necessarily all addresses of client contacts. Therefore, a new property, `BatchGeocode`, was added to indicate if an address needed to be geocoded, apart from whether it had geocode information or not.

The following changes make it possible to explicitly set whether or not an address gets geocoded:

- The new `Address` property `BatchGeocode` indicates whether or not an address is to be geocoded. The default value is `false`, meaning that addresses by default are not geocoded.
- There is a new ContactManager rule that ensures that vendor contacts continue to be automatically geocoded. The rule—`Set Vendor's Primary Address BatchGeocode`—is a `Preupdate` rule in the `ABContactPreupdate` rule set. This rule sets the `BatchGeocode` property of the primary address of an `ABContact` to `true` if the contact is tagged as a vendor.
- There is a batch process that picks up addresses for which `BatchGeocode` is set to `true` and `GeocodeStatus` is set to `none`. This combination means that the address needs to be geocoded and has not been geocoded yet.

## Using ContactManager 7.0 Web Services with ClaimCenter 6

If you have installed ClaimCenter 6, PolicyCenter 7, and ContactManager 7, you must make the changes described in this topic to use ContactManager 7 with ClaimCenter 6. After you make the changes, ClaimCenter can use the ContactManager web service `IContactAPI` for vendor contacts. You also need to migrate your ContactCenter 6 data to ContactManager 7 so you can access it in the new version of ContactManager.

**Note:** To get the *ClaimCenter Upgrade Guide* for ClaimCenter 7, contact your service representative. It is included in the ClaimCenter 7 download files.

You must make the following changes in ClaimCenter 6 to make it compatible with ContactManager 7:

- “Step 1: Refresh ContactManager Web Services” on page 28
- “Step 2: Exclude the ContactManager Tags Field and Foreign Key” on page 29
- “Step 3: Enable Contact Auto Sync Between ClaimCenter 6 and ContactManager 7” on page 29

After setting up the two applications to communicate with each other, see the following topic concerning merging duplicate contacts and its effect on ClaimCenter:

- “Merging Contacts in ContactManager 7.0 and Linked Contacts in ClaimCenter 6” on page 30

### Step 1: Refresh ContactManager Web Services

The description of how to use ClaimCenter 6 with ContactCenter 6 in the version 6 *Contact Management Guide* is mostly accurate for integrating with ContactManager 7. There are additional requirements. The first one is that you refresh the ContactManager web service in ClaimCenter after setting ClaimCenter up to integrate with ContactManager. The following instructions walk you through setting up ClaimCenter to work with ContactManager and include the steps to refresh the ContactManager web service.

#### To integrate ClaimCenter with ContactManager

**1. Start ContactManager.**

At a command prompt, navigate to the ContactManager `bin` directory and enter the following command:

```
gwab dev-start
```

**2. Start ClaimCenter studio.**

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

```
gwcc studio
```

**3. In ClaimCenter Studio in the Resources pane on the left, expand configuration → Plugins.**

**4. 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 ContactManager.

**5. Click Remove.**

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

**6. After the tab page refreshes, click Add → Gosu to create a new plugin.**

**7. In the Class field enter the following value:**

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

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

Name	Value
password	gw
username	ClientAppCC

**9. In Studio in the Resources pane on the left, expand configuration → Web Services.**

**10. Click abintegration to open this web service in the editor.**

**11. 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
```

**12. If the port number is not correct, click Edit to the right of the URL field and change it.**

**13. Ensure that ContactManager is running, and then click Refresh to get the latest web service definitions from ContactManager.**

14. Save your changes.
15. You could continue with the changes in the next topic, or you could stop and restart ClaimCenter now. To restart ClaimCenter:
  - a. Stop ClaimCenter.  
Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:  
`gwcc dev-stop`
  - b. Start the ClaimCenter application and server:  
`gwcc dev-start`

### Step 2: Exclude the ContactManager Tags Field and Foreign Key

ContactManager 7 contacts have a `Tags` field that is not used in ClaimCenter 6. The field points to an array of `ABContactTag`, which has a foreign key back to the `ABContact` entity. You must exclude the contact field and the contact tag array's foreign key when transferring data from ContactManager to ClaimCenter. To exclude them, you add entries for them to the file `ab-to-cc-data-mapping.xml`.

#### To exclude the field and the foreign key

1. Start ClaimCenter Studio.  
At a command prompt, navigate to the `ClaimCenter/bin` directory and enter the following command:  
`gwcc studio`
2. In the Resources pane on the left, navigate to `configuration → Other Resources` and click `ab-to-cc-data-mapping.xml` to open it in an editor.
3. After the line `<EntityMapping source="ABContact" target="Contact">` add the following field mapping entry. If you see a message asking if you want to edit the file, click Yes.  

```
<FieldMapping source="Tags"
  mapperClassName="gw.api.util.mapping.NullFieldMapper"/>
```

This entry explicitly prevents the `ABContact` field `Tags` from being copied to the `Contact` in ClaimCenter.
4. Add the following entity mapping to the end of the file, just before the `</DataMapping>` tag:  

```
<EntityMapping source="ABContactTag" target="ContactTag">
  <FieldMapping source="ABContact"
    mapperClassName="gw.api.util.mapping.NullFieldMapper"/>
</EntityMapping>
```

This entry does the following:
  - Maps the ContactManager `ABContactTag` entity to an entity named `ContactTag` in ClaimCenter.
  - Prevents the `ABContactTag` entity's `ABContact` foreign key from being created in the `ContactTag` entity in ClaimCenter.
5. Save the file.
6. Restart the ClaimCenter application:
  - a. Stop ClaimCenter.  
Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:  
`gwcc dev-stop`
  - b. Start the ClaimCenter application:  
`gwcc dev-start`

### Step 3: Enable Contact Auto Sync Between ClaimCenter 6 and ContactManager 7

ClaimCenter 6 synchronizes contacts with ContactCenter 6 by default. To enable ClaimCenter 6 to synchronize its contacts with ContactManager 7, you must make the following changes in ContactManager 7 Studio.

1. Start ContactManager Studio.

At a command prompt, navigate to the ContactManager bin directory and enter the following command:

```
gwab studio
```

2. In the Resources pane on the left, navigate to configuration → Rule Sets → Preupdate → ABContactPreupdate.
3. Right-click Abcontact Pre-update and click New Rule. If you are prompted to create a copy of the rule set for editing, click Yes.
4. Enter the rule name VendorsSync and click OK.
5. For Rule Conditions, enter:  

```
aBContact typeis ABCompanyVendor or aBContact typeis ABPersonVendor
```
6. For Rule Actions enter:  

```
aBContact.InitiateAutoSync=true
```

**Note:** InitiateAutoSync will have a line through it, and this line of code will be marked with a yellow warning because InitiateAutoSync has been deprecated. You can still use it.
7. In the Resources pane on the left, navigate to configuration → Rule Sets → EventMessage → EventFired.
8. Click ContactAutoSync to enable it. If you see a message asking if you want to edit this rule, click Yes.
9. Navigate to configuration → RPC-Encoded Web Services → ccgwsoap and click ccgwsoap to edit it.
10. Ensure that the URL is set to the location for ClaimCenter 6. The default URL is <http://localhost:8080/cc-soap/IContactAutoSyncAPI?wsdl>.
11. Make sure that ClaimCenter 6 is running, and then click Refresh for the ccgwsoap web service. If you see a message asking if you want to open it for edit, click Yes.
12. Restart ContactManager, and then log in as an administrator. For example, user su with password gw.
13. Click the Administration tab, and click Event Messages. Ensure that the event message ABContact AutoSync Broadcast is Started.
14. Log in to ClaimCenter 6 as an administrator. For example, user su with password gw.
15. Click the Administration tab, and click Event Messages. Ensure that the event message Contact Auto Sync Failure is Started.

## Merging Contacts in ContactManager 7.0 and Linked Contacts in ClaimCenter 6

ContactManager has a Merge Contacts feature that enables a user to log in to ContactManager and resolve duplicate contacts into a single contact record.

When two duplicate contacts are merged in ContactManager, one contact is retired, and the other is kept. The kept contact becomes the contact of record. The retired contact's AddressBookUID can no longer be accessed by ClaimCenter 6. Therefore, it is possible that, as a result of merging contacts in ContactManager, the link between a contact in ClaimCenter 6 and a contact in ContactManager can become broken. ClaimCenter detects when a contact link becomes broken and shows an appropriate status message. A ClaimCenter user can then relink one of these contacts by clicking the Relink button for the contact.

When you relink a contact from ClaimCenter, ContactManager tries to find a matching contact. If ContactManager finds an exact match, it links the ClaimCenter contact to the matching ContactManager contact. If ContactManager is unable to find an exact match for the contact, it sends a list of potential matches that the ClaimCenter user can choose from. If ContactManager finds no potential matches, it will automatically add the

contact to ContactManager if the ClaimCenter user has the proper permissions. If the ClaimCenter user does not have the permissions required to add a contact to ContactManager, then ContactManager will reject the link.

**Note:** This behavior is not new for ClaimCenter. It has always been possible to delete a contact in ContactCenter, which would break any link to that contact in ClaimCenter. Relinking a contact also works the same as always in ClaimCenter 6. However, after integrating with ContactManager 7, ClaimCenter 6 linked contacts might become unlinked more often than they were previously because of the Merge Contacts feature.



# Managing Integrated Contacts

This topic provides an overview of managing and working with your contacts in Guidewire core applications.

This topic includes:

- “Client Data Management” on page 33
- “Vendor Data Management” on page 35

## Client Data Management

Client Data Management is available as an optional add-on module for Guidewire core applications that integrates and synchronizes client-related information across InsuranceSuite. Client Data Management supports a comprehensive view of the customer across core insurance processes through unifying the customer record. Some Client Data Management capabilities are supported by ContactManager.

In conjunction with Guidewire InsuranceSuite core systems, Client Data Management enables you to manage customer contact data across underwriting, policy administration, billing, and claims processes. If you license the Client Data Management add-on module, you can install ContactManager and integrate it with PolicyCenter and with other InsuranceSuite core applications. With the applications integrated, Client Data Management supports your customer and agent interactions and can enable you to maintain data integrity across different systems.

ContactManager is designed to be your system of record, the central repository for your customer data. This centralized repository stores a unique customer record and synchronizes updates across your policy, billing, and claims systems.

Various types of users, such as billing specialists, underwriters, and agents, can edit customer information in their respective applications without having to navigate to another system. You can add new clients through PolicyCenter, see current client information for policies in ClaimCenter, make updates directly in ClaimCenter, and obtain the latest client data in BillingCenter. To aid data cleanup, duplicate contact detection identifies potential duplicate contacts resulting from creating new contacts and from changing existing contacts, and enables you to merge or override these contacts.

ContactManager’s comprehensive customer view, available in PolicyCenter, enables service representatives to personalize interactions to match customers’ needs. The search and auto-fill features support updating of contact

records, and contact history tracks which updates have been made and provides visibility into previous customer interactions. Additionally, the PolicyCenter comprehensive view enables underwriters to find a customer's policy, billing, and claim information in a single place. Underwriters no longer have to search for this information in multiple systems. And, to ensure that the proper controls are in place, user permissions dictate who can update contact information at various points in the lifecycle.

## PolicyCenter Role in Managing Client Data

With the Client Data Management module, after integrating PolicyCenter with ContactManager, PolicyCenter can store client data in ContactManager. Typically, you create a new client when you open an account or create a policy, or when you add contacts to an account or policy. Additionally, in the PolicyCenter **Contact** tab, you can view information associated with a client, such as personal details like address, date of birth, and phone numbers. On the **Account** tab, you can view contact information such as all accounts, policies, and work orders associated with the client. You can also see claims and billing information if you use ClaimCenter and BillingCenter with PolicyCenter or if you integrate this feature with another claim system or billing system.

**Note:** While you can create a new client when you open an account or a new policy, the client data is not immediately stored in ContactManager. Contact data is stored in ContactManager only when you bind a policy or when you associate a contact with an account that has a bound policy. Additionally, contacts associated with reinsurance treaties or programs are also stored in ContactManager.

Using the **Contact** tab, you can create new contacts, search for existing contacts, select a recently viewed contact, and change contact information. PolicyCenter additionally enables you to work with contacts in its **Account** and **Policy** screens, where you can add, remove, and update contacts in various roles.

All these contacts are stored locally with the associated account or policy. Contacts that were originally in ContactManager and contacts in accounts with at least one bound policy are linked to a central record in ContactManager. If you make changes to a linked contact, PolicyCenter saves these changes locally and to the central contact record in ContactManager.

ContactManager also notifies PolicyCenter of any changes made to linked contacts outside PolicyCenter, such as a change originating in ClaimCenter when the client makes a claim. These notifications keep a contact's records synchronized across all accounts, policies, work orders, and so on.

## PolicyCenter Linked Addresses and Side Effects

PolicyCenter has a feature that enables you to share the same address with more than one contact. When you share an address across contacts in PolicyCenter, the contact addresses continue to be stored separately, but PolicyCenter links them. However, in the other Guidewire applications, contact addresses are not linked. ContactManager stores each contact address as a separate, unlinked address.

If you make a change to one of these addresses from ClaimCenter, BillingCenter, or ContactManager, there is a side effect in the base configurations. When ContactManager sends the changed contact address to PolicyCenter, PolicyCenter applies the change to the other addresses to which the original changed address is linked. PolicyCenter then sends these changed contact addresses back to ContactManager, and the result is that more than one contact gets an address change. There is no indication in ClaimCenter, BillingCenter, or ContactManager of this side effect—the linked addresses are changed automatically. You can configure the applications to behave differently.

If you change a linked address in PolicyCenter, you have the option of linking or unlinking it from other contacts' addresses. However, since this feature exists only in PolicyCenter, you do not have this option in the other Guidewire applications.

## ClaimCenter Role in Managing Client Data

ClaimCenter can receive client data from PolicyCenter when ClaimCenter requests policy information during claims processing. ClaimCenter also can receive client data updates from ContactManager when the information

for a client stored in ContactManager changes. For example, a client might notify a PolicyCenter customer service representative that their address has changed. The customer service representative updates the client's address, and PolicyCenter sends the change to ContactManager, which subsequently sends the change to ClaimCenter.

Additionally, a client might notify a ClaimCenter customer service representative that their information has changed, such as a change of phone number or address. The ClaimCenter customer service representative can change that information for the client in a claim, and then ClaimCenter can send the change to ContactManager. ContactManager then broadcasts the change to any Guidewire application that is integrated with ContactManager, such as PolicyCenter.

## BillingCenter Role in Managing Client Data

BillingCenter can receive client data from PolicyCenter when PolicyCenter sends billing information to BillingCenter. BillingCenter also can receive client data updates from ContactManager when the information for a client stored in ContactManager changes. For example, a client might notify a PolicyCenter customer service representative that their address has changed. The customer service representative updates the client's address, and PolicyCenter sends the change to ContactManager, which subsequently sends the change to BillingCenter.

Additionally, BillingCenter can send changes in client data to ContactManager. For example, a customer contacts a billing clerk to report a change of address.

# Vendor Data Management

Vendor data management is available only in ClaimCenter.

## Components of Vendor Data Management

This topic introduces vendor data management and contains the following topics:

- “Vendor Contacts” on page 35
- “Using and Configuring Vendor Data Management” on page 35

### Vendor Contacts

A vendor contact is a person or company that provides services for claims. In ClaimCenter, a vendor contact can be a person like a doctor or attorney. Additionally, a vendor contact can be a company, such as a repair shop, a bank, or a hospital. Additionally, a vendor 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 Vendor Data Management

Vendor data management is typically a set of tasks related to adding, modifying, searching for, and deleting vendor contacts in a contact management system, like ContactManager. 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 vendor contact or search for a contact, such as an attorney, in the claim creation wizard. You can also create and search for contacts in other areas of the ClaimCenter application where you need to specify a contact for a claim.

If ContactManager is integrated with ClaimCenter, you can define a set of services to be associated with vendor contacts. See “Vendor Services” on page 181.

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 in the correct format.

## ClaimCenter Integration with ContactManager

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

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

- “Centralized Vendor Data Management” on page 36
- “Which Contacts to Store in ContactManager” on page 36
- “Locally and Centrally Managed Contacts” on page 36
- “Deciding Whether to Use ContactManager for Vendor Management” on page 37

For detailed steps showing how to integrate ClaimCenter with ContactManager, see “Integrating ContactManager with Guidewire Core Applications” on page 45.

### Centralized Vendor Data Management

As the system of record for vendor contacts, ContactManager enables you to manage and serve vendor contact data centrally. ClaimCenter enables you to search for these centrally-maintained contacts on the **Address Book** tab and from claims. In claims, you can add contacts and edit the contact data as needed, and your changes can be saved in ContactManager.

After you integrate ContactManager and ClaimCenter, users searching for contacts have access not only to contacts stored locally in ClaimCenter, but also to contacts stored in ContactManager. For example, an adjuster working in ClaimCenter can search for an auto shop for a repair and access a list of approved service providers provided by ContactManager.

You can define different users for each Guidewire application. For example, you can have a group of ContactManager users whose primary function is to manage contact data and ensure that it is accurate. 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 ContactManager or ClaimCenter, the change appears in both applications.

### Which Contacts to Store in ContactManager

You need not configure ClaimCenter and ContactManager to store all company contacts in ContactManager. 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 ContactManager and locally in ClaimCenter.

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

Contacts that are specific to a claim are best managed locally in ClaimCenter. These contacts 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 might be 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 what kind of contact it is, the contact can be stored in two ways. ClaimCenter can store the contact locally only in ClaimCenter. Or, ClaimCenter can store the contact locally in ClaimCenter and centrally in ContactManager and link the contacts. If you create a new, non-vendor contact

directly on a claim, ClaimCenter can store it locally and not in ContactManager. This contact, associated only with the claim, is an *unlinked* contact.

Unlinked contacts are not associated with ContactManager. For an unlinked contact, ClaimCenter does not attempt to determine if a contact associated with one claim appears elsewhere on another claim. Thus, any unlinked contact can be a duplicate of one or more other unlinked contacts associated with different claims. When such duplicates exist, changes to one contact do not propagate to another because they are not related to one another.

Conversely, contacts stored in ContactManager, such as vendor contacts, do propagate changes when stored in ClaimCenter. In ClaimCenter, you can associate a single, centrally-managed contact with any number of claims. ClaimCenter makes a copy of each contact and stores it locally with the claim, and each copy is also linked to the ContactManager contact. Therefore, ClaimCenter can keep the data for all these copies of a contact in sync, even though there are multiple copies stored in ClaimCenter.

You can add a new contact and then **Link** the contact to make it centrally managed and available for use with other claims. If the new contact is a vendor, it is sent automatically to ContactManager and does not require any other action in ClaimCenter to become linked.

**Note:** The contact might require action in ContactManager, however. If ClaimCenter sends the contact as a pending create, the contact might require verification from a user in ContactManager, as described later.

For example, from a contact management perspective, a claimant might be a contact that other adjusters do not reuse. Therefore, a claimant might be an unlinked contact, stored locally with the claim but not in ContactManager. Even so, you could add a linked, centrally managed claimant to a claim. For example, you could click **Add an Existing Contact** and find the contact in ContactManager database and then add that contact to the claim.

To link a contact, ClaimCenter and ContactManager define a connection between a specific record in the ClaimCenter database and another record in the ContactManager database. In effect, linking the two contacts declares that though they are in two different databases, they are the same and are to show the same information. Users of ClaimCenter can link contacts from ClaimCenter to ContactManager. ClaimCenter can also automatically link a vendor contact to ContactManager. From ContactManager, users cannot link contacts to ClaimCenter.

After a contact is linked, the contact can be updated from either application, and both applications can get the update. ContactManager notifies ClaimCenter of changes to linked contacts. A linked contact in ClaimCenter whose data has changed in ContactManager is marked as not in sync. To keep contacts in ClaimCenter current, you can copy the data for a linked contact from ContactManager. Copying causes ClaimCenter to copy the latest information from ContactManager's central repository and mark the contact as in sync.

When you update a contact in ClaimCenter that is linked to ContactManager, ClaimCenter sends the change to ContactManager. For example, you can edit a linked contact on the **Contacts** screen of a claim and then update the contact. If you have permission to edit ContactManager contacts, ClaimCenter instructs ContactManager to save the change. If you do not have this permission, ClaimCenter instructs ContactManager to create a pending change, which then has to be reviewed by a ContactManager user.

You can unlink a contact. An unlinked contact in ClaimCenter becomes a claim-specific contact and is no longer centrally managed. However, even though the contact is no longer linked to ContactManager, a contact that you unlink continues to be in the ContactManager database.

## Deciding Whether to Use ContactManager for Vendor Management

In the base configuration, ClaimCenter has a registered address book plugin that is useful only for demonstration purposes. Guidewire also supplies a fully functional Gosu plugin implementation that works with ContactManager and is suitable for production. Do not go into a production environment with the demonstration plugin registered. If you decide not to use the integration with ContactManager 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 “Integrating with a Contact Management System” on page 501 in the *Integration Guide*.

You are not required to use either ContactManager 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 its address book functionality changes as follows:

- The **Address Book** tab is present, but attempts to use it for searching result in an error.
- The claim **Parties Involved** interface has reduced functionality. You are unable to view the Address Book from the screen or copy a contact or add an existing contact from ContactManager.

The **Address Book** tab, buttons, and other related options are always present in the base application. If you want to remove them, you must edit the related PCF files.

**Note:** Even without having a contact management application integrated, you can use the **Search** tab to search for local contacts. However, without this integration, you cannot check for duplicate contacts.

#### See also

- “Client Data Management” on page 33
- “Integrating ContactManager with Guidewire Core Applications” on page 45

# Installing ContactManager

This topic describes how to install ContactManager in a development environment. To use this topic, you must be familiar with the installation process defined in the *ClaimCenter Installation Guide*.

You can install ContactManager for development by using the Guidewire QuickStart installation setup, or you can use any supported application and database server. Use QuickStart only in a demonstration or development environment.

ContactManager must start with its own application and database server. Do not run ContactManager in the same server as a core application.

For production, you need a dedicated Java Virtual Machine (JVM) for each Guidewire application. Running a Guidewire core application and ContactManager in the same JVM can result in memory conflicts and other problems. If you do not use a QuickStart setup, install the products in separate JVMs as described later in “Enhancing Performance” on page 42.

For production, you can install ContactManager with any supported combination of application server and database server. For more information, see the *Guidewire Platform Support Matrix* on the Guidewire Resource Portal at <http://guidewire.custhelp.com>.

This topic includes:

- “Installing ContactManager with QuickStart for Development” on page 39
- “Installing ContactManager with Tomcat and SQL Server for Development” on page 41

## Installing ContactManager with QuickStart for Development

A QuickStart installation enables you to immediately use and test the product. This topic provides only the information necessary to get ContactManager working in the QuickStart environment.

For more information on this environment, see “Installing the QuickStart Development Environment” on page 46 in the *Installation Guide*. For example:

- For information on setting the port number for the Jetty server used in this environment, see “Configuring QuickStart Ports” on page 48 in the *Installation Guide*.

- For information on configuring the H2 Database Engine used in this environment, see “Using the QuickStart Database” on page 51 in the *Installation Guide*.
- For information on using SQL Server or Oracle in the QuickStart environment, see “Using SQL Server or Oracle in a Development Environment” on page 52 in the *Installation Guide*.

## Step 1: Install ContactManager

These instructions assume that you have installed your Guidewire core application as described in “Installing the QuickStart Development Environment” on page 46 in the *Installation Guide*.

### To install ContactManager

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

## Step 2: Load Sample Data

Each installation contains sample data you can use to learn application functionality. This data can be useful for learning purposes and for completing integration examples in this book, but it is not intended to be used in a production system. To import the data:

1. If you have not already done so, follow the instructions in the previous topic to start ContactManager and log in as the superuser.
2. Press **Alt+Shift+T**.
3. Click the **Internal Tools** tab.
4. Click **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:
  - a. On the Options menu , click **Return to ContactManager**.
  - b. Click the **Administration** tab.
  - c. 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 log in to ContactManager as a user with permission to add, edit, and delete contacts:

- a. On the Options menu , click Log Out.
- b. Log in with user name aapplegate and password gw.

## Installing ContactManager with Tomcat and SQL Server for Development

This topic provides some general guidelines for installing and configuring ContactManager in a development environment in which you are using your own application and database server. Although the QuickStart Jetty application server is suitable for most development needs, some organizations prefer to use Tomcat in their development environment.

For production, you can install ContactManager with any supported combination of application server and database server. For more information, see the *Guidewire Platform Support Matrix* on the Guidewire Resource Portal at <http://guidewire.custhelp.com>.

This topic describes some aspects of the servers, database, and versions of Java that are required to support your Guidewire installation. For a full description that applies to the core application, and also applies to ContactManager, see “Installation Environments Overview” on page 15 in the *Installation Guide*.

**Note:** Guidewire provides testing APIs in JAR files with names ending in `-gunit.jar` for use during configuration and development. These APIs are available only when the application is running in development mode or from Guidewire Studio. When WAR or EAR files are built, the testing API JAR files are excluded. If your deployed code makes calls to any testing APIs, these calls cause `ClassNotFoundException`s and other problems and prevent the application from running properly.

See also “Using Multiple ClaimCenter Development Instances” on page 46 in the *Installation Guide*.

This topic includes:

- “Tomcat Development Environment Prerequisites” on page 41
- “Enhancing Performance” on page 42
- “Considerations for Running on an Oracle Database” on page 42
- “Installing in a Development Environment with Tomcat and SQL Server” on page 43

### Tomcat Development Environment Prerequisites

- Your environment must meet the minimum requirements for a development workstation. See the *Guidewire Platform Support Matrix* for current system and patch level requirements. The *Guidewire Platform Support Matrix* is available from the Guidewire Resource Portal at <https://guidewire.custhelp.com/app/resources/products/platform>. Also see “Development Workstation Information” on page 41 in the *Installation Guide*.
- Do not include spaces in the path to your Tomcat installation.
- ContactManager requires Java and Ant. Guidewire recommends that you also install the Dynamic Code Evolution Virtual Machine (DCE VM) on development environments. If you do not install the DCE VM, you will not be able to see dynamic changes to Gosu code.

For specific required versions and installation guidelines, see:

- “Installing Java” on page 41 in the *Installation Guide*
- “Installing Ant” on page 43 in the *Installation Guide*

## Enhancing Performance

This topic provides specific information for configuring and using ContactManager with Tomcat and SQL Server in a development environment.

See the following topics for important information on configuring Tomcat and SQL Server with a Guidewire application:

- “Configuring Tomcat” on page 22 in the *Installation Guide*
- “Configuring SQL Server for ClaimCenter” on page 37 in the *Installation Guide*

To enhance performance, configure ContactManager to run in its own Java Virtual Machine (JVM). See “Installing Java” on page 41 in the *Installation Guide*.

Set the `XXMaxPermSize` value on the ContactManager application server instance to at least 256MB for a 64-bit JVM. For more information, see “JVM Heap Size Considerations” on page 19 in the *Installation Guide*.

Install ContactManager and each Guidewire core application with which it integrates in separate Tomcat instances with separate installation directories. You can install the Tomcat binaries once and share them between all installation directories by using Tomcat’s `CATALINA_BASE` system property to support this type of behavior. Each Tomcat instance must be running on its own unique port.

**Note:** If you are using the Oracle JVM, add the `-server` flag to `CATALINA_OPTS`.

Use separate database instances for ContactManager and each Guidewire core application, especially in larger installations. Also, ensure that the Guidewire core applications and ContactManager run on different ports. The default port number for ContactManager is 8280. The Guidewire core applications by default all have port numbers that are different from one another and from ContactManager.

---

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

---

## Considerations for Running on an Oracle Database

If you are running ContactManager on an Oracle database, you must create additional tablespaces in your existing database. As with the Guidewire core applications, ContactManager 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 ContactManager data tables
TYPELIST	Stores system code tables
INDEX	Stores system indexes
STAGING	Stores inbound staging data tables

Because ContactManager has its own set of objects, you must configure database user names that are different from those used in the Guidewire core applications. For example, in the base product, the `jdbcURL` parameter in the ContactManager `database-config.xml` file sets a different database user from the `jdbcURL` parameters in the Guidewire core applications’ `database-config.xml` files.

There are additional considerations for running a Guidewire application with an Oracle database. See “Configuring Oracle for ClaimCenter” on page 32 in the *Installation Guide*.

## Installing in a Development Environment with Tomcat and SQL Server

This topic describes how to install ContactManager in a development environment in which you have installed your Guidewire core application with your own application server and database server. 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 for ContactManager. The *Installation Guide* has information on installing with other servers and databases that you can apply to a ContactManager installation.

This topic includes:

- “Step1: Install and Set Up ContactManager, Tomcat, and SQL Server” on page 43
- “Step 2: Test the ContactManager Development Environment on Tomcat” on page 44
- “Step 3: Load Sample Data” on page 44

### Step1: Install and Set Up ContactManager, Tomcat, and SQL Server

These directions assume that you have installed your Guidewire core application in a development environment. See “Installing a ClaimCenter Development Environment” on page 45 in the *Installation Guide*.

1. Install and unpack the contents of `ContactManager.zip` into a directory on your system. These steps use `ContactManager` as the directory name.
2. Set up the Java Virtual Machine (JVM) as described in “Installing Java” on page 41 in the *Installation Guide*.
3. Set up ANT as described in “Installing Ant” on page 43 in the *Installation Guide*.
4. Set up Apache Tomcat as described under “Configuring Tomcat” on page 22 in the *Installation Guide*.
5. Build a ContactManager environment on Tomcat by using the instructions under “Building a ClaimCenter Development Environment with Tomcat” on page 49 in the *Installation Guide*. In that topic, there are references to `cc` and `ClaimCenter` that you need to translate to `ab` and `ContactManager`. For example, the following steps use `ab` and `ContactManager`:
  - a. Edit the Tomcat `conf/server.xml` file.
  - b. Find the definition for the HTTP connector.
  - c. Set the connector port to 8280:

```
<Connector port="8280" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"/>
```
  - d. Create your ContactManager SQL Server database.

Use the same options for the ContactManager database as those you set for your Guidewire core application. For information about how to set up your database, see “Configuring SQL Server for ClaimCenter” on page 37 in the *Installation Guide*.

- e. In ContactManager Studio, navigate in the Project window to `configuration` → `config` and double-click `database-config.xml` to open it in the editor.
- f. Find the SQL Server section and modify it to reference your new SQL Server database. For more information, see “Configuring a Database Connection” on page 62 in the *Installation Guide*.
- g. Continue with this procedure after you have created the database account and configured ContactManager to connect to the database.
- h. At a command prompt, navigate to the `ContactManager/bin` directory and run the following command:  
`gwab build-tomcat-war-dbcp`

This command creates the `ab.war` file including JDBC drivers in the `ContactManager/dist/war` directory.

- i. Deploy the package to Tomcat by copying the ab.war file to the webapps directory in your Tomcat server.
- j. Use the Tomcat bin/startup.bat command to start Tomcat and expand the WAR file.

When Tomcat starts up, it automatically recognizes this new application and unpacks the ab.war file into a directory structure in webapps, such as 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.

- k. Delete the Tomcat webapps/ab/modules/configuration directory.
- l. Create a symbolic link from the deployed ContactManager application on Tomcat to the ContactManager/modules directory of the original ContactManager installation location.

**Windows 7 and Vista:** Use the mklink command to create the link, as in the example below:

```
mklink /d C:\apache-tomcat-7.0.39\webapps\ab\modules\configuration  
C:\ContactManager\modules\configuration
```

**Windows XP:** Use the Windows SysInternals program Junction.exe to create the symbolic link. This program is available at:

<http://technet.microsoft.com/en-us/sysinternals/bb896768.aspx>

- m. Restart the Tomcat server.

## Step 2: Test the ContactManager Development Environment on Tomcat

1. If necessary, start Tomcat. For example:

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

2. Open a browser and navigate to <http://localhost:8280/ab>.

The login screen includes a **User name** field. You change the label for the **User name** field in this procedure.

3. At a command prompt, navigate to ContactManager/bin and start ContactManager Studio by running the following command:

```
gwab studio
```

4. In the Project window, navigate to configuration → config → Localizations → en\_US and double-click display.properties to open this file in the editor.

5. In the editor, find Web.Login.Username.

6. Change the value of the Web.Login.Username display key in an obvious manner. For example, change it to Enter your user name.

7. Log in to ContactManager with username su and password gw. At this point the **User name** field is unchanged.

8. Press Alt+Shift+T to open the Server Tools screen.

9. Click the Internal Tools tab.

10. In the sidebar, click Reload.

11. Click Reload PCF Files.

12. On the Options menu , click Log Out Super User. ContactManager redirects you to the login screen.

13. Ensure that the label for **User name** field changed to your setting. If you properly set up your development environment, the label for the field now shows the new name you entered for the Web.Login.Username display key.

## Step 3: Load Sample Data

The steps for loading sample data are the same as those described under “Step 2: Load Sample Data” on page 40.

# Integrating ContactManager with Guidewire Core Applications

This topic describes how to integrate ContactManager with the Guidewire core applications ClaimCenter, PolicyCenter, and BillingCenter.

As described in “ContactManager Integration Reference” on page 269, the Guidewire core applications communicate with ContactManager through SOAP APIs. The applications work with ContactManager through its published web service ABContactAPI.

Each Guidewire core application defines a plugin interface for invoking the integration layer to communicate with an external contact management system. In each application, there is also a plugin implementation of this interface—a class that implements this interface—that communicates with ContactManager. The plugin interface and the plugin implementation are different for each Guidewire core application.

You configure the integration between ContactManager and the Guidewire core applications in Guidewire Studio.

This topic includes:

- “Integrating ContactManager Using QuickStart” on page 46
- “Integrating ContactManager Using Tomcat and SQL Server” on page 65
- “Configuring Core Application Authentication with ContactManager” on page 69
- “Configuring ContactManager Authentication with Core Applications” on page 73

This topic discusses only integrating Guidewire core applications with ContactManager.

- For information on integrating Guidewire core applications with third-party applications, see the “Integration Overview” on page 15 in the *Integration Guide*.
- For information on integrating ContactManager with third-party applications, see “ContactManager Integration Reference” on page 269.

# Integrating ContactManager Using QuickStart

A QuickStart installation enables you to quickly use and test the product. This topic covers integrating ContactManager with Guidewire core applications in a QuickStart environment.

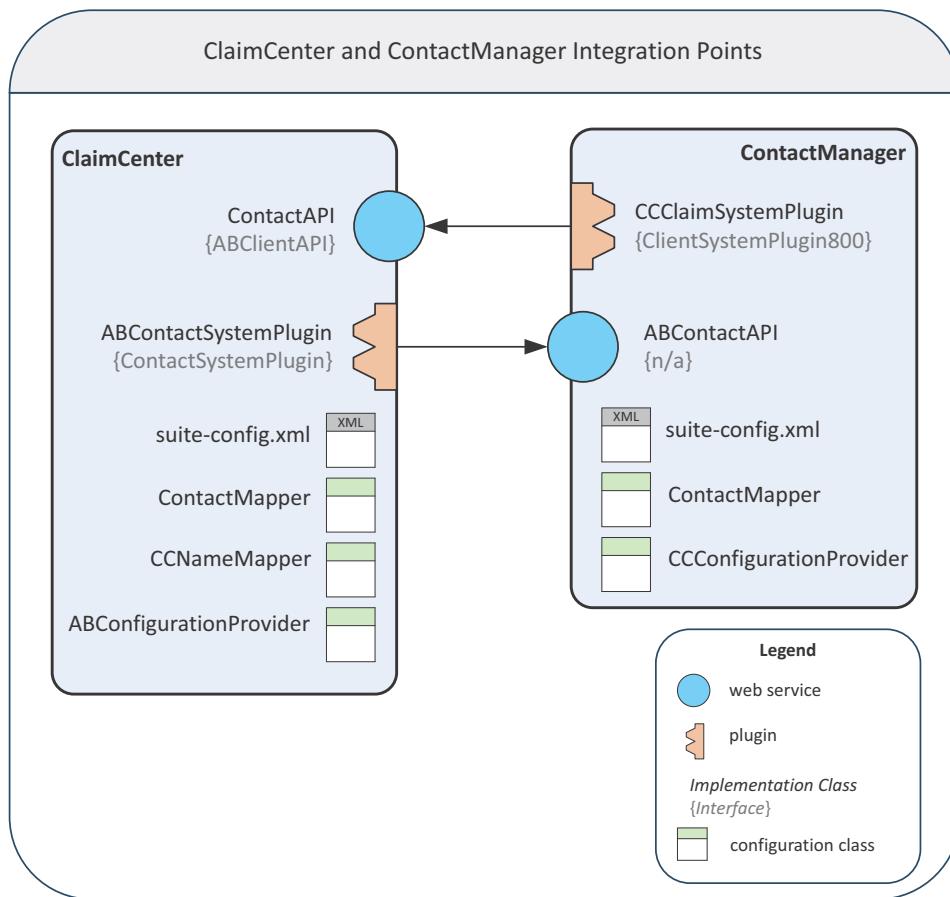
Before continuing with this topic, you must have installed ContactManager. For more information, see “Installing ContactManager with QuickStart for Development” on page 39.

This topic covers the following topics:

- “Integrating ContactManager with ClaimCenter in QuickStart” on page 46
- “Integrating ContactManager with PolicyCenter in QuickStart” on page 52
- “Integrating ContactManager with BillingCenter in QuickStart” on page 59

## Integrating ContactManager with ClaimCenter in QuickStart

The following figure shows the primary integration points used to integrate ClaimCenter and ContactManager.



To get the basic integration working, you edit the `suite-config.xml` files in ClaimCenter and ContactManager. You also register the `CCClaimSystemPlugin` plugin implementation in the ContactManager `ClaimSystemPlugin` registry. Additionally, you register the `ABContactSystemPlugin` in the ClaimCenter `ContactSystemPlugin` registry. All these steps are described in the topics that follow.

If you extend the contact model, you edit the `ContactMapper` classes in both `ContactManager` and `ClaimCenter`, and possibly the `CCNameMapper` class. When you change the authentication user and password, you edit the `ABConfigurationProvider` and `CCConfigurationProvider` classes.

This topic includes:

- “Step 1: Integrate `ContactManager` with `ClaimCenter`” on page 47
- “Step 2: Test the Integration” on page 50
- “Troubleshooting the `ClaimCenter` Connection with `ContactManager`” on page 51
- “Integrating `ClaimCenter` 7 with `ContactManager` 8” on page 51

#### See also

- “`ABClientAPI` Interface” on page 282
- “`ABContactAPI` Web Service” on page 275
- “`ContactMapper` Class” on page 285
- “Configuring `ClaimCenter`-to-`ContactManager` Authentication” on page 69
- “Configuring `ContactManager`-to-`ClaimCenter` Authentication” on page 74
- “Extending the Vendor Contact Data Model” on page 142

### [Step 1: Integrate `ContactManager` with `ClaimCenter`](#)

After installing `ContactManager`, you can use both applications separately, but they are not integrated. To integrate the two applications you must configure services in Guidewire Studio. This topic starts with the changes you make in `ClaimCenter` Studio and then continues with the changes you make in `ContactManager` 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 `ClaimCenter` Studio, open the `Project` window.

3. Press `Ctrl+Shift+N` and enter `suite-config.xml` to find this file, and then double-click the search result to open the file in the editor.

This file defines the URLs of the applications in the Guidewire InsuranceSuite. If the entries in this file are commented out, you must remove the comments for the applications you have installed. In the default configuration, the Guidewire applications have the following settings in this file:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
  <!--<product name="ab" url="http://localhost:8280/ab"/>-->
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

4. Remove the comments around the `ContactManager` (`ab`) entry, as follows:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
  <product name="ab" url="http://localhost:8280/ab"/>
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

5. Save your changes.

6. In the `Project` window, expand `configuration` → `config` → `Plugins` → `registry`.

7. Double-click `ContactSystemPlugin.gwp` to open it in the Registry editor.

**Note:** By default, the system uses the plugin implementation

`gw.plugin.addressbook.demo.DemoContactSystemPlugin`. This adapter is a Java plugin implementation provided only for product demonstrations, and it is not to be used for any other purpose. In the steps that follow, you replace this demo class with the plugin implementation that connects ClaimCenter with ContactManager.

8. In the Registry editor, click Remove Plugin .

9. Click Add Plugin  and, in the drop-down menu, choose **Add Gosu Plugin** to register the new plugin implementation.

10. In the **Gosu Class** field, enter the following class:

```
gw.plugin.contact.ab800.ABContactSystemPlugin
```

11. If necessary, start ContactManager.

At a command prompt, navigate to the `ContactManager/bin` directory and enter the following command:  
`gwab dev-start`

12. In the ClaimCenter Studio Project window, navigate to **configuration** → **gsrc** and then to `wsi.remote.gw.webservice.ab.ab801.wsc`.

**Note:** The following steps describe specific settings in the editor for web services collections. For general information, see “Using the WS-I Web Service Editor” on page 122 in the *Configuration Guide*.

13. Double-click the `ab801.wsc` web services collection to open the editor, and then, in the editor, click the following resource:

```
 ${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl
```

The  `${ab}` variable in this path corresponds to the `product name="ab"` entry in the `suite-config.xml` file that specifies the URL for ContactManager.

14. With ContactManager running, click **Fetch Updates** to get the latest updates to ABContactAPI.

15. In the same editor on the **Settings** tab, there is a setting for the Configuration Provider

`wsi.remote.gw.webservice.ab.ABConfigurationProvider`. This class defines the user name and password that ClaimCenter uses to authenticate with ContactManager when ClaimCenter makes calls to ABContactAPI. In the base configuration, `ABConfigurationProvider` defines these two settings as follows:

```
config.Guidewire.Authentication.Username = "ClientAppCC"  
config.Guidewire.Authentication.Password = "gw"
```

**Note:** You must load sample data into ContactManager for this default user name to work. This user name and password can be useful for development. However, for security reasons, you need to change this default user name and password before going to production. For more information, see “Configuring Core Application Authentication with ContactManager” on page 69.

16. Save your changes and close ClaimCenter Studio.

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

```
gwab studio
```

18. In ContactManager Studio, press **Ctrl+Shift+N** and enter `suite-config.xml` to find the suite configuration file. Then double-click the search result to open the file in the editor.

This file defines the URLs of the applications in the Guidewire InsuranceSuite. If the entries in this file are commented out, you must remove the comments for the applications you have installed. In the default configuration, the Guidewire applications have the following settings in this file:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">  
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->  
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
```

```
<!--<product name="ab" url="http://localhost:8280/ab"/>-->
<!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

- 19.** Remove the comments around the ClaimCenter (cc) entry, as follows:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <product name="cc" url="http://localhost:8080/cc"/>
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
  <!--<product name="ab" url="http://localhost:8280/ab"/>-->
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

- 20.** Save your changes.

- 21.** Open the Project window and expand **configuration** → **config** → **Plugins** → **registry**.

- 22.** Double-click **ClaimSystemPlugin.gwp** to open it in the Registry editor.

You need to register a plugin implementation so ContactManager can use it to synchronize address book changes with ClaimCenter.

**Note:** By default, the system uses `gw.plugin.integration.StandAloneClientSystemPlugin`. This adapter is a plugin implementation that does not broadcast Contact changes and does not communicate with core applications.

- 23.** In the Registry editor, click **Remove Plugin** —

- 24.** Click **Add Plugin** + and, in the drop-down menu, choose **Add Gosu Plugin** to register the new plugin implementation.

- 25.** In the **Gosu Class** field, enter the following class:

```
gw.plugin.claim.cc800.CCClaimSystemPlugin
```

- 26.** Navigate to **configuration** → **gsrc** and then to **wsi.remote.gw.webservice.cc.cc800.wsc**.

- 27.** Double-click the **cc800.wsc** web services collection to open the editor, and then, in the editor, click the following resource:

```
 ${cc}/ws/gw/webservice/cc/cc800/contact/ContactAPI?wsdl
```

ContactManager calls this ClaimCenter webservice when it has Contact updates for ClaimCenter. The ClaimCenter web service `ContactAPI` implements the interface `ABCClientAPI`. The  `${cc}` variable in this path corresponds to the `product name="cc"` entry in the `suite-config.xml` file, which specifies the URL for ClaimCenter.

- 28.** With the ClaimCenter server running, click **Fetch Updates** to get the latest updates to `ContactAPI`.

- 29.** In the same editor on the **Settings** tab is the configuration provider class. This class,

`wsi.remote.gw.webservice.cc.CCConfigurationProvider`, defines the user name and password that ContactManager must use to connect with ClaimCenter. By default, ContactManager uses the user name `su` and password `gw`. Guidewire recommends that you change this user name as described at “Configuring ContactManager-to-ClaimCenter Authentication” on page 74.

- 30.** Close ContactManager studio.

- 31.** To pick up your changes, stop and restart the two servers.

- a.** Stop ClaimCenter.

Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:  
`gwcc dev-stop`

- b.** Stop ContactManager.

Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`

- c.** Start the ContactManager application:

```
gwab dev-start
```

d. Start the ClaimCenter application:

```
gwcc dev-start
```

## Step 2: Test the Integration

Ensure that both ContactManager and ClaimCenter have started. 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/`
2. Log in as a user who has permissions to view, create, edit, and search for a ContactManager contact in ClaimCenter. For example, log in as the sample user `ssmith` with password `gw`.
3. Open an existing claim and then click **Parties Involved** to open the **Contacts** screen.
4. Click **New Contact** and create a new contact.
5. Give the contact enough data to be able to save it in ContactManager, such as name, address, and tax ID information.
6. Give the contact a role on the claim and click **Update** to save it.
7. When the **Contacts** screen opens, select the new contact. Because you are logged in with a role that permits writing to ContactManager, the contact links automatically to ContactManager. If the link is successful, you see the message, **This contact is linked to the Address Book and is in sync**.
8. Click the **Address Book** tab to search for a contact. With sample data loaded in ContactManager, you can search for a Vendor (Company) with a name that starts with the letter a. That search returns contacts with names like AB Construction and Allendale, Myers & Associates.
9. Open a browser window and enter the following ContactManager URL:  
`http://localhost:8280/ab/`
10. Log in as a user who can view or edit a contact in ContactManager, such as the sample user `aapplegate` with password `gw`.
11. In ContactManager, click **Search** and verify that you can search for and locate the contact you just created in ClaimCenter.
12. Click the contact in the search results and edit it. For example, change the contact's phone number. Click **Update** to save the changes.
13. In ClaimCenter, open the claim to which you added the new contact, and click **Parties Involved** to open the **Contacts** screen.
14. Select the contact that you added. On the Basics card, if you have not changed default settings, you see the following message:  
**This contact is linked to the Address Book and is in sync.**
15. Click **View in Address book**.  
ClaimCenter fetches the contact data from ContactManager and displays it in a new screen.  
**Note:** In this screen, you can click **Edit in ContactManager** to open a browser window and connect to the ContactManager server. You can then log in and edit the contact data directly. You must have a ContactManager user name to be able to use this feature. Additionally, that user must have a role in ContactManager that enables editing contacts, such as the Contact Manager role.
16. Click **Return to Contacts** in the top right of the screen.  
You see the Contacts screen again, with your contact still selected.

## Troubleshooting the ClaimCenter Connection with ContactManager

### Bad Username or Password Error

You might see an error message saying that there was an exception caused by a bad user name or password. If so, you probably need to create the ContactManager user that ClaimCenter uses to communicate with ContactManager. In the base configuration, this user is ClientAppCC with password gw. This user is available if you import the sample data for ContactManager. To create a user in ContactManager, see “Configuring ContactManager Authentication with Core Applications” on page 73.

### See also

- “Configuring Core Application Authentication with ContactManager” on page 69.
- “Step 2: Load Sample Data” on page 40 for instructions on how to load sample data for ContactManager

### No Response from ContactManager

If you are not able to get any response at all from ContactManager, it is possible that the message queue that ClaimCenter uses to communicate with ContactManager is suspended.

### To check the ClaimCenter message queue and reactivate it

1. If they are not already running, start ClaimCenter and ContactManager.
2. Log in to ClaimCenter as a user with administrative privileges. For example log in with user name su and password gw.
3. Click the Administration tab, and then click **Monitoring → Message Queues** in the sidebar.
4. On the **Message Queues** screen, click the check box next to the **Contact Message Transport** entry.
5. If the **Status** is **Suspended**, click the **Resume** button at the top of the screen.
6. The **Status** changes to **Started**, and ClaimCenter can now send messages to ContactManager.

## Integrating ClaimCenter 7 with ContactManager 8

If your installation of ClaimCenter is version 7.0.x, you might have ContactManager 8.0.x installed if you have PolicyCenter 8.0.x and a license for Client Data Management. In that case, Guidewire recommends that you upgrade ContactManager 7.0.x to ContactManager 8.0.x and integrate it with ClaimCenter 7.0.x.

**Note:** Compatibility does not go in the other direction. ClaimCenter 8.0 supports only ContactManager 8.0. The 8.0 version of ClaimCenter does not support ContactManager 7.0.

On the ClaimCenter side, the integration between ClaimCenter 7.0.x and ContactManager 8.0.x uses the same plugins, classes, and XML configuration files as with ContactManager 7.0.x. See the *ClaimCenter 7.0.x Contact Management Guide*.

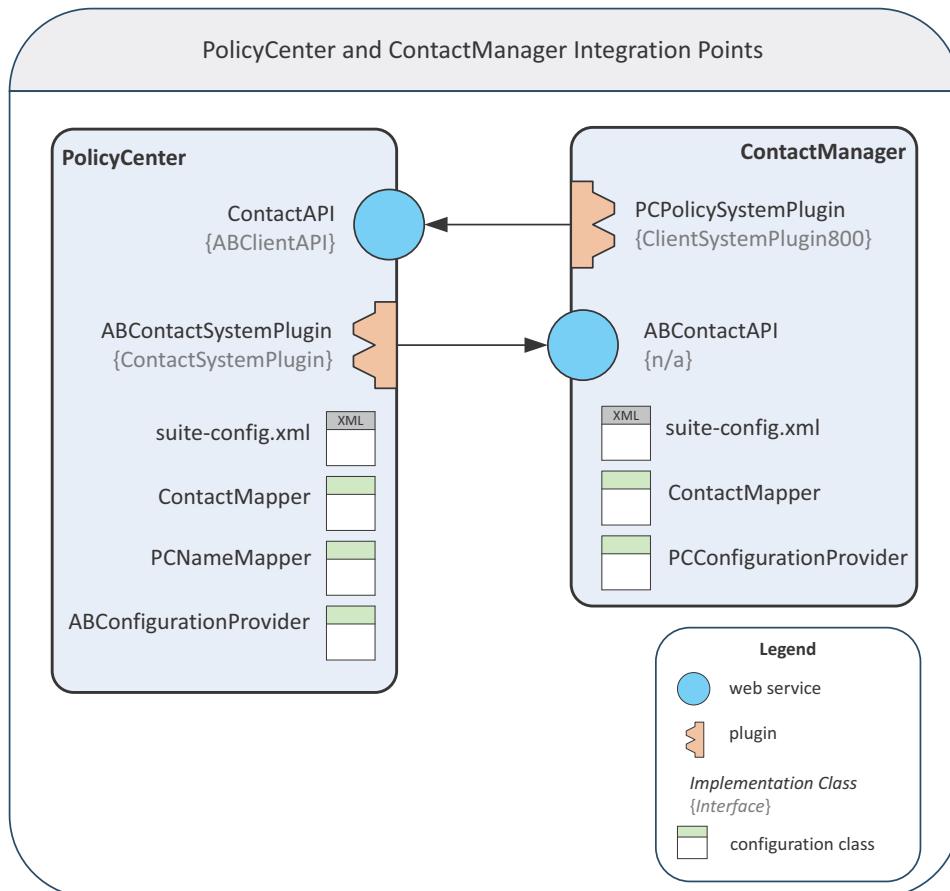
To support integration with ClaimCenter 7.0.x, ContactManager 8.0.x provides the following plugins, web services, classes, and XML configuration files:

- `ClaimSystemPlugin.gwp` – Registry for plugin that enables communication with ClaimCenter.
- `gw.plugin.claim.cc700.CCClaimSystemPlugin` – Plugin implementation you register in `ClaimSystemPlugin.gwp` to communicate with ClaimCenter 7.0.x.
- `gw.webservice.ab.ab700.abcontactapi.ABContactAPI` – ContactManager web service supporting SOAP calls from ClaimCenter 7.0.x.
- `gw.webservice.contactapi.ab700.ABClientAPI` – ContactManager interface used by ClaimCenter 7.0.x to implement the ClaimCenter web service `gw.webservice.cc.cc700.contact.ContactAPI`.
- `gw.contactmapper.ab700.ContactIntegrationXMLMapper.gs` – ContactManager XML mapping file for use with ClaimCenter 7.0.x.

- `gw.webservice.ab.ab700.reviewsummary.IReviewSummaryAPI.gs` – An RPC-E web service available to ClaimCenter 7.0.x for creating and deleting review summaries corresponding to vendor service provider reviews in ClaimCenter.
- `gw.webservice.ab.ab700.abcontactapi.ABContactSearchCriteriaInfo.gs` – A class that specifies the contact search criteria that the Guidewire core applications can use.
- `gw.webservice.ab.ab700.abcontactapi.ABContactSearchResult.gs` – A class that specifies the fields included in the contact search results that ContactManager sends back to the Guidewire core application.
- `gw.webservice.ab.ab700.abcontactapi.ABContactAPISearchSpecInfo.gs` – Provides the search criteria used by the web service method `ABContactAPI.findDuplicates`.
- `gw.webservice.ab.ab700.abcontactapi.AddressSearchInfo.gs` – Used by `ABContactSearchCriteriaInfo` to declare the `Address` and `ProximitySearchCenter` variables.
- `gw.webservice.ab.ab700.abcontactapi.ProximitySearchParametersInfo.gs` – Used by `ABContactSearchCriteriaInfo` to declare the `ProximitySearchParameters` variable.

## Integrating ContactManager with PolicyCenter in QuickStart

The following figure shows the primary integration points used to integrate PolicyCenter and ContactManager.



To get the basic integration working, you edit the `suite-config.xml` files in PolicyCenter and ContactManager. You also register the `PCPolicySystemPlugin` plugin implementation in the ContactManager `PolicySystemPlugin` registry. Additionally, you register the `ABContactSystemPlugin` in the PolicyCenter

ContactSystemPlugin registry. The detailed steps to integrate PolicyCenter and ContactManager are described in the topics that follow.

If you extend the contact model, you edit the ContactMapper classes in both ContactManager and PolicyCenter, and possibly the PCNameMapper class. When you change the authentication user and password, you edit the ABConfigurationProvider and PCConfigurationProvider classes.

This topic includes:

- “Step 1: Integrate ContactManager with PolicyCenter” on page 53
- “Step 2: Test the Integration” on page 56
- “Troubleshooting the PolicyCenter Connection with ContactManager” on page 57
- “Integrating PolicyCenter 7 with ContactManager 8” on page 58

#### See also

- “ABClientAPI Interface” on page 282
- “ABContactAPI Web Service” on page 275
- “ContactMapper Class” on page 285
- “Configuring PolicyCenter-to-ContactManager Authentication” on page 71
- “Configuring ContactManager-to-PolicyCenter Authentication” on page 76
- “Extending the Client Data Model” on page 141

### Step 1: Integrate ContactManager with PolicyCenter

After installing ContactManager, you can use both applications separately, but they are not integrated. To integrate the two applications you must configure services in Guidewire Studio. This topic starts with the changes you make in PolicyCenter Studio and then continues with the changes you make in ContactManager Studio.

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

```
gwpc studio
```

2. In PolicyCenter Studio, open the Project window.

3. Press Ctrl+Shift+N and enter suite-config.xml to find this file, and then double-click the search result to open the file in the editor.

This file defines the URLs of the applications in the Guidewire InsuranceSuite. If the entries in this file are commented out, you must remove the comments for the applications you have installed. In the default configuration, the Guidewire applications have the following settings in this file:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
  <!--<product name="ab" url="http://localhost:8280/ab"/>-->
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

4. Remove the comments around the ContactManager (ab) entry, as follows:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
  <product name="ab" url="http://localhost:8280/ab"/>
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

5. Save your changes.

6. In the Project window, expand configuration → config → Plugins → registry.

7. Double-click **ContactSystemPlugin** to change the plugin implementation used by the system to connect to ContactManager.

**Note:** By default, the system uses `gw.plugin.contact.impl.StandAloneContactSystemPlugin`. This adapter is a Gosu plugin that PolicyCenter uses when not connected to a contact management system. In the steps that follow, you replace this class with the plugin implementation that connects BillingCenter with ContactManager.

8. In the Registry editor, click Remove Plugin .

9. Click Add Plugin  and, in the drop-down menu, choose **Add Gosu Plugin** to register the new plugin implementation.

10. In the **Gosu Class** field, enter the following class:

```
gw.plugin.contact.ab800.ABContactSystemPlugin
```

11. If necessary, start ContactManager.

At a command prompt, navigate to the `ContactManager/bin` directory and enter the following command:  
`gwab dev-start`

12. In the PolicyCenter Studio Project window, navigate to **configuration** → **gsrc** and then to `wsi.remote.gw.webservice.ab.ab801.wsc`.

**Note:** The following steps describe specific settings in the editor for web services collections. For general information, see “Using the WS-I Web Service Editor” on page 122 in the *Configuration Guide*.

13. Double-click the `ab801.wsc` web services collection to open the editor, and then, in the editor, click the following resource:

```
 ${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl
```

The  `${ab}` variable in this path corresponds to the `product name="ab"` entry in the `suite-config.xml` file that specifies the URL for ContactManager.

14. With ContactManager running, click **Fetch Updates** to get the latest updates to ABContactAPI.

15. On the **Settings** tab, there is a setting for the Configuration Provider

`wsi.remote.gw.webservice.ab.ABConfigurationProvider`. This class defines the user name and password that PolicyCenter uses to authenticate with ContactManager when PolicyCenter makes calls to ABContactAPI. In the base configuration, `ABConfigurationProvider` defines these two settings as follows:

```
config.Guidewire.Authentication.Username = "ClientAppPC"  
config.Guidewire.Authentication.Password = "gw"
```

**Note:** You must load sample data into ContactManager for this default user name to work. This user name and password can be useful for development. However, for security reasons, you must change this default user name and password before going to production. For more information, see “Configuring Core Application Authentication with ContactManager” on page 69.

16. In the Project window, press **Ctrl+N** and enter `ContactMessageTransport`. In the search results, double-click `ContactMessageTransport` to open this Gosu class in the editor.

17. Press **Ctrl+F** and enter `getAdminUserForIntegrationHandling` to find that method in the class. The method is defined as follows:

```
private function getAdminUserForIntegrationHandling() : User {  
    return PLDependenciesGateway.getUserFinder().findByName("admin")  
}
```

This method gets the PolicyCenter user that will be assigned an activity if ContactManager throws an unhandled exception during communication with PolicyCenter. The generic text for the activity is:

**Subject:** Failed to add the contact '{Contact}' to the CMS.

**Description:** An unexpected error occurred when adding the contact to the contact management system\:{Error Message}

The `getAdminUserForIntegrationHandling` method retrieves the user by login name. In the base configuration, this PolicyCenter user has the role Superuser. You must designate a user of your own to handle this activity. Assign the user roles with permissions that enable working with contacts. The contact and tag permissions in the base configuration are `abcreate`, `ctccreate`, `anytagcreate`, `abdelete`, `ctcdelete`, `anytagdelete`, `abedit`, `ctcedit`, `anytagedit`, `abview`, `abviewsearch`, `ctcview`, and `anytagview`. In response to the activity, this user reviews the contact's data and makes the needed corrections. After the user updates the contact, PolicyCenter sends it to ContactManager again.

When you determine the user who will handle these activities, you can create your own class and copy the code in the `ContactMessageTransport` class into your class. In your class's `getAdminUserForIntegrationHandling` method, replace the parameter `admin` with the login name of your user. Then navigate to `configuration → config → Plugins → registry` and register your class in the plugin registry `ContactMessageTransport.gwp`.

18. Save your changes and close PolicyCenter Studio.
19. At a command prompt, navigate to the `ContactManager/bin` directory, and then start ContactManager Studio by entering the following command:  
`gwab studio`
20. In ContactManager Studio, press `Ctrl+Shift+N` and enter `suite-config.xml` to find the suite configuration file. Then double-click the search result to open the file in the editor.

This file defines the URLs of the applications in the Guidewire InsuranceSuite. If the entries in this file are commented out, you must remove the comments for the applications you have installed. In the default configuration, the Guidewire applications have the following settings in this file:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
  <!--<product name="ab" url="http://localhost:8280/ab"/>-->
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

21. Remove the comments around the PolicyCenter (pc) entry, as follows:  

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->
  <product name="pc" url="http://localhost:8180/pc"/>
  <!--<product name="ab" url="http://localhost:8280/ab"/>-->
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```
22. Save your changes.
23. Open the Project window and expand `configuration → config → Plugins → registry`.

24. Double-click `PolicySystemPlugin.gwp` to open it in the Registry editor.

You need to register a plugin implementation so ContactManager can use it to synchronize address book changes with PolicyCenter.

**Note:** By default, the system uses `gw.plugin.integration.StandAloneClientSystemPlugin`. This adapter is a plugin implementation that does not broadcast Contact changes and does not communicate with core applications.

25. In the Registry editor, click Remove Plugin 
26. Click Add Plugin  and, in the drop-down menu, choose `Add Gosu Plugin` to register the new plugin implementation.
27. In the `Gosu Class` field, enter the following class:  
`gw.plugin.policy.pc800.PCPolicySystemPlugin`
28. Navigate to `configuration → gsrc` and then to `wsi.remote.gw.webservice.pc.pc800.wsc`.

29. Double-click the pc800.wsc web services collection to open the editor, and then, in the editor, click the following resource:

```
 ${pc}/ws/gw/webservice/pc/pc800/contact/ContactAPI?wsdl
```

ContactManager calls this PolicyCenter webservice when it has **Contact** updates for PolicyCenter. The PolicyCenter web service **ContactAPI** implements the interface **ABCClientAPI**. The \${pc} variable in this path corresponds to the product name="pc" entry in the **suite-config.xml** file, which specifies the URL for PolicyCenter.

30. With PolicyCenter running, click **Fetch Updates** to get the latest updates to **ContactAPI**.

At the bottom of the window is the **Settings** tab. The default setting in this tab is the configuration provider class. This class, **wsi.remote.gw.webservice.pc.PCConfigurationProvider**, defines the user name and password that ContactManager uses to connect with PolicyCenter. By default, ContactManager uses the user name **su** and password **gw**. Guidewire recommends that you change this user name as described at “Configuring ContactManager-to-PolicyCenter Authentication” on page 76.

31. Save your changes and close ContactManager studio.

32. To pick up your changes, stop and restart ContactManager and PolicyCenter.

- a. Stop PolicyCenter.

Open a command prompt in the **PolicyCenter/bin** directory and then enter the following command:

```
gwpc dev-stop
```

- b. Stop ContactManager.

Open a command prompt in the **ContactManager/bin** directory and then enter the following command:

```
gwab dev-stop
```

- c. Start the ContactManager application:

```
gwab dev-start
```

- d. Start the PolicyCenter application:

```
gwpc dev-start
```

## Step 2: Test the Integration

Ensure that you have restarted both ContactManager and PolicyCenter.

**Note:** You must be working with either a policy that is in force or an account that has at least one policy in force for contacts to be saved in ContactManager. If PolicyCenter appears not to be working with ContactManager, check the policy or account status.

You can verify that the two applications are integrated as follows:

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

```
http://localhost:8180/pc/
```

2. Log in as a user who can create a contact in PolicyCenter, such as the sample user **aapplegate** with password **gw**.

3. Open an existing account from the **Account** tab.

4. Click **Contacts** in the sidebar on the left under **Actions**.

5. At the top of the **Account File Contacts** screen, click **Create New Contact**.

6. On the list, click **Additional Interest** → **New Person**, and create a new contact of type **Person**.

7. Navigate to **Search → Contacts** and search for a contact that is stored in ContactManager. With sample data loaded in ContactManager, you could search for the person William Andy. In the search results, look for a contact that has its **External** column set to **Yes**. That company is stored in ContactManager.

**Note:** If a contact is stored in both PolicyCenter and ContactManager, its **External** column is set to **No**. If you do not see any contacts that are external, either ContactManager has not found a contact stored only in ContactManager or it has suspended its PolicyCenter message queue.

- If you are sure the contact is stored only in ContactManager, the contact's tag might not be set to **Client**. Any contact created in PolicyCenter and stored in ContactManager does have a Client tag. For information on contact tags, see “Contact Tag Overview” on page 177.
- An additional possibility is that PolicyCenter has stopped the message queue it uses for ContactManager. See “Troubleshooting the PolicyCenter Connection with ContactManager” on page 57.

8. Open a browser window and enter the following ContactManager URL:

`http://localhost:8280/ab/`

9. Log in as a user who can view or edit a contact in ContactManager, such as the sample user `aapplegate` with password `gw`.

10. In ContactManager, click **Search** and verify that you can search for and locate the contact you just created in PolicyCenter.

## Troubleshooting the PolicyCenter Connection with ContactManager

### Bad Username or Password Error

You might see an error message saying that there was an exception caused by a bad user name or password. If so, you probably need to create the ContactManager user that PolicyCenter uses to communicate with ContactManager. In the base configuration, this user is `ClientAppPC` with password `gw`. This user is available if you import the sample data for ContactManager. To create a user in ContactManager, see “Configuring ContactManager Authentication with Core Applications” on page 73.

### See also

- “Configuring Core Application Authentication with ContactManager” on page 69.
- “Step 2: Load Sample Data” on page 40 for instructions on how to load sample data for ContactManager

### No Response from ContactManager

If you are not able to get any response at all from ContactManager, it is possible that the message queue that PolicyCenter uses to communicate with ContactManager has been suspended.

### To check the PolicyCenter message queue and reactivate it

1. If they are not already running, start PolicyCenter and ContactManager.
2. Log in to PolicyCenter as a user with administrative privileges. For example, log in with user name `su` and password `gw`.
3. Click the **Administration** tab, and then click **Monitoring → Message Queues** in the sidebar on the left.
4. On the **Message Queues** screen, select the **ContactMessageTransport** entry.
5. If the **Status** is **Suspended**, click the **Resume** button at the top of the screen.
6. The **Status** changes to **Started**, and PolicyCenter can now send messages to ContactManager.

## Integrating PolicyCenter 7 with ContactManager 8

If your installation of PolicyCenter is version 7.0.1 or later, you can integrate with ContactManager. You might have ContactManager 8.0.x installed if you have ClaimCenter 8.0.x and a license for Client Data Management. If you upgrade ContactManager 7.0.x to ContactManager 8.0.x, you can still integrate it with PolicyCenter 7.0.x.

On the PolicyCenter side, the integration between PolicyCenter 7.0.x and ContactManager 8.0.x uses the same plugins, classes, and XML configuration files as with ContactManager 7.0.x. See the *PolicyCenter 7.0.x Contact Management Guide*.

To support integration with PolicyCenter 7.0.x, ContactManager 8.0.x provides the following plugins, web services, classes, and XML configuration files:

- `ClaimSystemPlugin.gwp` – Registry for plugin that enables communication with PolicyCenter.
- `gw.plugin.policy.pc700.PCPolicySystemPlugin` – Plugin implementation you register in `ClaimSystemPlugin.gwp` to communicate with PolicyCenter 7.0.x.
- `gw.webservice.ab.ab700.abcontactapi.ABContactAPI` – ContactManager web service supporting SOAP calls from PolicyCenter 7.0.x.
- `gw.webservice.contactapi.ab700.ABClientAPI` – ContactManager interface used by PolicyCenter 7.0.x to implement the PolicyCenter web service `gw.webservice.pc.pc700.contact.ContactAPI`.
- `gw.contactmapper.ab700.ContactIntegrationXMLMapper.gs` – ContactManager XML mapping file for use with PolicyCenter 7.0.x.
- `gw.webservice.ab.ab700.abcontactapi.ABContactSearchCriteriaInfo.gs` – A class that specifies the contact search criteria that PolicyCenter 7.0.x can use.
- `gw.webservice.ab.ab700.abcontactapi.ABContactSearchResult.gs` – A class that specifies the fields included in the contact search results that ContactManager sends back to PolicyCenter 7.0.x.
- `gw.webservice.ab.ab700.abcontactapi.ABContactAPISearchSpecInfo.gs` – Provides the search criteria used by the web service method `ABContactAPI.findDuplicates`.
- `gw.webservice.ab.ab700.abcontactapi.AddressSearchInfo.gs` – Used by `ABContactSearchCriteriaInfo` to declare the Address and ProximitySearchCenter variables.
- `gw.webservice.ab.ab700.abcontactapi.ProximitySearchParametersInfo.gs` – Used by `ABContactSearchCriteriaInfo` to declare the ProximitySearchParameters variable.

## Integrating PolicyCenter 8 with ContactManager 7

If your installation of PolicyCenter is version 8.0.x, you might have ContactManager 7.0.x installed, but not ContactManager 8.0.x. For example, you have ClaimCenter 7.0.x and a 7.0.x license for Client Data Management.

On the ContactManager side, the integration between PolicyCenter 8.0.x and ContactManager 7.0.x uses the same plugins, classes, and XML configuration files as with PolicyCenter 7.0.x. See the *Contact Management Guide* for your 7.0.x installation.

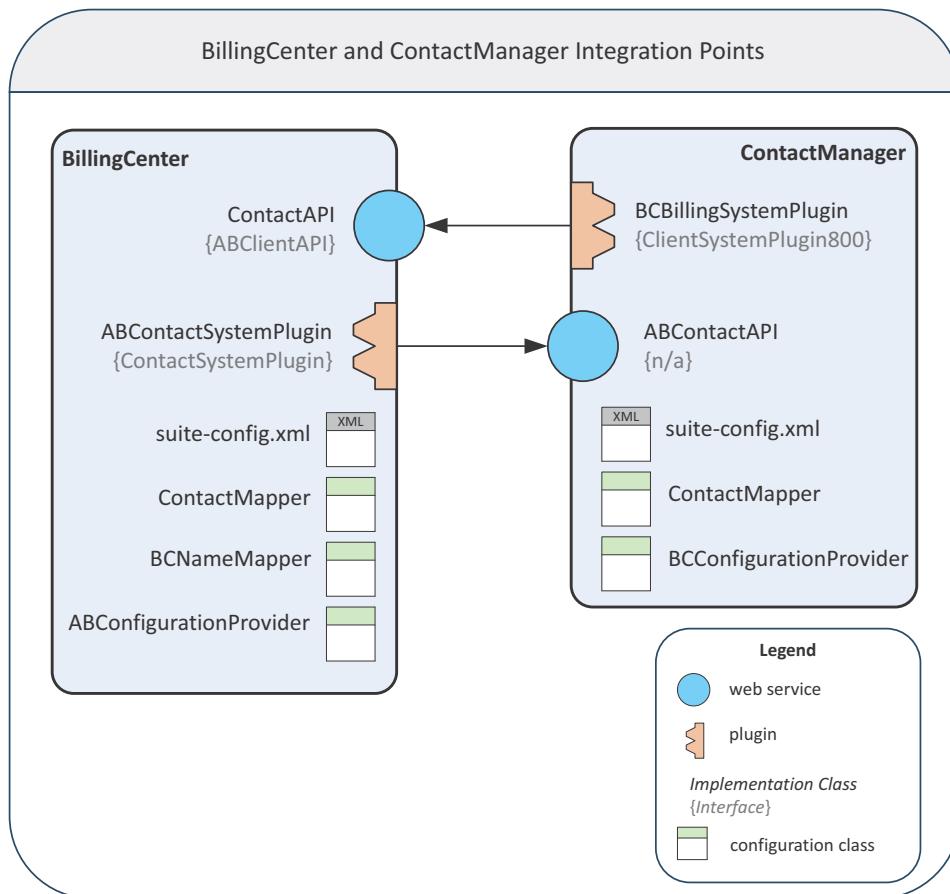
You can integrate PolicyCenter 8.0.x with ContactManager 7.0.x by using the following ab700 versions of the PolicyCenter contact integration files:

- `ContactSystemPlugin.gwp` – Registry for plugin that enables communication with ContactManager.
- `gw.plugin.contact.ab700.ABContactSystemPlugin` – Plugin implementation that you register in `ContactSystemPlugin.gwp` to communicate with ContactManager 7.0.x.
- `gw.contactmapper.ab700.ContactIntegrationXMLMapper` – Class that maps contact data as XML between PolicyCenter and ContactManager 7.0.x. Do not use `ContactMapper.gs` for this purpose.
- `pc-to-cm-type-mapping.xml` and `cm-to-pc-type-mapping.xml` – Configuration files used to exclude fields from being mapped between PolicyCenter and ContactManager 7.0.x. Use these files instead of `ContactMapper.gs` and `PCNameMapper.gs`.

- `gw.webservice.pc.pc700.contact.ContactAPI` – PolicyCenter web service used by ContactManager 7.0.x to make web service calls to PolicyCenter.

## Integrating ContactManager with BillingCenter in QuickStart

The following figure shows the primary integration points used to integrate BillingCenter and ContactManager.



To get the basic integration working, you edit the `suite-config.xml` files in BillingCenter and ContactManager. You also register the `BCBillingSystemPlugin` plugin implementation in the ContactManager `BillingSystemPlugin` registry. Additionally, you register the `ABContactSystemPlugin` in the BillingCenter `ContactSystemPlugin` registry. The detailed steps to integrate BillingCenter and ContactManager are described in the topics that follow.

If you extend the contact model, you edit the `ContactMapper` classes in both ContactManager and BillingCenter, and possibly the `BCNameMapper` class. When you change the authentication user and password, you edit the `ABConfigurationProvider` and `BCConfigurationProvider` classes.

This topic includes:

- “Step 1: Integrate ContactManager with BillingCenter” on page 60
- “Step 2: Test the Integration” on page 62
- “Troubleshooting the BillingCenter Connection with ContactManager” on page 63
- “Integrating BillingCenter 7 with ContactManager 8” on page 64

**See also**

- “ABClientAPI Interface” on page 282
- “ABContactAPI Web Service” on page 275
- “ContactMapper Class” on page 285
- “Configuring BillingCenter-to-ContactManager Authentication” on page 72
- “Configuring ContactManager-to-BillingCenter Authentication” on page 79
- “Extending the Client Data Model” on page 141

**Step 1: Integrate ContactManager with BillingCenter**

After installing ContactManager, you can use both applications separately, but they are not integrated. To integrate the two applications you must configure services in Guidewire Studio. This topic starts with the changes you make in BillingCenter Studio and then continues with the changes you make in ContactManager Studio.

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

```
gwbc studio
```

2. In BillingCenter Studio, open the Project window.

3. Press `Ctrl+Shift+N` and enter `suite-config.xml` to find this file, and then double-click the search result to open the file in the editor.

This file defines the URLs of the applications in the Guidewire InsuranceSuite. If the entries in this file are commented out, you must remove the comments for the applications you have installed. In the default configuration, the Guidewire applications have the following settings in this file:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
  <!--<product name="ab" url="http://localhost:8280/ab"/>-->
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

4. Remove the comments around the ContactManager (ab) entry, as follows:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->
  <product name="ab" url="http://localhost:8280/ab"/>
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->
</suite-config>
```

5. Save your changes.

6. In the Project window, expand **configuration** → **config** → **Plugins** → **registry**.

7. Double-click `ContactSystemPlugin.gwp` to open it in the Registry editor.

**Note:** By default, the system uses the plugin implementation

`gw.plugin.contact.impl.StandAloneContactSystemPlugin`. This adapter is a Gosu plugin that BillingCenter uses when not connected to a contact management system. In the steps that follow, you replace this class with the plugin implementation that connects BillingCenter with ContactManager.

8. In the Registry editor, click Remove Plugin .

9. Click Add Plugin  and, in the drop-down menu, choose **Add Gosu Plugin** to register the new plugin implementation.

10. In the **Gosu Class** field, enter the following class:

```
gw.plugin.contact.ab800.ABContactSystemPlugin
```

11. If necessary, start ContactManager.

At a command prompt, navigate to the `ContactManager/bin` directory and enter the following command:

```
gwab dev-start
```

12. In the BillingCenter Studio Project window, navigate to **configuration** → **gsrc** and then to **wsi.remote.gw.webservice.ab.ab801.wsc**.

**Note:** The following steps describe specific settings in the editor for web services collections. For general information, see “Using the WS-I Web Service Editor” on page 122 in the *Configuration Guide*.

13. Double-click the **ab801.wsc** web services collection to open the editor, and then, in the editor, click the following resource:

```
 ${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl
```

The  **\${ab}** variable in this path corresponds to the product name="ab" entry in the **suite-config.xml** file that specifies the URL for ContactManager.

14. With ContactManager running, click **Fetch Updates** to get the latest updates to ABContactAPI.

15. In the same editor on the **Settings** tab, there is a setting for the Configuration Provider **wsi.remote.gw.webservice.ab.ABConfigurationProvider**. This class defines the user name and password that BillingCenter uses to authenticate with ContactManager when BillingCenter makes calls to ABContactAPI. In the base configuration, ABConfigurationProvider defines these two settings as follows:

```
 config.Guidewire.Authentication.Username = "ClientAppBC"  
 config.Guidewire.Authentication.Password = "gw"
```

**Note:** You must load sample data into ContactManager for this default user name to work. This user name and password can be useful for development. However, for security reasons, you need to change this default user name and password before going to production. For more information, see “Configuring Core Application Authentication with ContactManager” on page 69.

16. Save your changes and close BillingCenter Studio.

17. At a command prompt, navigate to the **ContactManager/bin** directory, and then start ContactManager Studio by entering the following command:

```
gwab studio
```

18. In ContactManager Studio, press **Ctrl+Shift+N** and enter **suite-config.xml** to find the suite configuration file. Then double-click the search result to open the file in the editor.

This file defines the URLs of the applications in the Guidewire InsuranceSuite. If the entries in this file are commented out, you must remove the comments for the applications you have installed. In the default configuration, the Guidewire applications have the following settings in this file:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">  
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->  
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->  
  <!--<product name="ab" url="http://localhost:8280/ab"/>-->  
  <!--<product name="bc" url="http://localhost:8580/bc"/>-->  
</suite-config>
```

19. Remove the comments around the BillingCenter (cc) entry, as follows:

```
<suite-config xmlns="http://guidewire.com/config/suite/suite-config">  
  <!--<product name="cc" url="http://localhost:8080/cc"/>-->  
  <!--<product name="pc" url="http://localhost:8180/pc"/>-->  
  <!--<product name="ab" url="http://localhost:8280/ab"/>-->  
  <product name="bc" url="http://localhost:8580/bc"/>  
</suite-config>
```

20. Save your changes.

21. Open the Project window and expand **configuration** → **config** → **Plugins** → **registry**.

22. Double-click **BillingSystemPlugin.gwp** to open it in the Registry editor.

You need to register a plugin implementation so ContactManager can use it to synchronize address book changes with BillingCenter.

**Note:** By default, the system uses `gw.plugin.integration.StandAloneClientSystemPlugin`. This adapter is a plugin implementation that does not broadcast Contact changes and does not communicate with core applications.

23. In the Registry editor, click Remove Plugin —.
24. Click Add Plugin + and, in the drop-down menu, choose **Add Gosu Plugin** to register the new plugin implementation.
25. In the **Gosu Class** field, enter the following class:  
`gw.plugin.billing.bc800.BCBillingSystemPlugin`
26. In the column under the add and remove plugin buttons, click **Gosu[ ]** to update the plugin entry. After clicking, the entry changes to the following:  
`Gosu [gw.plugin.billing.bc800.BCBillingSystemPlugin]`
27. Navigate to **configuration** → **gsrc** and then to **wsi.remote.gw.webservice.bc.bc800.wsc**.
28. Double-click the **bc800.wsc** web services collection to open the editor, and then, in the editor, click the following resource:  
 `${bc}/ws/gw/webservice/bc/bc800/contact/ContactAPI?wsdl`

ContactManager calls this BillingCenter webservice when it has Contact updates for BillingCenter. The BillingCenter web service `ContactAPI` implements the interface `ABCClientAPI`. The  `${bc}` variable in this path corresponds to the `product name="bc"` entry in the `suite-config.xml` file, which specifies the URL for BillingCenter.
29. With BillingCenter running, click **Fetch Updates** to get the latest updates to `ContactAPI`.
30. In the same editor on the **Settings** tab is the configuration provider class. This class, `wsi.remote.gw.webservice.bc.BCConfigurationProvider`, defines the user name and password that ContactManager uses to connect with BillingCenter. By default, ContactManager uses the user name `su` and password `gw`. Guidewire recommends that you change this user name as described at “Configuring ContactManager-to-BillingCenter Authentication” on page 79.
31. Save your changes and close ContactManager studio.
32. To pick up your changes, stop and restart ContactManager and BillingCenter.
  - a. Stop BillingCenter.  
Open a command prompt in the `BillingCenter/bin` directory and then enter the following command:  
`gwbc dev-stop`
  - b. Stop ContactManager.  
Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`
  - c. Start the ContactManager application:  
`gwab dev-start`
  - d. Start the BillingCenter application:  
`gwbc dev-start`

## Step 2: Test the Integration

Ensure that you have started both ContactManager and BillingCenter.

You can verify that the two applications are integrated as follows:

1. When BillingCenter is ready, open a browser window and enter the following BillingCenter URL:  
`http://localhost:8580/bc/`
2. Log in as a user who can create a contact in BillingCenter, such as the sample user `aapplegate` with password `gw`.
3. Open an existing account from the **Account** tab.
4. Click **Contacts** in the left Sidebar under **Actions**.
5. On the **Contacts** screen, click the **Edit** button.
6. Click **Add Existing Contact**, and search for a company you know is in ContactManager and has a **Client** tag, such as the sample company Albertson's.

**Note:** If a contact is stored in both BillingCenter and ContactManager, its **External** column is set to **No**. If you do not see any contacts that are external, either ContactManager has not found contacts stored only in ContactManager or it has suspended its BillingCenter message queue.

  - If you are sure the contact is stored only in ContactManager, the contact's tag might not be set to **Client**. Any contact created in BillingCenter and stored in ContactManager does have a Client tag. For information on contact tags, see “Contact Tag Overview” on page 177.
  - An additional possibility is that BillingCenter has stopped the message queue it uses for ContactManager. See “Troubleshooting the BillingCenter Connection with ContactManager” on page 63.
7. Click **Select** next to the company name.
8. On the **Contacts** screen, select the company name.
9. Below, on the **Contact Info** card, change contact data, like the **Address 1** field, and then click **Update** to save the change.
10. Open a browser window and enter the following ContactManager URL:  
`http://localhost:8280/ab/`
11. Log in as a user who can view or edit a contact in ContactManager, such as the sample user `aapplegate` with password `gw`.
12. In ContactManager, click **Search** and find the contact you changed in BillingCenter.
13. Verify that the data you changed for the contact in BillingCenter was also changed for the contact in ContactManager.

## Troubleshooting the BillingCenter Connection with ContactManager

### Bad Username or Password Error

You might see an error message saying that there was an exception caused by a bad user name or password. If so, you probably need to create the ContactManager user that BillingCenter uses to communicate with ContactManager. In the base configuration, this user is `ClientAppBC` with password `gw`. This user is available if you import the sample data for ContactManager. To create a user in ContactManager, see “Configuring ContactManager Authentication with Core Applications” on page 73.

### See also

- “Step 2: Load Sample Data” on page 40 for instructions on how to load sample data for ContactManager
- “Configuring Core Application Authentication with ContactManager” on page 69

### No Response from ContactManager

If you are not able to get any response at all from ContactManager, it is possible that the message queue that BillingCenter uses to communicate with ContactManager has been suspended. To check the BillingCenter message queue and reactivate it:

1. If they are not already running, start BillingCenter and ContactManager.
2. Log in to BillingCenter as a user with administrative privileges. For example, log in with user name `su` and password `gw`.
3. Click the **Administration** tab, and then click **Monitoring → Message Queues** in the sidebar on the left.
4. On the **Message Queues** screen, select the **ContactMessageTransport** entry.
5. If the **Status** is **Suspended**, click the **Resume** button at the top of the screen.
6. The **Status** changes to **Started**, and BillingCenter can now send messages to ContactManager.

### Integrating BillingCenter 7 with ContactManager 8

If your installation of BillingCenter is version 7.0.x, you might have ContactManager 8.0.x installed if you have ClaimCenter 8.0.x or PolicyCenter 8.0.x and a license for Client Data Management. In that case, if you upgrade ContactManager 7.0.x to ContactManager 8.0.x, you can still integrate it with BillingCenter 7.0.x.

On the BillingCenter side, the integration between BillingCenter 7.0.x and ContactManager 8.0.x uses the same plugins, classes, and XML configuration files as with ContactManager 7.0.x. See the *BillingCenter 7.0.x Contact Management Guide*.

To support integration with BillingCenter 7.0.x, ContactManager 8.0.x provides the following plugins, web services, classes, and XML configuration files:

- `ClaimSystemPlugin.gwp` – Registry for plugin that enables communication with BillingCenter.
- `gw.plugin.billing.bc700.BCBillingSystemPlugin` – Plugin implementation you register in `ClaimSystemPlugin.gwp` to communicate with BillingCenter 7.0.x.
- `gw.webservice.ab.ab700.abcontactapi.ABContactAPI` – ContactManager web service supporting SOAP calls from BillingCenter 7.0.x.
- `gw.webservice.contactapi.ab700.ABClientAPI` – ContactManager interface used by BillingCenter 7.0.x to implement the PolicyCenter web service `gw.webservice.bc.bc700.contact.ContactAPI`.
- `gw.contactmapper.ab700.ContactIntegrationXMLMapper.gs` – ContactManager XML mapping file for use with BillingCenter 7.0.x.
- `gw.webservice.ab.ab700.abcontactapi.ABContactSearchCriteriaInfo.gs` – A class that specifies the contact search criteria that BillingCenter 7.0.x can use.
- `gw.webservice.ab.ab700.abcontactapi.ABContactSearchResult.gs` – A class that specifies the fields included in the contact search results that ContactManager sends back to BillingCenter 7.0.x.
- `gw.webservice.ab.ab700.abcontactapi.ABContactAPISearchSpecInfo.gs` – Provides the search criteria used by the web service method `ABContactAPI.findDuplicates`.
- `gw.webservice.ab.ab700.abcontactapi.AddressSearchInfo.gs` – Used by `ABContactSearchCriteriaInfo` to declare the `Address` and `ProximitySearchCenter` variables.
- `gw.webservice.ab.ab700.abcontactapi.ProximitySearchParametersInfo.gs` – Used by `ABContactSearchCriteriaInfo` to declare the `ProximitySearchParameters` variable.

### Integrating BillingCenter 8 with ContactManager 7

If your installation of BillingCenter is version 8.0.x, you might have ContactManager 7.0.x installed, but not ContactManager 8.0.x. For example, you have PolicyCenter 7.0.x and a 7.0.x license for Client Data Management.

On the ContactManager side, the integration between BillingCenter 8.0.x and ContactManager 7.0.x uses the same plugins, classes, and XML configuration files as with BillingCenter 7.0.x. See the *Contact Management Guide* for your 7.0.x installation.

You can integrate BillingCenter 8.0.x with ContactManager 7.0.x by using the following ab700 versions of the BillingCenter contact integration files:

- `ContactSystemPlugin.gwp` – Registry for plugin that enables communication with ContactManager.
- `gw.plugin.contact.ab700.ContactManagerSystemPlugin` – Plugin implementation that you register in `ContactSystemPlugin.gwp` to communicate with ContactManager 7.0.x.
- `gw.contactmapper.ab700.ContactIntegrationXMLMapper` – Class that maps contact data as XML between BillingCenter and ContactManager 7.0.x. Do not use `ContactMapper.gs` for this purpose.
- `bc-to-cm-type-mapping.xml` and `cm-to-bc-type-mapping.xml` – Configuration files used to exclude fields from being mapped between BillingCenter and ContactManager 7.0.x. Use these files instead of `ContactMapper.gs` and `BCNameMapper.gs`.
- `gw.webservice.bc.bc700.contact.ContactAPI` – PolicyCenter web service used by ContactManager 7.0.x to make web service calls to BillingCenter.

## Integrating ContactManager Using Tomcat and SQL Server

This topic describes how to configure Guidewire core applications to run with ContactManager in a development environment other than QuickStart. You can use any single supported combination of application and database server with ContactManager and the applications.

Before continuing with this section, you must have installed ContactManager. For information on installing ContactManager in a development environment, see “[Installing ContactManager with Tomcat and SQL Server for Development](#)” on page 41.

These examples use Tomcat and SQL Server. Because you need a separate JVM for each application, you must install and configure an instance of Tomcat just for ContactManager.

For information on installing with other servers and databases that you can apply to ContactManager, see “[Installing a ClaimCenter Production Environment](#)” on page 55 in the *Installation Guide*.

This topic includes:

- “[Integrating ContactManager and ClaimCenter with Tomcat and SQL Server](#)” on page 65
- “[Integrating ContactManager and PolicyCenter with Tomcat and SQL Server](#)” on page 66
- “[Integrating ContactManager and BillingCenter with Tomcat and SQL Server](#)” on page 68

## Integrating ContactManager and ClaimCenter with Tomcat and SQL Server

### Step 1: Integrate Both Applications

This part of integrating ContactManager and ClaimCenter is the same as for QuickStart. See “[Step 1: Integrate ContactManager with ClaimCenter](#)” on page 47.

### Step 2: Deploy New ClaimCenter and ContactManager WAR Files and Start the Applications

After changing the plugin settings in ContactManager and ClaimCenter, you must generate new .war files and deploy them to your servers.

1. Generate a new ContactManager .war file, deploy it to your server, and start ContactManager.
  - a. At a command prompt, navigate to the `ContactManager/bin` directory and run the following command:  
`gwab build-tomcat-war-dbcp`

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

- b. Deploy the package to Tomcat by copying the ab.war file to the webapps directory in your Tomcat server.
- c. Use the Tomcat bin/startup.bat command to start Tomcat and expand the WAR file. When Tomcat starts, it automatically recognizes the new application and unpacks the ab.war into a directory structure within webapps. For this example, Tomcat creates a webapps/ab directory. Each time you deploy a new copy of an ab.war file, delete the existing ab directory structure before you start Tomcat.
- d. Delete the Tomcat webapps\ab\modules\configuration directory.
- e. Create a symbolic link from the deployed ContactManager application on Tomcat to the ContactManager\modules\configuration directory of the original ContactManager installation location.
  - Windows 7 and Vista: Use the mklink command to create the link, as in the example below:

```
mklink /d C:\apache-tomcat-6.0.18\webapps\ab\modules\configuration
C:\ContactManager\modules\configuration
```
  - Windows XP: Use the Windows SysInternals program Junction.exe to create the symbolic link. The Junction.exe program is available at:  
<http://technet.microsoft.com/en-us/sysinternals/bb896768.aspx>
- f. Restart the Tomcat server.
- g. Open a browser and navigate to <http://localhost:8280/ab>, and then log in as the superuser with:  
User name: su  
Password: gw

**Note:** Your ports might be different.

For more information, see “Step1: Install and Set Up ContactManager, Tomcat, and SQL Server” on page 43.

## 2. Generate a new ClaimCenter .war file, deploy it to your server, and start ClaimCenter.

- a. See the instructions for deploying a .war file at “Building a ClaimCenter Development Environment with Tomcat” on page 49 in the *Installation Guide*.

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

- b. Start Tomcat. For example:

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

- c. Open a browser and navigate to <http://localhost:8080/cc>, and then log in as the superuser with:

User name: su

Password: gw

**Note:** Your ports might be different.

## Step 3: Test the Integration

Ensure that you have started both ContactManager and ClaimCenter. You can verify that the two applications are integrated by following the steps at “Step 2: Test the Integration” on page 50.

# Integrating ContactManager and PolicyCenter with Tomcat and SQL Server

## Step 1: Integrate Both Applications

This part of integrating ContactManager and PolicyCenter is the same as for QuickStart. See “Step 1: Integrate ContactManager with PolicyCenter” on page 53.

## Step 2: Deploy New ContactManager and PolicyCenter WAR Files

After changing the plugin settings in ContactManager and PolicyCenter, you must generate new .war files and deploy them to your servers.

1. Generate a new ContactManager .war file, deploy it to your server, and start ContactManager.
  - a. At a command prompt, navigate to the ContactManager/bin directory and run the following command:

```
gwab build-tomcat-war-dbc
```

This command creates the ab.war file in the ContactManager/dist/war directory.
  - b. Deploy the package to Tomcat by copying the ab.war file to the webapps directory in your Tomcat server.
  - c. Use the Tomcat bin/startup.bat command to start Tomcat and expand the WAR file. When Tomcat starts, it automatically recognizes the new application and unpacks the ab.war into a directory structure within webapps. For this example, Tomcat creates a webapps/ab directory. Each time you deploy a new copy of an ab.war file, delete the existing ab directory structure before you start Tomcat.
  - d. Delete the Tomcat webapps\ab\modules\configuration directory.
  - e. Create a symbolic link from the deployed ContactManager application on Tomcat to the ContactManager\modules\configuration directory of the original ContactManager installation location.
    - Windows 7 and Vista: Use the mklink command to create the link, as in the example below:

```
mklink /d C:\apache-tomcat-6.0.18\webapps\ab\modules\configuration  
C:\ContactManager\modules\configuration
```
    - Windows XP: Use the Windows SysInternals program Junction.exe to create the symbolic link. The Junction.exe program is available at:  
<http://technet.microsoft.com/en-us/sysinternals/bb896768.aspx>
  - f. Restart the Tomcat server.
  - g. Open a browser and navigate to <http://localhost:8280/ab>, and then log in as the superuser with:  
User name: su  
Password: gw
- Note:** Your ports might be different.
- For more information, see “Step1: Install and Set Up ContactManager, Tomcat, and SQL Server” on page 43.
2. Generate a new PolicyCenter .war file, deploy it to your server, and start PolicyCenter.
  - a. See the instructions for deploying a .war file at “Building a ClaimCenter Development Environment with Tomcat” on page 49 in the *Installation Guide*.
  - Note:** You must use a different Tomcat installation for PolicyCenter than for ContactManager. For example, you might use c:\tomcat for ContactManager and c:\tomcat3 for PolicyCenter.
  - b. Start Tomcat. For example:

```
startup -Dgw.server.mode=dev -Dcatalina.base="c:\tomcat3"
```
  - c. Open a browser and navigate to <http://localhost:8180/pc>, and then log in as the superuser with:  
User name: su  
Password: gw
  - Note:** Your ports might be different.

## Step 3: Test the Integration

Ensure that you have started both ContactManager and PolicyCenter. You can verify that the two applications are integrated by following the steps at “Step 2: Test the Integration” on page 56.

## Integrating ContactManager and BillingCenter with Tomcat and SQL Server

### Step 1: Integrate ContactManager with BillingCenter

This part of integrating ContactManager and PolicyCenter is the same as for QuickStart. See “Step 1: Integrate ContactManager with BillingCenter” on page 60.

### Step 2: Deploy New ContactManager and BillingCenter WAR Files

After changing the plugin settings in ContactManager and BillingCenter, you must generate new .war files and deploy them to your servers.

1. Generate a new ContactManager .war file, deploy it to your server, and start ContactManager.
  - a. At a command prompt, navigate to the ContactManager/bin directory and run the following command:

```
gwab build-tomcat-war-dbc
```

This command creates the ab.war file in the ContactManager/dist/war directory.
  - b. Deploy the package to Tomcat by copying the ab.war file to the webapps directory in your Tomcat server.
  - c. Use the Tomcat bin/startup.bat command to start Tomcat and expand the WAR file. When Tomcat starts, it automatically recognizes the new application and unpacks the ab.war into a directory structure within webapps. For this example, Tomcat creates a webapps/ab directory. Each time you deploy a new copy of an ab.war file, delete the existing ab directory structure before you start Tomcat.
  - d. Delete the Tomcat webapps\ab\modules\configuration directory.
  - e. Create a symbolic link from the deployed ContactManager application on Tomcat to the ContactManager\modules\configuration directory of the original ContactManager installation location.
    - Windows 7 and Vista: Use the mklink command to create the link, as in the example below:

```
mklink /d C:\apache-tomcat-6.0.18\webapps\ab\modules\configuration  
C:\ContactManager\modules\configuration
```
    - Windows XP: Use the Windows SysInternals program Junction.exe to create the symbolic link. The Junction.exe program is available at:  
<http://technet.microsoft.com/en-us/sysinternals/bb896768.aspx>
  - f. Restart the Tomcat server.
  - g. Open a browser and navigate to <http://localhost:8280/ab>, and then log in as the superuser with:  
User name: su  
Password: gw
- Note:** Your ports might be different.
- For more information, see “Step1: Install and Set Up ContactManager, Tomcat, and SQL Server” on page 43.
2. Generate a new BillingCenter .war file, deploy it to your server, and start BillingCenter.
  - a. See the instructions for deploying a .war file at “Building a ClaimCenter Development Environment with Tomcat” on page 49 in the *Installation Guide*.
  - Note:** You must use a different Tomcat installation for BillingCenter than for ContactManager. For example, you might use c:\tomcat for ContactManager and c:\tomcat4 for BillingCenter.
  - b. Start Tomcat. For example:

```
startup -Dgw.server.mode=dev -Dcatalina.base="c:\tomcat4"
```
  - c. Open a browser and navigate to <http://localhost:8180/bc>, and then log in as the superuser with:  
User name: su  
Password: gw

**Note:** Your ports might be different.

### Step 3: Test the Integration

Ensure that you have started both ContactManager and BillingCenter. You can verify that the two applications are integrated by following the steps at “Step 2: Test the Integration” on page 62.

## Configuring Core Application Authentication with ContactManager

When you integrate a Guidewire core application with ContactManager, you register a plugin implementation with the core application’s plugin registry. For integration of ContactManager 8.0 with a core application of the same version, you register `gw.plugin.contact.ab800.ABContactSystemPlugin` with the core application’s `ContactSystemPlugin` registry.

Each core application defines a user name and password that it uses to authenticate with ContactManager when the application uses its plugin to call `ABContactAPI` methods. The definition is in the class `wsi.remote.gw.webservice.ab.ABConfigurationProvider`. For security purposes, Guidewire recommends that you define your own user names and passwords.

- ClaimCenter defines the two authentication parameters, `username` and `password`, as `ClientAppCC` and `gw`.
- PolicyCenter defines the two authentication parameters, `username` and `password`, as `ClientAppPC` and `gw`.
- BillingCenter defines the two authentication parameters, `username` and `password`, as `ClientAppBC` and `gw`.

The base ContactManager application’s sample code contains users with names and passwords that match the default settings in the core applications. The sample code has users with names `ClientAppCC`, `ClientAppPC`, and `ClientAppBC`, all with password `gw`.

This topic describes how to change the `username` and `password` that each application uses to authenticate with ContactManager when making calls to `ABContactAPI`. The process, which is generally the same for all applications, follows:

1. Set up a user and password in ContactManager to match the user name and password you define in the core application.
2. Define the user name and password in the core application’s `ABConfigurationProvider` class.

This topic includes:

- “Configuring ClaimCenter-to-ContactManager Authentication” on page 69
- “Configuring PolicyCenter-to-ContactManager Authentication” on page 71
- “Configuring BillingCenter-to-ContactManager Authentication” on page 72

## Configuring ClaimCenter-to-ContactManager Authentication

To change the user name and password that ClaimCenter uses to authenticate with ContactManager:

1. Start ContactManager.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:  
`gwab dev-start`

2. Log in as a user who can create new ContactManager users. For example, log in with user name `su` and password `gw`.

3. Click the **Administration** tab and choose **Actions → New User**.

**4.** Enter the following values:

Name	Value
First Name	CC
Last Name	NewAuthUser
Username	CCNewAuthUser
Password	Xc8899Lm
Confirm Password	Xc8899Lm
Active	Yes
Locked	No

**5.** Under Roles, click **Add**.

**6.** Click the **Name** field and choose **Client Application**.

**7.** Click **Update**.

**8.** Start ClaimCenter Studio.

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

```
gwcc studio
```

**9.** Open the **Project** window.

**10.** Press **Ctrl+N** and enter **ABConfigurationProvider**, and then click the class name in the search results to open it in the editor.

**11.** Change the **Username** and **Password** definition in the **configure** method. If you see a message asking if you want to edit the class, click **Yes**.

For example, change the values to `CCNewAuthUser` and `Xc8899Lm`, as follows:

```
override function configure( serviceName : QName, portName : QName, config : WsdlConfig ) {
    config.Guidewire.Authentication.Username = "CCNewAuthUser"
    config.Guidewire.Authentication.Password = "Xc8899Lm"
}
```

**12.** In the **Project** window, navigate to **configuration** → **gsrc** and then to **wsi.remote.gw.webservice.ab.ab801.wsc**.

**13.** Double-click the **ab801.wsc** web resource collection to open the editor.

**14.** In the editor, click  `${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl`.

**15.** With **ContactManager** still running, click **Fetch Updates**.

For more information on this editor, see “Using the WS-I Web Service Editor” on page 122 in the *Configuration Guide*.

**16.** Save your changes, and then stop **ClaimCenter** if it is running and restart it to pick up this change.

**a.** Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:

```
gwcc dev-stop
```

**b.** After the application stops, enter the following command to restart it:

```
gwcc dev-start
```

**17.** Log in to **ClaimCenter** as a user with permission to work with the Address Book, such as user `ssmith` with password `gw`.

**18.** Verify that you can use the **Address Book** tab to search for **ContactManager** contacts, such as the sample contact William Weeks.

## Configuring PolicyCenter-to-ContactManager Authentication

To change the user name and password that PolicyCenter uses to authenticate with ContactManager:

1. Start ContactManager.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:

```
gwab dev-start
```

2. Log in as a user who can create new ContactManager users. For example, log in with user name `su` and password `gw`.

3. Click the **Administration** tab and choose **Actions → New User**.

4. Enter the following values:

Name	Value
First Name	PC
Last Name	NewAuthUser
Username	PCNewAuthUser
Password	Xc8899Lm
Confirm Password	Xc8899Lm
Active	Yes
Locked	No

5. Under Roles, click **Add**.

6. Click the **Name** field and choose **Client Application**.

7. Click **Update**.

8. Start PolicyCenter Studio.

At a command prompt, navigate to `PolicyCenter/bin` and enter the following command:

```
gwpc studio
```

9. Press **Ctrl+N** and enter `ABConfigurationProvider`, and then click the class name in the search results to open it in the editor.

10. Change the `Username` and `Password` definition in the `configure` method. If you see a message asking if you want to edit the class, click **Yes**.

For example, change the values to `PCNewAuthUser` and `Xc8899Lm`, as follows:

```
override function configure( serviceName : QName, portName : QName, config : WsdlConfig ) {  
    config.Guidewire.Authentication.Username = "PCNewAuthUser"  
    config.Guidewire.Authentication.Password = "Xc8899Lm"  
}
```

11. In the **Project** window, navigate to **configuration → gsrc** and then to `wsi.remote.gw.webservice.ab.ab801.wsc`.

12. Double-click the `ab801.wsc` web services collection to open the editor.

13. In the editor, click  `${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl`.

14. With ContactManager still running, click **Fetch Updates**.

For more information on this editor, see “Using the WS-I Web Service Editor” on page 122 in the *Configuration Guide*.

15. Save your changes, and then stop PolicyCenter if it is running and restart it to pick up this change.

- a. Open a command prompt in the `PolicyCenter/bin` directory and then enter the following command:

```
gwpc dev-stop
```

- b.** After the application stops, enter the following command to restart it:  

```
gwpc dev-start
```
- 16.** Log in to PolicyCenter as a user with permission to search for and change contacts, such as user name **aapplegate** with password **gw**.
- 17.** Verify that you can use the **Contact** tab to search for ContactManager contacts. You can be sure that a contact is stored in ContactManager if its **External** field has the value **Yes**. For example, search for the person Adam Hinds, a client contact in the ContactManager sample data.

## Configuring BillingCenter-to-ContactManager Authentication

To change the user name and password that BillingCenter uses to authenticate with ContactManager:

**1. Start ContactManager.**

At a command prompt, navigate to **ContactManager/bin** and enter the following command:  

```
gwab dev-start
```

**2. Log in as a user who can create new ContactManager users.** For example, log in with user name **su** and password **gw**.

**3. Click the Administration tab and choose Actions → New User.**

**4. Enter the following values:**

Name	Value
First Name	BC
Last Name	NewAuthUser
Username	BCNewAuthUser
Password	Xc8899Lm
Confirm Password	Xc8899Lm
Active	Yes
Locked	No

**5. Under Roles, click Add.**

**6. Click the Name field and choose Client Application.**

**7. Click Update.**

**8. Start BillingCenter Studio.**

At a command prompt, navigate to **BillingCenter/bin** and enter the following command:  

```
gwbc studio
```

**9. Press Ctrl+N and enter ABConfigurationProvider, and then click the class name in the search results to open it in the editor.**

**10. Change the Username and Password definition in the configure method. If you see a message asking if you want to edit the class, click Yes.**

For example, change the values to **BCNewAuthUser** and **Xc8899Lm**, as follows:

```
override function configure( serviceName : QName, portName : QName, config : WsdlConfig ) {
    config.Guidewire.Authentication.Username = "BCNewAuthUser"
    config.Guidewire.Authentication.Password = "Xc8899Lm"
}
```

**11. In the Project window, navigate to configuration → gsrc and then to wsi.remote.gw.webservice.ab.ab801.wsc.**

**12. Double-click the ab801.wsc web services collection to open the editor.**

13. In the editor, click \${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl.
14. With ContactManager still running, click Fetch Updates.  
For more information on this editor, see “Using the WS-I Web Service Editor” on page 122 in the *Configuration Guide*.
15. Save your changes, and then stop BillingCenter if it is running and restart it to pick up this change.
  - a. Open a command prompt in the BillingCenter/bin directory and then enter the following command:  
`gwbc dev-stop`
  - b. After the application stops, enter the following command to restart it:  
`gwbc dev-start`
16. Log in to BillingCenter as a user with permission to search for and change contacts, such as user name su with password gw.
17. Verify that you can use the Contact tab to search for ContactManager contacts. You can be sure that a contact is stored in ContactManager if its External field has the value Yes. For example, search for the person Adam Hinds, a client contact in the ContactManager sample data.

## Configuring ContactManager Authentication with Core Applications

This topic shows you how to change the user names and passwords that ContactManager uses to send ABContact updates to the core applications. In the base configuration, ContactManager defines this user as su, a user that by default has all permissions to perform any action in a core application. Guidewire recommends for security reasons that you change this default setting to a user with permission only to create, update, and delete contacts.

This topic includes:

- “Overview of ContactManager Authentication Configuration” on page 73
- “Configuring ContactManager-to-ClaimCenter Authentication” on page 74
- “Configuring ContactManager-to-PolicyCenter Authentication” on page 76
- “Configuring ContactManager-to-BillingCenter Authentication” on page 79

### Overview of ContactManager Authentication Configuration

Initially, to set up ContactManager to send ABContact updates to a Guidewire core application, you open ContactManager Studio and register a plugin with the core application’s plugin registry. For example:

- To communicate with ClaimCenter 8.0, you register `gw.plugin.claim.cc800.CCClaimSystemPlugin` with the `ClaimSystemPlugin` plugin registry. See “Integrating ContactManager with ClaimCenter in QuickStart” on page 46.
- To communicate with PolicyCenter 8.0, you register `gw.plugin.policy.pc800.PCPolicySystemPlugin` with the `PolicySystemPlugin` registry. See “Integrating ContactManager with PolicyCenter in QuickStart” on page 52.
- To communicate with BillingCenter 8.0, you register `gw.plugin.billing.bc800.BCBillingSystemPlugin` with the `BillingSystemPlugin` registry. See “Integrating ContactManager with BillingCenter in QuickStart” on page 59.

ContactManager uses these plugins to make calls to core application web services that implement `ABCClientAPI`.

ContactManager defines core application authorization for each web service in a configuration provider class, which is associated with the web service in a web service collection. To see the web services that ContactManager uses with a core application, you open the editor for the web service collection. For general information on this editor, see “Using the WS-I Web Service Editor” on page 122 in the *Configuration Guide*.

ContactManager provides the following web collections for each core application:

- ContactManager uses the web service collection cc800.wsc to manage web services it uses with ClaimCenter 8.0. You can find cc800.wsc with CTRL+SHFT+N, and then double-click the search result to open the editor. In the editor, select the web service \${cc}/ws/gw/webservice/cc/cc800/contact/ContactAPI?wsdl1. At the bottom of the editor in the **Settings** tab is the configuration provider class. This class, `wsi.remote.gw.webservice.cc.CCConfigurationProvider`, defines the user name and password that ContactManager uses to connect with ClaimCenter.
- ContactManager uses the web service collection pc800.wsc to manage web services it uses with PolicyCenter 8.0. You can find pc800.wsc with CTRL+SHFT+N, and then double-click the search result to open the editor. In the editor, select the web service \${pc}/ws/gw/webservice/pc/pc800/contact/ContactAPI?wsdl1. At the bottom of the editor in the **Settings** tab is the configuration provider class. This class, `wsi.remote.gw.webservice.pc.PCConfigurationProvider`, defines the user name and password that ContactManager uses to connect with PolicyCenter.
- ContactManager uses the web service collection bc800.wsc to manage web services it uses with BillingCenter 8.0. You can find bc800.wsc with CTRL+SHFT+N, and then double-click the search result to open the editor. In the editor, select the web service \${bc}/ws/gw/webservice/bc/bc800/contact/ContactAPI?wsdl1. At the bottom of the editor in the **Settings** tab is the configuration provider class. This class, `wsi.remote.gw.webservice.bc.BCConfigurationProvider`, defines the user name and password that ContactManager uses to connect with BillingCenter.

In general, you change the user name and password as follows:

1. Set up a user with a name and password in the core application that matches the user name and password you define in the configuration provider class in ContactManager.
2. Give the user a role with limited permissions, with at least the permissions to create, edit, and delete contacts.
3. In ContactManager, define the user name and password in the configuration provider class for the core application web service.

## Configuring ContactManager-to-ClaimCenter Authentication

To change the user name and password that ContactManager uses to authenticate with ClaimCenter:

1. Start ClaimCenter.

At a command prompt, navigate to ClaimCenter/bin and enter the following command:  
`gwcc dev-start`

2. Log in as a user who can create new ClaimCenter users and roles. For example, log in with user name `su` and password `gw`.
3. Create a user role with all the permissions that enable working with contacts whose contact information is stored in ContactManager. In the base configuration, the minimum required permission codes are `abcreate`, `abcreatepref`, `anytagcreate`, `ctccreate`, `abdelete`, `abdeletepref`, `anytagdelete`, `abedit`, `abeditpref`, `anytagedit`, `ctcedit`, and `soapadmin`.
  - a. Click the **Administration** tab and then, in the left Sidebar under **Actions**, click **Roles**.
  - b. On the **Roles** screen, click **Add Role**.
  - c. For **Name** enter **Contact Data Admin**.
  - d. For **Type**, choose **User Role**.
  - e. For **Description** enter **Permissions for virtual user required by ContactManager for updating contacts**.
  - f. Beneath the **Description** field, click **Add**.
  - g. Click the **Permission** field and choose **Create address book contacts** from the list.

- h.** Repeat the process for the following permissions. For each permission, click **Add** and choose the permission from the drop-down list:

- Create address book preferred vendors
- Create contact with any tag
- Create local contacts
- Delete address book contacts
- Delete address book preferred vendors
- Delete contact with any tag
- Edit address book contacts
- Edit address book preferred vendors
- Edit contact with any tag
- Edit local contacts
- SOAP administration

- i.** Click **Update** to create the new Contact Data Admin role.

- 4.** Choose **Actions → New User**.

- 5.** Enter the following values for a new user named CMUpdateCC:

Name	Value
First Name	CMUpdate
Last Name	CC
Username	CMUpdateCC
Password	Xc8899Lm
Confirm Password	Xc8899Lm
Active	Yes
Locked	No

- 6.** Under Roles, click **Add**.

- 7.** Click the Name field and choose Contact Data Admin.

**Note:** If this role has more permissions than you would like, you can create a new role and assign the new user to that role. The new role must have at least the following permissions: abcreate, abdelete, abedit, abview, abcreatepref, abdeletepref, abeditpref, abviewsearch, anytagcreate, anytagdelete, anytagedit, anytagview, ctccreate, ctcedit, and ctcview. For more information on setting up roles, see “Role-Based Security” on page 448 in the *Application Guide*.

- 8.** Click **Update**.

- 9.** Start ContactManager Studio.

At a command prompt, navigate to **ContactManager/bin** and enter the following command:

```
gwab studio
```

- 10.** In the Project window, navigate to **configuration → gsrc**, and then navigate to **wsi.remote.gw.webservice.cc.CCConfigurationProvider**.

- 11.** Double-click **CCConfigurationProvider** to open it in the editor.

- 12.** Change the Username and Password definition in the **configure** method and then save your changes.

For example, define Username and Password to be CMUpdateCC and Xc8899Lm, as follows:

```
override function configure( serviceName : QName,
                           portName : QName,
                           config : WsdlConfig ) {
    config.Guidewire.Authentication.Username = "CMUpdateCC"
    config.Guidewire.Authentication.Password = "Xc8899Lm"
}
```

13. In the editor, the cc800.wsc web services collection is at the same level as the CCConfigurationProvider class. Double-click cc800.wsc.  
Alternatively, you can press **Ctrl+Shift+N** and enter cc800, and then double-click cc800.wsc in the search results.
14. In the editor, select the following resource:  
 `${cc}/ws/gw/webservice/cc/cc800/contact/ContactAPI?wsdl`
15. Ensure that the **Settings** tab has the **Setting Type** for  
`!WebserviceCollection.SettingType.ConfigurationProvider!` set to  
`wsi.remote.gw.webservice.cc.CCConfigurationProvider`, the class you just edited.
16. With ClaimCenter still running, above the **Settings** tab, click **Fetch Updates** to update the web service's WSDL file.
17. Save your changes, and then stop ContactManager if it is running and restart it to pick up this change.
  - a. Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`
  - b. After the application stops, enter the following command to restart it:  
`gwab dev-start`
18. Log in to ClaimCenter as a user with permission to work with the Address Book, such as user `ssmith` with password `gw`.
19. Add to a claim a contact that is stored in ContactManager. For example, open a claim, click **Parties Involved**, and on the **Contacts** screen, click **Add Existing Contact**. Search for a contact that you know is stored in ContactManager, like the sample contact Stephen Marshall. Select the contact and give the contact a role on the claim, and then click **Update**.
20. Log in to ContactManager as a user who can work with contacts. For example, log in as the sample user `aapplegate` with password `gw`.
21. Make a change to the contact that you added to the claim, such as a different phone number, and save it.
22. Back in ClaimCenter, on the **Contact** screen, select another contact, and then select the contact you added previously.
23. On the **Basics** card for that contact is the message, **This contact is linked to the Address Book but is out of sync**. Click **Copy from Address Book** to update the contact.

## Configuring ContactManager-to-PolicyCenter Authentication

To change the user name and password that ContactManager uses to authenticate with PolicyCenter:

1. Start PolicyCenter.  
At a command prompt, navigate to `PolicyCenter/bin` and enter the following command:  
`gwpc dev-start`
2. Log in as a user who can create new PolicyCenter users and roles. For example, log in with user name `su` and password `gw`.
3. Create a user role with all the permissions that enable working with clients whose contact information is stored in ContactManager. The permission codes in the base configuration are `abcreate`, `ctccreate`, `anytagcreate`, `abdelete`, `anytagdelete`, `abedit`, `anytagedit`, `ctcedit`, and `soapadmin`.
  - a. Click the **Administration** tab and then, in the left **Sidebar** under **Actions**, click **Roles**.
  - b. On the **Roles** screen, click **New Role**.
  - c. For **Name** enter **Client Data Admin**.

- d. For Type, choose User Role.
- e. For Description enter Permissions for virtual user required by ContactManager for updating contacts.
- f. Under Permissions, click Add.
- g. Click the Permission field and choose Create address book contacts from the list.
- h. Repeat the process for the following permissions. For each permission, click Add and choose the permission from the drop-down list:
  - Create contact with any tag
  - Create local contacts
  - Delete address book contacts
  - Delete contact with any tag
  - Edit address book contacts
  - Edit contact with any tag
  - Edit local contacts
  - SOAP administration
- i. Click Update to create the new Client Data Admin role.

4. On the Administration tab, choose Actions → New User.

5. Enter the following values for a new user named CMUpdatePC:

Name	Value
First Name	CMUpdate
Last Name	PC
Username	CMUpdatePC
Password	Xc8899Lm
Confirm Password	Xc8899Lm
Active	Yes
Locked	No
Vacation Status	At work
User Type	Other
Primary Phone	Work
Work Phone	800-555-5555

**Note:** Primary Phone and Work Phone are required for a new user, but they do not require real values for this particular user.

6. Click the Roles tab, and then click Add.

7. Click the Name field and choose Client Data Admin.

8. Click Update.

9. Start ContactManager Studio.

At a command prompt, navigate to ContactManager/bin and enter the following command:

```
gwab studio
```

10. In the Project window, navigate to configuration → gsrc, and then navigate to wsi.remote.gw.webservice.pc.PCConfigurationProvider.

11. Double-click PCConfigurationProvider to open it in the editor.

12. Change the Username and Password definition in the configure method. If you see a message asking if you want to edit the class, click Yes.

For example, define Username and Password to be CMUpdatepc and Xc8899Lm, as follows:

```
override function configure( serviceName : QName,
                            portName : QName,
                            config : WsdlConfig ) {
    config.Guidewire.Authentication.Username = "CMUpdatePC"
    config.Guidewire.Authentication.Password = "Xc8899Lm"
}
```

**13.** Save the changed file.

**14.** In the Project window, the pc800.wsc web services collection is at the same level as the PCConfigurationProvider class. Double-click pc800.wsc.

Alternatively, you can press **Ctrl+Shift+N** and enter pc800, and then double-click pc800.wsc in the search results.

**15.** In the editor, select the following resource:

```
 ${pc}/ws/gw/webservice/pc/pc800/contact/ContactAPI?wsdl
```

**16.** Ensure that the **Settings** tab has the **Setting Type** for `!WebserviceCollection.SettingType.ConfigurationProvider!` set to `wsi.remote.gw.webservice.pc.PCConfigurationProvider`, the class you just edited.

**17.** With PolicyCenter still running, above the **Settings** tab, click **Fetch Updates** to update the web service's WSDL file.

**18.** Save your changes, and then stop ContactManager if it is running and restart it to pick up this change.

**a.** Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`

**b.** After the application stops, enter the following command to restart it:  
`gwab dev-start`

**19.** Log in to PolicyCenter as a user with permission to work with the **Contact** tab and with accounts, such as the user `aapplegate` with password `gw`.

**20.** Add a contact stored in ContactManager to an active account. For example:

**a.** Open an active account, such as the sample data account S000212121 named Armstrong Cleaners, and, in the Sidebar on the left under **Actions**, click **Contacts**. The **Account File Contacts** screen opens.

**b.** Click **Create New Contact** → **Additional Insured** → **From Address Book**.

**c.** Search for a contact who has the **Client** tag set in ContactManager, such as the sample person contact `Mark Stone`.

**d.** Click **Select** for a contact that the search results list says is stored in ContactManager. The value of the contact's **External** field must be **Yes**.

**e.** PolicyCenter adds the selected contact to the list of contacts on the **Account File Contacts** screen.

**21.** Log in to ContactManager as a user who can work with contacts. For example, log in as the sample user `aapplegate` with password `gw`.

**22.** Make a change to the contact that you added to the account, such as entering a different primary phone number, and then click **Update** to save the change.

**23.** Back in PolicyCenter, on the **Account File Contacts** screen, select a different contact from the one you added. PolicyCenter will update the added contact's data when you select that contact again.

**24.** Select the contact you added and ensure that the data has changed for that contact.

## Configuring ContactManager-to-BillingCenter Authentication

To change the user name and password that ContactManager uses to authenticate with BillingCenter:

**1. Start BillingCenter.**

At a command prompt, navigate to `BillingCenter/bin` and enter the following command:

```
gwbc dev-start
```

**2. Log in as a user who can create new BillingCenter users.** For example, log in with user name `su` and password `gw`.

**3. Create a user role with all the permissions that enable working with clients whose contact information is stored in ContactManager.** The permission codes are `acctcntcreate`, `acctcntdelete`, `acctcntedit`, `p1cycntcreate`, `p1cycntdelete`, `p1cycntedit`, `prodcntcreate`, `prodcntdelete`, `prodcntedit`, and `soapadmin`.

- a. Click the **Administration** tab and then, in the Sidebar on the left under **Actions**, click **Roles**.
- b. On the **Roles** screen, click **New Role**.
- c. For **Name** enter **Client Data Admin**.
- d. For **Description** enter **Permissions for virtual user required by ContactManager for updating contacts**.
- e. Below the **Description** field, click **Add**.
- f. Click the **Permission** field and choose **Create account contact** from the list.

- g. Repeat the process for the following permissions. For each permission, click **Add** and choose the permission from the drop-down list:

Create policy contact  
Create producer contact  
Delete account contact  
Delete policy contact  
Delete producer contact  
Edit account contact  
Edit policy contact  
Edit producer contact  
SOAP administration

- h. Click **Update** to create the new Client Data Admin role.

**4. On the Administration tab, choose Actions → New User.**

**5. Enter the following values for a new user named CMUpdateBC:**

Name	Value
First Name	CMUpdate
Last Name	BC
Username	CMUpdateBC
Password	Xc8899Lm
Confirm Password	Xc8899Lm

**6. Click Add User Role.**

**7. Click the Name list and choose Client Data Admin.**

**8. Click Next.** and add some fake data for the required fields **Address**, **City**, **Country**, **Primary Phone**, and the telephone number for the type of primary phone you selected.

**9. Click Next.**

**10.** Click **Finish**.

**11.** Start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:

```
gwab studio
```

**12.** In the Project window, navigate to `configuration → gsrc`, and then navigate to `wsi.remote.gw.webservice.bc.BCConfigurationProvider`.

**13.** Double-click `BCConfigurationProvider` to open it in the editor.

**14.** Change the `Username` and `Password` definition in the `configure` method. If you see a message asking if you want to edit the class, click **Yes**.

For example, define `Username` and `Password` to be `CMUpdateBC` and `Xc8899Lm`, as follows:

```
override function configure( serviceName : QName,
                            portName : QName,
                            config : WsdlConfig ) {
    config.Guidewire.Authentication.Username = "CMUpdateBC"
    config.Guidewire.Authentication.Password = "Xc8899Lm"
}
```

**15.** Save the changed file.

**16.** In the Project window, the `bc800.wsc` web services collection is at the same level as the `BCConfigurationProvider` class. Double-click `bc800.wsc`.

Alternatively, you can press `Ctrl+Shift+N` and enter `bc800`, and then double-click `bc800.wsc` in the search results.

**17.** In the editor, select the following resource:

```
 ${bc}/ws/gw/webservice/bc/bc800/contact/ContactAPI?wsdl
```

**18.** Ensure that the **Settings** tab has the **Setting Type** for `!WebserviceCollection.SettingType.ConfigurationProvider!` set to `wsi.remote.gw.webservice.bc.BCConfigurationProvider`, the class you just edited.

**19.** With BillingCenter still running, above the **Settings** tab, click **Fetch Updates** to update the web service's WSDL file.

**20.** Save your changes, and then stop ContactManager if it is running and restart it to pick up this change.

**a.** Open a command prompt in the `ContactManager/bin` directory and then enter the following command:

```
gwab dev-stop
```

**b.** After the application stops, enter the following command to restart it:

```
gwab dev-start
```

**21.** Log in to BillingCenter as a user with permission to work with the **Contacts** screen for accounts, such as the user `su` with password `gw`.

**22.** Add a contact stored in ContactManager to an active account. For example:

**a.** Open an active account, such as the sample account named Standard Account, and, in the Sidebar on the left under **Actions**, click **Contacts**. The **Contacts** screen opens.

**b.** Click **Edit**, and then click **Add Existing Contact**.

**c.** Search for a contact that has the **Client** tag set in ContactManager, such as the sample person contact Stan Newton. If necessary, log in to ContactManager and add a **Client** tag to a contact.

**d.** Click **Select** for a contact that the search results list says is stored in ContactManager. The value of the contact's **External** field must be **Yes**.

**e.** BillingCenter adds the selected contact to the list of contacts on the **Contacts Edit** screen.

**f.** Click **Update** to add the contact to the account.

- 23.** Log in to ContactManager as a user who can work with contacts. For example, log in as the sample user aapplegate with password gw.
- 24.** Make a change to the contact that you added to the account, such as entering a different address, and click **Update** to save the change.
- 25.** Back in BillingCenter, on the Contacts screen select the contact you added and ensure that the data has changed for that contact.



# Searching for Contacts

This topic provides the following information:

- How search works with contacts in Guidewire core applications and in ContactManager.
- Configuration points for search in ContactManager and the core applications.
- How to add a search criterion to ClaimCenter contact search.
- How to configure ClaimCenter to perform a vendor proximity search.

This topic includes:

- “Overview of Contact Search” on page 83
- “ContactManager Support for Contact Searches” on page 84
- “ClaimCenter Support for Contact Searches” on page 87
- “PolicyCenter Support for Contact Searches” on page 106
- “BillingCenter Support for Contact Searches” on page 107

## Overview of Contact Search

Guidewire core applications store contacts locally, and, if integrated with ContactManager, a core application can also store contacts centrally in the ContactManager database. PolicyCenter and BillingCenter support searching for contacts both locally and in an external contact management application, like ContactManager. ClaimCenter supports searching for contacts only in an external contact management application, like ContactManager.

For example, in ClaimCenter, if you have added a contact to a claim, you can see a list of contacts by navigating to the **Parties Involved** → **Contacts** screen. All the contacts on the list are stored locally with the claim. Contacts can also be stored in ContactManager. If you click a contact in the list, you see its detailed information below the list. If the contact is stored in ContactManager, at the top of the **Basics** card is a message starting with the text, **This contact is linked to the Address Book...**

In ClaimCenter, you can search for a contact in the **Address Book** by clicking the **Address Book** tab and entering your search criteria. All contact data returned by this search comes from contacts stored in ContactManager. You can

also search for contacts when creating existing contacts for claims. For example, with a claim open, you can navigate to the **Parties Involved** → **Contacts** screen and click **Add Existing Contact**.

In PolicyCenter, you can search for a contact by clicking the **Contact** tab and then entering your search criteria. Alternatively, you can click the **Search** tab menu and choose **Contact**. Additionally, you can open the **Contacts** screen for an account or policy and select a contact. Then you can click **Add Existing Contact**, choose a contact type, and then choose **From Address Book** to open the contact search screen.

In BillingCenter, you can search for a contact by clicking the **Search** tab menu and choosing **Contacts**, and then entering your search criteria. Additionally, if you edit the **Contacts** screen for an account or policy period, clicking **Add Existing Contact** opens the contact search screen.

Searching for contacts in PolicyCenter and BillingCenter returns information both about locally stored contacts and about contacts stored in an external contact management system, such as ContactManager. The list of search results has an **External** field that provides the following information:

- If the value in this field is **Yes**, the contact is not local, but is stored in an external contact management system.
- If the value in this field is **No**, the contact is stored locally and might also be stored in an external contact management system.

In all cases, searching for contacts returns a subset of information about the contact, which you see listed in the set of returned contacts.

---

**IMPORTANT** Limiting the fields retrieved by a search improves performance. In the base configuration, ContactManager filters search results to reduce the size of the objects returned to the calling application. To avoid performance issues, ensure that your searches are optimized where possible to return only the parts of objects that you need and not the full objects.

---

The following topics describe how contact search works between the core applications and ContactManager:

- “ContactManager Support for Contact Searches” on page 84
- “ClaimCenter Support for Contact Searches” on page 87
- “PolicyCenter Support for Contact Searches” on page 106
- “BillingCenter Support for Contact Searches” on page 107

## ContactManager Support for Contact Searches

ContactManager supports contact search that is initiated both locally, from its own user interface, and externally, through web services. External searches most commonly originate from Guidewire core applications.

This topic includes:

- “Local ContactManager Search Criteria” on page 85
- “Defining Minimum Search Criteria for a Contact” on page 85
- “ContactManager Support for Core Application Searches” on page 86

## Local ContactManager Search Criteria

If you run ContactManager and log in as a user who has permission to search for and view contacts, you can search for contacts by clicking the **Contacts** tab. The criteria you use for searches on this screen are defined in ContactManager's `search-config.xml` file and in the entity `ABContactSearchCriteria`.

**IMPORTANT** There are restrictions on adding new `CriteriaDef` elements. See “The `<CriteriaDef>` Element” on page 349 in the *Configuration Guide*. Additionally, there are limitations on how you can use the `ArrayCriterion` element. See “The `<ArrayCriterion>` Subelement” on page 354 in the *Configuration Guide*.

### See also

- For an example, see “Step 1: Add Search Capability in ContactManager” on page 88.
- For complete details on the `search-config.xml` file, see “Configuring ClaimCenter Database Search” on page 348 in the *Configuration Guide*.

## Defining Minimum Search Criteria for a Contact

There is a plugin class in the base configuration of ContactManager in which you can define the minimum criteria required to find a contact, `gw.plugin.contact.ValidateABContactSearchCriteriaPluginImpl`. Settings in this class affect both local searches and searches from Guidewire core applications. You can add new, required search fields by editing this class. Because the class is written in Gosu, you have considerable flexibility in how you specify which fields to require for a search.

The following code snippet from this class shows the default search criteria defined for an `ABPerson` entity:

```
protected function abPersonCanSearch(searchCriteria : ABContactSearchCriteria) : boolean {
    if (searchCriteria.FirstName != null || searchCriteria.FirstNameKanji != null) {
        if (searchCriteria.Keyword == null & searchCriteria.KeywordKanji == null) {
            return false
        }
    }
    if (searchCriteria.Keyword == null
        and searchCriteria.KeywordKanji == null
        and searchCriteria.FirstName == null
        and searchCriteria.FirstNameKanji == null
        and searchCriteria.TaxID == null
        and satisfiesNoLocaleSpecificCriteriaRequirements(searchCriteria)
        and searchCriteria.Address.PostalCode == null
        and not searchCriteria.isValidProximitySearch()) {
        return false
    }
    return true
}
```

This topic includes:

- “Effect of Locale and Country on Minimum Search Criteria” on page 85
- “Minimum Search Criteria Error Messages” on page 86

### Effect of Locale and Country on Minimum Search Criteria

Different locales have different requirements for searches. ContactManager determines the locale for the name fields of a search and uses the country specified to determine the other fields required.

The following code snippet from `ValidateABContactSearchCriteriaPluginImpl.ValidateCanSearch` shows the code that picks up the locale and the country that the user specified in the search:

```
var country = searchCriteria.Address.Country ?: gw.api.admin.BaseAdminUtil.getDefaultCountry()
if (searchSpec != null && searchSpec.Locale == null)
    searchSpec.Locale = LocaleType.get(PLConfigParameters.DefaultApplicationLocale.Value)
}
```

**Note:** If the country or the locale or both are not passed in, the method uses default values.

The `ValidateCanSearch` method throws a `TooLooseContactSearchCriteriaException` if the search criteria are not met:

```
var exception = new TooLooseContactSearchCriteriaException(  
    searchCriteria.ContactSubtypeType, country, searchSpec.Locale)  
em = exception.Message
```

The message sent uses the locale and country to send the appropriate search criteria back to the calling application. The next topic describes how these messages are defined.

## Minimum Search Criteria Error Messages

The `ValidateABContactSearchCriteriaPluginImpl` class uses the `TooLooseContactSearchCriteriaException` message definitions to pass locale and country specific error messages to the calling application. In the base configuration, these messages are display keys in the `en_US_display.properties` file.

To see this file, open ContactManager Studio and navigate in the Project window to `configuration → config → Localizations` and then to `en_US → display.properties`. Double-click `display.properties` to open this file in the editor.

In the file, if you press `Ctrl+F` and search for `TooLooseContactSearchCriteriaException`, you see the set of display keys defined for this exception. For example:

```
Java.TooLooseContactSearchCriteriaException.ABCompany.AU.ja_JP =  
    Please specify one of the following\: Name (phonetic), Name, Tax ID, Postcode, or City and State
```

The display keys are grouped first by entity type, `ABCompany`, `ABContact`, `ABPerson`, and `ABPlace`. In each set, there are display keys for the countries Australia (AU), Canada (CA), Germany (DE), France (FR), Great Britain (GB), Japan (JP), and the United States (US). For each country, there is a display key for one locale, `ja_JP`.

**Note:** You can add display keys for additional countries, locales, and entity types. If you add additional country or locale display keys, be sure to add them for all entity types, which in the base configuration are `ABCompany`, `ABContact`, `ABPerson`, and `ABPlace`.

This classification of messages enables ContactManager to return different contact search error messages based on country and locale. For example, a ClaimCenter user searches for a `MedicalCareOrg`, a `Company` subtype, located in the United States. The user has previously set the locale to `ja_JP` (Japan). ContactManager determines that there are not enough search criteria to find a company in that country and locale, and sends an exception message defined by the following display key:

```
Java.TooLooseContactSearchCriteriaException.ABCompany.US.ja_JP =  
    Please specify one of the following\: Name (phonetic), Name, Tax ID, Zip Code, or City and State
```

**Note:** Each entity has one display key that does not specify a country or locale. This display key is a default in case the country or locale is not specified. Additionally, the display keys that have only a country and no locale are the default display keys for all locales that are not defined. In the base configuration, the only locale for which a message is defined is `ja_JP`.

## ContactManager Support for Core Application Searches

The ContactManager web service `gw.webservice.ab.ab801.abcontactapi`.`ABContactAPI` provides methods that the Guidewire 8.0 core applications call to search for, create, retrieve, update, and delete contacts. Additionally, this class provides a method for finding vendor contacts by associated service. The method that supports search is `ABContactAPI.searchContact`. This class is read-only and cannot be directly edited. For more information on this class, see “`ABContactAPI` Web Service” on page 275.

You configure search in other classes and in a configuration parameter.

## Configuring Contact Search Criteria and Search Results in ContactManager

The Gosu class `ABContactAPISearchCriteria` specifies the contact search criteria that the Guidewire core applications can use. The package for this class is `gw.webservice.ab.ab801.abcontactapi`.

You can edit this class to add new search criteria for core application contact searches. See “Step B: Adding Search Support in ContactManager for Guidewire Core Applications” on page 89.

You can also define the fields included in the search results that ContactManager sends back to a Guidewire core application. To add a field to the search results, add code for the field to the Gosu class `gw.webservice.ab.ab801.abcontactapi.ABContactAPISearchResult`. For more information, see “Step B: Adding Search Support in ContactManager for Guidewire Core Applications” on page 89.

These classes, `ABContactAPISearchCriteria` and `ABContactAPISearchResult`, are classes used as parameters in the web service API. After changing either of these classes, restart ContactManager to regenerate these WS-I web services. Then start Studio for the Guidewire core application, navigate to the `ab801.wsc` web collection, and fetch updates for the ContactManager web service `ABContactAPI`.

### [Limiting the Number of Service Elements Specified in a Contact Search](#)

If there are too many services specified in a contact search, the search can fail. To limit this number, there is a ContactManager configuration parameter, `MaxNumberServicesInSearchQuery`, that you can set in `config.xml`. This configuration parameter defaults to a value of 20. If a contact search specifies more services than the number set in this parameter, ContactManager sends an error message to the core application.

If `MaxNumberServicesInSearchQuery` is set to too large a number, ContactManager can experience SQL errors. In that case, ContactManager returns an error message to the core application with a note to check the logs in ContactManager for the details of the error.

## [ClaimCenter Support for Contact Searches](#)

There are two files defining contact search criteria that ClaimCenter sends to ContactManager, the entity `ContactSearchCriteria.etc` and the Gosu class `ContactSearchMapper`. The entity defines the search criteria that appear in the search screens, and the Gosu class maps the search criteria to ContactManager. ClaimCenter uses these criteria when it calls the ContactManager `ABContactAPI` method `searchContacts`. ClaimCenter calls this method in the plugin `ABContactSystemPlugin`, which implements `ContactSystemPlugin`.

There is a third Gosu class, `ContactSearchResultMapper`, that `ABContactSystemPlugin` uses to map fields sent from ContactManager search results so that ClaimCenter can display them in the lists of search results.

**Note:** ClaimCenter displays `Contact` entities in the search results. Therefore, the result mapper maps from ContactManager results to `Contact` entities.

The files are:

- `ContactSearchCriteria.etc` – Defines the search criteria that are shown to the user in the search screens. If you add entries to `ContactSearchCriteria`, you must also add them to `ContactSearchMapper`. ClaimCenter uses both files.
- `gw.plugin.addressbook.ab800.ContactSearchMapper` – Defines how search criteria map to ContactManager `ABContact` search criteria.
- `gw.plugin.addressbook.ab800.ContactSearchResultMapper` – Defines the search results received from ContactManager to display in the search results screen.

These files work in concert with the ContactManager search files described under “ContactManager Support for Core Application Searches” on page 86.

This topic includes:

- “Adding the InterpreterSpecialty Property to Search” on page 88
- “Adding an Address Field to Contact Search” on page 93
- “Adding the Service State Property to Search” on page 94
- “Geocoding and Proximity Searches for Vendor Contacts” on page 99

## Adding the InterpreterSpecialty Property to Search

This topic uses the example subtype ABInterpreter. This subtype of ABPersonVendor has one field, `InterpreterSpecialty`. For this example to work, you must complete all the steps in “Example of Adding a Vendor Contact Subtype” on page 147.

**IMPORTANT** To improve search performance, Guidewire recommends that you add an index in the `ContactManager` entity definition for every field used in the search. `ABInterpreter` does have an index defined.

This topic includes:

- “Step 1: Add Search Capability in `ContactManager`” on page 88
- “Step 2: Restart `ContactManager` and Refresh the Web Services in `ClaimCenter`” on page 90
- “Step 3: Add `ContactManager` Search Capability in `ClaimCenter`” on page 90
- “Step 4: Configure the Address Book Search Interface in Both Applications” on page 91
- “Step 5: Test Your Changes” on page 93

### Step 1: Add Search Capability in `ContactManager`

There are two primary aspects to adding search capability in `ContactManager`:

- Adding search support for the field to the `ContactManager` user interface through settings in `search-config.xml` and in the `ABContactSearchCriteria` entity. For a complete description of `search-config.xml`, see “Configuring `ClaimCenter` Database Search” on page 348 in the *Configuration Guide*.
- Adding search support for Guidewire core applications in the Gosu class `ABContactAPISearchCriteria`. You can also have the search field show in the search results that `ContactManager` sends back to the Guidewire core application. To add the field to the search results, add code for the field to the Gosu class `ABContactAPISearchResult`.

**Note:** These classes are used as parameters in web service calls, and changing them results in a WSDL change. After changing one of these classes, you must restart `ContactManager`. Then, in Studio for the Guidewire core application, you must refresh the `ContactManager` web services.

#### Step A: Adding Search Support to the `ContactManager` User Interface

In this series of steps, you make changes that affect only the `ContactManager` user interface. You add `InterpreterSpecialty` to `ABContactSearchCriteria.etcx`. Then you add a `CriteriaDef` element to the `ContactManager` `search-config.xml` file to specify where the `InterpreterSpecialty` column is located.

1. In `ContactManager` Studio, in the `Project` window, navigate to `configuration` → `config` → `Extensions` → `Entity`, and then double-click `ABContactSearchCriteria.etcx`.
2. In the editor, at the top of the `Element` hierarchy, click `nonPersistentEntity (extension)`.
3. Click the drop-down list next to  and choose `column`.  
You use the `column` element rather than `typekey` because the data type is `varchar` and is not a `typekey`.
4. To the new `column`, add the following values that support searches for `InterpreterSpecialty`.

Name	Value
name	InterpreterSpecialty
type	varchar
nullok	true
desc	Interpreter language specialties

5. Click the drop-down list next to  and choose **params**.
6. Enter the following values to specify the size of this varchar column:

Name	Value
name	size
value	30

7. In the Project window, navigate to **configuration → config → search** and double-click **search-config.xml** to edit the file.

8. 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">
```

9. Find the following comment:

```
<!-- Specific fields in ABContact subtypes -->
```

10. After the last **CriteriaDef** element in this group, add a new **CriteriaDef** element for the target entity **ABInterpreter** that tests **InterpreterSpecialty** for equality:

```
<CriteriaDef entity="ABContactSearchCriteria" targetEntity="ABInterpreter">
    <Criterion property="InterpreterSpecialty" matchType="eq"/>
</CriteriaDef>
```

11. Save the file.

### Step B: Adding Search Support in ContactManager for Guidewire Core Applications

In this topic, you add code for the new field to the Gosu class **ABContactAPISearchCriteria**. This Gosu object is populated by the core application to pass a search criterion to ContactManager.

Additionally, for this example, the new field will be added to the search results that ContactManager sends back to the Guidewire core application. To add the field, you enter code for the field in the Gosu class **ABContactAPISearchResult**.

1. Add an entry for **InterpreterSpecialty** to the class **ABContactAPISearchCriteria**.

a. In ContactManager Studio, navigate in the Project window to **configuration → gsrc** and then to **gw.webservice.ab.ab801.abcontactapi**.

b. Double-click **ABContactAPISearchCriteria** to open the class in the editor.

c. Add the following variable definition to the list of variables at the beginning of the class.

```
public var InterpreterSpecialty : String
```

d. In the method **toSearchCriteria**, before **:AllTagsRequired**, add the following entry:

```
:InterpreterSpecialty = this.InterpreterSpecialty,
```

2. If you want the field to be added to the search results sent back to a Guidewire core application, add an entry for **InterpreterSpecialty** to the class **ABContactAPISearchResult**.

a. Press **Ctrl+N** and enter **ABContactAPISearchResult**.

b. Click the class with package **gw.webservice.ab.ab801.abcontactapi** in the search results to open the class in the editor.

c. Add the following variable definition to the list of variables at the beginning of the class.

```
public var InterpreterSpecialty : String
```

- d. In the constructor `construct(contact : ABContact)`, find the `if` statement block for `ABPerson`:

```
if (contact typeis ABPerson) {
```

- e. After that `if` statement block, add a new `if` statement for `ABInterpreter`, as follows:

```
// Search results for ABInterpreter
if (contact typeis ABInterpreter) {
    this.InterpreterSpecialty = contact.InterpreterSpecialty
}
```

## Step 2: Restart ContactManager and Refresh the Web Services in ClaimCenter

At this point, ClaimCenter does not have the changes you made in ContactManager. You have to stop and then restart ContactManager to force regeneration of the WS-I SOAP APIs and make the changes available. Before restarting ContactManager, you regenerate the ContactManager data dictionary to ensure that your changes to the entity `ABContactSearchCriteria`. are valid. If they are, then you restart ContactManager and, in ClaimCenter studio, you refresh the ContactManager web services.

1. Open a command prompt in the `ContactManager/bin` directory and then enter the following command:

```
gwab dev-stop
```

2. Regenerate the dictionaries:

```
gwab regen-dictionary
```

3. If the dictionary regenerates without errors, start the ContactManager application:

```
gwab dev-start
```

4. In ClaimCenter Studio, refresh the ContactManager plugin.

- a. Press `Ctrl+Shift+N` and enter `ab801.wsc` to find that web service collection, and then double-click the file in the search results to open it in the editor.

- b. In the editor, select the following resource:

```
 ${ab}/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl
```

- c. At the bottom of the pane, click **Fetch Updates** to update the WSDL for this web service.

## Step 3: Add ContactManager Search Capability in ClaimCenter

This topic shows you how to set search criteria and search results in ClaimCenter that work with ContactManager. The topic uses a new Contact subtype named `Interpreter` that has one field, `InterpreterSpecialty`. For more information on this subtype, see “Example of Adding a Vendor Contact Subtype” on page 147.

1. To support search on the `InterpreterSpecialty` field, add it to the entity `ContactSearchCriteria.etc`.

- a. In ContactManager Studio, in the Project window, navigate to `configuration` → `config` → `Extensions` → `Entity`, and then double-click `ContactSearchCriteria.etc`.

- b. In the editor, at the top of the Element hierarchy, select `nonPersistentEntity (extension)`.

- c. Click the drop-down list next to  and choose `column`.

You use the `column` element because the data type is `varchar` and is not a typekey.

- d. To the new column, add the following values that support searches for `InterpreterSpecialty`:

Name	Value
name	InterpreterSpecialty
type	varchar
nullok	true
desc	Interpreter language specialties

- e. Click the drop-down list next to  and choose **params**.
- f. Enter the following values to specify the size of this varchar column:

Name	Value
name	size
value	30

- 2. Add an entry for **InterpreterSpecialty** to the Gosu class **ContactSearchMapper** so you can use this field when you search for a contact.
  - a. In ClaimCenter Studio, navigate in the Project window to **configuration** → **gsrc** and then to **gw.plugin.contact.ab800**.
  - b. Double-click **ContactSearchMapper** to open it in the editor
  - c. Press **CTRL+F** and enter **convertToABContactAPISearchCriteria** to find this method.
  - d. Find the following line of code:
 

```
searchCriteriaInfo.FirstNameKanji = searchCriteria.FirstNameKanji
```
  - e. After that line, add the following code:
 

```
searchCriteriaInfo.InterpreterSpecialty = searchCriteria.InterpreterSpecialty
```
- 3. Add an entry for **InterpreterSpecialty** to the Gosu class **ContactSearchResultMapper** so you can see this field in the search results that come back from **ContactManager**.
  - a. In the same Project window folder, **gw.plugin.contact.ab800**, double-click the class **ContactSearchResultMapper** to open it in the editor.
  - b. Press **CTRL+F** and enter **populateContactFromSearchResult** to find that method, and then in the method enter the following statement:
 

```
if (contact.type is Interpreter) {
    contact.InterpreterSpecialty = searchResult.InterpreterSpecialty
}
```

#### Step 4: Configure the Address Book Search Interface in Both Applications

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

##### Step A. Configure ContactManager

1. In **ContactManager Studio** in the Project window, navigate to **configuration** → **config** → **Localizations** → **en\_US** and double-click **display.properties** to open the file in the editor
2. Press **CTRL+F** and enter **Web.ContactSearch.IncludePendingCreates** to find this entry.
3. Add a line and enter the following display key:  
`Web.ContactSearch.InterpreterSpecialty = Interpreter Specialty`
4. Press **Ctrl+Shift+N** and enter **ContactSearchDV** to find this PCF file, and then double-click the file in the search results to open it in the editor.
5. Select the **ContactSearchDV** detail view panel.
6. Click the **Code** tab at the bottom of the window and find the following line of code:
 

```
function isDoctor(c : ABContactSearchCriteria) : boolean {
    return entity.ABDoctor.Type.isAssignableFrom(c.ContactSubtypeType )}
```

7. Add a line after that one and enter on one line the following line of code:

```
function isInterpreter(c : ABContactSearchCriteria) : boolean {
    return entity.ABInterpreter.Type.isAssignableFrom(c.ContactSubtypeType )}
```

8. In the **InputColumn** on the left, locate the **InputSet** containing the Organization Name input.

9. From the **Toolbox** on the right, drag an **InputSet** widget and drop it under the **InputSet** containing the Organization Name input.

10. Click the new **InputSet** and set the following property:

---

```
visible  isInterpreter(SearchCriteria)
```

---

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

---

<b>editable</b>	<b>true</b>
<b>id</b>	<b>InterpreterSpecialty</b>
<b>label</b>	<b>displaykey.Web.ContactSearch.InterpreterSpecialty</b>
<b>value</b>	<b>SearchCriteria.InterpreterSpecialty</b>

---

### Step B. Configure ClaimCenter

1. In ClaimCenter Studio in the Project window, navigate to **configuration** → **config** → **Localizations** → **en\_US** and double-click **display.properties** to open the file in the editor.

2. Press **CTRL+F** and enter the search text **Web.AddressBook.Search.IncludeSpecialistServices**, and then add a line after the one found by the search.

3. Enter the following display key on the new line:

```
Web.AddressBook.Search.InterpreterSpecialty = Interpreter Specialty
```

4. Press **Ctrl+Shift+N** and enter **AddressBookSearchDV** to find this PCF file, and then double-click the file in the search results to open it in the editor.

5. In the **InputColumn** on the left, locate the **InputSet** containing the Law Firm Specialty **TypeKeyInput**, and then copy the input set and paste the copy below it.

6. Select the new **InputSet** and set the following property:

---

```
visible  searchCriteria.isSearchFor(entity.Interpreter)
```

---

7. Right-click the **TypeKeyInput** widget inside the **InputSet** and click **Change element type**.

8. Click **Input** in the list of element types, and then click **OK**.

9. Set the following properties for the input widget:

---

<b>editable</b>	<b>true</b>
<b>id</b>	<b>InterpreterSpecialty</b>
<b>label</b>	<b>displaykey.Web.AddressBook.Search.InterpreterSpecialty</b>
<b>value</b>	<b>searchCriteria.InterpreterSpecialty</b>

---

## Step 5: Test Your Changes

1. If ContactManager is running, stop and then restart ContactManager.
  - a. Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`
  - b. Restart ContactManager:  
`gwab dev-start`
2. Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:  
`gwcc dev-stop`
3. Regenerate the ClaimCenter data dictionary to test your changes to `ContactSearchCriteria`. At the command prompt, enter:  
`gwcc regen-dictionary`
4. If the data dictionary regenerates successfully, start the ClaimCenter application:  
`gwcc dev-start`
5. Log in to ContactManager as a user who can create and edit contacts, such as user `aapplegate` with password `gw`.
6. Click **Search** on the left, and then choose **Interpreter** in the **Contact Type** list.
7. Search for one of the interpreters you created during your earlier tests.  
If you have not done so already, create a new interpreter and specify an interpreter specialty.
  - a. Do a search for that interpreter and use the interpreter specialty as a search criterion to see if the search returns that interpreter.
  - b. Search for the same interpreter and use a specialty that is not a specialty for that interpreter to see if you get zero returns.
8. Log in to ClaimCenter and use the Address Book to search for an interpreter. Use the same search strategies as you did for ContactManager.

## Adding an Address Field to Contact Search

You can add an `Address` field to Contact search. The process for `Address` fields that are not denormalized fields is similar to the process for adding the `InterpreterSpecialty` property to search. However, the ClaimCenter part of the process is different enough that you might want to refer to the steps that follow starting with step 6. For the ContactManager part of the process, see “Adding the `InterpreterSpecialty` Property to Search” on page 88.

A denormalized field can improve the speed of search. If you want to search on a denormalized address field, add fields to the `Address` entity and the `ABContact` entity to give ContactManager an actual field to search for. On the ContactManager side, the field you set up for search is the field you add to `ABContact`. On the core application side, the field you search for is the `Address` field, as it would be for a regular, non-denormalized `Address` field search.

The following steps show what you do differently for denormalized `Address` column searches:

1. Add a column for the denormalized field to the `Address` entity and set `SupportsLinguisticSearch` to `true`.
2. Add a `searchColumn` to the `ABContact` entity that has the same name as the column added to `Address`, plus `Denorm`. For example, if the `Address` column is named `County`, the `ABContact` column is named `CountyDenorm`. The `sourceColumn` is the same as the name for this search column, such as `CountyDenorm`, and the `sourceForeignKey` is `PrimaryAddress`.

3. The next steps are similar to those starting at “Step A: Adding Search Support to the ContactManager User Interface” on page 88. In general, for the ContactManager part of the process, use the ABContact search column as your field. For example:
  - a. Add a column to ABContactSearchCriteria.etcx, using the name of the ABContact search column.
  - b. Add to search-config.xml a <Criterion> for the column you just added to ABContactSearchCriteria.etcx. Add it to the following <CriteriaDef> for ABContact:

```
<CriteriaDef entity="ABContactSearchCriteria" targetEntity="ABContact">
```

The new criterion must use the name of the column in ABContactSearchCriteria.etcx and can have any match type you prefer.
4. The next steps are similar to those starting at “Step B: Adding Search Support in ContactManager for Guidewire Core Applications” on page 89. In general, for the ContactManager part of the process, use the ABContact search column as your field. For example:
  - a. Add an entry for your ABContact search field to the class ABContactAPISearchCriteria.
  - b. If you want the field to be added to the search results sent back to a Guidewire core application, add the ABContact search field to the class ABContactAPISearchResult.
5. Refresh the web services. See “Step 2: Restart ContactManager and Refresh the Web Services in ClaimCenter” on page 90.
6. Go through the steps for the ClaimCenter part of the configuration, but use the Address field you want to have in search. This part of the configuration starts at “Step 3: Add ContactManager Search Capability in ClaimCenter” on page 90.
  - a. The entity ContactSearchCriteria.etcx has a foreign key to Address, so any field on Address can be used in contact search, and there is no need to update this file.
  - b. Add an entry for the Address field to ContactSearchMapper in the following if statement after the first line of code in that statement:

```
if (searchCriteria.Address != null) {  
    var address = new ABContactAPIAddressSearch()
```

For example, for Address.County, add the following new line of code:  

```
address.County = searchCriteria.Address.County
```
  - c. The class ContactSearchResultMapper has a lot of the Address fields already in it. Check to see if a field you want to make visible for the address part of search might already be in the file. If you add a new field to Address that you want to search for, you must add an entry for the new Address field to this file.
7. Continue with the user interface configuration instructions, but use them as only a guideline for adding Address fields and do not follow them exactly. Address fields are located in different parts of the screens from Contact and ABContact fields. See “Step 4: Configure the Address Book Search Interface in Both Applications” on page 91.

## Adding the Service State Property to Search

This topic uses the ContactServiceState entity and ServiceState property from the example in “Extending Contacts with an Array” on page 158. In this topic, you make the new ServiceState property searchable in both ClaimCenter and ContactManager. The property names must match in both applications.

**Note:** Any object you add to the search system needs to have indexes declared for its fields in ContactManager to make the search reasonably fast. In the example, ContactServiceState does have indexes declared for each of its fields. Without indexes, searching for the object can be slow.

This topic includes:

- “Step 1: Add Search Capability in ContactManager” on page 95
- “Step 2: Regenerate ContactManager Web Services and Refresh Them in ClaimCenter” on page 96

- “Step 3: Add Search Capability in ClaimCenter” on page 97
- “Step 4: Configure the Contact Search Interface in Both Applications” on page 97
- “Step 5: Test Your Changes” on page 98

### Step 1: Add Search Capability in ContactManager

There are two primary aspects to adding search capability in ContactManager:

- Adding search support for the field to the ContactManager user interface through settings in `search-config.xml` and in the `ABContactSearchCriteria` entity. For a complete description of `search-config.xml`, see “Configuring ClaimCenter Database Search” on page 348 in the *Configuration Guide*.
- Adding search support for Guidewire core applications in the Gosu class `ABContactAPISearchCriteria`. You can also have the search field show in the search results that ContactManager sends back to the Guidewire core application. To add the field to the search results, add code for the field to the Gosu class `ABContactAPISearchResult`.

These classes provide parameters used in web service calls, and changing them changes the web service WSDL. After changing one of these classes, you must restart ContactManager to regenerate the web APIs. Then, in Studio for the Guidewire core application, you must refresh the ContactManager web APIs.

#### Step A: Add the Element to the ContactManager Search Screens

In this topic, you add the `ServiceState` typekey to `ABContactSearchCriteria.etcx`. Then you add an `ArrayCriterion` element to the ContactManager `search-config.xml` file to specify where the `ServiceState` column is located. For a detailed description of `ArrayCriterion`, see “The `<ArrayCriterion>` Subelement” on page 354 in the *Configuration Guide*.

**Note:** There are restrictions on using `ArrayCriterion`:

- For any `CriteriaDef` element in the `search-config.xml` file, you must have only one `ArrayCriterion` subelement. If you have more than one `ArrayCriterion` defined for an entity, a search can return duplicate results for that entity. For example, you have two `ArrayCriterion` properties for `ABContact`. A successful search returns the same contact twice, once for each matched property.
- If you want to search for multiple fields in an array entity, create a new field that combines the fields.
- Additionally, the system requires unique values for the field that you are searching on. For example, `City` and `State` are two fields on an array entity, and the combination of the two fields is what makes the search unique. In this case, you must create a third field on the entity that concatenates the two values, and then create the array search criterion on that third field.

#### To add the element:

1. In ContactManager Studio, in the Project window, navigate to `configuration` → `config` → `Extensions` → `Entity` and then double-click `ABContactSearchCriteria.etcx` to open it in the editor.
2. Right-click `nonPersistentEntity` (extension) at the top of the `Element` hierarchy and click `Add new`, and then choose `typekey` from the drop-down list.
3. Enter the following values:

Name	Value
<code>name</code>	<code>ServiceState</code>
<code>typelist</code>	<code>State</code>
<code>desc</code>	<code>State where contact provides services</code>

4. In the Project window, navigate to `configuration` → `config` → `search` and double-click `search-config.xml` to edit the file.

5. 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">
```

6. Press **Ctrl+F** and enter `ABContact` to find the `CriteriaDef` element for the target entity `ABContact`.

7. Add a new `ArrayCriterion` element to the end of this `CriteriaDef` element, as follows:

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

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`.

### Step B: Add Search Support in ContactManager for Guidewire Core Applications

In this topic, you add code for the `ServiceState` field to the `Gosu` class `ABContactAPISearchCriteria`. This web API class enables a Guidewire core application's `Search` screen to show the field to the user and pass its value to `ContactManager`.

1. In `ContactManager Studio`, navigate in the `Project` window to `configuration` → `gsrc` and then to `gw.webservice.ab.ab801.abcontactapi`.
2. Double-click `ABContactAPISearchCriteria` to open it in the editor.
3. Add the following variable definition to the list of variables at the beginning of the class.  
`public var ServiceState : typekey.State`
4. In the method `toSearchCriteria` add a comma after the line starting with `:AllTagsRequired` and add a new entry for `ServiceState`:  
`:AllTagsRequired = this.AllTagsRequired,`  
`:ServiceState = this.ServiceState`

### Step 2: Regenerate ContactManager Web Services and Refresh Them in ClaimCenter

At this point, `ClaimCenter` does not have the changes you made in `ContactManager`. To get them, restart `ContactManager` to regenerate the web services, and then, in `ClaimCenter Studio`, refresh the `ContactManager` web service `WSDL`.

#### To refresh the `ClaimCenter` copy of the `ContactManager` WSDL

1. Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`
2. Start the `ContactManager` application:  
`gwab dev-start`
3. After `ContactManager` starts up, in `ClaimCenter Studio`, refresh the `ContactManager` plugin.
  - a. In the `Project` window, press **Ctrl+Shift+N** and enter `ab801.wsc`. Then click `ab801.wsc` in the search results to open this web service collection in the editor.

- b.** In the editor, select the following resource:

```
 ${ab}/ab/ws/gw/webservice/ab/ab801/abcontactapi/ABContactAPI?wsdl
```

- c.** At the bottom of the pane, click **Fetch Updates**.

### Step 3: Add Search Capability in ClaimCenter

1. Using ClaimCenter Studio, in the Project window, navigate to **configuration** → **config** → **Extensions** → **Entity**.
2. Click **ContactSearchCriteria.etx** to open the entity extension in an editor.
3. Right-click **nonPersistentEntity (extension)** at the top of the Element hierarchy and click **Add new**, and then choose **typekey** from the drop-down list.
4. Enter the following values:

Name	Value
name	ServiceState
typelist	State
desc	State where contact provides services

5. Add an entry for **ServiceState** to the Gosu class **ContactSearchMapper** so you can use this field when you search for a contact.
  - a. In ClaimCenter Studio, press **Ctrl+N** and enter **ContactSearchMapper**, and then in the search results click **ContactSearchMapper (gw.plugin.contact.ab800)** to open this class in the editor.
  - b. Press **Ctrl+F** and enter **convertToABContactAPISearchCriteria** to find this method in the class.
  - c. Add a line after the following line of code:
 

```
searchCriteriaInfo.PreferredVendors = searchCriteria.PreferredVendors
```
  - d. Enter the following code for **ServiceState**:
 

```
searchCriteriaInfo.ServiceState = searchCriteria.ServiceState.Code
```

### Step 4: Configure the Contact Search Interface in Both Applications

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

#### Step A. Configure ContactManager

1. In **ContactManager** studio, open the **Project** window.
2. Navigate to **configuration** → **config** → **Localizations** → **en\_US** and then double-click **display.properties** to open it in the editor.
3. Press **CTRL+F** and enter the search text **Web.ContactSearch.Person.LastName** to find this line in the file, and then add a line after it.
4. Add the following new entry:
 

```
Web.ContactSearch.ServiceState = Service State
```
5. In the **Project** window, navigate to **configuration** → **config** → **Page Configuration** → **pcf** → **search**, and then click **ContactSearchDV** to edit this PCF file.
6. In the **InputColumn** on the left, locate the **InputSet** containing the **Organization Name** input, then add an **InputSet** widget under it.

7. Click the new InputSet and set the following property:

```
visible    isCompany(SearchCriteria)
```

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

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

### Step B. Configure ClaimCenter

1. In ClaimCenter studio, open the Project window.
2. Navigate to configuration → config → Localizations → en\_US and then double-click display.properties to open it in the editor.
3. Press CTRL+F and enter Web.AddressBook.Search.SearchType to find this line in the file, and then add a line after it.
4. Add the following new entry:  
`Web.AddressBook.Search.ServiceState = Service State`
5. In the Project window, navigate to configuration → config → Page Configuration → pcf → addressbook and double-click AddressBookSearchDV to edit this PCF file.
6. In the InputColumn on the left, locate the InputSet containing the Tax ID input, then add an InputSet widget under it.
7. Click the new InputSet and set the following property:

```
visible    searchCriteria.isSearchFor(entity.Company)
```

8. 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 5: Test Your Changes

1. Shut down and restart both ClaimCenter and ContactManager.
  - a. Open a command prompt in the ContactManager/bin directory and then enter the following command:  
`gwab dev-stop`
  - b. Start the ContactManager application:  
`gwab dev-start`
  - c. Open a command prompt in the ClaimCenter/bin directory and then enter the following command:  
`gwcc dev-stop`
  - d. Start the ClaimCenter application:  
`gwcc dev-start`

2. Log in to ContactManager and click **Search** on the left, then choose **Company** in the **Contact Type** list.
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 **Service 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 the first letter of that company name, but no state specified.
  - d. Do the same search with a state specified to see if the search results narrow to the companies with that value specified.
4. Log in to ClaimCenter and use the Address Book to search for a company that you know is stored in ContactManager. Use the same search strategies as you did for ContactManager.

## Geocoding and Proximity Searches for Vendor Contacts

As described in “Geocoding Plugin Integration” on page 235 in the *Integration Guide*, ClaimCenter and ContactManager support *geocoding*, which enables the assignment of latitude and longitude to an address. 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 or all the addresses in a certain radius.

This topic describes how to configure geocoding to work with ClaimCenter and ContactManager and provides examples.

This topic includes:

- “Configuring Geocoding for ClaimCenter and ContactManager” on page 99
- “Proximity Search Examples for Vendor Contacts” on page 105

### Configuring Geocoding for ClaimCenter and ContactManager

Configuring geocoding involves activating the geocoding plugin, setting `config.xml` parameters, and enabling the work queue and scheduler to run batch geocoding of addresses.

This topic includes:

- “Step 1: Activate the Plugin” on page 100
- “Step 2: Configure geocoding in config.xml” on page 101
- “Step 3: Configure the Work Queue” on page 102
- “Step 4: Schedule Geocoding” on page 103
- “Step 5: Log Geocode Information” on page 104
- “Step 6: Stop and Restart ClaimCenter and ContactManager” on page 104

---

**IMPORTANT** If your environment includes a ContactManager integration, you must perform these steps in both ClaimCenter and ContactManager for proper functioning of the geocoding feature.

### Step 1: Activate the Plugin

By default, the `GeocodePlugin` plugin interface uses the Microsoft Bing Maps web service and is disabled. In the base configuration, the geocoding plugin class that implements this interface is written in Gosu and 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. When you create a key, the application name is arbitrary, and no application URL is required.

#### To enable and use the plugin in ClaimCenter

1. Start ClaimCenter Studio.

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

```
gwcc studio
```

2. In the Project window, navigate to `configuration → config → Plugins → registry` and then double-click `GeocodePlugin.gwp` to open this registry file in the editor.

3. In the column on the right under the `Server` field, click the `Enabled` check box to enable the plugin.

4. Under `Parameters`, click the `Value` field for `applicationKey`, and then enter the application key you obtained from Bing Maps. For information on getting a Bing Maps application key, see “Step 1: Activate the Plugin” on page 100.

#### To enable and use the plugin in ContactManager

1. Start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:

```
gwab studio
```

2. In the Project window, navigate to `configuration → config → Plugins → registry` and then double-click `GeocodePlugin.gwp` to open this registry file in the editor.

3. Click the `Enabled` check box to enable the plugin.

4. In the `Parameters` section, click the `Value` field for `applicationKey`, and then enter the application key you obtained from Bing Maps. For more information, see “Step 1: Activate the Plugin” on page 100.

#### GeocodePlugin Parameters

There are five parameters you can set for the `GeocodePlugin`.

Parameter	Description
<code>applicationKey</code>	The application key that you obtained from Bing Maps. See “Step 1: Activate the Plugin” on page 100.
<code>geocodeDirectionsCulture</code>	The locale for geocoded addresses and routing instructions returned from Bing Maps. For example, use the locale code <code>ja-JP</code> for addresses and instructions for Japan. The base application plugin implementation uses <code>en-US</code> if you do not specify a value. For a current list of codes that Bing Maps supports, see <a href="http://msdn.microsoft.com/en-us/library/cc981048.aspx">http://msdn.microsoft.com/en-us/library/cc981048.aspx</a> .
<code>imageryCulture</code>	The language for map imagery. For example, use the language code <code>ja</code> for maps labeled in Japanese. The base application plugin implementation uses <code>en</code> if you do not specify a value. For a current list of codes that Bing Maps supports, see <a href="http://msdn.microsoft.com/en-us/library/cc981048.aspx">http://msdn.microsoft.com/en-us/library/cc981048.aspx</a> .
<code>mapURLHeight</code>	Width of maps, in pixels. The base application plugin implementation uses <code>500</code> if you do not specify a value.
<code>mapURLWidth</code>	Height of maps, in pixels. The base application plugin implementation uses <code>500</code> if you do not specify a value.

## 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 ContactManager integrated with ClaimCenter, change settings in the config.xml file in each application.

### To open config.xml:

1. If necessary, start ClaimCenter Studio and ContactManager Studio.
2. In Studio, in the Project window, navigate to configuration → config and go to the bottom of that node, and then double-click config.xml to open the file in an editor.
3. In the editor, press CTRL+F and then enter Geocoding to find that section of the configuration parameters.
4. Set the UseGeocoding... parameters defined in the following tables to true to enable geocoding.

The ClaimCenter config.xml file provides the following geocoding configuration parameters:

Parameter	Description
UseGeocodingInPrimaryApp	Set this parameter to true to enable geographical data and proximity search on application screens in ClaimCenter. The setting affects screens 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 screens if ClaimCenter is integrated with ContactManager. The default setting is false.
UseMetricDistancesByDefault	Use kilometers in the user interface. The default setting, false, specifies that miles are to be used.
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 ContactManager and ClaimCenter.
ProximityRadiusSearchDefaultMaxResultCount	Maximum number of results to return when performing a radius (n miles or kilometers) search from ClaimCenter. <ul style="list-style-type: none"><li>• ClaimCenter passes the value of this parameter to ContactManager when it makes a call for a proximity search.</li><li>• The default value in the base configuration is 1000.</li><li>• This parameter has no effect on ordinal (nearest n) proximity searches.</li><li>• This parameter applies only to searches originating from ClaimCenter and does not have to match the value of the corresponding parameter in the ContactManager config.xml file.</li></ul>

The ContactManager config.xml file provides the following geocoding configuration parameters:

Parameter	Description
UseGeocodingInAddressBook	Set this parameter to true to enable geocoding for contacts stored in ContactManager. The default is false. The setting for this parameter must be the same in ContactManager 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 ContactManager 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 ContactManager and ClaimCenter.
ProximityRadiusSearchDefaultMaxResultCount	Maximum number of results to return when performing a radius (n miles or kilometers) search from the ContactManager search screen. <ul style="list-style-type: none"> <li>• The default value in the base configuration is 1000.</li> <li>• This parameter has no effect on ordinal (nearest n) proximity searches.</li> <li>• This parameter applies only to searches originating from ContactManager and does not have to match the value of the corresponding parameter in the ClaimCenter config.xml file.</li> </ul>

---

**IMPORTANT** If you have integrated ClaimCenter and ContactManager, verify that UseGeocodingInAddressBook, UseMetricDistancesByDefault, and ProximitySearchOrdinalMaxDistance have the same values in both ClaimCenter and ContactManager.

---

### Step 3: Configure the Work Queue

The geocoding feature uses the Guidewire work queue infrastructure for asynchronous searches. For example, the ClaimCenter Geocode batch 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 103. The ClaimCenter Geocode batch process geocodes all addresses in the database that have not yet been geocoded. ContactManager has an ABGeocode batch 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 ContactManager 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 first verify that you have a good understanding of the work queue infrastructure. For example, one concept 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 ContactManager 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`.

You must restart ClaimCenter and ContactManager after changing any of these files.

For more information on work queues, see:

- “Distributable Work Queues” on page 124 in the *System Administration Guide*
- “Configuring Distributable Work Queues” on page 128 in the *System Administration Guide*

**IMPORTANT** If you have many new User or Contact objects in ClaimCenter or many new ABContact objects in ContactManager, 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 batch process, see “Step 4: Schedule Geocoding” on page 103.

### Running the Geocoding Batch Process Manually

You can run the geocoding batch process manually as a system administrator on the **Server Tools** screen.

For 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 selected in the sidebar.
3. Locate the batch process **Geocode Writer** and click its **Run** button.

For ContactManager:

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 sidebar, click **Batch Process Info**.
4. Locate the batch process **AB Geocode Writer** and click its **Run** button.

### Step 4: Schedule Geocoding

If you enable the Geocode plugin in ClaimCenter, you must also uncomment the code that schedules the geocoding batch process in the `scheduler-config.xml` file.

#### To edit the entry for the Geocode process:

1. In ClaimCenter Studio, press `Ctrl+Shift+N` and enter `scheduler-config.xml`, and then double-click `scheduler-config.xml` in the search results to open it in the editor.
2. In the editor, press `CTRL+F` and enter **Geocode** to find the following entry:

The following XML code shows the ClaimCenter entry, including the `<!-- ... -->` comment tags:

```
<!-- New addresses searched for geocoding at 1:30 am -->
<!--
<ProcessSchedule process="Geocode">
    <CronSchedule hours="1" minutes="30"/>
</ProcessSchedule>
-->
```

This entry instructs the batch process to start searching for new addresses to geocode every night at 1:30 AM.

3. Remove the opening `<!--` and closing `-->` comments to activate this entry the next time you start the application.

4. If ClaimCenter is integrated with ContactManager, you must also uncomment the entry in the ContactManager `scheduler-config.xml` file for ABGeocode. In ContactManager Studio, follow the same procedure you did for ClaimCenter to open the file and find the entry.

The following XML code shows the ContactManager entry, including the `<!-- ... -->` comment tags:

```
<!-- <ProcessSchedule process="ABGeocode">
    <CronSchedule minutes="0"/>
</ProcessSchedule> -->
```

This entry instructs the batch process to start searching for new addresses to geocode every hour on the hour.

#### Important Notes:

- Schedule the Geocode and the ABGeocode batch processes with enough time between runs to fully process the work items in the work queues. If you find duplicate work items in the work queues for the same address ID, make the time between runs a longer interval.
- 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 when general access to your server is restricted.

#### BatchGeocode and GeocodeStatus Properties in ContactManager

The ABGeocode batch process determines which addresses to geocode by checking the `BatchGeocode` and `GeocodeStatus` properties. ContactManager handles these properties as follows:

- The `Address` property `BatchGeocode` indicates whether or not an address is to be geocoded. The default value is `false`, meaning that addresses by default are not geocoded.
- There is a ContactManager rule that sets vendor contacts to be automatically geocoded. The rule—`Set Vendor's Primary Address BatchGeocode`—is a `Preupdate` rule in the `ABContactPreupdate` rule set. This rule sets the `BatchGeocode` property of the primary address of an `ABContact` to `true` if the contact is tagged as a vendor.
- The batch process picks up addresses for which `BatchGeocode` is set to `true` and `GeocodeStatus` is set to `none`. This combination means that the address needs to be geocoded and has not been geocoded yet.

If you are adding many new contacts, especially into ContactManager, tune this parameter to match your expected daily load of new addresses. For detailed information on scheduling batch processes see “Defining a Schedule Specification” on page 144 in the *System Administration Guide*.

#### Step 5: Log Geocode Information

The `logging.properties` file controls logging for the application. You can edit this file in ClaimCenter and ContactManager Studio by navigating in the Project window to `configuration → config → logging` and then double-clicking `logging.properties`.

The file includes sections that you can configure to add details of all plugin activity, including geocoding, to the log. See “Configuring Logging” on page 25 in the *System Administration Guide* for details on configuring logging.

#### Step 6: Stop and Restart ClaimCenter and ContactManager

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 ContactManager when they are running in development mode:

##### 1. Stop ClaimCenter.

Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:  
`gwcc dev-stop`

##### 2. Stop ContactManager.

Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`

3. Start the ContactManager application:

```
gwab dev-start
```

4. Start the ClaimCenter application:

```
gwcc dev-start
```

**See also**

- “Geocoding Plugin Integration” on page 235 in the *Integration Guide*
- “Configuring Geocoding” on page 20 in the *System Administration Guide*

## Proximity Search Examples for Vendor Contacts

**Note:** To try the examples in this section, you must first configure Geocoding as described at “Configuring Geocoding for ClaimCenter and ContactManager” on page 99. Additionally, ensure that the geocoding batch process has run. If you are not on a production system, you could run the batch process manually as described at “Running the Geocoding Batch Process Manually” on page 103.

Your search results are returned from ContactManager ordered by distance, closest to farthest away. In the base configuration, this order of search results is the only one available.

The system sorts the results of a proximity search, but it does not display driving directions with the search results. To obtain driving directions, you must select a return address and click the **Return Driving Directions** button. Then 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 driving direction results.

### Distanced-based Search Example

This example shows you how to request a distance-based proximity search from the ClaimCenter user interface and describes what the system does in response to the search request. In this example, you request the closest auto repair shops within 25 miles of San Mateo, California, USA.

**Note:** To walk through this example in ClaimCenter, you must have done the following:

- Integrated ContactManager with ClaimCenter as described in “Integrating ContactManager with Guidewire Core Applications” on page 45.
- Loaded sample data as described at “Step 2: Load Sample Data” on page 40.
- Set up geocoding for both applications as described in “Configuring Geocoding for ClaimCenter and ContactManager” on page 99.
- Run the Geocode batch process for ClaimCenter and the ABGeocode batch process for ContactManager as described in “Running the Geocoding Batch Process Manually” on page 103.
- Started up both applications.

### To run the proximity search

- Log in to ClaimCenter.
- Click the **Address Book** tab.
- On the **Search Address Book** screen, choose the following settings:

Type	Auto Repair Shop
Search Radius	25 miles
City (under Search Radius)	San Mateo
ZIP Code	94404

4. Click **Search**.
5. ClaimCenter determines the center of the proximity search by synchronously geocoding this one address.
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**. 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 **Search Results**.

## PolicyCenter Support for Contact Searches

PolicyCenter enables you to search both for locally stored contacts and for contacts stored externally in ContactManager.

To configure Contact search in PolicyCenter, you edit the following files in PolicyCenter Studio:

- `search-config.xml` – Defines search criteria for addresses.
- `gw.plugin.contact.ab800.ABContactSystemPlugin` – The plugin implementation used to call ContactManager web services. This plugin implementation has a `searchContacts` method that calls the ContactManager `ABContactAPI.searchContact` method.
- `gw.plugin.contact.ab800.ABContactAPISearchCriteriaEnhancement` – The class that defines additional PolicyCenter search criteria to be used with ContactManager contact searches. The class enhances the ContactManager web service `ABContactAPISearchCriteria`, adding a `sync` method that is used by `ABContactSystemPlugin.searchContacts`.
- `gw.plugin.contact.ab800.ABContactAPISearchResultEnhancement` – The class that defines the search result fields that PolicyCenter displays in the search results returned from ContactManager.
- `ContactSearchCriteria.eix` – You can extend this entity by creating a `ContactSearchCriteria.etx` file. This entity defines the search criteria that are shown to the user in the search screens. If you add search criteria to `ContactSearchCriteria.etx`, you must also add them to `ABContactAPISearchCriteriaEnhancement`. PolicyCenter uses both files.
- `gw.plugin.contact.impl.ContactSearchCriteriaEnhancement` – The class in which the contact search method for both internal and external contact search is defined. The primary contact search method in this class is `performSearch`.
- `ContactSearchScreen.pcf` – This PCF file displays the contact search criteria. It also instantiates an instance of the `ContactSearchCriteria` entity and uses it as a parameter to its `doSearch` method defined in the **Code** tab. That method calls the method `ContactSearchCriteriaEnhancement.performSearch`.

In the base configuration, PolicyCenter uses the `Address` fields defined in `search-config.xml` to search only for address fields. PolicyCenter does not use this file to configure searching for Contact fields.

To search for contacts stored in ContactManager, PolicyCenter uses the plugin implementation `gw.plugin.contact.ab800.ABContactSystemPlugin`. This class's `searchContacts` method instantiates an `ABContactAPISearchCriteria` object and calls its `sync` method, which in turn calls the ContactManager web service `ABContactAPI.searchContact`. The PolicyCenter `searchContacts` method returns data from the `ABContactAPISearchResult` Gosu objects returned by the ContactManager `searchContact` method. PolicyCenter shows these returned objects as contacts and their fields in the search results.

One class where you can customize this contact search behavior is `gw.plugin.contact.ab800.ABContactAPISearchCriteriaEnhancement`. This class extends the search criteria defined in the ContactManager web service `ABContactAPISearchCriteria`. For example, the `sync` method

specifically searches for the client tag. If you want to search for other tags, you can modify the code of this method to do so.

A corresponding class in which you can customize contact search behavior is in `gw.plugin.contact.ab800.ABContactAPISearchResultEnhancement`. This class defines extensions to the search results returned by ContactManager. It enhances the ContactManager web service `ABContactAPISearchResult`.

For information on the ContactManager files involved in an external contact search, see “ContactManager Support for Core Application Searches” on page 86.

## BillingCenter Support for Contact Searches

BillingCenter enables you to search both for locally stored contacts and for contacts stored externally in ContactManager.

To configure Contact search in BillingCenter, you edit the following files in BillingCenter Studio:

- `search-config.xml` – Defines search criteria for addresses.
- `gw.plugin.contact.ab800.ABContactSystemPlugin` – The plugin implementation used to call ContactManager web services. This plugin implementation has a `searchContacts` method that calls the ContactManager `ABContactAPI.searchContact` method.
- `gw.plugin.contact.ab800.ABContactSearchCriteriaInfoEnhancement` – The class that defines additional BillingCenter search criteria to be used with ContactManager contact searches. The class enhances the ContactManager web service `ABContactAPISearchCriteria`, adding a `sync` method that is used by `ABContactSystemPlugin.searchContacts`.
- `gw.plugin.contact.ab800.ContactResultFromSearch` – The class that defines the search result fields that BillingCenter displays in the search results returned from ContactManager.
- `ContactSearchCriteria.eix` – You can extend this entity by creating a `ContactSearchCriteria.etx` file. `ContactSearchCriteria` defines the search criteria that are shown to the user in the search screens. If you add search criteria to `ContactSearchCriteria.etx`, you must also add them to `ABContactAPISearchCriteriaInfoEnhancement`. BillingCenter uses both files.
- `gw.plugin.contact.impl.ContactSearchCriteriaEnhancement` – The class in which the contact search method for both internal and external contact search is defined. The primary contact search method in this class is `performSearch`.
- `ContactSearchScreen.pcf` – This PCF file contains the display view in which the contact search criteria are shown. `ContactSearchScreen` also instantiates an instance of the `ContactSearchCriteria` entity and uses it as a parameter to its `doSearch` method defined in the `Code` tab. That method calls the method `ContactSearchCriteriaEnhancement.performSearch`.
- `ContactSearchDV.pcf` – This PCF file contains the contact search criteria shown in the search screen. This file is where you add new search criteria.

In the base configuration, BillingCenter uses the Address fields defined in `search-config.xml` to search only for address fields.

To search for contacts stored in ContactManager, BillingCenter uses the plugin implementation `gw.plugin.contact.ab800.ABContactSystemPlugin`. This class’s `searchContacts` method instantiates an `ABContactAPISearchCriteria` object and calls its `sync` method, which in turn calls the ContactManager web service `ABContactAPI.searchContact`. The BillingCenter `searchContacts` method returns data from the `ABContactAPISearchResult` Gosu objects returned by the ContactManager `searchContact` method. BillingCenter shows these returned objects as contacts and their fields in the search results.

One class in which you can customize this contact search behavior is `gw.plugin.contact.ab800.ABContactSearchCriteriaInfoEnhancement`. This class enhances the search

criteria defined in the ContactManager web service `ABContactAPISearchCriteria`. For example, the `sync` method of this class specifically searches for the Client tag. If you want to search for other tags, you can modify the code of this method to do so.

A corresponding class where you can customize contact search behavior is `gw.plugin.contact.ab800.ContactResultFromSearch`. This class defines extensions to the search results returned by ContactManager. It extends the ContactManager web service `ABContactAPISearchResult`.

There are also ContactManager files involved in an external contact search. For more information, see “ContactManager Support for Core Application Searches” on page 86.

# Securing Access to 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 109
- “ContactManager Contact Security” on page 110
- “PolicyCenter Contact Security” on page 120
- “BillingCenter Contact Security” on page 120
- “ClaimCenter Contact Security” on page 122

## Overview of Contact Security

A fundamental component of enforcing security on shared contacts is checking permissions for users of the Guidewire core application. Contact security can be enforced both by a core application and by ContactManager as users create, update, and delete contacts. Additionally, users of ContactManager with appropriate permissions can merge duplicate contacts and review pending contact updates and creation.

Each Guidewire application has its own rules and security setup.

**For information on ContactManager security, see**

- “ContactManager Contact Security” on page 110

**For information on PolicyCenter security, see**

- “PolicyCenter Contact Security” on page 120
- “Security: Roles, Permissions, and the Community Model” in the *PolicyCenter Application Guide*
- “Authority Profiles” in the *PolicyCenter Application Guide*

**For information on BillingCenter security, see**

- “BillingCenter Contact Security” on page 120
- “Security” in the *BillingCenter Application Guide*

**For information on ClaimCenter security, see**

- “ClaimCenter Contact Security” on page 122
- “Configuring ClaimCenter Contact Security” on page 126
- “Security: Roles, Permissions, and Access Controls” on page 447 in the *Application Guide*

## ContactManager Contact Security

ContactManager is built on the same platform as the Guidewire core applications and provides roles and permissions that you can configure to control access to contact data. Additionally, the permissions have a set of permission check expressions you can configure to limit access to specific screens and widgets.

*Role-based security* defines what actions a user of a Guidewire application is allowed to do. This type of security includes defining permissions, grouping related permissions into roles, and assigning these roles to users based on the work they perform in the Guidewire application.

For most purposes, other than merging duplicate contacts and reviewing pending contact changes, you typically access ContactManager data through a core application, like ClaimCenter, PolicyCenter, or BillingCenter. Those applications provide contact security to control which users of the application can access contact data. For more information on core application contact security, see:

- “PolicyCenter Contact Security” on page 120
- “BillingCenter Contact Security” on page 120
- “ClaimCenter Contact Security” on page 122

You can set up ContactManager users to access contact data directly in ContactManager. The base configuration provides a role for this purpose, the ContactManager role. For more information, see “ContactManager Roles” on page 110.

The primary tasks that users have to perform in ContactManager are:

- **Merging duplicate contacts** – See “Detecting and Merging Duplicate Contacts” on page 245.
- **Reviewing pending contact changes** – See “Reviewing Pending Changes to Contacts” on page 252.

**This topic includes:**

- “ContactManager Roles” on page 110
- “ContactManager Contact Subtype and Tag Permissions” on page 112
- “Contact Search Security Configuration Parameters in ContactManager” on page 115
- “Configuring ContactManager Contact Security” on page 115

## ContactManager Roles

A role is a collection of permissions. By grouping permissions into roles, you can define the authority of a user of ContactManager by assigning the user a few roles rather than a larger list of permissions. A user can have multiple roles and must have at least one role.

The permissions used for contact access in the base configuration of ContactManager are `abcreate`, `abcreateref`, `abdelete`, `abdeleteref`, `abedit`, `abeditref`, `abview`, `abviewpending`, `abviewmerge`, `abviewpending`, `abviewsearch`, `anytagcreate`, `anytagdelete`, `anytagedit`, and `anytagview`. These permissions are described in more detail in the next topic, “ContactManager Contact Subtype and Tag Permissions” on page 112.

You can create additional permissions for certain contacts, contact subtypes, or tags. For example, you can create a permission for working with client contacts and another for working with vendor contacts. To add permissions to roles and assign roles to users, use the **ContactManager Roles** screen. Log in as a user with administration privileges and click the **Administration** tab, and then navigate in the sidebar to **Users and Security → Roles**. See “Configuring ContactManager Contact Security” on page 115.

The base configuration of ContactManager provides the following roles, each of which has a set of default permissions. It is likely that you will add your own roles and permissions as well.

Role	Permissions	Description
Client Application	<ul style="list-style-type: none"> <li>• Client Application – clientapp</li> <li>• Create address book contacts – abcreate</li> <li>• Create address book preferred vendors – abcreatepref</li> <li>• Create contact with any tag – anytagcreate</li> <li>• Delete address book contacts – abdelete</li> <li>• Delete address book preferred vendors – abdeletepref</li> <li>• Delete contact with any tag – anytagdelete</li> <li>• Edit address book contacts – abedit</li> <li>• Edit address book preferred vendors – abeditpref</li> <li>• Edit contact with any tag – anytagedit</li> <li>• Edit user language – usereditlang</li> <li>• View address book contact search screens – abviewsearch</li> <li>• View address book contacts – abview</li> <li>• View contact with any tag – anytagview</li> </ul>	This role is used by core applications to communicate with ContactManager. It contains permissions that allow the core applications to perform specific tasks.
ContactManager	<ul style="list-style-type: none"> <li>• Create address book contacts – abcreate</li> <li>• Create address book preferred vendors – abcreatepref</li> <li>• Create contact with any tag – anytagcreate</li> <li>• Delete address book contacts – abdelete</li> <li>• Delete address book preferred vendors – abdeletepref</li> <li>• Delete contact with any tag – anytagdelete</li> <li>• Edit address book contacts – abedit</li> <li>• Edit address book preferred vendors – abeditpref</li> <li>• Edit contact with any tag – anytagedit</li> <li>• Edit user language – usereditlang</li> <li>• View address book contact search screens – abviewsearch</li> <li>• View address book contacts – abview</li> <li>• View contact with any tag – anytagview</li> <li>• View merge – abviewmerge</li> <li>• View pending – abviewpending</li> </ul>	User with full permission to create, edit, and delete contacts.
Contact Subtype Changer	<ul style="list-style-type: none"> <li>• Change Contact Subtype – changecontactsubtype</li> <li>• SOAP administration – soapadmin</li> </ul>	User with permissions to change the subtype of a contact instance.
Contact Viewer	<ul style="list-style-type: none"> <li>• Edit user language – usereditlang</li> <li>• View address book contact search screens – abviewsearch</li> <li>• View address book contacts – abview</li> <li>• View contact with any tag – anytagview</li> </ul>	User with view-only permissions for contacts
Rule Admin	<ul style="list-style-type: none"> <li>• Administer rules – ruleadmin</li> <li>• Edit user language – usereditlang</li> </ul>	Rule administrator
Superuser	All permissions	Super user with all permissions

<b>Role</b>	<b>Permissions</b>	<b>Description</b>
Tools View	<ul style="list-style-type: none"> <li>• View BatchProcess tools screen – toolsBatchProcessview</li> <li>• View Cache Info screen – toolsCacheinfoview</li> <li>• View Cluster tools screen – toolsClusterview</li> <li>• View Info tools screen – toolsInfoview</li> <li>• View Log tools screen – toolsLogview</li> <li>• View ManagementBeans tools screen – toolsJMXBeansview</li> <li>• View Profiler tools screen – toolsProfilerview</li> <li>• View StartablePlugin tools screen – toolsPluginview</li> <li>• View WorkQueue tools screen – toolsWorkQueueview</li> </ul>	User with permission to work on the Server Tools screens.  To access Server Tools, press Alt+Shift+T and click Server Tools.
User Admin	<ul style="list-style-type: none"> <li>• Create groups – groupcreate</li> <li>• Create users – usercreate</li> <li>• Delete groups – groupdelete</li> <li>• Delete users – userdelete</li> <li>• Edit groups – groupedit</li> <li>• Edit user language – usereditlang</li> <li>• Edit users – useredit</li> <li>• Grant roles to users – usergrantroles</li> <li>• Manage regions – regionmanage</li> <li>• Manage roles – rolemanage</li> <li>• Manage script parameters – scrprmmange</li> <li>• Resync message – resyncmessage</li> <li>• Retry message – retrymessage</li> <li>• Skip message – skipmessage</li> <li>• SOAP administration – soapadmin</li> </ul>	User who handles administration of ContactManager users.

## ContactManager Contact Subtype and Tag Permissions

The Guidewire core applications and ContactManager provide contact subtype and tag permissions that you can use to control access to contacts. The `SystemPermissionType` typelist lists all the subtype and tag permissions in ContactManager.

The following table lists the subtype and tag permissions provided in the base configuration of ContactManager for contacts:

<b>Code</b>	<b>Enables a ContactManager User to</b>
abview	View the details of contact entries in ContactManager.
abviewsearch	View ContactManager contact search screens.
anytagcreate	Create a new contact regardless of which contact tag it requires.
anytagdelete	Delete a contact that has any contact tag.
anytagedit	Edit a contact that has any contact tag.
anytagview	See a contact that has any contact tag.
abcreate	Create a new contact in ContactManager.
abcreatepref	Create a new preferred vendor in ContactManager.
abdelete	Delete an existing contact from the address book.
abdeletepref	Delete an existing preferred vendor address book entry.
abedit	Edit an existing contact in ContactManager.
abeditpref	Edit an existing preferred vendor in ContactManager.
abviewmerge	Review and merge duplicate contacts in the Merge Contacts screens.
abviewpending	Review and approve or disapprove pending contacts in the Pending Changes screens.

The system uses role-based security for these permissions. As described in the previous topic, 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` and `anytagcreate` permissions enables the user who has this role to create any type of contact.

The contact and tag permissions supplied in the base configuration apply across all contact subtypes and tags. If you create a permission that applies to a contact subtype, that permission also applies to all the subtypes of that contact subtype.

ContactManager enables you to restrict permissions according to contact subtype or tag. For example, you can enable a user with `abcreate` permission to create only `PersonVendor` contacts, but not `CompanyVendor` contacts. You configure contact and tag permissions through the `SystemPermissionType` typelist and the `security-config.xml` resource.

**Note:** If you create a set of tag permissions for a specific tag, these permissions enable access to contacts that have only that tag. For example you create a set of Vendor tag permissions and a user has a role with only those tag permissions. That user will not be able to work with a contact that has both Claim Party and Vendor tags. You could also create a set of tag permissions for Claim Party tags. In that case, a user with both Vendor and Claim Party tag permissions would be able to work with contacts that have both Vendor and Claim Party tags.

For examples showing how to create and use permissions for a contact subtype and a contact tag, see “Configuring ContactManager Contact Security” on page 115.

## ContactManager Contact and Tag 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 permission to perform an operation.

For example, in the `ContactManager>NewContact.pcf` file, the `canVisit` attribute is set to:

```
perm.ABContact.create(ContactType) and  
ContactTagType.userHasCreatePermissionForAtLeastOneContactTagType()
```

These expressions control access to this page. They allow access only to users who have both subtype permission to create the contact and at least one tag permission to create a tag.

**Note:** To see this PCF file, in ContactManager Studio, press `Ctrl+Shift+N` and enter `NewContact`, and then double-click `NewContact.pcf` (`configuration\config\web\pcf\contacts`) in the list of objects that the system finds.

### Check Expression Parameters

Some Gosu permission check expressions require a parameter, and some do not.

For example, the ContactManager tag create permission check expression is the same as the `ABContact.create` permission check expression: `perm.ABContact.create`. When this expression has a `ContactTagType` argument, the expression verifies that the user has permission to create a contact with this tag. The `ContactTagType` arguments for this expression in the base configuration support the following enumeration constants:

- `ContactTagType.TC_CLIENT`
- `ContactTagType.TC_VENDOR`
- `ContactTagType.TC_CLAIMPARTY`

The expression `perm.ABContact.create(ContactTagType.TC_CLIENT)` verifies that the user has permission to create a new contact with the Client tag. If the user has `anytagcreate` permission, which grants permission for creating contacts with all contact tag types, this permission check returns `true`.

The contact and tag create permission expression can take an `ABContact` type or subtype parameter or a `ContactTagType` typecode specified as an enumeration constant. The contact and tag delete, edit, and view

permission expressions take an ABContact instance as a parameter. All these expressions check for both contact and tag permissions, so no additional tag permission check expression is needed. These expressions are:

- perm.ABContact.create
- perm.ABContact.delete
- perm.ABContact.edit
- perm.ABContact.view

The following table lists the ContactManager contact and tag permission check expressions:

ContactManager Expression	Description
perm.ABContact.create	Takes an input parameter that is either an ABContact type or subtype or a ContactTagType typecode specified as an enumeration constant. Depending on the parameter, verifies that the user has the permission to create either a contact with the specified tag or a contact of the specified type.
perm.ABContact.createpreferred	Does not take an input parameter. Verifies that the user has permission to add the preferred contact in the address book.
perm.ABContact.delete	Takes an ABContact instance as an input parameter. Determines the subtype and contact tag or tags from the instance, and then verifies that the user has the contact and tag permissions to delete the contact.
perm.ABContact.deletepreferred	Does not take an input parameter. Verifies that the user has permission to delete the preferred vendor from the address book.
perm.ABContact.edit	Takes an ABContact instance as an input parameter. Determines the subtype and contact tag or tags from the instance, and then verifies that the user has the contact and tag permissions to edit the contact.
perm.ABContact.editpreferred	Does not take an input parameter. Verifies that the user has permission to edit the preferred vendor in the address book.
perm.ABContact.view	Takes an ABContact instance as an input parameter. Determines the subtype and contact tag or tags from the instance, and then verifies that the user has the contact and tag permissions to view the contact.
perm.ABContact.viewmerge	Does not take an input parameter. Verifies that the user has permission to merge contacts in the address book.
perm.ABContact.viewpending	Does not take an input parameter. Verifies that the user has permission to view pending contacts in the address book.
perm.ABContact.viewsearch	Does not take an input parameter. Verifies that the user has permission to search for contacts in the address book.

You can use the Security Dictionary to get information on the application permission keys. For example, if you click the **System Permissions** filter at the top of the left pane, you see all the permissions listed on the left by code name. If you click the permission code **abedit**, you see that it has the related application permission key **ABContact edit**, which has the associated Gosu check expression **perm.ABContact.edit**. You can filter the list by application permission keys, pages, system permissions, and roles.

#### To build and view the ClaimCenter Security Dictionary:

1. At a command prompt, run the following command from ContactManager/bin:

```
gwab regen-dictionary
```

This command builds the dictionary in the following location:

```
ContactManager/build/dictionary/security/index.html
```

2. To see the dictionary, navigate in a web browser to the location of the dictionary. For example, open:

```
file:///C:/ContactManager/build/dictionary/security/index.html
```

#### Using the security dictionary, you can determine the following:

- The system permission related to an application permission key
- PCF files and widgets that use an application permission key

- Roles, application permission keys, PCF pages, and widgets that use a system permission
- Gosu application permission expressions called from each PCF page
- Permissions assigned to each role

## Contact Search Security Configuration Parameters in ContactManager

There are two parameters you can set in the `ContactManager config.xml` file to configure security for contact searches, `RestrictSearchesToPermittedItems` and `RestrictContactPotentialMatchToPermittedItems`.

### To edit `config.xml`

1. Start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:

```
gwab studio
```

2. In the Project window, navigate to `configuration → config` and then double-click `config.xml`.

You can use the `RestrictSearchesToPermittedItems` configuration parameter to control the interaction between the permissions `abviewsearch` and `abview`. The `abviewsearch` permission determines which users can search for contacts. 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 contact and tag permissions. For example, if a user can view only the `Person` subtype with any tag 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.

## Configuring ContactManager Contact Security

Use the `SystemPermissionType` typelist and the `security-config.xml` configuration file to define new ContactManager contact security permissions. These resources enable you to create more finely grained system permissions for specific contact subtypes and tags.

This topic includes:

- “Creating Permissions for an ABContact Subtype in ContactManager” on page 115
- “Creating Client Tag Permissions in ContactManager” on page 117

### Creating Permissions for an ABContact Subtype in ContactManager

1. If necessary, start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:

```
gwab studio
```

2. In Studio, press `Ctrl+Shift+N` and search for `SystemPermissionType.ttx`.

3. Double-click `SystemPermissionType.ttx` in the search results to open this typelist in an editor.

4. Add one or more new typecodes to `SystemPermissionType.ttx`.

For example, for each of the following contact permission typecodes, right-click an existing typecode and choose **Add new → typecode**. Then enter the information for the new typecode.

Code	Name	Description
abpersoncreate	Create an ABPerson instance	Permission to create an instance of an ABPerson subtype
abpersonview	View an ABPerson instance	Permission to view an instance of an ABPerson subtype
abpersonedit	Edit an ABPerson instance	Permission to edit an instance of an ABPerson subtype
abpersondelete	Delete an ABPerson instance	Permission to delete an instance of an ABPerson subtype

5. In the Project window, navigate to **configuration → config → security** and then double-click **security-config.xml**.
6. Associate the new permissions with an ABContact subtype in the **security-config.xml** file. If you see a message asking if you want to edit a copy of the file, click **Yes**.

For example, map the new typecodes to the ABPerson contact subtype as follows:

```
<ContactPermissions>
  <ContactSubtypeAccessProfile entity="ABPerson">
    <ContactCreatePermission permission="abpersoncreate"/>
    <ContactViewPermission permission="abpersonview"/>
    <ContactEditPermission permission="abpersonedit"/>
    <ContactDeletePermission permission="abpersondelete"/>
  </ContactSubtypeAccessProfile>
</ContactPermissions>
```

**Note:** This entry might be the only **ContactPermissions** entry in the file. By default, the file contains a set of **StaticHandler** entries for system permissions.

7. Stop the ContactManager server, optionally regenerate the data and security dictionaries, and then restart ContactManager, as follows:
  - a. Open a command prompt in the **ContactManager/bin** directory and then enter the following command:  
`gwab dev-stop`
  - b. At a command prompt open in **ContactManager/bin**, optionally regenerate the data and security dictionaries:  
`gwab regen-dictionary`  
Regenerating the security dictionary is a good way to find out if there is a problem with the new definitions.
  - c. Restart ContactManager:  
`gwab dev-start`
8. Add the new permissions to a new user role.
  - a. Log in to ContactManager as a user that has the User Admin role, such as user name **su** with password **gw**.
  - b. Click the **Administration** tab.
  - c. In the sidebar, click **Users & Security → Roles**.
  - d. In the **Roles** screen, click **Add Role**.
  - e. For **Name** enter **ABPerson Manager**.
  - f. Add the following set of permission to the role.

For each of the following permissions, click **Add** and then click in the new field. Then choose a permission from the drop-down list:

**Create an ABPerson instance**

Create contact with any tag  
Delete an ABPerson instance  
Delete contact with any tag  
Edit an ABPerson instance  
Edit contact with any tag  
View address book contact search pages  
View an ABPerson instance  
View contact with any tag

- g. Click **Update** to add the new permissions to the role.
9. Assign the role to one or more users. You have a list of users if you loaded sample data, as described at “Step 2: Load Sample Data” on page 40.  
For example:
  - a. In the sidebar, navigate to **Users & Security** → **Roles**.
  - b. On the **Roles** screen, click **ABPerson Manager**.
  - c. Click the **Users** tab and click **Edit**, and then click **Add**.
  - d. On the **Search Users** screen, enter S in the **Last Name** field and click **Search**.
  - e. If you have the sample data loaded, **Steve Smith** is one of the **Search Results**. Click **Select** next to the name.
  - f. Click **Update** to add **Steve Smith** to the **ABPerson Manager** role.With the **ABPerson Manager** role, **Steve Smith** now has permission to create, edit, delete, and view a person or any **ABPerson** subtype. In addition, **Steve** has permission to create, edit, delete, or view a contact with any tag. These permissions are required so **Steve** can see the menus, screens, and widgets associated with creating, editing, or deleting a person.
10. Log in as **ssmith** with password **gw** and test to see if this user can create a new **ABPerson** or **ABPerson** subtype and edit and delete contacts of these types. Also, verify that searching for contacts shows only **ABPerson** or **ABPerson** subtypes.

### Creating Client Tag Permissions in ContactManager

The base application comes with a set of tag permissions that permit a user to create, delete, edit, and view contacts with any tag. You can add permissions that control access to contact tags at a more granular level. This topic shows how to create two sets of tag permissions in ContactManager.

In this topic, you create a set of permissions that control access to contacts that have the Client tag. Because ClaimCenter assigns the Claim Party tag automatically for participants in a claim, this topic also creates permissions for Claim Party tags. You then create a role that includes both sets of tags and assign it to a user.

**IMPORTANT** Tag permissions can cause more restriction on access to contacts than you might intend. A set of permissions for a single tag apply exclusively to that tag. If you want to control access to contact tags at this more granular level, create a set of tag permissions for every contact tag in your system. Then create roles using tag permissions and assign the roles to users based on the tags you expect each subset of contacts to have.

For example, a user is in a role that has only the Client tag permissions and no other tag permissions. That user cannot see contacts that have additional tags, even if those contacts have the Client tag. That user was previously able to see a contact with the Client tag, but now can no longer see that same contact. The reason is that the contact filed a claim, and ClaimCenter automatically tagged the contact with the ClaimParty tag. Since the user's role does not have the Claim Party tag permissions, that user can no longer see the Client contact. Adding the Claim Party tag permissions to the role enables the user to see and work with contacts who have both tags, Client and Claim Party, and no other tags. However, with only these two sets of tag permissions, this user would not be able to see a contact that is tagged both as a Client and a Vendor.

For more information on the tags supplied with the base configuration, see “ContactManager Contact Subtype and Tag Permissions” on page 112.

#### To add new Claim Party and Vendor permissions and a role that uses them

1. If necessary, start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:  
`gwab studio`

2. In the Project window, navigate to `configuration → config → extensions → Typelist` and then double-click `SystemPermissionType.ttx` to open this typelist in an editor.

3. Add the following typecodes.

For each of the following contact tag permission typecodes, right-click an existing typecode and choose **Add new → typecode**. Then enter the information for the new typecode.

Code	Name	Description
<code>claimpartytagcreate</code>	Create contact with Claim Party tag	Permission to create a contact with a Claim Party tag
<code>claimpartytagdelete</code>	Delete contact with Claim Party tag	Permission to delete a contact with a Claim Party tag
<code>claimpartytagedit</code>	Edit contact with Claim Party tag	Permission to edit a contact with a Claim Party tag
<code>claimpartytagview</code>	View contact with Claim Party tag	Permission to view a contact with a Claim Party tag
<code>clienttagcreate</code>	Create contact with Client tag	Permission to create a contact with a Client tag
<code>clienttagdelete</code>	Delete contact with Client tag	Permission to delete a contact with a Client tag
<code>clienttagedit</code>	Edit contact with Client tag	Permission to edit a contact with a Client tag
<code>clienttagview</code>	View contact with Client tag	Permission to view a contact with a Client tag

4. In the Project window on the left, navigate to **configuration** → **config** → **security** and double click **security-config.xml**.
5. Associate the new permissions with the Client and Claim Party tags in the **security-config.xml** file.

Add the following new typecodes:

```
<ContactPermissions>
  <ContactTagAccessProfile tag="ClaimParty">
    <ContactCreatePermission permission="claimpartytagcreate"/>
    <ContactDeletePermission permission="claimpartytagdelete"/>
    <ContactEditPermission permission="claimpartytagedit"/>
    <ContactViewPermission permission="claimpartytagview"/>
  </ContactTagAccessProfile>
  <ContactTagAccessProfile tag="Client">
    <ContactCreatePermission permission="clienttagcreate"/>
    <ContactDeletePermission permission="clienttagdelete"/>
    <ContactEditPermission permission="clienttagedit"/>
    <ContactViewPermission permission="clienttagview"/>
  </ContactTagAccessProfile>
</ContactPermissions>
```

**Note:** If you previously added other contact permissions, you already have a **ContactPermissions** element. In that case, add these two **ContactTagAccessProfile** elements to the existing **ContactPermissions** element.

6. Stop the ContactManager server, regenerate the data and security dictionaries, and then restart ContactManager, as follows:
  - a. If ContactManager is running, open a command prompt in the **ContactManager/bin** directory and then enter the following command:  
`gwab dev-stop`
  - b. To ensure that your new permissions are correctly formatted, at a command prompt, navigate to **ContactManager/bin** and then regenerate the data and security dictionaries:  
`gwab regen-dictionary`
  - c. Restart ContactManager:  
`gwab dev-start`
7. Add one or more new permissions to a user role. For example:
  - a. Log in to ContactManager as a user that has the User Admin role, such as user name **su** with password **gw**.
  - b. Click the **Administration** tab.
  - c. In the sidebar, click **Users and Security** → **Roles**.
  - d. In the **Roles** screen, click **Add Role**.
  - e. For **Name** enter **Client ContactManager**.
  - f. Add the following set of permission to the role.

For each of the following permissions, click **Add** and then click in the new field. Then choose a permission from the drop-down list:

Create address book contacts  
Create contact with Claim Party tag  
Create contact with Client tag  
Delete address book contacts  
Delete contact with Claim Party tag  
Delete contact with Client tag  
Edit address book contacts  
Edit contact with Claim Party tag  
Edit contact with Client tag  
View address book contact search pages  
View address book contacts  
View contact with Claim Party tag  
View contact with Client tag

g. Click **Update** to add the new permissions to the role.

8. Click **Actions** → **New User**.

9. Enter the following values for the new user:

Field	Value
First name	Pat
Last name	Hu
Username	phu
Password	gw

10. Under Roles, click **Add**.

11. Click the empty **Name** field and choose **Client ContactManager** from the list.

12. Click **Update** to save the new user.

13. Log in as phu with password gw and ensure that this user can edit and delete contacts that have the Client tag, the Claim Party tag, and both tags.

The sample data has contacts with all these tags set. You can also log in as another user, like su, and create users with various tags for testing. To load sample data, see “Step 2: Load Sample Data” on page 40.

14. Search for contacts with Tag specified as Vendor and verify that no contacts are returned.

15. Create a user and verify that you can assign only Claim Party and Client tags.

#### See also

- “Overview of Contact Security” on page 109

## PolicyCenter Contact Security

The primary discussion of PolicyCenter security is at “Security: Roles, Permissions, and the Community Model” in the *PolicyCenter Application Guide*.

Additionally, there is an example that uses PolicyCenter contact security at “Configuring ContactManager-to-PolicyCenter Authentication” on page 76.

There are *ab* and *anytag* contact permissions, such as `abedit`, `abcreate`, `anytagedit`, and `anytagcreate`, that are part of a general contact security infrastructure that supports all Guidewire applications. This infrastructure also provides permission check expressions like `perm.Contact.createab` and `perm.Contact.editab`. You can configure and extend these permissions and expressions in PolicyCenter.

The following ClaimCenter topics describe how these permissions and expressions work in ContactManager and ClaimCenter:

- “ClaimCenter Contact Subtype and Tag Permissions” on page 123
- “ClaimCenter Contact and Tag Permission Check Expressions” on page 124

## BillingCenter Contact Security

This topic discusses BillingCenter permissions and permission check expressions used to secure access to contact related tasks and screens.

This topic includes:

- “BillingCenter Contact-Related Permissions” on page 121
- “BillingCenter Contact-Related Permission Check Expressions” on page 121

## BillingCenter Contact-Related Permissions

As described at “Roles” in the *BillingCenter Application Guide*, BillingCenter uses roles and permissions to limit tasks that users can perform. For tasks related to contacts, such as editing data for an account contact, BillingCenter provides the following permissions:

Name	Code	Description
Create account contact	acctcntcreate	Permission to add a new contact to an account
Create policy contact	plcycntcreate	Permission to add a new contact to a policy period
Create producer contact	prodcntcreate	Permission to add a new contact to a producer
Delete account contact	acctcntdelete	Permission to remove a contact from an account
Delete policy contact	plcycntdelete	Permission to remove a contact from a policy period
Delete producer contact	prodcntdelete	Permission to remove a contact from a producer
Edit account contact	acctcntedit	Permission to edit information on an account contact
Edit policy contact	plcycntedit	Permission to edit information on an existing policy period contact
Edit producer contact	prodcntedit	Permission to edit information on an existing producer contact
View account contacts screen	acctcontview	Permission to view Accounts → Contacts screen
View policy contacts screen	plcycontview	Permission to view Policies → Contacts screen
View producer contacts screen	prodcontview	Permission to view Producers → Contacts screen

For an example that uses BillingCenter contact permissions, see “Configuring ContactManager-to-BillingCenter Authentication” on page 79.

Additionally, there are *ab* and *anytag* contact permissions, such as *abedit*, *abcreate*, *anytagedit*, and *anytagcreate* that are part of a general contact security infrastructure that supports all Guidewire applications. The following topic describes how these permissions and expressions work in ContactManager:

- “ContactManager Contact Subtype and Tag Permissions” on page 112

## BillingCenter Contact-Related Permission Check Expressions

BillingCenter uses permission check expressions to control access to contact-related screens and widgets. Each contact permission check expression is associated with a permission. For example, the `NewAccountContactPopup` widget has its `CanVisit` property set to the permission check expression `perm.AccountContact.create`. This setting allows only users who have the `acctcntcreate` permission to see this popup.

The base configuration of BillingCenter uses the following contact-related permission check expressions:

BillingCenter Expression	Description
<code>perm.AccountContact.create</code>	Related permission is <code>acctcntcreate</code> .
<code>perm.AccountContact.delete</code>	Related permission is <code>acctcntdelete</code> .
<code>perm.AccountContact.edit</code>	Related permission is <code>acctcntedit</code> .
<code>perm.PolicyPeriodContact.create</code>	Related permission is <code>plcycntcreate</code> .
<code>perm.PolicyPeriodContact.delete</code>	Related permission is <code>plcycntdelete</code> .
<code>perm.PolicyPeriodContact.edit</code>	Related permission is <code>plcyntedit</code> .
<code>perm.ProducerContact.create</code>	Related permission is <code>prodcntcreate</code> .

BillingCenter Expression	Description
perm.ProducerContact.delete	Related permission is prodcntdelete.
perm.ProducerContact.edit	Related permission is prodcntedit.

The Guidewire contact security infrastructure that provides the *ab* and *anytag* contact permissions also provides permission check expressions like `perm.Contact.createab` and `perm.Contact.editab`. You can configure and extend these permissions and expressions in BillingCenter.

You can use the Security Dictionary to get information on the application permission keys. For example, if you click the **System Permissions** filter at the top of the left pane, you see all the system permissions listed on the left by code name. If you click the permission code `acctcntcreate`, you see that it has the related application permission key **AccountContact create**, which has the Gosu check expression `perm.AccountContact.create`. You can filter the list by application permission keys, pages, system permissions, and roles.

#### To build and view the BillingCenter Security Dictionary:

- At a command prompt, run the following command from `BillingCenter/bin`:

```
gwbc regen-dictionary
```

This command builds the dictionary in the following location

```
BillingCenter/build/dictionary/security/index.html
```

- Navigate in a web browser to the location of the dictionary. For example:

```
file:///C:/billingCenter/build/dictionary/security/index.html
```

#### 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

The following ClaimCenter topic describes how permission check expressions work in ContactManager and ClaimCenter:

- “ClaimCenter Contact and Tag Permission Check Expressions” on page 124

## ClaimCenter Contact Security

To get the most out of this discussion of ClaimCenter contact security, you need to understand ClaimCenter permissions and configuration values. See the following references:

- “Security: Roles, Permissions, and Access Controls” on page 447 in the *Application Guide*
- “Securing Access to ClaimCenter Objects” on page 93 in the *System Administration Guide*

This topic includes:

- “ClaimCenter Roles” on page 123
- “ClaimCenter Contact Subtype and Tag Permissions” on page 123
- “ClaimCenter Contact and Tag Permission Check Expressions” on page 124
- “Contact Permission Search Configuration Parameters in ClaimCenter” on page 126
- “Configuring ClaimCenter Contact Security” on page 126

## ClaimCenter Roles

As described in “Security: Roles, Permissions, and Access Controls” on page 447 in the *Application Guide*, a role is a collection of permissions. By grouping permissions into roles, you can define a user’s authority with a few assigned roles, rather than with a much larger list of permissions. A user can have any number of roles, but has to have at least one.

You might need more granular control over who gets to view, edit, create, and delete contacts, rather than using the simple view and edit permissions. Some carriers have specific ContactManager users that manage certain subtypes of contacts and, therefore, want the system to enforce permissions at the contact subtype level. This role is especially important for the Service Provider Management feature, of which the list of contact subtypes—service providers—is an integral part. Only specific ContactManager users can manage the lists of these contact subtypes.

The permissions used in the base configuration of ClaimCenter for ContactManager are abview, abviewsearch, abedit, abeditpref, abccreate, abccreatepref, abdelete, abdeletepref, anytagcreate, anytagdelete, anytagedit, and anytagview. These permissions are described in more detail in the next topic.

With administrator privileges, you can assign permissions to particular users for certain contacts or contact subtypes. For example, you can grant one user the ability to manage the Auto Body Shops contact subtype and another the permission to manage other contact subtypes.

## ClaimCenter Contact Subtype and Tag Permissions

ClaimCenter provides subtype and tag 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 subtype and tag permissions in your system.

The following table lists the base subtype and tag permissions provided with ClaimCenter contacts:

Code	Enables a User to
anytagview	See a contact that has any contact tag.
anytagcreate	Create a new contact regardless of which contact tag it requires.
anytagdelete	Delete a local, unlinked contact that has any contact tag.
anytagedit	Edit a contact that has any contact tag.
ctcview	View and search local contacts.
ctccreate	Create a new, local contact.
ctcedit	Edit local contacts.
abview	View the details of contact entries retrieved from ContactManager.
abviewsearch	Search for contact entries in ContactManager.
abccreate	Create a new vendor contact in ContactManager. In the base configuration, this permission enables a ClaimCenter user to create a vendor contact and have it saved in ContactManager. Without this permission, a ClaimCenter user can create and save non-vendor contacts in ContactManager. Any vendor contacts created by a user without this permission are created in ContactManager with pending status and must be approved by a ContactManager user.
abccreatepref	Create a new preferred vendor in ContactManager.
abedit	Edit an existing vendor contact stored in ContactManager. In the base configuration, this permission enables a ClaimCenter user to edit a vendor contact and have it saved in ContactManager. Without this permission, a ClaimCenter user can edit and save non-vendor contacts in ContactManager. Any vendor contact changes by a user without this permission become pending changes in ContactManager and must be approved by a ContactManager user.
abeditpref	Edit an existing preferred vendor stored in ContactManager.

The system uses role-based security for these permissions. As described in the previous topic, to implement role-based security, a system administrator associates permissions with roles 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` and `anytagcreate` permissions enables the user who has this role to create any type of contact.

The base contact and tag permissions apply across all contact subtypes. If you grant a permission to a contact type, you grant the same permissions to all that contact's subtypes.

ClaimCenter enables you to restrict permissions according to contact subtype or tag. For example, you can enable a user with `abcreate` permission to create only `PersonVendor` contacts, but not `CompanyVendor` contacts. You configure contact and tag permissions through the `SystemPermissionType` typelist and the `security-config.xml` resource.

**Note:** If you create a set of tag permissions for a specific tag, these permissions enable access only to contacts that have that one tag. For example, you create a set of `Vendor` tag permissions and a user has a role with only those tag permissions. That user will not be able to work with a contact that has both `Claim Party` and `Vendor` tags. You could also create a set of tag permissions for `Claim Party` tags. In that case, a user with both `Vendor` and `Claim Party` tag permissions would be able to work with contacts that have both `Vendor` and `Claim Party` tags.

#### See also

- For examples of combinations of permissions that permit various kinds of work with contacts in ClaimCenter, see “Contact Permissions and Contacts” on page 522 in the *Application Guide*.
- For examples showing how to add typecodes to the `SystemPermissionType` typelist, set up `security-config.xml`, and implement role-based security, see “Configuring ClaimCenter Contact Security” on page 126.

## ClaimCenter Contact and Tag 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 permission to perform an operation.

For example, the setting for the ClaimCenter `AddressBookContactDetail` page's `canVisit` attribute is `perm.Contact.viewab(externalContact.Contact)`. This setting limits users of this page to viewing only contact types for which they have subtype permissions and tag permissions to view the contact. Additionally, the `canEdit` attribute setting is `externalContact.Source.supportsUpdate()` and `perm.Contact.editab(externalContact.Contact)`. This setting limits a user of this page to editing only contacts that support updates and for which the user has subtype permissions and tag permissions to edit the contact.

**Note:** To see this file, in ClaimCenter Studio, press `Ctrl+Shift+N` and enter `AddressBookContactDetail`, and then click `AddressBookContactDetail.pcf` (`configuration\config\web\pcf\addressbook`) in the list of objects that the system finds.

Some Gosu permission check expressions require an input parameter, and some do not. The following table lists the ClaimCenter contact and tag permission check expressions:

ClaimCenter Expression	Description
<code>perm.Contact.createab</code>	Takes an input parameter that is either a <code>Contact</code> type or subtype or a <code>ContactTagType</code> typecode specified as an enumeration constant. Depending on the parameter, verifies that the user has the permission to create either a contact with the specified tag or a contact of the specified type.
<code>perm.Contact.createpreferredab</code>	Does not take an input parameter. Verifies that the user has permission to create the preferred contact.

ClaimCenter Expression	Description
perm.Contact.deleteab	Takes a Contact instance as an input parameter. Determines the subtype and contact tag or tags from the instance, and then verifies that the user has the contact and tag permissions to delete the contact.
perm.Contact.deletepreferredab	Does not take an input parameter. Verifies that the user has permission to delete the preferred contact.
perm.Contact.editab	Takes a Contact instance as an input parameter. Determines the subtype and contact tag or tags from the instance, and then verifies that the user has the contact and tag permissions to edit the contact.
perm.Contact.editpreferredab	Does not take an input parameter. Verifies that the user has permission to edit the preferred contact.
perm.Contact.viewab	Takes a Contact instance as an input parameter. Determines the subtype and contact tag or tags from the instance, and then verifies that the user has the contact and tag permissions to view the contact.
perm.Contact.viewsearchab	Does not take an input parameter. Verifies that the user has permission to edit the preferred contact in the address book.
perm.Contact.createlocal	Does not take an input parameter. Verifies that the user has permission to create the local contact.
perm.Contact.editlocal	Requires a Contact instance as an input parameter. Verifies that the user has permission to edit either an existing local contact or a user contact. To edit a user contact, the user needs edit user permission.
perm.Contact.viewlocal	Does not take an input parameter. Verifies that the user has permission to view and search local contact entries.

ClaimCenter does not use tag permission checks to control access to screens or buttons. However, when a user tries to save a new contact to ContactManager, ClaimCenter applies the `createab` permission check expression. This check ensures that the user has permission to create the tag as well as the contact. If the user does not have this permission, ContactManager creates a pending contact that requires approval in ContactManager to become permanent.

You can use the Security Dictionary to get information on the application permission keys. For example, if you click the **System Permissions** filter at the top of the left pane, you see all the permissions listed on the left by code name. If you click the permission code `abedit`, you see that it has the related application permission key `Contact editab`, which has the associated Gosu check expression `perm.Contact.editab`. You can filter the list by application permission keys, pages, system permissions, and roles.

#### To build and view the ClaimCenter Security Dictionary:

- 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
```

- 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

## Contact Permission Search Configuration Parameters in ClaimCenter

There are two parameters you can set in the ClaimCenter config.xml file to control how searching for externally stored contacts interacts with contact permission. Those parameters are `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 Project window, navigate to configuration → config and double-click config.xml.

You can use the `RestrictSearchesToPermittedItems` configuration parameter to control the interaction between the permissions `abviewsearch` and `abview`. 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 contact and tag permissions. For example, if a user can view the Person subtype with any tag 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.

## Configuring ClaimCenter Contact Security

Use the `SystemPermissionType` typelist and the `security-config.xml` configuration file to define new ClaimCenter contact security permissions. These two files enable you to create more finely grained system permissions for contact subtypes and tags.

This topic includes:

- “Creating Contact Permissions for a Subtype in ClaimCenter” on page 126
- “Creating Claim Party and Vendor Tag Permissions in ClaimCenter” on page 129

### Creating Contact Permissions for a Subtype 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 Project window, navigate to configuration → configuration → Extensions → Typelist.

3. Double-click `SystemPermissionType.ttx` to open this typelist in the editor.

4. Add the following new typecodes to `SystemPermissionType.ttx`.

For each of the following contact permission typecodes, right-click an existing typecode and choose `Add new` → `typecode`. Then enter the information for the new typecode.

Code	Name	Description
companyvendorcreate	Create company vendor contacts	Permission to create company vendor contacts
companyvendoredit	Edit company vendor contacts	Permission to edit company vendor contacts
companyvendordelete	Delete company vendor contacts	Permission to delete company vendor contacts

Code	Name	Description
companyvendorview	View company vendor contacts	Permission to view company vendor contacts
personvendorcreate	Create person vendor contacts	Permission to create person vendor contacts
personvendoredit	Edit person vendor contacts	Permission to edit person vendor contacts
personvendordelete	Delete person vendor contacts	Permission to delete person vendor contacts
personvendorview	View person vendor contacts	Permission to view person vendor contacts

5. In the Project window, navigate to **configuration** → **config** → **security** and double-click **security-config.xml**.

6. Associate the new permissions with a Contact subtype in the **security-config.xml** file.

Add the new typecodes for the Vendor and VendorPerson contact subtype permissions to the **ContactPermissions** element as follows:

```
<ContactPermissions>
  ...
  <ContactSubtypeAccessProfile entity="CompanyVendor">
    <ContactCreatePermission permission="companyvendorcreate"/>
    <ContactEditPermission permission="companyvendoredit"/>
    <ContactDeletePermission permission="companyvendordelete"/>
    <ContactViewPermission permission="companyvendorview"/>
  </ContactSubtypeAccessProfile>
  <ContactSubtypeAccessProfile entity="PersonVendor">
    <ContactCreatePermission permission="personvendorcreate"/>
    <ContactEditPermission permission="personvendoredit"/>
    <ContactDeletePermission permission="personvendordelete"/>
    <ContactViewPermission permission="personvendorview"/>
  </ContactSubtypeAccessProfile>
  ...
</ContactPermissions>
```

7. Stop the ClaimCenter server, regenerate the data and security dictionaries, and then restart ClaimCenter, as follows:

- a. If ClaimCenter is running, open a command prompt in the **ClaimCenter/bin** directory and then enter the following command:

```
gwcc dev-stop
```

- b. To verify that the new permissions are correct, at a command prompt, navigate to **ClaimCenter/bin** and regenerate the data and security dictionaries:

```
gwcc regen-dictionary
```

- c. After you get a successful build of the dictionaries, restart ClaimCenter:

```
gwcc dev-start
```

8. Add the 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 **su** with password **gw**.

- b. Click the **Administration** tab.

- c. Click **Users & Security** → **Roles** in the sidebar.

- d. Click **Add Role**.

The New Role screen opens.

- e. For **Name**, enter **Vendor Admin**.

- f. For **Description**, enter **Manages vendor contacts. Requires Clerical role as well.**

- g. Click **Add** to add a new permission.

- h. Click the new field and choose the **Create company vendor contacts** permission from the drop-down list.

- i. Repeat step g and step h for each of the following permissions, substituting the actual permission for Create company vendor contacts in step h:

Create person vendor contacts  
Delete company vendor contacts  
Delete person vendor contacts  
Edit company vendor contacts  
Edit person vendor contacts  
View company vendor contacts  
View person vendor contacts

- j. Click **Update** to add the new role.

9. Remove the **abview** permission from the Clerical role.

**Note:** Removing this permission is a quick way to test the new Vendor Admin role. With **abview** removed, using the Clerical role in conjunction with Vendor Admin makes it possible to limit users with the two roles from viewing non-vendor contacts. However, after this change, users who have the Clerical role and no other will be unable to view any contacts. For this change to be permanent, another role with **abview** permission would need to be created and then used in conjunction with the Clerical role for all previous Clerical users.

- a. Click the **Administration** tab.
- b. Click **Users & Security** → **Roles** in the sidebar.
- c. Click **Clerical** and then click **Edit**.
- d. Click the **Code** column heading to list **abview** at the top.
- e. Click the check box for **abview**, and then click **Remove**.
- f. Click **Update** to save your changes.

10. Create a new user for the role.

- a. Click **Actions** → **New User**.  
The **New User** screen opens.
- b. For **First name**, enter **Devra**.
- c. For **Last name**, enter **Rajan**.
- d. For **User name**, enter **drajan**.
- e. For **Password** and **Confirm Password**, enter **gw**.
- f. Under **Roles**, click **Add**, and then click the **Name** field. Choose **Vendor Admin** from the drop-down list.
- g. Under **Roles**, click **Add** again, and then click the **Name** field. Choose **Clerical** from the drop-down list.
- h. Under **Groups**, click **Add** and then click the **Group** field. Choose a group from the list, such as the sample data group **Western Regional Claims Center**.
- i. Click **Update** to create the new user.

11. Log out.

For example, if you logged in as su, on the Options menu , click **Log Out Super User**.

12. Log in as **drajan** with password **gw**.
13. Verify that this user can create a new **CompanyVendor** or **PersonVendor** subtype and have it saved in **ContactManager** without being pending.
14. Verify that this user can edit and delete contacts of these types and have the changes saved in **ContactManager** without the contacts being made pending.

- 15.** Verify that in an Address Book search, this user can click in the search results and see details only for CompanyVendor and PersonVendor contact subtypes and not for non-vendor subtypes. For example, clicking a contact of type Company or Person displays a message saying that you do not have permission to view the contact detail popup.

## Creating Claim Party and Vendor Tag Permissions in ClaimCenter

The base application comes with a set of tag permissions that permit a user to create, delete, edit, and view all tags. You can add permissions that control access to tags at a more granular level. This topic shows how to create a set of tag permissions in ClaimCenter.

**Note:** In this topic, you create a set of Vendor tag permissions. These permissions enable a user to see and work with contacts that have only the Vendor tag. If the contact has any other tags, these permission do not enable working with that contact. For example, a user who has a single role with only Vendor tag permissions is not able to work with a contact that has both Claim Party and Vendor tags. You could create a set of tag permissions for Claim Party tags and add them to the role that has the Vendor tag permissions. A user with that role would be able to work with contacts that have both Vendor and Claim Party tags. However, that user would not be able to work with contacts that have both Vendor and Client tags.

For more information on the application tags in the base configuration, see “ClaimCenter Contact Subtype and Tag Permissions” on page 123.

### To create Claim Party and Vendor tag permissions

1. If necessary, start ClaimCenter Studio.

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

```
gwcc studio
```

2. In the Project window, navigate to **configuration** → **config** → **Extensions** → **Typelist**.

3. Double-click `SystemPermissionType.etc` to open this typelist in an editor.

4. Add new typecodes to `SystemPermissionType.etc`.

For each of the following contact permission typecodes, right-click an existing typecode and choose **Add new** → **typecode**. Then enter the information for the new typecode.

Code	Name	Description
<code>claimpartytagcreate</code>	Create contact with Claim Party tag	Permission to create a contact with a Claim Party tag
<code>claimpartytagdelete</code>	Delete contact with Claim Party tag	Permission to delete a contact with a Claim Party tag
<code>claimpartytagedit</code>	Edit contact with Claim Party tag	Permission to edit a contact with a Claim Party tag
<code>claimpartytagview</code>	View contact with Claim Party tag	Permission to view a contact with a Claim Party tag
<code>vendortagcreate</code>	Create contact with Vendor tag	Permission to create a contact with a Vendor tag
<code>vendortagdelete</code>	Delete contact with Vendor tag	Permission to delete a contact with a Vendor tag
<code>vendortagedit</code>	Edit contact with Vendor tag	Permission to edit a contact with a Vendor tag
<code>vendortagview</code>	View contact with Vendor tag	Permission to view a contact with a Vendor tag

5. In the Project window, navigate to **configuration** → **config** → **security** and double-click `security-config.xml`.

6. Associate the new permissions with a contact tag in the `security-config.xml` file.

For example, add the new typecodes for the Vendor tag permissions to the `ContactPermissions` element as follows:

```
<ContactPermissions>
  ...
  <ContactTagAccessProfile tag="Vendor">
    <ContactCreatePermission permission="vendortagcreate"/>
    <ContactDeletePermission permission="vendortagdelete"/>
```

```
<ContactEditPermission permission="vendortagedit"/>
<ContactViewPermission permission="vendortagview"/>
</ContactTagAccessProfile>
...
</ContactPermissions>
```

7. Stop the ClaimCenter server, regenerate the data and security dictionaries, and then restart ClaimCenter, as follows:

- If ClaimCenter is running, open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:

```
gwcc dev-stop
```

- To verify that the new permission are correctly formatted, at a command prompt in `ClaimCenter/bin`, regenerate the data and security dictionaries:

```
gwcc regen-dictionary
```

- Restart ClaimCenter:

```
gwcc dev-start
```

8. Add the new permissions to a user role. For example:

- Log in to ClaimCenter as a user that has the User Admin role, such as user name `su` with password `gw`.

- Click the **Administration** tab.

- Click **Users & Security** → **Roles** in the sidebar.

- Click **Add Role**.

The New Role screen opens.

- For **Name**, enter **Vendor & Claim Party Tag Admin**.

- For **Description**, enter **Manages contacts with Vendor and Claim Party tags. Requires Clerical role as well.**

- Click **Add** to add a new permission.

- Click the new field and choose the **Create contact with Vendor tag** permission from the drop-down list.

- Repeat step g and step h for each of the following permissions, substituting the actual permission for **Create contact with Vendor tag** in step h:

Create address book contact  
Create contact with Claim Party tag  
Delete address book contact  
Delete contact with Claim Party tag  
Delete contact with Vendor tag  
Edit address book contact  
Edit contact with Claim Party tag  
Edit contact with Vendor tag  
View address book contact  
View contact with Claim Party tag  
View contact with Vendor tag

- Click **Update** to add the new role.

9. Remove the `anytagview` permission from the Clerical role.

**Note:** Removing this permission is a quick way to test the new Vendor & Claim Party Admin role. However, after this change, users who have the Clerical role and no other will be unable to view contacts that have tags. For contacts stored in ContactManager, all contacts must have tags, so they would not be visible. For this change to be permanent, another role with `anytagview` permission would need to be created and then used in conjunction with the Clerical role for all previous Clerical users.

- Click the **Administration** tab.

- b. Click **Users & Security** → **Roles** in the sidebar.
  - c. Click **Clerical** and then click **Edit**.
  - d. Click the **Code** column heading to list anytagview on the first page.
  - e. Click the check box for anytagview, and then click **Remove**.
  - f. Click **Update** to save your changes.
10. Create a new user for the role.
  - a. Click **Actions** → **New User**.  
The **New User** screen opens.
  - b. For **First name**, enter **Mira**.
  - c. For **Last name**, enter **NoLo**.
  - d. For **User name**, enter **mnolo**.
  - e. For **Password** and **Confirm Password**, enter **gw**.
  - f. Under **Roles**, click **Add**, and then click the **Name** field. Choose **Vendor & Claim Party Tag Admin** from the drop-down list.
  - g. Under **Roles**, click **Add** again, and then click the **Name** field. Choose **Clerical** from the drop-down list.
  - h. Under **Groups**, click **Add** and then click the **Group** field. Choose a group from the list, such as the sample data group **Eastern Regional Claims Center**.
  - i. Click **Update** to create the new user.
11. Log out.  
For example, if you logged in as su, on the Options menu , click **Log Out Super User**.
12. Log in as **mnolo** with password **gw**.
13. Verify that this user can find contacts with Vendor and Claim Party tags set, or with only one tag or the other set. If necessary, either create those contacts in ContactManager or import ContactManager sample data.
14. Verify that this user can create, edit, and delete contacts with these tags and have the changes saved in ContactManager without the contacts being made pending.

**See also**

- “Overview of Contact Security” on page 109
- “Security: Roles, Permissions, and Access Controls” on page 447 in the *Application Guide*
- “Securing Access to ClaimCenter Objects” on page 93 in the *System Administration Guide*



# Extending the Contact Data Model

This topic describes the `Contact` and `ABContact` entity hierarchies. It presents information about extending these entities, and provides examples of extensions, including changes to the user interface to support the new subtypes.

This topic includes:

- “Overview of Contact Entities” on page 133
- “Extending the Client Data Model” on page 141
- “Extending the Vendor Contact Data Model” on page 142
- “Changing the Subtype of a Contact Instance” on page 172

## Overview of Contact Entities

This topic describes the contact data model and provides an overview of what to consider when planning to extend that data model.

This topic includes:

- “`ABContact` Data Model” on page 134
- “`Contact` Data Model” on page 135
- “Guidewire Core Application and `ContactManager` Contact Entity Hierarchy” on page 137
- “General Guidelines for Extending Contacts” on page 139
- “Deciding Whether to Create a Subtype” on page 139
- “Limitations on Configuring Extensions” on page 140
- “Working with Contact Mapping Files” on page 140

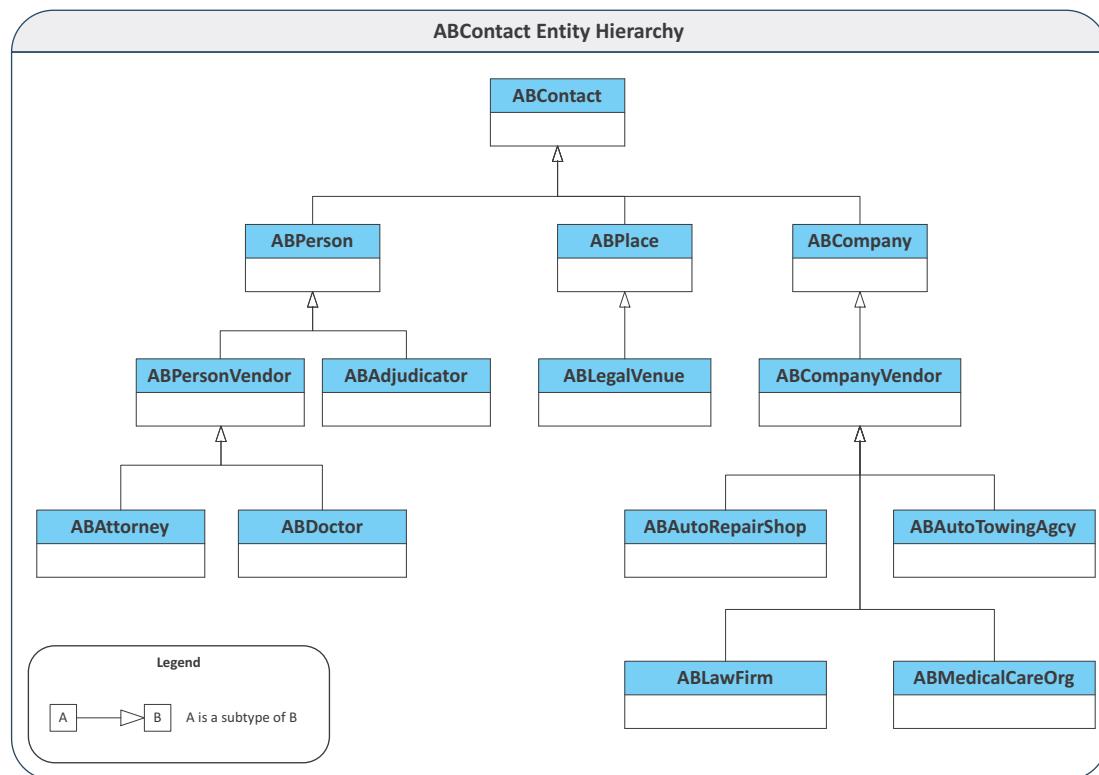
## ABContact Data Model

The ABContact entity is the ContactManager data model entity used in both client and vendor contact management. This entity has fields for name, phone number, email address, and so forth. For many vendor contacts it is necessary to maintain a tax ID for tax reporting. The ABContact entity tracks this data as well.

The ABContact entity has three primary subtypes for various types of contacts, ABPerson, ABCompany, and ABPlace.

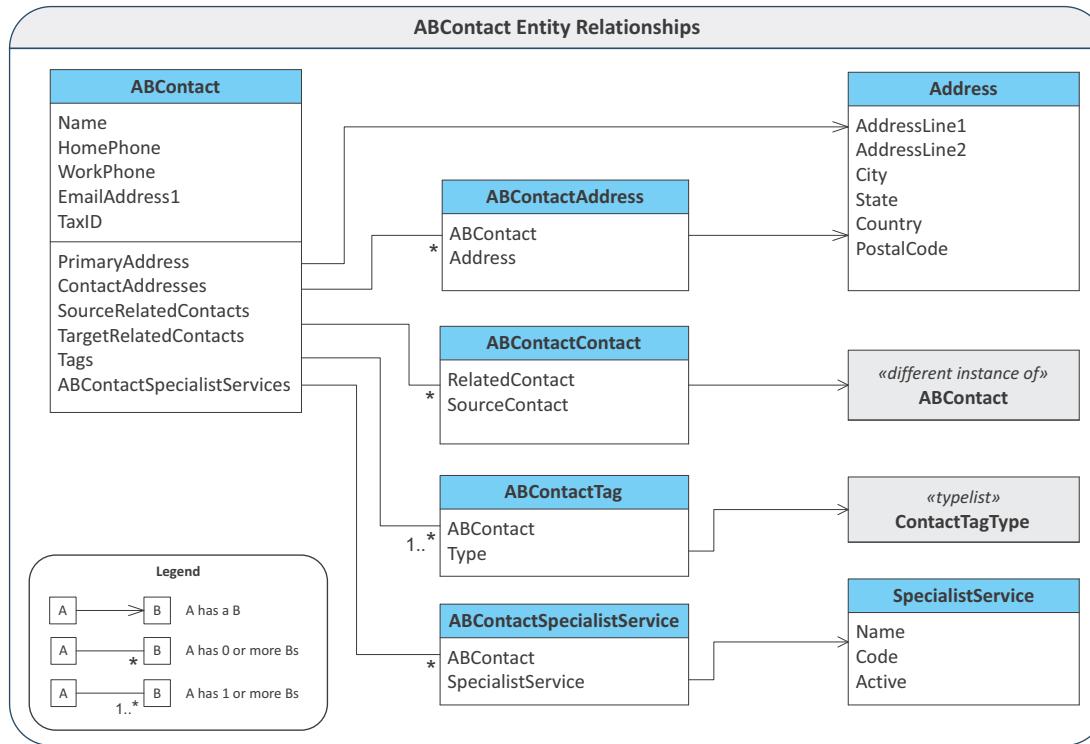
These subtypes have additional subtypes, like ABAdjudicator, ABCompanyVendor, ABLegalVenue, and so on. The following figure shows the ABContact entity hierarchy. This entity hierarchy has a parallel in the Contact entity hierarchy in the core applications. See “Guidewire Core Application and ContactManager Contact Entity Hierarchy” on page 137.

**Note:** This figure omits the subtypes ABUserContact, ABPolicyCompany, and ABPolicyPerson. Although they are available in the base configuration, these entities are not intended to be used in ContactManager.



The ABContact entity is associated with other entities as well. In the base configuration, a contact must have at least one tag. A contact can have multiple addresses, related contacts, tags, and services. References to those

entities are handled with arrays of join entities, such as `ContactAddress` for addresses and `ContactContact` for related contacts, and derived properties, as shown in the following figure:



The physical location, the address, of an `ABContact` is not maintained in the `ABContact` entity. Instead, the `ABContact` entity references another entity, the `Address` entity, which maintains the contact's street or mailing address. An `ABContact` entity can reference a primary address and, through the `ABContactAddress` entity, other secondary addresses.

Contacts can have relationships with other contacts. For example, an `ABPerson` contact might be employed by a particular `ABCompany` contact. The `ABContactContact` entity maintains data about the relationships a contact can have with other contacts.

**Note:** For simplicity, the diagram shows `ABContactContact` connecting to a different instance of `ABContact`. However, `ABContactContact` can also point back to the original contact. For example, you can be your own primary contact.

Contacts can have tags, like Client and Vendor, and specialist services that the contact provides, like carpentry or independent appraisal. An `ABContact` entity references its tags by using an array of `ABContactTag` entities. An `ABContact` entity references its services by using an array of `ABContactSpecialistService` entities.

## Contact Data Model

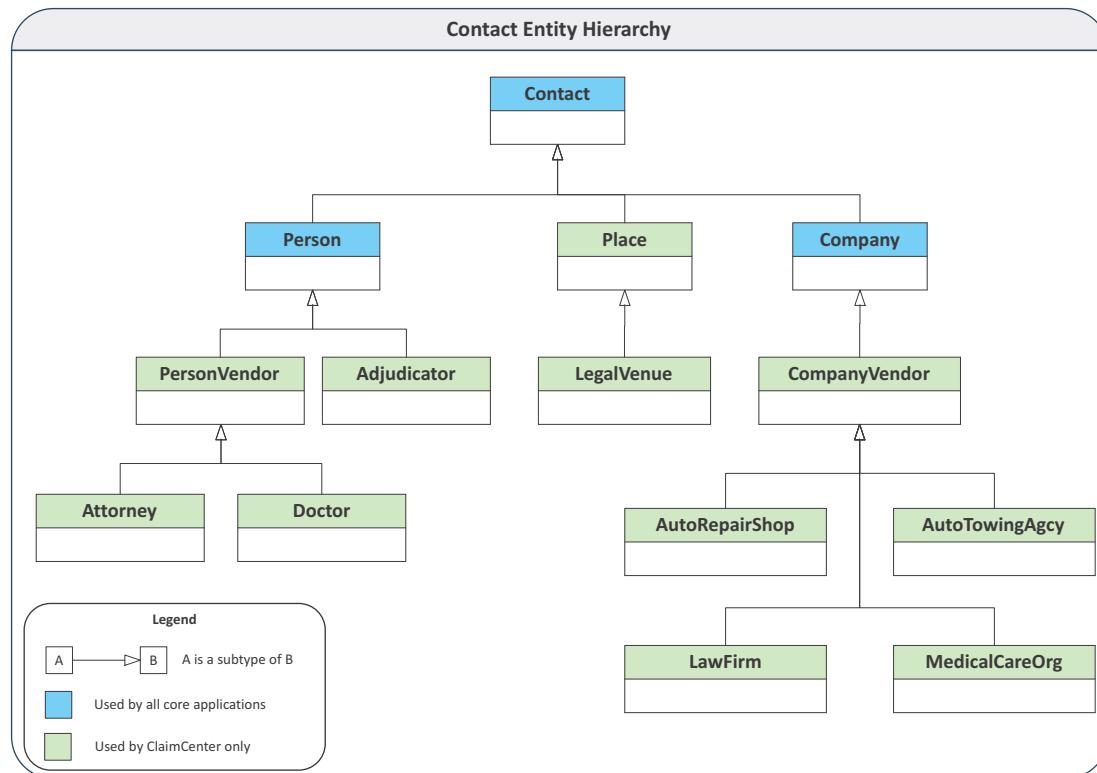
A `Contact` is the Guidewire core application data model entity used in both client and vendor contact management. In the base configuration, this entity is the core application equivalent of the `ContactManager` entity `ABContact`, described previously in “`ABContact` Data Model” on page 134.

As with `ABContact`, the `Contact` entity has subtypes for various types of contacts, like `Person`, `Company`, and `Place`. In their base configurations, `PolicyCenter` and `BillingCenter` use just the `Person` and `Company` subtypes.

In `ClaimCenter`, these subtypes have additional subtypes, like `Adjudicator`, `CompanyVendor`, `LegalVenue`, and so on. The following figure shows the `Contact` entity hierarchy. This entity hierarchy has a parallel in the

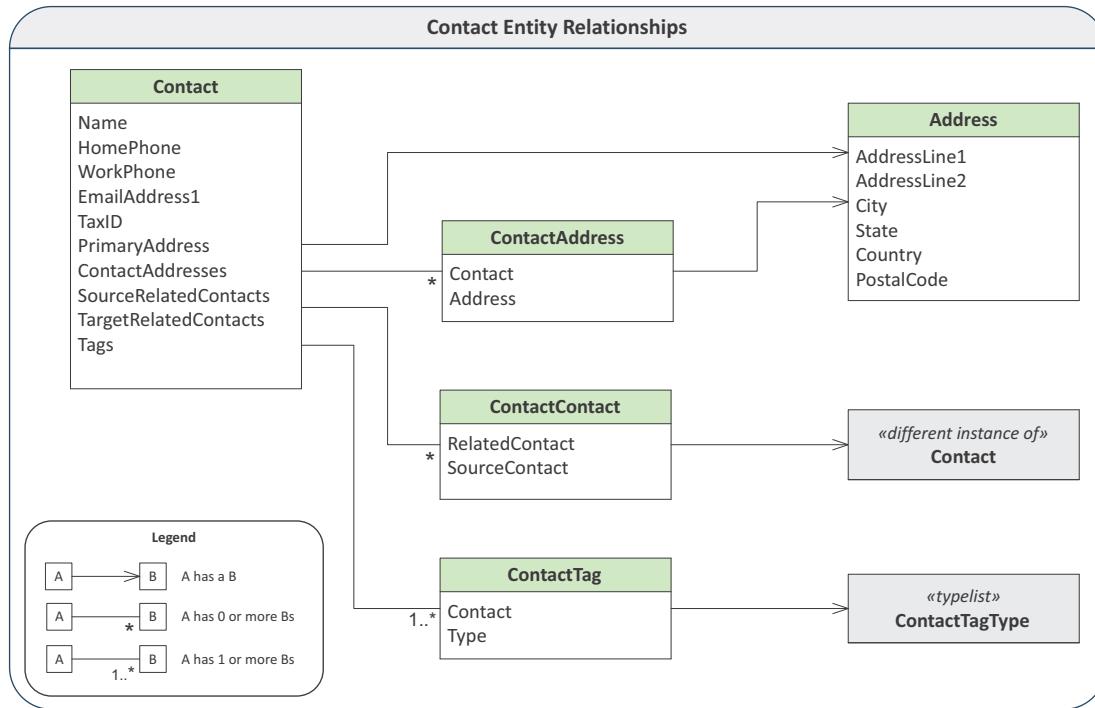
ABContact entity hierarchy in ContactManager. See “Guidewire Core Application and ContactManager Contact Entity Hierarchy” on page 137.

**Note:** The following figure does not include the Contact subtype UserContact. This entity is used by the User entity, which represents a user of the application. The User entity has a foreign key to UserContact so it can store data like the user’s address and phone number. However, UserContact entities are not intended to be used as either vendor contacts or client contacts, and in the base configuration they cannot be stored in ContactManager.



The **Contact** entity is associated with other entities as well. A contact can have multiple addresses, related contacts, and tags. A **Contact** entity must have at least one tag. Except for the primary address, references to

those entities are handled with arrays of join entities. For example, there is `ContactAddress` for addresses and `ContactContact` for related contacts, as shown in the following figure:



The physical location, the address, of a `Contact` is stored in the `Address` entity, which maintains the contact's street or mailing address. A `Contact` can reference a primary address and, through the `ContactAddress` entity, other secondary addresses.

Contacts can have relationships with other contacts. For example, a `Person` might be employed by a particular `Company`. The `ContactContact` entity maintains data about the relationships a contact can have with other contacts.

**Note:** For simplicity, the diagram shows `ContactContact` connecting to a different instance of `Contact`. However, `ContactContact` can also point back to the original contact. For example, you can be your own primary contact.

Contacts can have tags, like Client and Vendor, and specialist services that the contact provides, like carpentry or independent appraisal. A `Contact` entity references its tags by using the `ContactTag` entity. The relationship between a contact and its services are maintained by `ContactManager`, which is why there is no property accessing services in the `Contact` entity relationship model.

#### See also

- “Contact Tags” on page 177
- “Vendor Services” on page 181

## Guidewire Core Application and ContactManager Contact Entity Hierarchy

Both the Guidewire core application and ContactManager data models group and classify contact data as hierarchies. By default, ContactManager provides a contact entity hierarchy that supports both the Client Data Management add-on module and ClaimCenter vendor data management. Additionally, ContactManager entities have the prefix AB (for *Address Book*).

PolicyCenter and BillingCenter use only part of the hierarchy for client data management: **Contact**, **Person**, and **Company**.

ClaimCenter uses the entire hierarchy. In the default application, all the ClaimCenter contact types appear in ContactManager with AB prefixes. The following table shows the ClaimCenter hierarchy and the supporting hierarchy in ContactManager supplied in the base products:

<b>ClaimCenter</b>	<b>ContactManager</b>
Contact	ABContact
Person	ABPerson
Adjudicator	ABAdjudicator
PersonVendor	ABPersonVendor
Attorney	ABAAttorney
Doctor	ABDoctor
Company	ABCompany
CompanyVendor	ABCompanyVendor
AutoRepairShop	ABAutoRepairShop
AutoTowingAgcy	ABAutoTowingAgcy
LawFirm	ABLAWFIRM
MedicalCareOrg	ABMedicalCareOrg
Place	ABPlace
LegalVenue	ABLegalVenue

Each core application **Contact** entity and subentity has a corresponding **ABContact** entity or subentity in ContactManager. When ContactManager creates a contact, an instance of an **ABContact** entity or subentity, ContactManager also creates a unique identifier for that instance. This unique identifier is used both by core applications and by ContactManager to ensure that each application is referencing the same contact. In the core applications, **AddressBookUID** is the field used to uniquely identify a contact that is stored in ContactManager. In ContactManager, the corresponding field is **LinkID**.

For a contact stored both in ContactManager and in a core application, ContactManager sends the unique identifier in the **LinkID** field to each core application that uses the contact. The core application stores this unique identifier in the **AddressBookUID** field of the contact. If a contact in a core application has a non-null value in its **AddressBookUID** field, that value means that the contact is stored in ContactManager. Otherwise, the **AddressBookUID** field is **null**, meaning that the contact is stored only locally in the core application, and not in ContactManager.

To enable all applications to reference the same contact, you must not change an **AddressBookUID** or a **LinkID** field. For more information, see “ContactManager Link IDs and Comparison to Other IDs” on page 271.

Guidewire designed the data models to enable the Guidewire core applications and ContactManager to be compatible. If you make a contact-related extension to a Guidewire core application’s data model, you typically extend the ContactManager 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. If you have more than one core application installed, you might have to make the same extension in your other core applications as well.

Mapping of entities between applications ensures that contact records are stored with the correct entity in both the Guidewire core application and ContactManager. Mapping is not the same as linking contacts or *synchronizing* them, determining if the records are the same between the two applications. Linking and synchronizing are separate operations from mapping, as described in “Linking and Synchronizing Contacts” on page 191.

### See also

- “General Guidelines for Extending Contacts” on page 139

- “Deciding Whether to Create a Subtype” on page 139
- “Limitations on Configuring Extensions” on page 140
- “Working with Contact Mapping Files” on page 140
- “Extending the Vendor Contact Data Model” on page 142

## General Guidelines for Extending Contacts

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

- You cannot delete or move the root entity **Contact**.
- You cannot delete or move the three subtypes **Person**, **Company**, and **Place**.
- You cannot create a contact entity that is a peer to **Contact**.
- You cannot create a contact entity that is a parent to **Person**, **Company**, or **Place**.

You can add a field to any of these entities. If you add a field to an entity that is higher in the hierarchy, all entities below it inherit the field. Before adding a field, ensure that the field makes sense for all the entity’s subtypes.

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 the other subtypes.

If you have integrated your Guidewire core application with ContactManager, you typically extend both the core application and the ContactManager hierarchies to mirror each other. Most contacts require central management. If you have a contact subtype that does not require central management, you can create that type just in the Guidewire core application and not in ContactManager.

The Guidewire core applications provide a Gosu class that you use when extending the **Contact** data model—**ContactMapper**. The corresponding class you use in ContactManager is `gw.webservice.ab.ab800.abcontactapi.ContactMapper`. Each Guidewire core application also has a pair of XML configuration files for mapping contact names and typelists to and from ContactManager. For more information on mapping files, see “Working with Contact Mapping Files” on page 140.

The matching and searching functions for contacts require each subtype to have a collection of fields that makes it unique. See “Linking and Synchronizing Contacts” on page 191 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, limit the number of subtypes you need to represent your contacts.

If you have ClaimCenter installed, consider using vendor services rather than creating new vendor contact subtypes. See “Vendor Services” on page 181.

Another alternative is to create your own tags and apply them to contacts. See “Contact Tags” on page 177.

The main reason to limit creation of new subtypes is that the more you specialize the subtype hierarchy, the more restrictive and the less flexible your model becomes. For example, there are subtypes for **AutoRepairShop** and **AutoTowingAgcy**. If you work with an auto repair shop that also does towing, you cannot create a single contact that does both. However, you can add a service for Towing to an auto repair shop contact.

If a contact has different roles or skill sets, you can use one subtype and add contact tags or a specialty array to represent those skills and specialties.

Additionally, each time you create a new subtype, you must modify PCF files to support both creating the subtype and searching for it. You might also need to make supporting modifications to the screens that reference contacts.

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

1. Stop the applications 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 enables you to verify that your changes work before rebuilding the application.
3. If you have made a data model change to ContactManager, start ContactManager to refresh its web services. Then start Studio for the core application and refresh the web service collection for that web service.
4. Start the applications and confirm your changes.

For examples of modifications to contact entities and the class hierarchy, see the following topics:

- “Example of Adding a Field to a Contact Subtype” on page 142
- “Example of Adding a Vendor Contact Subtype” on page 147
- “Extending Contacts with an Array” on page 158

## Limitations on Configuring Extensions

There are limitations on the kinds of things you can do with contact type extensions. Your subtype can inherit from only a single parent entity—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 fields of both entities. To create the subtype you want, you can select one entity or the other and subtype it as something like `Practice`. Then, you add the fields that are missing from the other entity 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 ContactManager. For example, suppose you configure ClaimCenter to make `Adjudicator` not a choice in the user interface. However, in the ContactManager database there are a number of `ABAdjudicator` contact entities. Searching the `Address Book` for a `Person` returns `ABAdjudicator` matches if they exist in ContactManager. For information about restricting access to contact subtypes, see “Securing Access to Contact Information” on page 109.

## Working with Contact Mapping Files

If you extend your Guidewire core application’s contact data model, for ContactManager to be able to work with the extension, you must also make a matching extension in ContactManager. You then map the entities to each other in both the Guidewire core application and in ContactManager.

A Guidewire core application uses the `ContactMapper` class to map the fields of contact entities sent to and received from ContactManager. There is a mapping class in each Guidewire core application and in ContactManager. The mapping classes, set up by default for the base application `Contact` types and the base ContactManager `ABContact` types, match Guidewire core application entity types to entity types in ContactManager.

If you extend the contact data model, you must edit these classes and map every field for the new entity that you want to send and receive. In each `ContactMapper` class in the Guidewire core application and in ContactManager, you map both incoming and outgoing entities. If you leave a field out, it is not processed.

Additionally, you might need to map `Contact` subtypes and typecodes whose names in a Guidewire core application are different from the names in ContactManager. For example, Guidewire core applications use the `Person`

subtype, which in ContactManager is `ABPerson`. All contact data passed through the ContactManager web services use the ContactManager domain namespace. Therefore, it is up to the core applications to map names, like `Contact`, between the core application domain namespace and the ContactManager domain namespace, which uses `ABContact`. There is a Gosu class in each Guidewire core application for this purpose as described in “Core Application Mapping” on page 141.

**Note:** There might be situations in which you add a contact subtype to a Guidewire core application and not to ContactManager. If you have a contact subtype that does not require central management, you can create that type in the Guidewire core application only and not use the mapping files.

## ContactManager Mapping

In ContactManager, you use the class `gw.contactmapper.ab800.ContactMapper` to map contact entity fields between ContactManager and ClaimCenter or one of the other Guidewire core applications.

You can access this class in ContactManager Studio from the Project window. Navigate to `configuration → gsrc`, and then open `gw.contactmapper.ab800.ContactMapper`.

For reference information on this class, see “ContactManager ContactMapper Class” on page 286.

## Core Application Mapping

In core applications, you use the class `gw.contactmapper.ab800.ContactMapper` to map the fields of contact entities between ClaimCenter and ContactManager.

You can access this class in Guidewire Studio from the Project window. Navigate to `configuration → gsrc` and open `gw.contactmapper.ab800.ContactMapper`.

For reference information on this class, see “ContactMapper Class” on page 285.

While `ContactMapper` maps the fields of the entities, it does not map differing entity names. ContactManager contact entity names, such as `ABContact` or `ABPerson`, typically start with AB. Core application contact entity names, such as `Contact` or `Person`, typically do not start with AB. Additionally, there can be differences in typecodes between the entities. Core applications do their own name mapping—no changes are needed in ContactManager, and there is no equivalent class in ContactManager.

To support mapping of differing entity names and typecodes, the core applications use the following name mapping classes:

- **ClaimCenter** – `gw.contactmapper.ab800.CCNameMapper`
- **PolicyCenter** – `gw.contactmapper.ab800.PCNameMapper`
- **BillingCenter** – `gw.contactmapper.ab800.BCNameMapper`

For an example of how to use one of these classes, see “Step 3: Map the Subtype Names” on page 149.

## Extending the Client Data Model

The techniques required to extend the client data model are essentially the same as those required to extend the vendor data model. See “Extending the Vendor Contact Data Model” on page 142.

The primary differences are:

- The client data model includes only the `Contact`, `Person`, and `Company` entities.
- All three core Guidewire applications use client data. If you have more than one core application installed, you must apply your extension changes to all the applications. For example, you have installed the entire InsuranceSuite set of applications. If you extend the client data model in PolicyCenter and ContactManager, you must also extend it in ClaimCenter and BillingCenter.
- PolicyCenter and BillingCenter PCF files are different from those of ClaimCenter.

**See also**

- “Overview of Contact Entities” on page 133
- “Extending the Vendor Contact Data Model” on page 142

## Extending the Vendor Contact Data Model

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

This topic includes:

- “Example of Adding a Field to a Contact Subtype” on page 142
- “Example of Adding a Vendor Contact Subtype” on page 147
- “Extending Contacts with an Array” on page 158

### Example of Adding a Field to a Contact Subtype

This topic describes how to extend a contact with a new field. In the example, you add a new `BoardCertified_Ext` field to the `Doctor` subtype in ClaimCenter and to the `ABDoctor` subtype in ContactManager. You then update the necessary files and screens to make the new field usable across the applications.

**Note:** Before beginning this example, if you have not done so already, follow the instructions for installing ContactManager in “Installing ContactManager” on page 39. Additionally, you must have integrated ContactManager with ClaimCenter as described in “Integrating ContactManager with Guidewire Core Applications” on page 45. If you do not have any users or claims defined, you might also want to load sample data for both applications. See “Step 2: Load Sample Data” on page 40.

The following topics walk you through adding a field to a subtype:

- “Step 1: Add the Subtype Field in ClaimCenter” on page 142
- “Step 2: Add the Field to the ContactMapper Class in ClaimCenter” on page 143
- “Step 3: Add the Subtype Field in ContactManager” on page 143
- “Step 4: Add the Field to the ContactMapper Class in ContactManager” on page 144
- “Step 5: Modify the ClaimCenter Address Book User Interface” on page 144
- “Step 6: Modify the ClaimCenter Claim Contacts User Interface” on page 145
- “Step 7: Modify the ContactManager User Interface” on page 145
- “Step 8: Restart Both Applications and Test” on page 146

#### Step 1: Add the Subtype Field in ClaimCenter

**To extend the entity in ClaimCenter**

1. If necessary, start ClaimCenter Studio.

At a command prompt, navigate to the `ClaimCenter/bin` directory and enter:  
`gwcc studio`

2. In the Project window, navigate to `configuration → config → Extensions → Entity`.
3. Double-click `Doctor.eti` to open it in the Entity editor.
4. Click `subtype` at the top of the Element hierarchy.
5. Click the drop-down list for  and choose `column`.

6. Enter the following values for the new column:

Name	Value
name	BoardCertified_Ext
type	bit
nullok	false
desc	Is the doctor Board certified in the specialty?
default	false

7. Regenerate the data dictionary to ensure that your changes were correct.

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

```
gwcc regen-dictionary
```

## Step 2: Add the Field to the ContactMapper Class in ClaimCenter

For ClaimCenter to be able to send the new field to ContactManager and receive updates for the field, you need to add it to the `ContactMapper` class. For more information on this class, see “ContactMapper Class” on page 285.

### To update this class

1. In ClaimCenter Studio, press `Ctrl+N` and enter `ContactMapper`, and then double-click the class in the search results to open it in the Gosu editor.
2. Press `Ctrl+F` and search for the following line of code:  
`fieldMapping(Doctor#MedicalLicense),`
3. On a new line after the `MedicalLicense` line, enter `fieldMapping(Doctor#Board` and press `CTRL+SPACE` to complete the property name, and then add a comma. The new line will be:  
`fieldMapping(Doctor#BoardCertified_Ext),`
4. If necessary, stop ClaimCenter as follows:
  - a. Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:  
`gwcc dev-stop`
  - b. Wait to restart the ClaimCenter application and pick up the data model changes until you have made the ClaimCenter user interface changes later in this topic.

## Step 3: Add the Subtype Field in ContactManager

### To extend the entity in ContactManager

1. If necessary, start ContactManager Studio as follows:

At a command prompt, navigate to the `ContactManager/bin` directory and enter:  
`gwab studio`

2. In the Project window, navigate to `configuration → config → Extensions → Entity`.
3. Double-click `ABDoctor.eti` to open it in the Entity editor.
4. Click `subtype` at the top of the Element hierarchy.
5. Click the drop-down list next to  and choose `column`.

6. Enter the following values for the new column:

Name	Value
name	BoardCertified_Ext
type	bit
nullok	true
desc	Is the doctor Board certified in the specialty?
default	false

7. Regenerate the data dictionary to ensure that your changes are correct.

At a command prompt, navigate to the `ContactManager/bin` directory and enter:

```
gwab regen-dictionary
```

#### Step 4: Add the Field to the ContactMapper Class in ContactManager

To be able to send the new field to a Guidewire core application and receive updates for the field, you need to add it to the `ContactMapper` class. For more information on this class, see “ContactMapper Class” on page 285.

##### To update this class

1. In ContactManager Studio, click `Ctrl+N` and enter `ContactMapper`, and then double-click the class in the search results to open it in the Gosu editor.
2. Press `Ctrl+F` to find the following line of code:  
`fieldMapping(ABDoctor#MedicalLicense),`
3. Add the following code for the new field after the line for the `MedicalLicense` field:  
`fieldMapping(ABDoctor#BoardCertified_Ext),`
4. If necessary, stop ContactManager as follows:
  - a. Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`
  - b. Wait to restart the ContactManager application and pick up the data model changes. First make the user interface changes in “Step 7: Modify the ContactManager User Interface” on page 145.

#### Step 5: Modify the ClaimCenter Address Book User Interface

In this topic you modify the ClaimCenter Address Book user interface so you can see the new field with the `Doctor` subtype. To do so, you update the input sets used by the detail view.

1. In Studio, in the Project window in Project view, navigate to `configuration → config → Localizations → en_US`.
2. Double-click `display.properties` to open it in the editor.
3. Press `CTRL+F` and enter `Web.ContactDetail.Doctor.MedicalLicense` to find that entry in the file.
4. Add a line under that entry, and then enter the following key and value:  
`Web.ContactDetail.Doctor.BoardCertified = Board Certified`
5. In the Project window, navigate to `configuration → config → Page Configuration → pcf → addressbook`. Then double-click `AddressBookDoctorAdditionalInfoInputSet.Doctor` to open it in the editor.
6. Select the Input widget `Medical Specialty` and copy it, and then paste the copy below this widget.
7. Right-click the new field and choose `Change element type`.
8. Click `Boolean Radio Button Input` in the drop-down list.

9. Click the new BooleanRadioInput widget and set the following properties:

---

<code>editable</code>	<code>true</code>
<code>id</code>	<code>DoctorBoardCertified</code>
<code>label</code>	<code>displaykey.Web.ContactDetail.Doctor.BoardCertified</code>
<code>required</code>	<code>false</code>
<code>value</code>	<code>(personVendor as Doctor).BoardCertified_Ext</code>

---

### Step 6: Modify the ClaimCenter Claim Contacts User Interface

This part of the configuration enables you to edit the new Doctor field when working with contacts for a claim.

1. In the Project window, navigate to `configuration → config → Page Configuration → pcf → shared → contacts`. Then double-click `DoctorAdditionalInfoInputSet.Doctor` to open it in the editor.
2. Select the Input widget `Medical Specialty` and copy it, and then paste the copy below this widget.
3. Right-click the new field and choose `Change element type`.
4. Click `Boolean Radio Button Input` in the drop-down list.
5. Click the new BooleanRadioInput widget and set the following properties:

---

<code>editable</code>	<code>true</code>
<code>id</code>	<code>DoctorBoardCertified</code>
<code>label</code>	<code>displaykey.Web.ContactDetail.Doctor.BoardCertified</code>
<code>required</code>	<code>false</code>
<code>value</code>	<code>Doctor.BoardCertified_Ext</code>

---

### Step 7: Modify the ContactManager User Interface

This topic describes how to modify the ContactManager user interface to enable users to use the new field when they create and edit ABDocor contacts. You add the new field to the input set for the ABDocor subtype for use in the `ContactBasicsDV.ABPerson` detail view.

1. In Studio, in the Project window in Project view, navigate to `configuration → config → Localizations → en_US`.
2. Double-click `display.properties` to open it in the editor.
3. Click `Ctrl+F` and enter `Web.ContactDetail.Doctor.MedicalLicense` to find that entry in the file.
4. Add a line after that entry and then enter the following key and value:  
`Web.ContactDetail.Doctor.BoardCertified = Board Certified`
5. In the Project window, navigate to `configuration → config → Page Configuration → pcf → contacts → basics`.
6. Double-click `ABPersonVendorSpecialtyInputSet.ABDocor` to open it in the editor.
7. Select the Input widget `Medical Specialty` and copy it, and then paste the copy below the `Medical Specialty` widget.
8. Right-click the new field and choose `Change element type`.
9. Click `Boolean Radio Button Input`.
10. Click the new BooleanRadioInput widget and set the following properties:

---

<code>editable</code>	<code>true</code>
<code>id</code>	<code>DoctorBoardCertified</code>

---

label	displaykey.Web.ContactDetail.Doctor.BoardCertified
required	false
value	(person as ABDoctor).BoardCertified_Ext

### Step 8: Restart Both Applications and Test

1. If necessary, stop both ClaimCenter and ContactManager, and then restart both applications.
2. Log in to ClaimCenter as a user who can create new contacts. For example, log in as user **ssmith** with password **gw**.
3. Open an existing claim.
4. Click **Parties Involved** in the left Sidebar, and then on the **Contacts** screen choose **New Contact → Vendor → Doctor** to open the **New Doctor** screen.
5. Verify that **Board Certified** is listed in the **Additional Info** section.
6. Enter enough information to create a Doctor contact. The minimum is **First name**, **Last name**, and **Tax ID**. Include a **Medical Specialty** and click **Yes** for **Board Certified**.
7. At the top of the edit screen in the **Roles** list view table, click **Add**.
8. Click the **Role** cell for the new entry and choose a role from the drop-down list that the new Doctor vendor will take on the claim. For example, choose **Doctor**.
9. Click **Update** to save the new contact to the claim.

ClaimCenter adds the new **Doctor** contact to the list of contacts on the **Contacts** screen. ClaimCenter also ensures that the **ClaimParty** and **Vendor** tags are set for the contact and sends it to ContactManager.

10. Select the new doctor vendor in the **Contacts** list view table.
11. If you logged in as a user with permission to create contacts, below, on the **Basics** card, the message above the contact says:

**This contact is linked to the Address Book and is in sync**

This message means that the new contact has been saved to ContactManager, and the contact data for this contact on this claim is the same for ClaimCenter and ContactManager.

**Note:** It is possible that the contact is still being saved to ContactManager and ClaimCenter has not yet been notified. In that case, the message you see is **Waiting for link message from ContactManager. Refresh screen to get updated status**. You can refresh the screen by clicking another contact or screen and then clicking this contact again.

12. Click **View in Address Book** to see a screen showing the data saved for this contact in ContactManager.
13. On this screen, click **Edit in ContactManager** to open ContactManager and edit the contact.
- You might have to log in to ContactManager if your ClaimCenter user name is not in ContactManager. Log in as a ContactManager user who has **ABContact** view, edit, update, and delete permissions, such as the sample user **aapplegate**.
14. Click **Edit** and make some changes to the contact, such as a new address. Change the setting for the **Board Certified** field. Click **Update** to save your changes.
15. In ClaimCenter, click the **Address Book** tab.
16. On the **Search Address Book** screen, pick **Doctor** from the **Type** drop-down list and search for the doctor you added.
17. Click the contact found by **Search** and verify that the entry found by ClaimCenter in ContactManager is correct and that the data matches the changes you made in ContactManager.
18. Click the drop-down list for the **Claim** tab and choose the claim you previously edited.

19. Click **Parties Involved** in the left sidebar, and then, on the **Contacts** screen, select the doctor you added to this claim.

20. Below, on the **Basics** card, the message above the contact says:

This contact is linked to the Address Book but is out of sync

This message means that the contact data you changed in ContactManager is now different from the contact data for this contact on this claim.

21. Click **Copy from Address Book** to update the contact data for this claim. You see the changes on the **Basic** card and the following message:

This contact is linked to the Address Book and is in sync

22. Click **Edit** and change the setting for the **Board Certified** field, and then click **Update**.

Because you are logged in as a user who can edit contacts, this change is sent to ContactManager.

23. The message above the contact changes to:

This contact is linked to the Address Book but is out of sync

This message means that the change you made to the contact has not yet registered in ContactManager. It can take a few seconds for the message sent to ContactManager to take effect.

24. Click another contact in the **Contacts** list view table, and then click your original Doctor contact to refresh the **Basics** card. If the message has not changed to say that the contact is in sync, wait a few seconds, and then click another contact and then this one again. Eventually, you see the following message:

This contact is linked to the Address Book and is in sync

#### See also

- “Overview of Contact Entities” on page 133
- “Example of Adding a Vendor Contact Subtype” on page 147

## Example of Adding a Vendor Contact Subtype

This topic describes how to extend contacts with a new subtype. In this example, you add a new **Interpreter** subtype to ClaimCenter and its counterpart, **ABInterpreter** to ContactManager. You then update the necessary files and screens to make the new field usable across the applications.

#### Notes:

Creating a Contact subtype is usually not necessary if you define vendor services. See “Vendor Services” on page 181. Additionally, see “Deciding Whether to Create a Subtype” on page 139.

Before beginning this example, if you have not done so already, follow the instructions for installing ContactManager in “Installing ContactManager” on page 39. Additionally:

- You must have integrated ContactManager with ClaimCenter as described in “Integrating ContactManager with Guidewire Core Applications” on page 45.
- After installing ContactManager, for testing purposes, it would be helpful to import sample data as described in “Step 2: Load Sample Data” on page 40.

The following topics walk you through adding a new subtype:

- “Step 1: Add the Subtype to ClaimCenter” on page 148
- “Step 2: Add the Subtype to the ContactMapper Class in ClaimCenter” on page 149
- “Step 3: Map the Subtype Names” on page 149
- “Step 4: Add the Subtype to ContactManager” on page 149
- “Step 5: Add the Subtype to the ContactMapper Class in ContactManager” on page 151
- “Step 6: Modify the ClaimCenter Address Book User Interface” on page 151

- “Step 7: Modify the ContactManager User Interface” on page 155
- “Step 8: Restart Both Applications and Test” on page 157

### Step 1: Add the Subtype to ClaimCenter

1. Start ClaimCenter Studio.

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

```
gwcc studio
```

2. In the Project window open in Project view, navigate to **configuration** → **config** → **Extensions** → **Entity**.

3. Right-click **Entity** and choose **New** → **Entity**.

4. In the **Entity** field, enter the following name:

Interpreter

5. Click the **Entity Type** drop-down list and choose **subtype**.

6. In the **Desc** field, enter the following description:

Interpreter

7. Click the **Supertype** search button  and then choose **PersonVendor** from the drop-down list.

8. Click **OK**.

9. In the Entity editor for the new entity, click **subtype** at the top of the **Element** hierarchy.

10. For **displayName**, enter **Interpreter**.

11. Click the drop-down list next to  and choose **column**.

12. Enter the following values for the new column:

Name	Value
name	InterpreterSpecialty
type	varchar
nullok	false
desc	Interpreter's language specialties

13. Click the drop-down list next to  and choose **params**.

14. Enter the following values to specify the size of this varchar column:

Name	Value
name	size
value	30

15. Regenerate the data dictionary to ensure that your changes are correct.

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

```
gwcc regen-dictionary
```

16. After you have verified that the data model extension is correct, close ClaimCenter Studio, and then restart it so Studio can pick up the data model change.

## Step 2: Add the Subtype to the ContactMapper Class in ClaimCenter

To be able to send the new subtype to ContactManager and receive updates for it, you add it to the ContactMapper class. For more information on this class, see “ContactMapper Class” on page 285.

### To update this class

1. In ClaimCenter Studio, press **Ctrl+N** and enter **ContactMapper**, and then double-click the class in the search results to open it in the Gosu editor.
2. Press **Ctrl+F** and search for the following line of code:  
`fieldMapping(MedicalCareOrg#MedicalOrgSpecialty),`
3. After the line for **MedicalCareOrg**, add the following code for the new entity:  
`fieldMapping(Interpreter#InterpreterSpecialty),`

**Note:** You do not have to put this `fieldMapping` method call in this exact location. It must be in the method with declaration `override` property `get Mappings() : Set<CCPropertyMapping>`. Additionally, it is useful to group it with the other method calls after the comment `//Other Contact subtypes`.

## Step 3: Map the Subtype Names

To be able to send the new subtype to ContactManager and receive updates for it, you must map the ClaimCenter subtype name to the ContactManager subtype name. You do this mapping only in ClaimCenter. The ClaimCenter subtype is **Interpreter** and the ContactManager subtype, which you will create later, is **ABInterpreter**. You edit the following class:

```
gw.contactmapper.ab800.CCNameMapper
```

### To set up mapping for the subtype names

1. In ClaimCenter Studio, press **Ctrl+N** and enter **CCNameMapper**, and then double-click the class in the search results to open it in the Gosu editor.
2. In the editor, press **Ctrl+F** and search for **MedicalCareOrg**. The search finds the following line of code:  
`.entity(MedicalCareOrg, "ABMedicalCareOrg")`
3. Add the following entry below the line for **MedicalCareOrg**.  
`.entity(Interpreter, "ABInterpreter")`
4. If ClaimCenter is running, stop ClaimCenter as follows:
  - a. Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:  
`gwcc dev-stop`
  - b. Wait to restart the ClaimCenter application and pick up the data model changes. First make the ClaimCenter user interface changes in “Step 6: Modify the ClaimCenter Address Book User Interface” on page 151.

## Step 4: Add the Subtype to ContactManager

1. If necessary, start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin`, and enter the following command:

```
gwab studio
```

2. In the Project window open in Project view, navigate to `configuration → config → Extensions → Entity`.
3. Right-click **Entity** and choose **New → Entity**.
4. In the **Entity** field, type the following name:  
`ABInterpreter`
5. Click the **Entity Type** drop-down list and choose **subtype**.

6. In the **Desc** field, type the following description:

Interpreter

7. Click the **Supertype** search button  and choose **ABPersonVendor** from the drop-down list.

8. Click **OK**.

9. In the Entity editor for the new entity, click **subtype** at the top of the **Element** hierarchy.

10. Enter the following value:

Name	Value
Display name	Interpreter

11. Click the drop-down list next to  and choose **column**.

12. Enter the following values for the new column:

Name	Value
name	InterpreterSpecialty
type	varchar
nullok	false
desc	Interpreter's language specialties

13. Click the drop-down list next to  and choose **params**.

14. Enter the following values to specify the size of this varchar column:

Name	Value
name	size
value	30

15. Click **subtype** at the top of the **Element** hierarchy.

16. Click the drop-down list next to  and choose **index**.

17. Enter the following values for the new index:

Name	Value
name	intrprtrindx1
desc	Speed up searches using InterpreterSpecialty field
unique	true

18. Right-click **index** in the **Element** column, and choose **Add new → indexcol**.

This index column is the first of three to add to the index.

19. Enter the following values for the new **indexcol**:

Name	Value
name	InterpreterSpecialty
keyposition	1

20. Right-click **index** in the **Element** column, and choose **Add new → indexcol**.

21. Enter the following values for the new **indexcol**:

Name	Value
name	Retired
keyposition	2

22. Right-click **index** in the **Element** column, and choose **Add new → indexcol**.

23. Enter the following values for the new **indexcol**:

Name	Value
name	id
keyposition	3

24. Regenerate the data dictionary to ensure that your changes are correct.

At a command prompt, navigate to the **ContactManager/bin** directory and enter:

```
gwab regen-dictionary
```

## Step 5: Add the Subtype to the ContactMapper Class in ContactManager

To be able to send the new subtype to a Guidewire core application and receive updates for the subtype, you must add it to the **ContactMapper** class. For more information on this class, see “ContactMapper Class” on page 285.

### To update this class

1. In ContactManager Studio, press **Ctrl+N** and enter **ContactMapper**, and then double-click the class in the search results to open it in the Gosu editor.

2. Press **Ctrl+F** and search for the following line of code:

```
fieldMapping(ABMedicalCareOrg#MedicalOrgSpecialty),
```

3. After the line for **ABMedicalCareOrg**, add the following code for the new entity:

```
fieldMapping(ABIInterpreter#InterpreterSpecialty),
```

**Note:** You do not have to put this **fieldMapping** method call in this exact location. It must be in the method with declaration **override** property **get Mappings() : Set<PropertyMapping>**. Additionally, it is useful to group it with the other method calls after the comment **//Other ABContact subtypes**.

4. If ContactManager is running, you must stop and restart it to pick up the data model change. Wait to restart the ContactManager application until you have made the ClaimCenter user interface changes in “Step 7: Modify the ContactManager User Interface” on page 155.

To stop ContactManager:

- Open a command prompt in the **ContactManager/bin** directory and then enter the following command:  
`gwab dev-stop`

## Step 6: 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.

**Note:** In general, when you create new subtypes, the PCF files you need to edit depend on the parent of 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. Additionally, the PCF files can use different modes to show appropriate fields for different subtypes.

### Step A. Add the Interpreter Display Keys

The new menu items and inputs for Interpreter have labels that require display keys, which support localization.

#### To create the Interpreter display keys

1. In ClaimCenter Studio, in the Project window open in Project view, navigate to configuration → config → Localizations → en\_US.

2. Double-click display.properties to open it in the editor.

3. In the editor, press CTRL+F and enter Web.NewContactMenu.Doctor to find that entry in the file.

4. Add a line under that entry, and then enter the following key and value:

```
Web.NewContactMenu.Interpreter = Interpreter
```

5. In the search field at the top of the editor, enter Web.ContactDetail.Email.Secondary to find that entry in the file.

6. Add a line under that entry, and then enter the following keys and values:

```
Web.ContactDetail.Interpreter = Interpreter
Web.ContactDetail.InterpreterSpecialty = Interpreter Specialty
```

### Step B. Add the Subtype to the New Contact Menu

1. In Studio, navigate in the Project window to configuration → config → Page Configuration → pcf → claim → partiesinvolved.

2. Double-click ClaimContacts to open that PCF file in the editor.

3. On the toolbar above the list view panel with the ID PeopleInvolvedDetailedLV, there is a New Contact button. Click the button to open the embedded menu.

4. Select the Vendor submenu, which has the id ClaimContacts\_NewVendor.

This submenu has menu elements for vendors like Autobody Repair Shop, Doctor, and Medical Care Organization.

5. Right-click the menu item widget for Medical Care Organization and copy it.

6. Paste the copy below the menu item for Medical Care Organization.

This action puts the new menu item in the Vendor submenu. The new item turns red and stays that way until you enter the properties in the next step.

7. Click the newly pasted menu item widget and set the following properties:

action	NewPartyInvolvedPopup.push(claim, typekey.Contact.TC_INTERPRETER)
id	ClaimContacts_Interpreter
label	displaykey.Web.NewContactMenu.Interpreter
showConfirmMessage	true

8. Navigate in the Project window to configuration → config → Page Configuration → pcf → shared → contacts,

9. Double-click ClaimNewServiceRequestSpecialistPickerMenuItemSet to open it in the editor.

10. Right-click the menu item widget for Medical Care Organization and copy it.

11. Paste the copy below the menu item for Medical Care Organization.

This action puts the new menu item in the New Vendor submenu. The new item turns red and stays that way until you enter the properties in the next step.

- 12.** Click the newly pasted menu item widget and set the following properties:

action	NewContactPopup.push(typekey.Contact.TC_INTERPRETER, parentContact, claim)
id	NewInterpreter
label	displaykey.Web.NewContactMenu.Interpreter
showConfirmMessage	true

#### Step C. Edit the Address Book Input Sets

Update the input sets used by the Address Book detail view so you can select an Interpreter contact and see the new Interpreter Specialty field.

- 1.** In the Project window, navigate to configuration → config → Page Configuration → pcf → addressbook.
- 2.** Right-click the node AddressBookDoctorAdditionalInfoInputSet.Doctor and click **Copy**.
- 3.** Right-click again and choose **Paste**.
- 4.** In the **Copy** dialog box, enter the new name `AddressBookInterpreterAdditionalInfoInputSet.Interpreter.pcf`, and then click **OK**.  
The new widget opens in the editor.
- 5.** In the Project window, right-click `AddressBookInterpreterAdditionalInfoInputSet.Interpreter` and choose **Change PCF Mode**.
- 6.** In the **Change Mode** dialog box, verify that **Interpreter** is the mode.
- 7.** In the editor, select the **Medical License** input widget and set the following properties:

editable	true
id	InterpreterSpecialty
label	displaykey.Web.ContactDetail.Interpreter.InterpreterSpecialty
required	false
value	(personVendor as Interpreter).InterpreterSpecialty

- 8.** Select the **Medical Specialty** widget and delete it.
- 9.** If it exists, select the **Board Certified** widget and delete it.
- 10.** In the Project window, navigate to configuration → config → Page Configuration → pcf → addressbook.
- 11.** Right-click `AddressBookContactBasicsDV.Person` and choose **Change PCF Mode**.
- 12.** In the **Change Mode** dialog box, add the **Interpreter** mode to the list:  
`Person|PersonVendor|Adjudicator|UserContact|Doctor|Attorney|Interpreter`
- 13.** Click **OK**.
- 14.** Right-click `AddressBookAdditionalInfoInputSet.PersonVendor` and choose **Change PCF Mode**.
- 15.** In the **Change Mode** dialog box, add the **Interpreter** mode to the list:  
`PersonVendor|Attorney|Doctor|Interpreter`
- 16.** Double-click `AddressBookAdditionalInfoInputSet.PersonVendor` to open it in the editor.
- 17.** Drag a new **Input Set Ref** widget from the **Toolbox** and drop it below the second **InputSetRef** widget, the one containing the input set `AddressBookAttorneyAdditionalInfoInputSet`.

- 18.** Click the new InputSetRef widget and set the following properties:

def	AddressBookInterpreterAdditionalInfoInputSet(contact as PersonVendor)
id	AddressBookInterpreterAdditionalInfoInputSet
mode	contact type is Interpreter ? "Interpreter" : null

The mode statement enables the **Interpreter Specialty** field to display in the **Additional Info** section when the contact type is **Interpreter**.

- 19.** Click Refresh PCF  in the toolbar so you can see the input set you just added to the new InputSetRef widget.

#### Step D. Edit the Shared Contacts Detail View and Input Sets

Update the input sets used by the shared contacts detail view so you can create an **Interpreter** contact on the new contact screen and see the new **Interpreter Specialty** field:

1. In the Project window, navigate to **configuration** → **config** → **Page Configuration** → **pcf** → **shared** → **contacts**.
2. Right-click **ContactBasicsDV.Person** and choose **Change PCF Mode**.
3. In the Change Mode dialog box, add **Interpreter** to the list of modes, as follows:  
**Person|PersonVendor|Adjudicator|UserContact|Doctor|Attorney|Interpreter**
4. In the same Project window folder, **pcf** → **shared** → **contacts**, right-click **DoctorAdditionalInfoInputSet.Dcoter** and click **Copy**.
5. Right-click again and choose **Paste**.
6. In the **Copy** dialog box, enter the new name **InterpreterAdditionalInfoInputSet.Interpreter.pcf**, and then click **OK**.  
The new widget opens in the editor.
7. In the Projects window, right-click **InterpreterAdditionalInfoInputSet.Interpreter** and chose **Change PCF Mode**.
8. In the **Change Mode** dialog box, verify that **Interpreter** is the mode for this input set.
9. In the editor, select the new widget by clicking its identifier at the top of the widget: **InputSet: InterpreterAdditionalInfoInputSet**.
10. In the **Properties** area below, click the **Code** tab, and replace the existing code with the following Gosu statement:  

```
property get Interpreter(): Interpreter { return contactHandle.Contact as Interpreter; }
```
11. Select the **Medical License** input widget and change it to an **Interpreter Specialty** widget by setting the following properties:

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

12. Select the **Medical Specialty** widget and delete it.
13. If it exists, select the **Board Certified** widget and delete it.

14. In the same Project window location, pcf → shared → contacts, right-click AdditionalInfoInputSet.PersonVendor and choose Change PCF Mode.
15. In the Change Mode dialog box, add the Interpreter mode to the list:  
PersonVendor|Attorney|Doctor|Interpreter
16. Double-click AdditionalInfoInputSet.PersonVendor to open it in the editor.
17. Drag a new Input Set Ref widget from the Toolbox and drop it below the widget containing the input set AttorneyAdditionalInfoInputSet.
18. Click the new InputSetRef widget and set the following properties:

---

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

---

The mode statement enables the **Interpreter Specialty** field to show in the **Additional Info** section when the contact type is **Interpreter**.

19. Click Refresh PCF  in the toolbar so you can see the input set you just added to the new InputSetRef widget.

## Step 7: Modify the ContactManager User Interface

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

### Step A. Add Display Keys for ABInterpreter Fields

1. In ContactManager Studio, in the Project window open in Project view, navigate to configuration → config → Localizations → en\_US.
2. Double-click display.properties to open it in the editor.
3. Press CTRL+F and enter Web.NewContactMenu.Doctor to find that entry in the file.
4. Add a line under that entry, and then enter the following key and value:  
Web.NewContactMenu.Interpreter = Interpreter
5. In the Search field at the top of the editor, enter Web.ContactDetail.Email.Secondary to find that entry in the file.
6. Add a line under that entry, and then enter the following keys and values:  
Web.ContactDetail.Interpreter = Interpreter  
Web.ContactDetail.Interpreter.InterpreterSpecialty = Interpreter Specialty

### Step B. Edit the Actions Menu on the ContactManager Contacts Tab

1. In Studio, navigate in the Project window to configuration → config → Page Configuration → pcf → contacts and double-click ContactsMenuActions to open it in the editor.
2. 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.
3. Click the new MenuItem and enter the following values for the Basic properties of this Interpreter menu item:

---

action	NewContact.go(entity.ABInterpreter)
id	ContactsMenuActions_InterpreterMenuItem
label	displaykey.Web.NewContactMenu.Interpreter
showConfirmMessage	true

---

### Step C. Add Interpreter Fields to Modal Input Sets

1. In the Project window, navigate to configuration → config → Page Configuration → pcf → contacts → basics.
2. Right-click ABPersonVendorSpecialtyInputSet.ABDoctor and choose Copy.
3. Right-click again and choose Paste.
4. Name the new file ABPersonVendorSpecialtyInputSet.ABInterpreter.pcf and click OK.
5. Right-click ABPersonVendorSpecialtyInputSet.ABInterpreter and choose Change PCF Mode, and verify that the mode is ABInterpreter.
6. In the new PCF file, select the Medical License input widget and change it to an Interpreter Specialty widget by setting the following properties:

---

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

---

7. Select the Medical Specialty widget and delete it.
8. If it exists, select the Board Certified widget and delete it.
9. In the same location in the Projects window, pcf → contacts → basics, right-click ABPersonVendorInputSet.ABPersonVendor, and choose Change PCF Mode.
10. In the Change Mode dialog box, add ABInterpreter to the list of modes, as follows:  
**ABPersonVendor|ABAttorney|ABDoctor|ABInterpreter**
11. In the same location in the Projects window, pcf → contacts → basics, right-click ContactBasicsDV.ABPerson and choose Change PCF Mode.
12. In the Change Mode dialog box, add ABInterpreter to the list of modes, as follows:  
**ABPerson|ABPersonVendor|ABAdjudicator|ABUserContact|ABDoctor|ABAttorney|ABPolicyPerson|ABInterpreter**
13. Click OK to save the new mode.

### Step D. Add the Subtype to the ContactManager Contact Picker Menus

1. In Studio, navigate to configuration → config → Page Configuration → pcf → contacts and double-click NewContactPickerMenuItemSet to open it in the editor.
2. Select the submenu labeled Vendor, which has the following id property:  
**NewContactPickerMenuItemSet\_PersonVendorMenuItem**
3. Copy the Doctor menu item and paste the copy under the existing Doctor menu item.
4. Select the new menu item.  
The menu item turns red, indicating an error. It stays red until you set the properties in the next step.
5. Set the following properties for this new menu item:

---

action	NewContactPopup.push(entity.ABInterpreter, parentContact)
id	NewContactPickerMenuItemSet_InterpreterMenuItem
label	displaykey.Web.NewContactMenu.Interpreter
showConfirmMessage	true
visible	requiredContactType.isAssignableFrom(entity.ABInterpreter)

---

## Step 8: Restart Both Applications and Test

1. If they are running, stop ClaimCenter and ContactManager, and then restart them to refresh each application with your PCF and data model changes.
2. Log in to ClaimCenter as a user who can create new contacts. For example, log in as user `ssmith` with password `gw`.
3. Open a claim and navigate to the **Parties Involved** → **Contacts** screen.
4. Choose **New Contact** → **Vendor** → **Interpreter** to see the **New Interpreter** dialog. Verify that **Interpreter Specialty** is listed in the **Additional Info** section.
5. Enter enough information to create an interpreter contact, such as name, address, interpreter specialty, and tax ID.
6. Add a role for the contact on the claim, and then click **Update** to save the new contact information.
7. Because you are logged in as a user with permission to create contacts, ClaimCenter sends the new contact to ContactManager. If you click the contact on the Contacts screen right away, you are likely to see the following message:  
**Waiting for link message from ContactManager. Refresh screen to get updated status**
8. You can refresh the screen by clicking another contact on the list and then clicking this one again. Eventually, you see the following message:  
**This contact is linked to the Address Book and is in sync**
9. Click the **Address Book** tab, and then, on the **Search Address Book** screen, pick **Interpreter** from the **Type** drop-down list and search for the new contact.
10. Select the new contact found by **Search** and verify that the entry is correct and that you can see the **Interpreter Specialty** field.
11. Return to the claim and click **Actions** → **New** → **Service**.
12. For **Request Type**, choose **Perform Service**.
13. For **Vendor Name**, click the contact picker and choose **New Vendor** → **Interpreter** from the drop-down list.  
The *contact picker* is a second, round drop-down button to the right of the field:  

14. Enter a name, an address, an interpreter specialty, and a tax ID, and then click **OK**.
15. On the **Create Service Requests** screen under **Services to Perform**, click **Add** to specify a service to perform.  
**Note:** There is no Interpreter service in the base configuration. Just pick any service on the list.
16. At the bottom of the screen, choose a **Service Address**, and then click **Submit**.
17. In the sidebar, click **Parties Involved**, and then on the **Contacts** screen, verify that your new contacts have been added. Click one of the contacts to open the **Basics** card and verify that **Interpreter Specialty** is listed under **Additional Info**.
18. Log in to ContactManager as a user who can create new contacts. For example, log in as user `aapplegate` with password `gw`.
19. In the sidebar, choose **Search**, pick **Interpreter** as the **Contact Type**, and search for one of the interpreter contacts you created in ClaimCenter.
20. Click the contact's name to see the details screen for the contact.
21. On the **Basics** tab, click **Edit** and change the value of **Interpreter Specialty**, and then click **Update** to save the change.

22. From the sidebar, choose **Actions** → **New Person** → **Vendor** → **Interpreter** to see the **New Interpreter** screen. Verify that **Interpreter Specialty** is listed in the **Additional Info** section.
23. Enter enough information to create an **Interpreter** contact. For **Tags**, click **Vendor**. Click **Update** to save the new contact.
24. Return to ClaimCenter.
25. Click the **Address Book** tab.
26. On the **Search Address Book** screen, pick **Interpreter** from the dropdown and search for the new contact you created in ContactManager.
27. Click the contact found by **Search** and verify that the entry is correct.
28. Search for the interpreter that you originally created in ClaimCenter and then edited in ContactManager.
29. Click that contact and verify that the changes you made in ContactManager are in this contact's data.

#### See also

- “Overview of Contact Entities” on page 133
- “Example of Adding a Field to a Contact Subtype” on page 142

## Extending Contacts with an Array

This topic describes how to extend the **ABContact** entity in ContactManager and the **Contact** entity in ClaimCenter with an array. The array is composed of states that comprise the service area for the contact. After creating and extending entities, you update the necessary files and screens to make the new array usable across the applications.

The following topics walk you through adding this functionality:

- “Step 1: Create New Entities in ContactManager and ClaimCenter” on page 158
- “Step 2: Add Extension Arrays to Parent Entities” on page 164
- “Step 4: Map Entities and Array Extensions in ClaimCenter” on page 166
- “Step 5: Extend the ClaimCenter User Interface” on page 167
- “Step 6: Extend the ContactManager User Interface” on page 170
- “Step 7: Test Your Changes” on page 171

### Step 1: Create New Entities in ContactManager and ClaimCenter

In this topic, you add a service state entity to ContactManager and to ClaimCenter. As with the contact subentity **Interpreter** added in “Example of Adding a Vendor Contact Subtype” on page 147, you must add the entity to both products.

These entities have the following features:

- Their names begin with a capital letter.
- Their 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 indexes for all searchable elements to make search work at a reasonable speed.

**Note:** An 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 ContactManager object must implement `ABLinkable`.

**Step A. Add the Service State Entity to ContactManager**

1. If necessary, start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin`, and enter the following command:

```
gwab studio
```

2. In the Project window, navigate to `configuration → config → Extensions → Entity`, and then right-click `Entity` and choose `New → Entity`.

3. Enter the following file name:

```
ContactServiceState
```

4. Enter the following for `Desc`:

`Represents a state where the contact provides services`

5. In addition to `Extendable` and `Final`, which are already selected, select the `Exportable` check box.

6. Click `OK`.

7. In the editor, click `entity` at the top of the `Element` hierarchy

8. Click the drop-down list for  and choose `implementsEntity`, and then choose `ABLinkable` from the drop-down list for the `name` value:

9. Click `entity` at the top of the `Element` hierarchy.

10. Click the drop-down list for  and choose `foreignKey`, and then enter the following values:

Name	Value
<code>name</code>	<code>Contact</code>
<code>fkentity</code>	<code>ABContact</code>
<code>nullok</code>	<code>false</code>
<code>columnName</code>	<code>ContactID</code>
<code>desc</code>	<code>Contact that this Service State row relates to</code>

11. Click `entity` at the top of the `Element` hierarchy.

12. Click the drop-down list for  and choose `typekey`, and then enter the following values:

Name	Value
<code>name</code>	<code>ServiceState</code>
<code>typelist</code>	<code>State</code>
<code>desc</code>	<code>State serviced by the contact</code>
<code>exportable</code>	<code>true</code>
<code>loadable</code>	<code>true</code>

13. Click `entity` at the top of the `Element` hierarchy.

14. Click the drop-down list next to  and choose `index`.

15. Enter the following values for the new index:

Name	Value
name	absrvstatealinkid
desc	Preserve uniqueness of LinkID
unique	true

16. Right-click **index** in the **Element** column, and choose **Add new → indexcol**.

This index column is the first of two to add to the index.

17. Enter the following values for the new **indexcol**:

Name	Value
name	LinkID
keyposition	1

18. Right-click **index** in the **Element** column, and choose **Add new → indexcol**.

19. Enter the following values for the new **indexcol**:

Name	Value
name	Retired
keyposition	2

20. Click **entity** at the top of the **Element** hierarchy.

21. Click the drop-down list next to  and choose **index**.

22. Enter the following values for the new index:

Name	Value
name	ind1
unique	true

23. Right-click the **ind1 index** in the **Element** column, and choose **Add new → indexcol**.

This index column is the first of three to add to the index.

24. Enter the following values for the new **indexcol**:

Name	Value
name	ServiceState
keyposition	1

25. Right-click the **ind1 index** in the **Element** column, and choose **Add new → indexcol**.

26. Enter the following values for the new indexcol:

Name	Value
name	Retired
keyposition	2

27. Right-click the ind1 index in the Element column, and choose **Add new → indexcol**.

28. Enter the following values for the new indexcol:

Name	Value
name	ContactID
keyposition	3

29. Click **entity** at the top of the Element hierarchy.

30. Click the drop-down list next to  and choose **index**.

31. Enter the following values for the new index:

Name	Value
name	ind2
unique	true

32. Right-click the ind2 index in the Element column, and choose **Add new → indexcol**.

This index column is the first of three to add to the index.

33. Enter the following values for the new indexcol:

Name	Value
name	ContactID
keyposition	1

34. Right-click the ind2 index in the Element column, and choose **Add new → indexcol**.

35. Enter the following values for the new indexcol:

Name	Value
name	Retired
keyposition	2

36. Right-click the ind2 index in the Element column, and choose **Add new → indexcol**.

37. Enter the following values for the new indexcol:

Name	Value
name	ServiceState
keyposition	3

### Step B. Add the Service State Entity 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 Project window, navigate to `configuration → config → Extensions → Entity`, and then right-click `Entity` and choose `New → Entity`.

3. Enter the following file name:

`ContactServiceState`

4. Enter the following for `Desc`:

`Represents a state where the contact provides services`

5. In addition to `Extendable` and `Final`, which are already selected, click `Exportable` to select its check box.

6. Click `OK`.

7. In the editor, click `entity` at the top of the `Element` hierarchy

8. Click the drop-down list for and choose `implementsEntity`, and then choose `Extractable` from the drop-down list for the `name` value.

9. Repeat steps 7 and 8 to create `implementsEntity` elements named `OverlapTable` and `AddressBookLinkable`.

10. Click `entity` at the top of the `Element` hierarchy.

11. Click the drop-down list for and choose `column`, and then enter the following values:

Name	Value
<code>name</code>	<code>AddressBookUID</code>
<code>type</code>	<code>varchar</code>
<code>nullok</code>	<code>true</code>
<code>desc</code>	<code>Represents ID of associated object in Address Book. Null if object not linked to Address Book.</code>
<code>loadable</code>	<code>true</code>

12. Right click the column you just added and choose `Add new → params`, and then enter the following values:

Name	Value
<code>name</code>	<code>size</code>
<code>value</code>	<code>30</code>

13. Click `entity` at the top of the `Element` hierarchy.

14. Click the drop-down list for and choose `foreignKey`, and then enter the following values:

Name	Value
<code>name</code>	<code>Contact</code>
<code>fkentity</code>	<code>Contact</code>
<code>nullok</code>	<code>false</code>
<code>columnName</code>	<code>ContactID</code>
<code>desc</code>	<code>Contact that this Service State row relates to</code>

15. Click entity at the top of the Element hierarchy.

16. Click the drop-down list for  and choose typekey, and then enter the following values:

Name	Value
name	ServiceState
nullok	false
typelist	State
desc	State serviced by the contact
exportable	true
loadable	true

17. Click entity at the top of the Element hierarchy, and then click the drop-down list next to  and choose index.

18. Enter the following values for the new index:

Name	Value
name	ind1
unique	true

19. Right-click the ind1 index in the Element column, and choose Add new → indexcol.

This index column is the first of three to add to the index.

20. Enter the following values for the new indexcol:

Name	Value
name	ServiceState
keyposition	1

21. Right-click the ind1 index in the Element column, and choose Add new → indexcol.

22. Enter the following values for the new indexcol:

Name	Value
name	Retired
keyposition	2

23. Right-click the ind1 index in the Element column, and choose Add new → indexcol.

24. Enter the following values for the new indexcol:

Name	Value
name	AddressBookUID
keyposition	3

25. Click entity at the top of the Element hierarchy.

26. Click the drop-down list next to  and choose index.

**27.** Enter the following values for the new index:

Name	Value
name	ind2
unique	true

**28.** Right-click the ind2 index in the Element column, and choose **Add new → indexcol**.

This index column is the first of three to add to the index.

**29.** Enter the following values for the new indexcol:

Name	Value
name	AddressBookUID
keyposition	1

**30.** Right-click the ind2 index in the Element column, and choose **Add new → indexcol**.

**31.** Enter the following values for the new indexcol:

Name	Value
name	Retired
keyposition	2

**32.** Right-click the ind2 index in the Element column, and choose **Add new → indexcol**.

**33.** Enter the following values for the new indexcol:

Name	Value
name	ServiceState
keyposition	3

## Step 2: Add Extension Arrays to Parent Entities

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

Key points to note with these extension arrays:

- Set the **triggersValidation** attribute to **true** for both entities. Setting this attribute enables validation rules to be triggered when an element of the array changes.
- Give the arrays the same name in both applications to simplify communications between ContactManager and ClaimCenter.

### Step A. Add the Array to ABContact in ContactManager

- In ContactManager Studio, in the Project window, navigate to **configuration → config → Extensions → Entity**, and then double-click **ABContact.etc** to open it in the editor.
- In the editor, click **entity (extension)** at the top of the Element hierarchy

3. Click the drop-down list for  and choose **array**, and then enter the following values:

Name	Value
name	ContactServiceArea
arrayentity	ContactServiceState
arrayfield	Contact
desc	Geographical area where the contact provides service
triggersValidation	true

4. If necessary, stop ContactManager.

Open a command prompt in the `ContactManager/bin` directory and then enter the following command:  
`gwab dev-stop`

5. Regenerate the data dictionary to ensure that the data model updates are correct.

At a command prompt, navigate to the `ContactManager/bin` directory and enter:  
`gwab regen-dictionary`

#### Step B. Add the Array to Contact in ClaimCenter

1. In ClaimCenter Studio, in the Project window, navigate to **configuration** → **config** → **Extensions** → **Entity**, and then double-click `Contact.etx` to open it in the editor.

2. In the editor, click **extension** at the top of the Element hierarchy

3. Click the drop-down list for  and choose **array**, and then enter the following values:

Name	Value
name	ContactServiceArea
arrayentity	ContactServiceState
arrayfield	Contact
desc	Geographical area where the contact provides service
triggersValidation	true

4. If necessary, stop ClaimCenter.

Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:  
`gwcc dev-stop`

5. Regenerate the data dictionary to ensure that the data model updates are correct.

At a command prompt, navigate to the `ClaimCenter/bin` directory and enter:  
`gwcc regen-dictionary`

#### Step 3: Map Entities and Array Extensions in ContactManager

As described in “Working with Contact Mapping Files” on page 140, you need to map the new entities between ClaimCenter and ContactManager. In this topic, you add the new array reference and the entity it references to the `ContactMapper` class in ContactManager.

For more information on this class, see “`ContactMapper` Class” on page 285.

1. In ContactManager Studio, press `Ctrl+N` and enter `ContactMapper`, and then double-click this class in the search results to open it in the editor.

- 2.** Find the `Mappings` method, which has the following declaration:

```
override property get Mappings() : Set<PropertyMapping>
```

In this method, you add a method for `ABContact` array reference `ContactServiceArea` and methods that map the elements of the array, `ContactServiceState`.

- 3.** At the end of the method, add a comma to the last line of code so you can add more code. For example:

```
fieldMapping(ABContactCategoryScore#Score),
```

- 4.** After this last line of code, add an `arrayMapping` method for `ABContact#ContactServiceArea`, preceded with a comment for this part of the `Mappings` method:

```
//ContactServiceState mapping  
arrayMapping(ABContact#ContactServiceArea),
```

- 5.** Add `fieldMapping` methods for the elements of the `ContactServiceState` entity, which the `ABContact` array reference refers to:

```
fieldMapping(ContactServiceState#LinkID)  
.withMappingDirection(TO_XML),  
fieldMapping(ContactServiceState#External_PublicID),  
fieldMapping(ContactServiceState#ServiceState)
```

#### Step 4: Map Entities and Array Extensions in ClaimCenter

As described in “Working with Contact Mapping Files” on page 140, you need to map the new entities between ClaimCenter and ContactManager. In this topic, you add the new array reference and the entity it references to the `ContactMapper` class in ClaimCenter.

For more information on this class, see “`ContactMapper Class`” on page 285.

- 1.** In ClaimCenter Studio, press `Ctrl+N` and enter `ContactMapper`, and then double-click this class in the search results to open it in the editor.

- 2.** Find the `Mappings` method, which has the following declaration:

```
override property get Mappings() : Set<PropertyMapping>
```

In this method, you add a method for `Contact` array reference `ContactServiceArea` and methods that map the elements of the array, `ContactServiceState`.

- 3.** At the end of the method, add a comma to the last line of code so you can add more code. For example:

```
.withEntityBeanBlock( \ l m, bp -> populateBeanFromXml(bp)),
```

- 4.** After this last line of code, add an `arrayMapping` method for `Contact#ContactServiceArea`, preceded with a comment for this part of the `Mappings` method:

```
//ContactServiceState mapping  
arrayMapping(Contact#ContactServiceArea),
```

- 5.** Add `fieldMapping` methods for the elements of the `ContactServiceState` entity, which the `ABContact` array reference refers to:

```
fieldMapping(ContactServiceState#AddressBookUID)  
.withABName(MappingConstants.LINK_ID),  
fieldMapping(ContactServiceState#PublicID)  
.withMappingDirection(TO_XML)  
.withABName(MappingConstants.EXTERNAL_PUBLIC_ID),  
fieldMapping(ContactServiceState#ServiceState)
```

This code maps between the following `ContactServiceState` fields in ClaimCenter and ContactManager:

ClaimCenter	ContactManager	Direction
AddressBookUID	LINK_ID	Both ways
PublicID	EXTERNAL_PUBLIC_ID	From ClaimCenter to ContactManager only
ServiceState	ServiceState	Both ways

**See also**

- “ContactManager Link IDs and Comparison to Other IDs” on page 271

**Step 5: Extend the ClaimCenter User Interface**

Using ClaimCenter Studio, make the following user interface changes.

**Step A. Enable ClaimCenter Address Book Search to Show States for a Company**

- In the Project window, navigate to **configuration** → **config** → **Localizations** → **en\_US** and double-click **display.properties** to open it in the editor.
- Press **Ctrl+N** and enter **Web.ContactDetail.RetiredMessage** to find that line in the file.
- Insert a new line below that one and add the following entries:  

```
Web.ContactDetail.ServiceStateName = State Name
Web.ContactDetail.ServiceStates = Service States
```
- In the Project window, navigate to **configuration** → **Page Configuration** → **pcf** → **addressbook**, and then right-click **addressbook** and click **New** → **PCF File**.
- Enter the **File name** **AddressBookServiceStates**.
- For **File type**, click **List View**, and then click **OK** to edit this PCF file.
- Click the new **ListViewPanel** to open its **Properties** window, and then click the **Required Variables** tab.
- Click  and add the following new required variable:

<b>name</b>	<b>contact</b>
<b>type</b>	<b>Contact</b>

- Click the **Expose** tab, and then click  and choose **Exposerterator**.

- Enter the following values:

<b>valueType</b>	<b>entity.ContactServiceState</b>
<b>widget</b>	<b>AddressBookServiceStatesLV</b>

- From the toolbox on the right, drag a **RowIterator** and drop it on the list view.

- Select the row iterator widget and set the following properties:

<b>editable</b>	<b>true</b>
<b>elementName</b>	<b>currentServiceState</b>
<b>toAdd</b>	<b>contact.addToContactServiceArea(currentServiceState)</b>
<b>toRemove</b>	<b>contact.removeFromContactServiceArea(currentServiceState)</b>
<b>value</b>	<b>contact.ContactServiceArea</b>
<b>canPick</b>	<b>true</b>

- From the toolbox on the right, drag a new **Row** and drop it on the **RowIterator**.

- 14.** From the toolbox on the right, drag a new Cell and drop it on the Row, and then set the following cell properties:

editable	true
id	ServiceState
label	displaykey.Web.ContactDetail.ServiceStateName
value	currentServiceState.ServiceState
unique	true

- 15.** If the outline of the ListViewPanel remains red, it is likely that the editor has lost the value type of the ExposerIterator AddressBookServiceStatesLV in the Exposes tab. To see if that is the case, click the ListViewPanel to open its Properties window. Then, as described in step 9, click the Exposes tab, and then click AddressBookServiceStatesLV and, if necessary, reenter the following valueType:  
entity.ContactServiceState
- 16.** In the Project window, navigate to configuration → config → Page Configuration → pcf → addressbook, and then double-click AddressBookContactBasicsDV.Company to edit this PCF file.
- 17.** On the right, in the InputColumn with the TextAreaInput called Notes at its bottom, drag a ListViewInput widget and drop it above Notes. Then enter the following attributes:

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

The new list view input remains red until you add the Toolbar widget in the next step.

- 18.** From the Toolbox on the right, drag a Toolbar and drop it in the new ListViewInput above the ListViewPanel.
- 19.** From the Toolbox on the right, drag an IteratorButtons widget and drop it on the toolbar so you can add and remove states.
- 20.** Click the new IteratorButtons widget and set the following property:

iterator	AddressBookServiceStatesLV.AddressBookServiceStatesLV
----------	---

#### Step B. Enable Addition of States to a Company Associated with a Claim

- In the Project window, navigate to configuration → config → Page Configuration → pcf → shared → contacts, and then right-click contacts and click New → PCF File. If you see a message asking if you want to add the PCF file to the folder, click Yes.
- Enter the File name VendorServiceStates and for File type choose List View, and then click OK to edit this PCF file.
- Click the new list view panel to open its Properties window, then click the Required Variables tab.
- Click  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. Click the **Exposes** panel, and then click and click **ExposeIterator**.

7. Enter the following values:

<b>valueType</b>	entity.ContactServiceState
<b>widget</b>	VendorServiceStatesLV

8. From the **Toolbox** on the right, drag a **RowIterator** and drop it on the new list view panel, and then set the following properties:

<b>editable</b>	true
<b>elementName</b>	currentServiceState
<b>toAdd</b>	contact.addToContactServiceArea(currentServiceState)
<b>toRemove</b>	contact.removeFromContactServiceArea(currentServiceState)
<b>value</b>	contact.ContactServiceArea
<b>canPick</b>	true

9. From the **Toolbox**, drag a new **Row** and drop it on the **RowIterator**.

10. From the **Toolbox**, drag a new **Cell**, drop it on the **Row**, and then set the following cell properties:

<b>editable</b>	true
<b>id</b>	ServiceState
<b>label</b>	displaykey.Web.ContactDetail.ServiceStateName
<b>value</b>	currentServiceState.ServiceState
<b>unique</b>	true

11. If the outline of the **ListViewPanel** remains red, it is likely that the editor has lost the **valueType** of the **ExposeIterator** in the **Exposes** tab, **VendorServiceStatesLV**. To see if that is the case, click the **ListViewPanel** to open its **Properties** window. Then, as described in step 6, click the **Exposes** tab, click **VendorServiceStatesLV**, and, if necessary, reenter the following **valueType**:

```
entity.ContactServiceState
```

12. In the **Project** window, navigate to **configuration** → **config** → **Page Configuration** → **pcf** → **shared** → **contacts**, and then double-click **ContactBasicsDV.Company** to edit this PCF file.

13. On the right, in the input column containing the **TextAreaInput** called **Notes**, drag a **ListViewInput** from the **Toolbox** and drop it above **Notes**.

14. Click the new **ListViewInput** widget and set the following properties:

<b>def</b>	VendorServiceStatesLV(contactHandle)
<b>label</b>	displaykey.Web.ContactDetail.ServiceStates
<b>labelAbove</b>	true
<b>boldLabel</b>	true

The new list view input remains red until you add the **Toolbar** widget in the next step.

15. From the **Toolbox** on the right, drag a **Toolbar** and drop it in the **ListViewInput**.

16. From the **Toolbox** on the right, drag an **IteratorButtons** widget and drop it on the toolbar so you can add and remove states.
17. Click the new **IteratorButtons** widget and set the following property:

iterator	VendorServiceStatesLV.VendorServiceStatesLV
----------	---

### Step 6: Extend the ContactManager User Interface

Using ContactManager Studio, make the following user interface changes.

1. In the **Project** window, navigate to **configuration** → **config** → **Localizations** → **en\_US** and double-click **display.properties** to open it in the editor.
2. Press **CTRL+F** and enter **Web.ContactDetail.RetiredMessage** to find this entry in the file.
3. Add a new line below that entry, and then enter the following display keys:  

```
Web.ContactDetail.ServiceStateName = State Name
Web.ContactDetail.ServiceStates = Service States
```
4. In the **Project** window, navigate to **configuration** → **config** → **Page Configuration** → **pcf** → **contacts** → **basics**, and then double-click **ContactBasicsDV.ABCompany** to edit this PCF file.
5. On the right, at the bottom of the input column containing the **TextAreaInput** called **Notes**, drag a **ListViewInput** from the **Toolbox** and drop it above **Notes**.
6. Click the new **ListViewInput** and set the following properties:

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

7. From the **Toolbox**, drag a **RowIterator** and drop it on the new list view panel, and then click it and set the following properties:

editable	true
elementName	currentServiceState
toAdd	contact.addToContactServiceArea(currentServiceState)
toRemove	contact.removeFromContactServiceArea(currentServiceState)
value	contact.ContactServiceArea
canPick	true

The new list view input remains red until you add the **Row** widget in the next step.

8. From the **Toolbox**, drag a **Row** and drop it on the **RowIterator**, and then drag a **Cell**, and drop it on the **Row**.
9. Click the cell and set the following cell properties:

editable	true
id	ServiceState
label	displaykey.Web.ContactDetail.ServiceStateName
value	currentServiceState.ServiceState
unique	true

10. When you dropped the RowIterator, the PCF editor created a ListViewPanel as a container for it. Click the ListViewPanel and set its id property to CurrentServiceStateLV.
11. From the Toolbox on the right, drag a Toolbar and drop it above the ListViewPanel named CurrentServiceStateLV.
12. From the Toolbox, drag an IteratorButtons widget and drop it on the toolbar so you can add and remove states.
13. Click the new IteratorButtons widget and set the following property:

Iterator	CurrentServiceStateLV
----------	-----------------------

## Step 7: Test Your Changes

### To test your changes:

1. Shut down ClaimCenter and ContactManager if they are running, and then restart both applications.
2. Log in to ClaimCenter as a user who can create new contacts. For example, if you have loaded sample data, log in as user ssmith with password gw.
3. Click the Claim tab and open an existing claim.
4. Click Parties Involved in the sidebar, and then on the Contacts screen click New Contact → Company.
5. On the New Company screen, Service States is listed above the Notes section on the right.
6. Click Add to add a row for a state, and then click the new State Name field and select a state from the list.
7. Select the new row, and then click Remove to delete it.
8. Add several states, and then enter enough information to create the new company.
9. Under Roles near the top of the screen, click Add, and then add a role for this company for the current claim, such as Vendor.
10. Click Update to add the company to the list on the Contacts screen.
11. Click the company in the list on the Contacts screen.  
On the Basics card, you see the following message:  
**This contact is linked to the Address Book and is in sync**  
Because you logged in as a user with permission to create and edit contacts, ClaimCenter saved the new contact in ContactManager.
12. Click the Address Book tab, and then search for the company you just created.
13. Click the company name in the search results to open its Basics card.
14. Look for the Service States list view above the Notes, and see that they are the states you added.
15. Click the Edit in ContactManager button, and, if necessary, log in as a user who can create new contacts. For example, log in as user aapplegate with password gw.
16. ContactManager opens showing the company you created in ClaimCenter.
17. Verify that there is a Service States list above Notes, and that the states are the same as the ones you added in ClaimCenter.
18. Click Edit and delete one of the states.
19. Add a different state, and then click Update.

20. Click **Actions** → **New Company** → **Vendor** and choose each kind of company in turn to verify that there is a **Service States** list for each type of company.
21. Choose a type of company to create, then add and delete a service state. Add several service states, and then enter enough information to create the new Company.
22. Click **Update** to save the company.
23. Go back to ClaimCenter and click the **Address Book** tab, and then search for the company you created in ContactManager to verify that ClaimCenter can find the company.
24. In the claim in which you created the company with states, click **Parties Involved**.
25. On the **Contacts** screen, click the company you created and verify that it is now out of sync.
26. Click **Copy from Address Book** and verify that the company now has the set of states that you set up in ContactManager.

## Changing the Subtype of a Contact Instance

### IMPORTANT

This feature must be used with caution by experienced database and configuration professionals who are familiar with the Contact data model. It applies only to ContactManager and ClaimCenter. To use this feature, you must log out of the application and set the application server runlevel to maintenance during the contact subtype change. The change of subtype directly affects the database. It causes ContactManager to be unable to synchronize the contact with ClaimCenter until you make the same subtype change in both ClaimCenter and ContactManager. Additionally, fields can be deleted and array fields can cause user interface exceptions. For example:

- A subtype change can delete fields that do not exist in the new subtype, as described at “Field Changes between Subtypes” on page 174. If feasible, determine which fields will change and where you want to move data in the new subtype. Then make any data updates you can in ContactManager before changing the subtype. You might also want to save the original contact data until you can review the contact after the subtype change.
- Related contacts and other arrays are not deleted when you change a subtype. However, they can cause problems in the user interface, as described at “Field Changes between Subtypes” on page 174. You can delete problematic data for the contact, such as related contacts that are not compatible with the new subtype, in ContactManager before making the subtype change. See “Subtype Changes and Incompatible Data” on page 174.

In ClaimCenter and ContactManager, you can change the subtype of a contact that was created with the wrong subtype without having to delete and re-create the contact. For example, you created a contact with subtype **Doctor** when you intended the contact to be an **Attorney**. If it is possible to delete the contact and re-create it with the correct subtype, do so. However, deleting and re-creating the contact might not be practical if the contact has history, such as being the payee on a check.

This feature is available through a new command-line utility available in ContactManager and in ClaimCenter:

```
maintenance_tools -user user-name  
                  -password user-password  
                  -changesubtypetargettype Subtype  
                  -changesubtypepublicid PublicID
```

**Note:** Do not copy this command and paste it directly into the command line. Beside adding your own parameters, you must get rid of the linefeed characters, inserted in this example for legibility.

This command takes the following parameters:

- *user-name* – User name of a user who has the role Contact Subtype Changer. This role includes the permissions required to run this command, **changecontactsubtype** and **soapadmin**. If you do not specify the **-user** parameter, the command defaults to Super User and requires that user’s password.

- *user-password* – Password for the user specified in the `-user` parameter, or for the Super User if there is no user specified.
- *Subtype* – New Contact or ABContact subtype for the contact, depending on whether you are running the command in ClaimCenter or ContactManager.
- *PublicID* – Value of the `PublicID` property of the contact. The `PublicID` value is not necessarily the same for a contact stored in ContactManager and the equivalent contact instances stored in ClaimCenter. Additionally, the `PublicID` value is not necessarily the same as that of `LinkID` in ContactManager or the `AddressBookUID` in ClaimCenter. See “ContactManager Link IDs and Comparison to Other IDs” on page 271.

This topic includes:

- “About Changing a Contact’s Subtype” on page 173
- “Error Messages from the Command-line Tool” on page 176

## About Changing a Contact’s Subtype

This topic describes information you need to know before changing the subtype of a contact instance. The contact instance can be stored in either ClaimCenter or in ContactManager, or both. In ClaimCenter, the contact would be on at least one claim’s **Parties Involved → Contacts** page. If the ClaimCenter contact is linked to ContactManager, you must make the subtype change separately in each application.

There can be multiple instances of the contact in ClaimCenter, each stored as a claim contact with a claim. If contacts are not linked to ContactManager, you can change any number of them, based on the reason for the subtype change.

If the contacts are linked with ContactManager, you need to change all the linked contact instances in ClaimCenter and the single instance in ContactManager. One way to find the ClaimCenter instances is to first get the value of the `LinkID` of the contact in ContactManager. Then use that value in a query that uses the equivalent ClaimCenter contact property, `AddressBookUID`, to find all the contact instances in ClaimCenter. The value that you need from each contact to make the subtype change is the `PublicID`. You then set the ClaimCenter runlevel to maintenance and run the command-line tool for each instance of the contact, specifying the value of its `PublicID`.

To change a contact subtype in ContactManager, set the runlevel to maintenance and then run the command-line tool once for the contact instance, using its `PublicID` to change its subtype. If the contact is linked to ClaimCenter, you set both applications’ runlevels to maintenance and perform the subtype changes at the same time.

### Subtype Changes for Client Contacts

If the contact has the Client tag, there are restrictions on which subtypes you can change it to. Contacts with the Client tag cannot have their subtypes changed across the three main contact subtypes, Person, Company, and Place. This restriction is in place because neither PolicyCenter nor BillingCenter can handle such changes. For example, if you try to make a subtype change in ContactManager for a client contact of type `ABDoctor` to the subtype `ABMedicalCareOrg`, the command fails with an error message. Because `ABDoctor` is an `ABPerson` subtype, the client contact’s subtype cannot be changed to `ABMedicalCareOrg`, which is an `ABCompany` subtype.

If your subtype change stays within a single subtype hierarchy, such as `ABPerson` in ContactManager and `Person` in ClaimCenter, you are allowed to make the change. The same is true for the `ABCompany` and `Company` subtype hierarchies and the `ABPlace` and `Place` subtype hierarchies in ContactManager and ClaimCenter. PolicyCenter and BillingCenter work only with `Company` and `Person` subtypes. Subtypes of `Person` all look the same to these applications, as do subtypes of `Company`. For example, a subtype change from a client contact of type `ABDoctor` to the subtype `ABAAttorney` is permitted, because PolicyCenter and BillingCenter work with the contact as a `Person` subtype.

For information on contact subtypes in the base configuration, see “Guidewire Core Application and ContactManager Contact Entity Hierarchy” on page 137.

## Field Changes between Subtypes

Changing the subtype of a contact can cause some predictable changes in contact data. This topic describes some common changes. Before making the subtype change, you can save the current data for the contact, and then add data to the new subtype if that subtype supports the data. After the subtype change, you can see which fields were changed in the Notes field for the contact.

### Name Field Changes when Changing between a Company or Place and a Person

A person subtype in the en\_US locale can have first name, last name, prefix, and suffix fields, while a company or place subtype has only a name field. If you change the subtype from a person subtype to a company or place subtype, the person's first name, last name, prefix, and suffix are combined into a single name. They become the name of the company or place subtype. If you make the opposite subtype change, the name of the company or place subtype becomes the last name of the person subtype, with no first name.

**Note:** The set of name fields for a Person subtype can vary by locale. However, for those locales, the conversion works the same way between the multiple name fields of a Person subtype and the single name field of a Company subtype.

For example, you change the subtype of the doctor with prefix Dr, first name Samantha, and last name Andrews to a medical care organization. The Name field of the contact that now has the MedicalCareOrg subtype becomes Dr Samantha Andrews.

### Subtype Changes and Incompatible Data

If there are fields on one subtype that are not on the new one, the fields are dropped or converted to similar fields as part of the subtype change. However, array fields are not dropped, and if they are not compatible with the new subtype, that can cause problems in the user interface.

For example, a contact can have related contacts, such as a company that has employees, or a person who has an employer. However, in the base configuration, a company cannot have an employer, and a person cannot have employees. If you change a person who has an employer to a company subtype, the employer field is preserved. However, the next time you edit the contact, you see an exception saying that a company cannot have an employer. If you go to the **Related Contacts** card and delete that relationship, you can then save the contact.

Additionally, if the person had the primary phone defined as a cell phone, that field is deleted during the transfer. In the base configuration, the primary contact number for a company contact cannot be a cell phone.

You can compare the fields for the subtypes in the Data Dictionary. If there are fields that will be dropped, you can record the data. After you change the subtype, you can add data that is appropriate for the new subtype. You can also consider deleting fields that will cause problems before making the contact subtype change.

#### See also

- “Relationships Between Contacts” on page 596 in the *Configuration Guide*

**“Adding a Bidirectional Contact Relationship: an Example” on page 597 in the *Configuration Guide***

## Changing the Subtype of a Contact

Be sure to read the important note at the beginning of the main topic and the preceding topics before starting this process. In particular, if the contact has the Client tag, you cannot change the subtype between Person, Company, or Place subtypes. For example, you cannot change the subtype of a Client contact from Doctor, a Person subtype, to MedicalCareOrg, a Company subtype. The tool will prevent you from making this change.

- For the Important note, see “Changing the Subtype of a Contact Instance” on page 172.
- For information you need to know before you start, see “About Changing a Contact’s Subtype” on page 173

If there is data in the contact that will be lost or will not be compatible with the new subtype, if possible, fix the data before you change the subtype. If the contact is stored in ClaimCenter and is linked to ContactManager, first

make the data changes to the contact in ContactManager. The changes are then propagated to ClaimCenter, which updates all linked instances of the contact. After you complete your data changes, you can proceed with the contact subtype change.

**Note:** It is not always possible to make all the data changes before the subtype change, so you might have to make some data changes afterwards.

You can change the subtype of a contact stored only in ClaimCenter, stored only in ContactManager, or stored in both and linked together. The server runlevel must be maintenance to make the change. If the contact is stored in both ClaimCenter and ContactManager, set both servers' runlevels to maintenance. Leave them at maintenance runlevel until you complete the subtype changes and any subsequent edit verifications in both applications.

#### To change the subtype of a contact in ContactManager

1. Run a query on the ContactManager database to get the PublicID and LinkID of the contact instance.
2. Ensure that all users are logged out.

**Note:** This step is an important one. ContactManager must not have its user interface open when you apply the subtype change.

3. Set the ContactManager server's runlevel to maintenance.

For example, at a command line open in *ContactManager/admin/bin*, enter:

```
system_tools -password superuser-password -maintenance
```

4. Run the command-line tool to change the subtype of the contact instance.

For example, your user with login alinu has the role Contact Subtype Changer and the password x2wTz@71P. To change the contact with PublicID ab:123 to the subtype ABDoctor, at a command line open in *ContactManager/admin/bin*, you enter the following command, all on one line:

```
maintenance_tools -user alinu -password x2wTz@71P -changesubtypetargettype ABDoctor  
-changesubtypepublicid ab:123
```

5. Note the messages in the console. If your command is successful, you see messages similar to the following:

```
Changing contact with publicId: ab:123 to subtype: ABDoctor
```

```
done
```

6. Log in to ContactManager and ensure that you can open the contact and save changes. If necessary, update data as needed.

7. Take ContactManager out of maintenance runlevel.

For example, at a command line open in *ContactManager/admin/bin*, enter:

```
system_tools -password superuser-password -multiuser
```

#### To change the subtype of a contact in ClaimCenter

1. Make a query for contact data. You might proceed differently depending on whether the contact is linked to ContactManager or is just local to ClaimCenter:

- If the contact is linked to ContactManager, run a query to find all instances of the contact with the AddressBookUID value equal to the LinkID value you got from ContactManager. Save the PublicID of each contact instance the query finds. Additionally, note each claim for which the contact is a claim contact.
- If the contact is local only to ClaimCenter, run a query in ClaimCenter to find the contact and save its PublicID. Note the claim for which the contact is a claim contact. You can run multiple queries if there are duplicate contact instances on other claims that you want to change.

2. Set the ClaimCenter server's runlevel to maintenance.

For example, at a command line open in *ClaimCenter/admin/bin*, enter:

```
system_tools -password superuser-password -maintenance
```

**3.** Run the command-line tool for each contact instance your query found.

For example, the user with login alinu has the role Contact Subtype Changer and password x2wTz@71P. To change the contact with PublicID ab:123 to the subtype Doctor, at a command line open in *ContactManager/admin/bin*, enter the following command, all on one line:

```
maintenance_tools -user alinu -password x2wTz@71P -changesubtypetargettype Doctor
-changesubtypepublicid ab:123
```

**4.** Note the messages in the console. For each command that completes successfully, you see messages similar to the following:

```
Changing contact with publicID: ab:123 to subtype: Doctor
done
```

**5.** Log in to ClaimCenter and ensure that you can open the contact in each claim and save changes. If necessary, update data as needed.

**6.** Take ClaimCenter out of maintenance runlevel.

For example, at a command line open in *ClaimCenter/admin/bin*, enter:

```
system_tools -password superuser-password -multiuser
```

## Error Messages from the Command-line Tool

When you run the command-line tool, you might see the following errors reported in the console.

Error Message	What caused it
Failed to change contact to subtype: Bad username or password	Any of the following entries could have caused this problem: <ul style="list-style-type: none"> <li>• You entered an invalid user name.</li> <li>• You entered the wrong password.</li> <li>• You entered the password without a user specified and you are not using the password of the Super User.</li> </ul>
Failed to change contact to subtype: Incorrect runlevel for subtype change. Required: MAINTENANCE, Actual: MULTIUSER	Unless you are user Super User, the default user for this command, you must use the -user parameter and specify a user that has the permissions <code>changecontactsubtype</code> and <code>soapadmin</code> . For example, the role Contact Subtype Changer has these two permissions, as does the Superuser role, which has all permissions.
Failed to change contact to subtype: Contact with PublicID <i>public-id-value</i> not found	You have not set the application runlevel to maintenance prior to running the command. Use the following command: <code>system_tools -password superuser-password -maintenance</code>
Failed to change contact to subtype: entity-name is not a valid entity type	You entered an incorrect value for the PublicID of the contact.
Failed to change contact to subtype: entity-name is not a valid subtype of Contact	The entity name you used is not valid. For example, you used ABDoctor in ClaimCenter, or Doctor in ContactManager.
Failed to change contact to subtype: Cannot change Person to Company/Place or Company to Person/Place for contacts with Client tags	The entity name you used for the new subtype is not a subtype of ABContact or Contact. For example, you tried to change the contact subtype from ABPerson to ABContact.
You see the list of command line options for <code>maintenance_tools</code> , but no error message	The contact has a Client tag, which makes the contact usable by PolicyCenter and BillingCenter. Those applications do not support changing between a Person contact subtype and a Company or Place subtype. See "Subtype Changes for Client Contacts" on page 173.
	Either you misspelled one of the command-line options or you did not enter one. Changing a contact subtype requires that you enter both options, <code>-changesubtypetargettype</code> and <code>-changesubtypepublicid</code> . See "Changing the Subtype of a Contact Instance" on page 172.

# Contact Tags

Contact tags enable you to classify contacts without having to add new subtypes.

This topic includes:

- “Contact Tag Overview” on page 177
- “Contact Tag-based Security” on page 178

## Contact Tag Overview

Contact tags enable you to classify contacts without having to add new subtypes. Every contact stored by ContactManager must have at least one tag. The three tags provided in the base configuration are:

- **Client** – A policy or account contact in PolicyCenter. For example, the holder of an account, the primary named insured on a policy, or a driver of a vehicle insured by a policy.
- **Vendor** – A ClaimCenter vendor providing services that help resolve claim losses. For example, a body shop, assessor, attorney, or physical therapy clinic.
- **Claim Party** – A party to a claim in ClaimCenter, such as the insured or a claimant.

These tags are used to ensure that a contact is either a vendor, and therefore of interest only to ClaimCenter, or a client, of interest to PolicyCenter. A client can also be of interest to ClaimCenter and BillingCenter. Additionally, the Claim Party tag indicates that a contact has been added as a party to a claim. You can add additional tags and change how the applications use them.

You can add new contact tag types to the base contact tags by using Guidewire Studio. In the Project window, navigate to **configuration** → **config** → **Metadata** → **Typelist** and right-click **ContactTagType.tti**. Then choose **New** → **Typelist Extension** and click **OK**. Studio creates the file **ContactTagType.ttx** in **configuration** → **config** → **Extensions** → **Typelist**.

## Applications and Contact Tags

The base configurations of PolicyCenter and BillingCenter save and retrieve only ContactManager contacts that have the Client tag. For example, a PolicyCenter user adds a client contact to a policy. PolicyCenter automati-

cally sets the tag to Client and saves the contact in ContactManager. Later a claims adjuster adds this client contact to a claim as a party to the claim and saves the contact. ClaimCenter applies the Claim Party tag to the contact before sending it to ContactManager. The contact now has both the Client tag and the Claim Party tag.

When you add a new contact or edit a contact in ContactManager, you see a drop-down list that enables you to choose one or more contact tags. Additionally, you can see the contact tag on the detail screen for every contact.

When you work with contacts in the base configurations of ClaimCenter, PolicyCenter, and BillingCenter, you do not see any tag information in the user interface. The applications work with contact tags in the background and add the appropriate tags before sending a contact to ContactManager. After the tags are added to the contact in ContactManager, those tags are synchronized as part of the contact data copied over from ContactManager to the local contact record.

**Note:** A contact saved only locally in a core application does not have to have a tag. However, in their base configurations, the core applications always set tags for contacts when they are created and when they are saved to the database. Core applications set tags for contacts that they store locally even if the contacts are not stored in ContactManager.

### ClaimCenter Contact Tag Generation

If a new contact created in ClaimCenter is a vendor subtype, ClaimCenter adds a Vendor tag when it saves the contact to ContactManager. If you add a contact to a claim, ClaimCenter ensures that it has the Claim Party tag before saving the contact in ContactManager.

The base configuration of ClaimCenter sets tags in the class `gw.api.contact.CCContactMinimalTagsImpl`. If you add new tags, you must also set them in this class.

### PolicyCenter Contact Tag Generation

PolicyCenter ensures that any contacts it sends to ContactManager have Client tags. PolicyCenter cannot find a contact in ContactManager unless the contact has a Client tag.

The base configuration of PolicyCenter sets tags in the class `gw.api.contact.PCContactLifecycle`. If you add new tags, you must also set them in this class.

### BillingCenter Contact Tag Generation

BillingCenter ensures that any contacts it sends to ContactManager have Client tags. BillingCenter cannot find a contact in ContactManager unless the contact has a Client tag.

The base configuration of BillingCenter sets tags in the Contact Preupdate rule **Add default tags**. If you add new tags and want to set them for contacts that BillingCenter sends to ContactManager, you must set them in this rule.

## Contact Tag-based Security

As with contact subtypes, there are permissions associated with tags and there are permission check expressions used to control access to screens.

### Contact Tag Permissions

The base application permissions, listed in the table that follows, are special permissions that enable a user to create, edit, delete, and view contacts that have any tag. You can add permissions for creating, editing, deleting, and viewing specific tags, just as you can for contact subtypes. For more information, see “Creating Client Tag Permissions in ContactManager” on page 117.

The tag permissions provided in the base configurations are:

Code	Enables a User to
anytagcreate	Create a new contact regardless of which contact tag it requires.
anytagedit	Edit a contact that has any contact tag.
anytagdelete	Delete a contact that has any contact tag.
anytagview	See a contact that has any contact tag.

For more information on contact tag permissions, see:

- “ContactManager Contact and Tag Permission Check Expressions” on page 113
- “ClaimCenter Contact Subtype and Tag Permissions” on page 123
- “BillingCenter Contact-Related Permissions” on page 121

You can associate permissions with tags in Studio. For information on adding new tag permissions, see:

- “Creating Client Tag Permissions in ContactManager” on page 117
- “Creating Claim Party and Vendor Tag Permissions in ClaimCenter” on page 129

## Contact Tag Permission Check Expressions

Guidewire applications use a set of permission check expressions to control access to screens and widgets related to contacts. In ContactManager, to perform an action, users need permission for both the contact's subtype and the contact's tags. For example, to edit a CompanyVendor contact, the user needs permission to edit contacts of that subtype. If the contact has the Vendor tag, the user also needs permission to edit contacts with the Vendor tag. If you have not extended the tag permissions, the base configuration `anytagedit` permission gives that user permission to edit vendor contacts.

- For information on ContactManager permission check expressions, see “ContactManager Contact and Tag Permission Check Expressions” on page 113.
- For information on ClaimCenter permission check expressions, see “ClaimCenter Contact and Tag Permission Check Expressions” on page 124.
- For information on BillingCenter permission check expressions, see “BillingCenter Contact-Related Permission Check Expressions” on page 121.



# Vendor Services

Vendor services in ContactManager support the Services feature in ClaimCenter. You can create the services that your vendors provide, like carpentry or auto towing, and connect them to the vendor contacts who provide them. ContactManager maintains the connection between each vendor and the vendor's services and enables ClaimCenter to search for services and get a list of vendors who provide them.

This topic includes:

- “Vendor Services Overview” on page 181
- “Working with Vendor Services in ContactManager” on page 182
- “Vendor Services Data Model” on page 184

**See also**

- “Services” on page 375 in the *Application Guide*

## Vendor Services Overview

Vendor services provide a way to classify vendor contacts, contacts that have the Vendor tag, without having to add new subtypes. For example, a construction company that is a **CompanyVendor** entity provides several construction services for property claims: carpentry, painting, and plumbing, as well as independent appraisal. A claim comes in to ClaimCenter that requires major plumbing work on a property. The claims adjuster creating the claim in the **New Claim** wizard in ClaimCenter can search in ContactManager for vendors who provide the plumbing service. The clerk can then choose a vendor from the list of contacts returned by ContactManager.

Only vendor contacts, contacts that have the Vendor tag, can have services. ContactManager maintains the connection between a vendor contact and the services that the contact provides. When ContactManager receives a search request for a service from ClaimCenter, ContactManager returns vendor contacts that match the service.

Both ContactManager and ClaimCenter support a services directory, a tree structure of **SpecialistService** entities, each of which can have child and parent **SpecialistService** entities. The purpose of this tree structure in ContactManager is to display the tree in contact editing screens so you can add and remove services for contacts. When you add services to a contact, ContactManager links them to that contact.

In the base configuration, the services directory is not populated. You must implement services in XML and import them through the administration interface to load your services into ContactManager. In the base configuration, there is a sample services directory in the file `vendorservicetree.xml`. You can edit this file and create your own services.

If you load sample data for testing purposes, the sample services directory also loads, and you can see the services directory in various places in the user interface.

---

**IMPORTANT** If you load sample data, the services that are loaded with it cannot be deleted from the database without dropping the entire database. Do not load sample services into an installation for which you cannot drop the database, such as a production system. While you cannot delete a service from the database, you can set a service's Active flag to `false`, which makes it inactive.

---

In ContactManager, if you have loaded sample data, you can see the services tree when you edit a contact that has the Vendor tag. An example is `ABCConstruction`, which has the Vendor tag and is a `VendorCompany` entity. If you search for that contact and then click that contact in the search results, on the **Services** card you can see all the services that contact provides. If you edit that contact and click the **Services** card, you see the full Services hierarchy, and you can add and remove services by clicking their check boxes.

## Working with Vendor Services in ContactManager

To be able to work with vendor services in ContactManager, you must first create a set of services in an XML file, such as those in the `vendorservicetree.xml` sample file. You then import the file into ContactManager. For information on this service tree data model and the XML syntax required, see “Vendor Services Directory Data Model” on page 184.

This topic focuses on the mechanics of importing and exporting the XML file, viewing the services for a contact, adding services to a contact, and removing services from a contact.

This topic includes:

- “Importing and Exporting Vendor Service Data” on page 182
- “Viewing, Adding, and Removing Vendor Services Data for Contacts” on page 183

### Importing and Exporting Vendor Service Data

You import and export vendor services data in ContactManager on the **Administration** tab. You must be logged in as an administrator with the permissions `soapadmin` and `viewadmin`. For example, you might import vendor service data to establish your service tree for the first time, to add more services, or to make an obsolete service inactive. Importing vendor services instantiates any new `SpecialistService` entities defined in the file and adds them to the tree. Any services you import become permanent and cannot be deleted from the database, although you can deactivate them.

---

**IMPORTANT** Guidewire recommends that you keep your vendor service tree definitions identical in ContactManager and ClaimCenter. As you make changes, import the same vendor service tree XML definitions into both ContactManager and ClaimCenter.

ClaimCenter has an additional set of vendor service details definitions that do not apply to ContactManager. Do not import the ClaimCenter `vendorservicedetails.xml` file into ContactManager. See “Importing Services” on page 495 in the *Configuration Guide*.

---

#### To import vendor service data

1. In ContactManager, navigate to **Administration** tab → **Utilities** → **Import Data**.

2. Click **Browse** to find the file in which you have defined the vendor services tree.  
For example, navigate to the sample `vendorservicetree.xml` file in `ContactManager/modules/configuration/config/sampleddata`.
3. Double-click the file to open it in ContactManager.  
ContactManager parses the file and verifies that it is valid and can be loaded.
4. If you have existing service tree definitions, you see a prompt asking what to do to resolve matches with existing records. In response to this prompt, choose:  
**Overwrite all existing records**
5. Click **Finish** to load the file.

#### To export vendor service data

1. In ContactManager, navigate to **Administration tab** → **Utilities** → **Export Data**.
2. Click the drop-down list for **Data to Export** and choose **Vendor Service Tree**.
3. Click **Export**.

ContactManager exports the data into the file `vendorservicetree.xml`, and your browser downloads the file. For example, if your browser is Google Chrome on Windows 7, exporting this file saves it by default in `C:\Users\yourlogin\Downloads`.

## Viewing, Adding, and Removing Vendor Services Data for Contacts

In ContactManager when you view a contact, if the contact is a vendor—has a Vendor tag—you see a **Services** card. If you click the **Services** card, you can see all the vendor services assigned to this contact.

When you edit a contact, if the contact is a vendor, there is a **Services** card in which you can add and remove contacts. If you click the **Services** card, you see the entire **Vendor Services Tree**. In this tree, you select check boxes to add vendor services, and you deselect checkboxes to remove services.

#### To see vendor services data for a contact

1. Log in to ContactManager as a user with the `anytagview`, `abview`, and `abviewsearch` permissions.  
If you have imported sample data, you can log in as a user that has the Contact Manager role, such as `aapplegate/gw`.
2. Click the **Contact** tab and click **Search**.
3. Search for a contact who is a vendor.  
For example, if you have imported sample data, you can click the **Contact Type** drop-down list and choose **Vendor (Company)**. Then enter `a` as the name and click **Search**.
4. Select a contact in the search results.  
For example, click the sample contact AB Construction.
5. When the contact details screen opens, click the **Services** card. On that card, all services defined for this contact are listed. Each service is shown with the category and subcategory to which it belongs, if any.  
When you click the **Service** card for the sample contact AB Construction, the services listed include carpentry, drying, flooring, and painting.

#### To add services to or remove services from a contact

1. Log in to ContactManager as a user who has the **ABContact** permissions `abcreate`, `abedit`, `abdelete`, `abview`, and `abviewsearch`, and the tag permissions `anytagview`, `anytagedit`, and `anytagcreate`.

If you have imported sample data, you can log in as a user that has the Contact Manager role, such as aapple-gate/gw.

**2.** Click the **Contact** tab and click **Search**.

**3.** Search for a contact who is a vendor.

For example, if you have imported sample data, you can click the **Contact Type** drop-down list and choose **Vendor (Company)**. Then enter a as the name and click **Search**.

**4.** Click a contact in the search results.

**5.** When the contact details screen opens, click **Edit**.

**6.** Click the **Services** card, which displays the entire vendor services tree. Each service you can select has a check box next to it.

**a.** Select a check box to add a service to the contact.

**b.** Deselect a check box to remove a service from the contact.

**7.** When you have finished selecting and deselecting all the services for this contact, click **Update**.

For example, click the sample contact AB Construction and then click **Edit**. On the **Services** card, you see that services like Carpentry, Drying, Flooring, and Painting are all selected. If you click a check box next to one of these services to deselect it, that service is removed when you click **Update**.

## Vendor Services Data Model

There are two aspects to the Services data model in ContactManager:

- The tree-structured Services directory, which is composed of **SpecialistService** entities, each of which can have child and parent **SpecialistService** entities.
- The data model for connecting the **ABContact** entity to **SpecialistService** entities.

This topic includes:

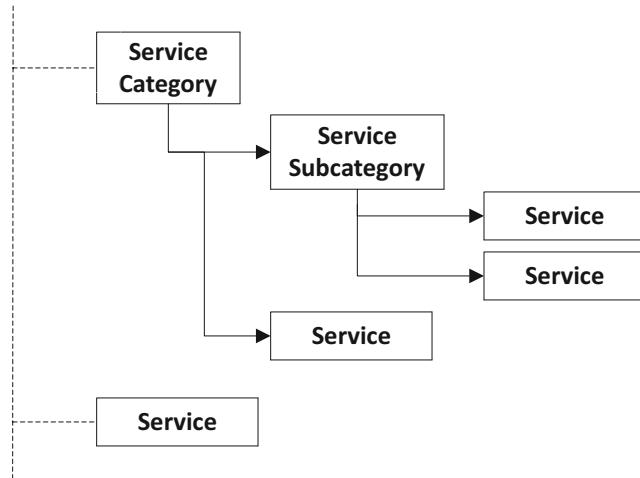
- “Vendor Services Directory Data Model” on page 184
- “ABContact and SpecialistService Entity Relationships” on page 190

### Vendor Services Directory Data Model

The data model for the vendor services directory consists of a hierarchy of parent services, or *categories*, and child services. Each node of the directory is an instance of a **SpecialistService** entity. This data model supports a tree-structured directory that is used in ContactManager screens in which you select services to add to or remove from a vendor contact. In those screens, the services you can add and remove are terminal nodes, also known as *leaf nodes*, that are not parents of other nodes.

Additionally, the vendor services directory enables ContactManager to show the category, subcategory, and service for each service that has been added to a contact.

In the base configuration, the Services Directory tree has three levels as shown in the following figure:



A leaf node, a *Service* in the figure, can appear at any level.

A node without a parent can be useful as a general service category, such as the Service Category node in the figure. It becomes a top level node in the tree.

A single service or category can have only one immediate parent. If you want a service or category to be listed in the tree under two different parents, you must create it twice and give each instance a separate parent.

You can create a tree with more than three levels, but you also have to configure PCF files to support the extra levels.

### Specialist Services Directory Example

For example, you might want to categorize some services at the top level as Auto and Property. The Auto category in this example, has one subcategory, Inspect/Repair, with two services, Audio Equipment and Auto Body. The Property category has two subcategories, Inspection and Construction Services. The Inspection subcategory has one child, the service Independent Appraisal. The Construction Services subcategory has two children, General Contractor and Carpentry.

The following table shows this set of categories and services. The Services column shows leaf nodes, services that can be assigned to a contact.

Category	Subcategory	Services
Auto	Inspect/Repair	Audio Equipment
		Auto Body
Property	Inspection	Independent appraisal
	Construction services	General contractor Carpentry

When you import the service tree XML file, ContactManager creates `SpecialistService` entities based on the definitions in the XML file and sets properties like parent, name, description, and public ID.

---

**IMPORTANT** You must import the same service tree XML file in both ContactManager and ClaimCenter so the service definitions match exactly. The base configuration provides a `vendorservicetree.xml` file with a set of services predefined. You can define your own services by using this file as an example, and then import it into ContactManager and ClaimCenter.

---

The XML for the previous example is as follows:

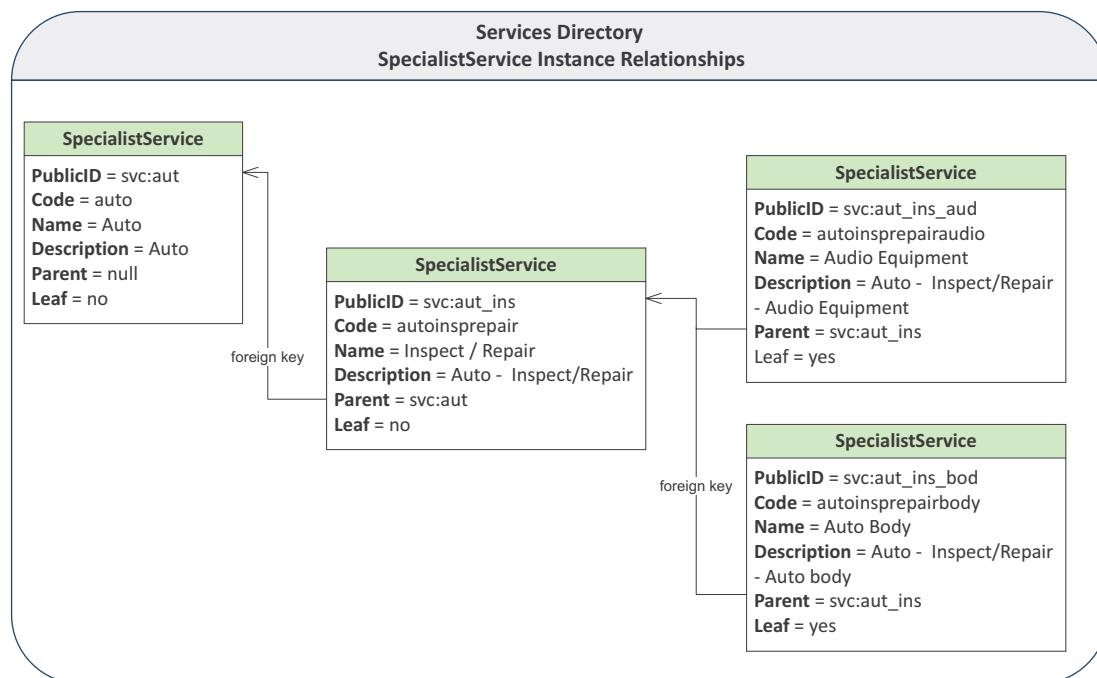
```
<?xml version="1.0"?>
<import>
<SpecialistService public-id="svc:aut">
  <Active>true</Active>
  <Code>auto</Code>
  <Description>Auto</Description>
  <Description_L10N_ARRAY>
    <Name>Auto</Name>
    <Name_L10N_ARRAY/>
    <Parent/>
  </SpecialistService>
<SpecialistService public-id="svc:aut_ins">
  <Active>true</Active>
  <Code>autoinsprepair</Code>
  <Description>Auto - Inspect / Repair</Description>
  <Description_L10N_ARRAY/>
  <Name>Inspect / Repair</Name>
  <Name_L10N_ARRAY/>
  <Parent public-id="svc:aut"/>
</SpecialistService>
<SpecialistService public-id="svc:aut_ins_aud">
  <Active>true</Active>
  <Code>autoinsprepairaudio</Code>
  <Description>Auto - Inspect / Repair - Audio equipment</Description>
  <Description_L10N_ARRAY/>
  <Name>Audio Equipment</Name>
  <Name_L10N_ARRAY/>
  <Parent public-id="svc:aut_ins"/>
</SpecialistService>
<SpecialistService public-id="svc:aut_ins_bod">
  <Active>true</Active>
  <Code>autoinsprepairbody</Code>
  <Description>Auto - Inspect / Repair - Auto body</Description>
  <Description_L10N_ARRAY/>
  <Name>Auto body</Name>
  <Name_L10N_ARRAY/>
  <Parent public-id="svc:aut_ins"/>
</SpecialistService>
<SpecialistService public-id="svc:pro">
  <Active>true</Active>
  <Code>prop</Code>
  <Description>Property</Description>
  <Description_L10N_ARRAY/>
  <Name>Property</Name>
  <Name_L10N_ARRAY/>
  <Parent/>
</SpecialistService>
<SpecialistService public-id="svc:pro_ins">
  <Active>true</Active>
  <Code>propinspect</Code>
  <Description>Property - Inspection</Description>
  <Description_L10N_ARRAY/>
  <Name>Inspection</Name>
  <Name_L10N_ARRAY/>
  <Parent public-id="svc:pro"/>
</SpecialistService>
<SpecialistService public-id="svc:pro_ins_ind">
  <Active>true</Active>
  <Code>propinspectindependent</Code>
  <Description>Property - Inspection - Independent appraisal</Description>
  <Description_L10N_ARRAY/>
  <Name>Independent appraisal</Name>
  <Name_L10N_ARRAY/>
  <Parent public-id="svc:pro_ins"/>
</SpecialistService>
<SpecialistService public-id="svc:pro_con">
```

```

<Active>true</Active>
<Code>propconstrserv</Code>
<Description>Property - Construction services</Description>
<Description_L10N_ARRAY/>
<Name>Construction services</Name>
<Name_L10N_ARRAY/>
<Parent public-id="svc:pro"/>
</SpecialistService>
<SpecialistService public-id="svc:pro_con_gen">
  <Active>true</Active>
  <Code>propconstrservgencontractor</Code>
  <Description>Property - Construction services - General contractor</Description>
  <Description_L10N_ARRAY/>
  <Name>General contractor</Name>
  <Name_L10N_ARRAY/>
  <Parent public-id="svc:pro_con"/>
</SpecialistService>
<SpecialistService public-id="svc:pro_con_car">
  <Active>true</Active>
  <Code>propconstrservcarpentry</Code>
  <Description>Property - Construction services - Carpentry</Description>
  <Description_L10N_ARRAY/>
  <Name>Carpentry</Name>
  <Name_L10N_ARRAY/>
  <Parent public-id="svc:pro_con"/>
</SpecialistService>
</import>

```

This XML creates a `SpecialistService` instance for each `<SpecialistService>` XML entry. Each entity definition has a public ID, a code, a name, a description, and, if the service has a parent, a pointer to the parent. The following figure shows the entity instances that would be created in ContactManager for the Auto portion of the XML in the previous example.



## SpecialistService XML Syntax

This topic defines the XML syntax for a `SpecialistService` entity. You can specify all the persistent `SpecialistService` fields in XML. There are also virtual properties, like `Leaf`, that are determined after `ContactManager` constructs the entire tree.

**Note:** You define vendor services in an XML file and import the file into `ContactManager` and `ClaimCenter`. For a sample file, you can open the `vendorservicetree.xml` file. See “Specialist Services Directory Example” on page 185.

The following sample code shows the beginning of the file and the definition of an Auto category, to help you identify the elements described in this topic.

```
<?xml version="1.0"?>
<import>
<SpecialistService public-id="svc:aut">
  <Active>true</Active>
  <Code>auto</Code>
  <Description>Auto</Description>
  <Description_L1ON_ARRAY/>
  <Name>Auto</Name>
  <Name_L1ON_ARRAY/>
  <Parent/>
</SpecialistService>
...
</import>
```

### <import>

Specifies that the enclosed XML uses the administrative data import XSD files `ab_import.xsd` and `ab_entities.xsd`.

When you export this file from `ContactManager`, the Administrative Export feature populates the following two attributes:

- `xmlns` – The namespace for this XML file. The default values is "<http://guidewire.com/ab/exim/import>".
- `version` – A version assigned to this XML file. For example, "p5.62.a10.225.26".

Both attributes are optional. It is not necessary to maintain these attributes in your files.

If you want to see the XSD files, you can generate them as follows:

1. At a command prompt open in the `ContactManager/bin` directory, enter the following command:  
`gwab regen-xsd`
2. `ContactManager` generates the XSD files in the following directory  
`ContactManager/build/xsd`

### <SpecialistService>

Specifies the name of the entity, `SpecialistService`.

This element takes the attribute `public-id`. In the `SpecialistService` entity, the value of the `public-id` attribute becomes the value of the `PublicID` field.

### Elements of <SpecialistService>

The following table describes the elements of the <SpecialistService> XML element.

XML Element	SpecialistService Field	Description
Active	Active	Indicates if the service is available to be used in new service requests. Setting this element to <code>false</code> is a way to make the service unavailable in the user interface if you no longer use it. You cannot delete a service from the database without dropping the database.
Code	Code	A unique string identifying the service. Used by ContactManager code to reference the service instance.
Description	Description	Text describing the service.
Description_L10N_ARRAY	Description	Use this element to provide the description in one or more specified languages. See “Specifying Localized Languages for a SpecialistService Instance” on page 189.
Name	Name	The display name of the service, used in the service tree in contact editing screens and as the name of a service for a contact in a contact’s Services card.
Name_L10N_ARRAY	Name	Use this element to provide the element name in one or more specified languages. See “Specifying Localized Languages for a SpecialistService Instance” on page 189.
Parent	Parent	The parent SpecialistService for this entity instance. If the XML tag is defined as empty, <Parent/>, the Parent field is null, and the service is a top-level category or service in the Services Directory.

### Specifying Localized Languages for a SpecialistService Instance

You can use <Language> elements with a SpecialistService entity if you require that the service have a different name or description depending on the user’s preferred language.

**Note:** You do not provide a language for the <Code> element because it is used only in code and is not visible to the ContactManager user.

The following XML definition for the Auto category provides Peruvian Spanish and German alternatives for the Name and Description fields.

**Note:** Before importing this file, you might have to open Guidewire Studio and add the typecodes `de_DE` and `es_PE` to the `LanguageType` typelist. See “Selecting Language and Regional Formats in ClaimCenter” on page 13 in the *Globalization Guide*. See also “About Display Languages” on page 21 in the *Globalization Guide*.

```
<SpecialistService public-id="svc:aut">
  <Active>true</Active>
  <Code>auto</Code>
  <Description>Auto</Description>
  <Description_L10N_ARRAY>
    <SpecialistService_Description_L10N public-id="svc:autd1">
      <Language>es_PE</Language>
      <Owner public-id="svc:aut"/>
      <Value>Vehiculo</Value>
    </SpecialistService_Description_L10N>
    <SpecialistService_Description_L10N public-id="svc:autd2">
      <Language>de_DE</Language>
      <Owner public-id="svc:aut"/>
      <Value>Kraftfahrt</Value>
    </SpecialistService_Description_L10N>
  </Description_L10N_ARRAY>
</SpecialistService>
```

```

</Description_L10N_ARRAY>
<Name>Auto</Name>
<Name_L10N_ARRAY>
  <SpecialistService_Name_L10N public-id="svc:autn1">
    <Language>es_PE</Language>
    <Owner public-id="svc:aut" />
    <Value>Vehiculo</Value>
  </SpecialistService_Name_L10N>
  <SpecialistService_Name_L10N public-id="svc:autn2">
    <Language>de_DE</Language>
    <Owner public-id="svc:aut" />
    <Value>Kraftfahrt</Value>
  </SpecialistService_Name_L10N>
</Name_L10N_ARRAY>
<Parent/>
</SpecialistService>

```

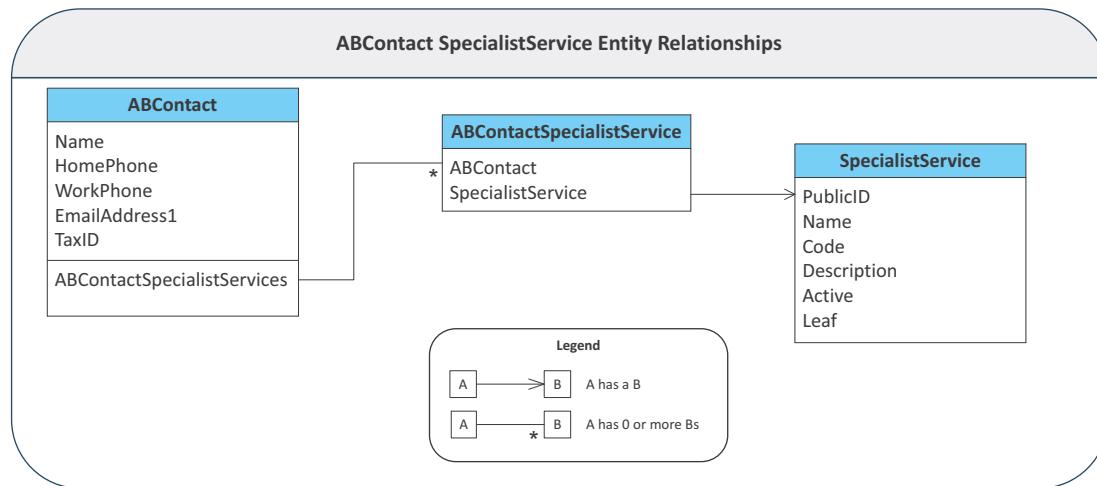
The public-id values must be unique for each instance of an entity type, including any existing entities of that type in the application. See “Public IDs and Integration Code” on page 27 in the *Integration Guide*.

Three types of entities defined in the XML code are `SpecialistService`, `SpecialistService_Description_L10N`, and `SpecialistService_Name_L10N`, each of which has instances with their own sets of unique `public-id` values.

Each L10N instance sets its `Owner` element to `svc:aut` to indicate that it is connected to the Auto `SpecialistService` instance.

## ABContact and SpecialistService Entity Relationships

ContactManager can link an array of specialist services to an ABContact. As described previously, a contact must have a Vendor tag for you to be able to add a service to it in the contact editing screen. After you add services, the contact references the services as an array, as shown in the following figure.



### See also

- “ABContact Data Model” on page 134

# Linking and Synchronizing Contacts

This topic covers setting up links for contacts and synchronizing contact information between Guidewire core applications and the Guidewire ContactManager contact management system. This topic applies only if you are managing contacts in an environment where ContactManager is integrated with your Guidewire core applications.

**Note:** This topic covers synchronizing contact data between a Guidewire core application and ContactManager, which is more complex and configurable in ClaimCenter than it is in PolicyCenter or BillingCenter. From the point of view of the core application, this type of synchronization is *external contact synchronization*. PolicyCenter also defines behavior for synchronizing contact data internally between accounts and policies. That type of synchronization is distinct from external contact synchronization.

This topic includes:

- “Linking a Contact” on page 191
- “Synchronizing with ContactManager” on page 196

## Linking a Contact

Before contact information can be synchronized between ContactManager and a Guidewire core application, it must be linked. Linking ensures that a contact stored locally in the core application is connected to a contact stored in ContactManager. ContactManager uniquely identifies an ABContact instance by its `LinkID`. The core applications use `AddressBookUID` for the same purpose. A contact is linked when the core application has verified the contact’s `AddressBookUID` with ContactManager.

This topic includes:

- “Creating and Linking a Contact” on page 192
- “Linking in ClaimCenter” on page 194
- “Linking in PolicyCenter” on page 194
- “Linking in BillingCenter” on page 195
- “Understanding Find Duplicates Behavior” on page 195

## Creating and Linking a Contact

The initial linking of a core application contact and a ContactManager contact typically happens when the core application creates the contact and sends it to ContactManager. In the base configuration, PolicyCenter specifies the unique ID for a new contact, while ClaimCenter and BillingCenter let ContactManager specify the unique ID for a new contact. In all three core applications, `AddressBookUID` holds the unique ID for the contact.

- In the base configuration, ClaimCenter and BillingCenter send the `createContact` request to ContactManager, and ContactManager creates a unique `LinkID` for the contact. ContactManager then returns the new contact data, including the `LinkID`, to the calling application. ContactManager also broadcasts the new contact data to the other applications. For other aspects of creating and linking a contact for ClaimCenter and BillingCenter, see:
  - “Linking in ClaimCenter” on page 194
  - “Linking in BillingCenter” on page 195
- In the base configuration, PolicyCenter generates a unique ID for the new contact and sends the `createContact` request to ContactManager. ContactManager populates `LinkID` for the new contact with the unique ID specified by PolicyCenter. See:
  - “PolicyCenter Contact Creation with External Unique IDs” on page 192
  - “Linking in PolicyCenter” on page 194

This topic includes:

- “PolicyCenter Contact Creation with External Unique IDs” on page 192
- “ContactManager Support for Handling External Unique IDs” on page 193
- “PolicyCenter Support for Creating External Unique IDs” on page 193

### PolicyCenter Contact Creation with External Unique IDs

PolicyCenter must be able to create a new contact and send it both to BillingCenter and ContactManager.

PolicyCenter must be able to uniquely identify a contact it sends to BillingCenter, and BillingCenter must be able to link to the contact created in ContactManager. Therefore, in the base configuration, PolicyCenter specifies a unique identifier for any contact it creates and sends to ContactManager.

If PolicyCenter needs to notify BillingCenter that the new contact has been created, PolicyCenter also sends a message to BillingCenter indicating the `AddressBookUID` of the new contact. The value of the `AddressBookUID` is the unique identifier that PolicyCenter created for the contact.

These messages are asynchronous and can arrive at BillingCenter and ContactManager at different times and in no particular order.

When ContactManager receives a contact create request for which a unique ID is specified, it uses that ID for the `LinkID` value of the new contact. ContactManager then sends create messages for update to BillingCenter and ClaimCenter if they are registered.

Depending on when BillingCenter receives the messages, it proceeds down different paths to link the contact:

- BillingCenter receives the message from PolicyCenter before ContactManager creates the new contact.
  - a. BillingCenter sends a request to ContactManager for contact data, specifying the `AddressBookUID`.
  - b. ContactManager returns `null`, indicating that the contact does not exist.
  - c. BillingCenter creates a local copy of the contact with the `AddressBookUID` it received from PolicyCenter.
  - d. ContactManager later notifies BillingCenter that the contact was created.
- BillingCenter verifies that the `LinkID` sent by ContactManager matches the `AddressBookUID` of the local contact that it created.

- f. BillingCenter updates its local copy of the contact with the data from ContactManager and links the contact. See “Linking in BillingCenter” on page 195.
- BillingCenter receives the message from ContactManager about the new contact before it receives the message from PolicyCenter.
  - a. BillingCenter ignores the message from ContactManager, because it does not have a matching contact.
  - b. BillingCenter receives a message from PolicyCenter saying that there is a new contact with a specified AddressBookUID.
  - c. BillingCenter sends a request to ContactManager for contact data, specifying the AddressBookUID.
  - d. ContactManager sends the data for the contact whose LinkID matches the AddressBookUID sent by BillingCenter.
  - e. BillingCenter updates its local copy of the contact with the data from ContactManager and links the contact. See “Linking in BillingCenter” on page 195.

#### See also

“PolicyCenter Support for Creating External Unique IDs” on page 193

### ContactManager Support for Handling External Unique IDs

ContactManager provides support for handling external unique IDs as follows:

- The `ContactMapper` class in ContactManager has field mapping definitions for the `External_UniqueID` fields in each entity that implements `ABLinkable`. These fields are listed in the topic “Mapping Externally Specified Unique IDs of a ContactManager Contact” on page 287.
- If a `createContact` request comes in that specifies an `External_UniqueID` field for the new contact, ContactManager uses the value of that field to populate the `LinkID` for the new contact. If that field is not in the data for the new contact or the field exists but its value is `null`, ContactManager creates its own `LinkID` for the new contact. See the description of the `createContact` method in “`ABContactAPI` Methods” on page 276.
- ContactManager provides a `CreatedApproved` rule in the `EventFired` ruleset for each core application. The rule is fired by the event `ABContactCreatedApproved`, which is added to an `ABContact` when it is created with an approved status by a call to `ABContactAPI`. The rule causes ContactManager to broadcast a create message as an update to each configured core application. This message enables any application that has been notified of a new contact that has an external ID provided to ContactManager to receive an update message when the create happens.

For example, PolicyCenter sends a new contact to BillingCenter with the `AddressBookUID` specified. When ContactManager creates the new contact, BillingCenter gets the create for update message from ContactManager. BillingCenter can then compare the `LinkID` in the message with the `AddressBookUID` of the contact that PolicyCenter sent to BillingCenter and update the contact with the data from ContactManager.

### PolicyCenter Support for Creating External Unique IDs

PolicyCenter provides support for creating external unique IDs with the following:

- The `Contact` entity in PolicyCenter is extended with the field `ExternalID`, which stores the value of the unique ID that PolicyCenter generates.
- The `ContactMapper` class in PolicyCenter has field mapping definitions for various `EXTERNAL_UNIQUE_ID` fields. Those definitions are described in “PolicyCenter Mapping of Externally Specified Unique IDs” on page 291.
- PolicyCenter has code that generates a new unique ID and assigns it to the `ExternalID` field of a new contact. PolicyCenter then calls the `ABContactAPI.createContact` method, passing it the data for that new contact. See the class `gw.api.contact.PCContactLifecycle`.

- PolicyCenter has an `EventFired` rule that responds to a `ContactAdded` event and sends a newly created contact to BillingCenter.

## Linking in ClaimCenter

ClaimCenter links a vendor contact automatically with ContactManager if the contact is created by a user that has the permissions `abcreate` or `abedit`. If the user does not have these permissions, in the base configuration, new vendor contacts are created, flagged as pending in ContactManager, and linked. These pending creates must be approved by a ContactManager user. After approval, ContactManager removes the pending flag.

### Note

This ClaimCenter behavior with vendor contacts is defined in the `Gosu` class `gw.plugin.contact.ab800.ContactSystemApprovalUtil`. You can edit this class and change how ClaimCenter determines the following:

- If a contact created in ClaimCenter will be created in ContactManager
- If a contact created or updated in ClaimCenter and sent ContactManager requires approval

In ClaimCenter, you can also manually link a local contact to ContactManager. For example, on the **Parties Involved** screen of a claim, you can click the name of a local contact and, below it in the **Basics** tab, click **Link**. After you click **Link**, ClaimCenter calls the ContactManager `findDuplicates` web service to see if there is a matching contact.

Depending on the results of the `findDuplicates` call, ContactManager and ClaimCenter behave as follows:

- If ContactManager finds an exact match, ClaimCenter links the local contact to the ContactManager contact by populating the local contact's `AddressBookUID` with the `LinkID` of the ContactManager contact.
- If ContactManager finds potential matches, ClaimCenter enables you to select one of the potential matches.
- If ContactManager does not find any matches, its behavior depends on the permissions of the ClaimCenter user and the type of contact:
  - If the ClaimCenter user has the `abcreate` permission, ClaimCenter sends a request to ContactManager to create a new contact.
  - If the ClaimCenter user does not have the `abcreate` permission and the contact is a vendor contact, ClaimCenter sends the new vendor to ContactManager with pending status. This new vendor must be approved by a user logged in to ContactManager.
  - If the contact is not a vendor contact, such as `Person`, ClaimCenter sends a request to ContactManager to create the new contact. The ClaimCenter user does not have to have `abcreate` permission.

**Note:** See the previous note on this vendor contact behavior and its definition in the class `ContactSystemApprovalUtil`.

For more information on this web service, see “Understanding Find Duplicates Behavior” on page 195.

After a successful link, the contact becomes subject to synchronizing rules, and the **Link** button turns into an **Unlink** button. If you click **Unlink**, the contact is no longer connected to ContactManager. It becomes a local-only contact.

### See also

- “Synchronizing ClaimCenter and ContactManager Contacts” on page 200.
- “Working with Contacts in ClaimCenter and ContactManager” on page 519 in the *Application Guide*

## Linking in PolicyCenter

The base PolicyCenter application links contacts with an integrated ContactManager under the following conditions:

- PolicyCenter always links contacts that are on accounts with at least one bound policy. At the time the user adds the contact, PolicyCenter checks to see if there are possible duplicate contacts. See “Understanding Find

Duplicates Behavior” on page 195.

- If the contact initially comes from ContactManager, PolicyCenter links the local contact when it creates the local contact. For example:
  - If you search for contacts and add one that is in ContactManager, that contact becomes linked regardless of the state of the policies on the account.
  - If you click to add a new contact, ContactManager can return a list of exact and potential duplicates. If you choose one of the duplicates as the contact that you want to use, that contact is linked.

In the base configuration, PolicyCenter does not send new client data to ContactManager while the client is still a prospect—when there is no bound policy on the account. PolicyCenter does store prospect information locally. If a prospect becomes a customer—at least one policy is bound for the account—PolicyCenter links the contact to the contact management system. You can configure this behavior differently.

The **Contact** entity and its subentities have an **AutoSync** property that controls synchronization with ContactManager. Setting this property to **Allow** enables the contact to be synchronized. After a contact is linked, PolicyCenter ensures that the **AutoSync** property is set to **Allow**. For more information, see “Configuring PolicyCenter External Contact Synchronization” on page 198.

## Linking in BillingCenter

The base BillingCenter application links contacts with an integrated ContactManager under the following conditions:

- BillingCenter links contacts that are on accounts and policy periods. At the time the user adds the contact, BillingCenter can check to see if there are possible duplicate contacts.
  - If you click **Add Existing Contact** for an account or policy period and search for a contact, BillingCenter shows a list of matching internal and external contacts.
- If you add an external contact, one that is stored only in ContactManager, BillingCenter creates a local instance of the contact and links it.
- If you add a contact that is not external, the contact has a local instance that BillingCenter might already have linked. If the contact instance is not linked, BillingCenter saves it as a new contact in ContactManager and links it.
  - If you click to add a new contact, BillingCenter checks to see if it exists in ContactManager. This check can return a list of exact and potential duplicates. If you choose one of the duplicates as the contact that you want to use, that contact is linked. If you create the contact without choosing from the list, BillingCenter sends it to ContactManager as a new contact, and at that point it gets linked.
- For producer contacts, BillingCenter creates a local instance of the contact. It does not link producer contacts with ContactManager.

BillingCenter find duplicates behavior is defined in the method

`gw.plugin.contact.ab800.ABContactSystemPlugin.findDuplicates`. For example, this method specifies that the only tag to be used to find duplicates is the client tag. If you want to use other tags for finding duplicates, you can modify this method to do so. See also “Understanding Find Duplicates Behavior” on page 195.

The **Contact** entity and its subentities have an **AutoSync** property that affects synchronization with ContactManager. Setting this property to **Allow** for a contact of an account or policy period enables the contact to be synchronized. BillingCenter sets the **AutoSync** property to **Allow** for all new contacts. For more information, see “Configuring BillingCenter External Contact Synchronization” on page 199.

## Understanding Find Duplicates Behavior

In their base configurations, the Guidewire core applications call the ContactManager web service `ABContactAPI.findDuplicates` to see if there are duplicates for a contact. The match types that ContactManager can return are *exact match* and *potential match*. For example:

- ClaimCenter calls this API when setting up a link for a contact. If there is an exact match, ContactManager establishes the link automatically. If the match is potential, ContactManager sends the potential matches to ClaimCenter, which displays them and enables you to choose one. Linking proceeds as described previously in “Linking in ClaimCenter” on page 194.
- Both PolicyCenter and ClaimCenter call this API when you click the **Check for Duplicates** button while adding a new contact. They also call this API when you save a new contact and you have not previously clicked **Check for Duplicates**.
  - If you click **Check for Duplicates**, ContactManager checks for exact and potential matches and sends the results to the core application, PolicyCenter or ClaimCenter. The core application shows the results and enables you to choose one.
  - If you click to save a new contact and have not clicked **Check for Duplicates**, the core application and ContactManager participate in the find duplicates exchange as described previously.
  - If you click to save a new contact and have already clicked **Check for Duplicates**, there is no additional find duplicates check. However, if there is an exact match that you have not chosen, the core application shows a warning that an exact match was available. You must either confirm that you do not want to choose the exact match, and therefore *do* want to create the new contact, or click to cancel the operation.
- BillingCenter calls this API when you add a new contact to an account or policy period, as follows:
  1. The first time you click to save a new contact, BillingCenter and ContactManager participate in a find duplicates exchange. ContactManager checks for exact and potential matches and sends the results to BillingCenter. BillingCenter shows the results and enables you to choose one.
  2. If you choose a contact from a list of exact or potential matches, BillingCenter links that contact and sets it up to be synchronized.
  3. You might save a new contact if there were no matches listed or if you do not see a good match on the list of contacts. In this case, there is no additional find duplicates check. However, if there is an exact or potential match on the list that you have not chosen, BillingCenter shows a warning that a matching contact was available. You must either click **Update** to continue and create the new contact or click **Check for Duplicates** to choose from the list.
- In ContactManager, to configure this matching behavior you can edit the `FindDuplicatesPlugin` plugin, which implements the interface `IFindDuplicatesPlugin`. Additionally, there is a set of Duplicate Finder classes that work with the plugin. You can edit these classes to add new subtypes and fields to the find duplicates process.

#### See also

- “[IFindDuplicatesPlugin Plugin Interface](#)” on page 298
- “[ABContactAPI Web Service](#)” on page 275

## Synchronizing with ContactManager

After a contact has been linked with ContactManager, the core applications and ContactManager work together to keep the contact data synchronized. This topic describes how contacts are synchronized, how the applications communicate changes to contact data, and how they ensure that this data is kept current across your Guidewire applications.

This topic includes:

- “[ContactManager Plugins for Broadcasting of Contact Changes](#)” on page 197
- “[Synchronizing PolicyCenter and ContactManager Contacts](#)” on page 197
- “[Synchronizing BillingCenter and ContactManager Contacts](#)” on page 198
- “[Synchronizing ClaimCenter and ContactManager Contacts](#)” on page 200

## ContactManager Plugins for Broadcasting of Contact Changes

ContactManager uses three plugins and an API to broadcast contact updates and changes to the Guidewire core applications. The registries for the plugins are ClaimSystemPlugin, PolicySystemPlugin, and BillingSystemPlugin. The base ContactManager application comes with plugin implementations that ContactManager uses to communicate with ClaimCenter, PolicyCenter, and BillingCenter. The plugin implementations are:

- `gw.plugin.claim.cc800.CCClaimSystemPlugin`
- `gw.plugin.policy.pc800.PCPolicySystemPlugin`
- `gw.plugin.billing.bc800.BCBillingSystemPlugin`

Depending on the Guidewire core applications you have installed, you configure the appropriate plugin registries. Then ContactManager can then send updates by calling each application's `ContactAPI` web service, which implements the interface `ABCClientAPI`. At that point, the application handles the update in its `ContactAPI` implementation of `ABCClientAPI`.

For more information on configuring this integration, see:

- “Integrating ContactManager with ClaimCenter in QuickStart” on page 46
- “Integrating ContactManager with PolicyCenter in QuickStart” on page 52

## Synchronizing PolicyCenter and ContactManager Contacts

This topic covers synchronizing contact data between PolicyCenter and ContactManager, and uses the term *external contact synchronization* to distinguish this type of synchronization from PolicyCenter internal synchronization. PolicyCenter also defines behavior for synchronizing contact data internally between accounts and policies. That type of synchronization is distinct from external contact synchronization.

PolicyCenter performs external contact synchronization with ContactManager for contacts that are shared across accounts, not for contacts at the policy level. PolicyCenter synchronizes contact data between policies and accounts internally.

In the PolicyCenter integration with ContactManager, linked client data is synchronized with ContactManager at all times. For PolicyCenter, ContactManager is the system of record for linked client data.

PolicyCenter uses the class `gw.plugin.contact.ab800.ABContactSystemPlugin` to create, retrieve, update, delete, and find duplicates of contacts in ContactManager. This class implements `gw.plugin.contact.ContactSystemPlugin` and calls the methods of the ContactManager web service `ABContactAPI`.

To enable ContactManager to send contact updates to PolicyCenter, PolicyCenter implements the ContactManager interface `ABCClientAPI` in the class `gw.webservice.pc.pc800.contact.ContactAPI`. ContactManager requires that all Guidewire core applications implement the `ABCClientAPI` interface. Implementing this interface ensures that ContactManager can broadcast contact changes to the core applications by using standard web services, and that the core applications have logic to handle these updates.

### Exceptions to Synchronization Updates

There are some cases in which PolicyCenter cannot perform a contact change or update:

- ContactManager notifies PolicyCenter that a contact has been deleted, but the contact is being used on at least one account in PolicyCenter.

This case is not likely to happen, because ContactManager checks to see if a contact can be deleted before it sends the request to delete the contact to PolicyCenter. However, if this case does happen, PolicyCenter unlinks the contact, does not delete the local instance, and creates an activity warning that an external system tried to delete the contact.

- ContactManager notifies PolicyCenter that an address has been deleted, but the address is being used as a policy address in PolicyCenter.

In this case, PolicyCenter unlinks the local instance of the address and does not delete it.

- PolicyCenter sends changes to a contact's data to ContactManager, but the contact's data was changed previously by another application or directly in ContactManager. The original data for the contact in PolicyCenter, prior to the change, must match the data currently in ContactManager for the change to be accepted. Since the original data that PolicyCenter sent does not match, ContactManager sends an exception to PolicyCenter and does not update the contact's data.

In this case, PolicyCenter creates an activity for a user to reapply the updates to the contact's data. The activity contains information about the contact and the data that could not be applied in ContactManager.

## Configuring PolicyCenter External Contact Synchronization

The PolicyCenter Contact entity has a typekey attribute called AutoSync that controls whether contacts are automatically synchronized with the external contact management system. Any PolicyCenter contact that is linked to ContactManager has the same value in its AddressBookUID property as the LinkID of the corresponding contact in ContactManager. Guidewire recommends that all contacts that are in both PolicyCenter and ContactManager be linked and have an AutoSync setting of Allow.

**Note:** In the default integration, PolicyCenter does not send a new contact to ContactManager until there is at least one bound policy on an account that the contact is associated with. When a contact is not linked to ContactManager, it has an AutoSync setting of null.

For any Contact instance, in addition to null, the AutoSync value can be one of three codes:

Allow	Allow the contact to be synchronized automatically. In a rule, this value would be TC_ALLOW.
Disallow	Do not allow the contact to be synchronized. In a rule, this value would be TC_DISALLOW. In its base configuration, PolicyCenter never sets a contact to Disallow.
Suspended	The contact was synchronized in the past, but synchronizing is not currently allowed. In a rule, this value would be TC_SUSPENDED. For example, this value could be set for all contacts during a ContactManager outage.

The previous code descriptions provide values you can set for AutoSync in a rule. For an example of a rule, open PolicyCenter Studio. Then navigate in the Projects window to configuration → config → Rule Sets → Preupdate → ContactPreupdate → Change Account Holder Name.

PolicyCenter uses the Java method `AccountContactImpl.markContactForAutoSync` internally to set the AutoSync value on a contact. This method is scriptable. The method sets AutoSync to Allow only if it is not already set to Disallow. Disallow is treated as a terminal state.

## Synchronizing PolicyCenter Contact Fields

When PolicyCenter synchronizes contact data with ContactManager, it updates the fields in the PolicyCenter contact with the centralized ContactManager data. The system overwrites all the fields defined in the `ContactMapper` class. This class also defines the fields that PolicyCenter sends to ContactManager. Use this class to control how contact data is synchronized between PolicyCenter and ContactManager.

For more information on modifying the `ContactMapper` class, see “Working with Contact Mapping Files” on page 140.

## Synchronizing BillingCenter and ContactManager Contacts

BillingCenter performs contact synchronization with ContactManager for contacts of accounts and policy periods, but not for contacts of producers. For BillingCenter, ContactManager is the system of record for linked client data.

BillingCenter uses the class `gw.plugin.contact.ab800.ABContactSystemPlugin` to create, retrieve, update, delete, search for, and find duplicates of contacts in ContactManager. This class implements

ContactSystemPlugin and IContactSystemPlugin and calls the methods of the ContactManager web service ABContactAPI.

To enable ContactManager to send contact updates to BillingCenter, BillingCenter implements the ContactManager interface ABClientAPI in the class gw.webservice.bc.bc801.contact.ContactAPI. ContactManager requires that all Guidewire core applications implement the ABClientAPI interface. Implementing this interface ensures that ContactManager can broadcast contact changes to the core applications by using standard web services, and that the core applications have logic to handle these updates.

### Exceptions to Synchronization Updates

There are some cases in which BillingCenter cannot perform a contact change or update:

- ContactManager notifies BillingCenter that a contact has been deleted, but the contact is being used on at least one account in BillingCenter.

This case is not likely to happen, because ContactManager checks to see if a contact can be deleted before it sends the request to delete the contact to BillingCenter. However, if this case does happen, BillingCenter unlinks the contact and does not delete the local instance.

- BillingCenter sends changes to a contact's data to ContactManager, but the contact's data was changed previously by another application or directly in ContactManager. The original data for the contact in BillingCenter, prior to the change, must match the data currently in ContactManager for the change to be accepted. Since the original data that BillingCenter sent does not match, ContactManager sends an exception to BillingCenter and does not update the contact's data.

In this case, BillingCenter creates an activity for a user to reapply the updates to the contact's data. The activity contains information about the contact and the data that could not be applied in ContactManager.

### Configuring BillingCenter External Contact Synchronization

The BillingCenter Contact entity has a typekey attribute called AutoSync that is important in determining if contacts are automatically synchronized with the external contact management system. Any BillingCenter contact that is linked to ContactManager has the same value in its AddressBookUID property as the LinkID of the corresponding contact in ContactManager. BillingCenter sets AutoSync for all new contacts to Allow.

**Note:** In the default integration, BillingCenter does not synchronize contacts of producers. BillingCenter adds a property to Contact entities, ShouldSendToContactSystem, that determines if the contact is synchronized. This boolean property is set in a get method in gw.api.address.ContactEnhancement, described later in this topic.

For any Contact instance, the AutoSync value can be one of three codes, in addition to null:

Allow	Allow the contact to be synchronized automatically. In a rule, this value would be TC_ALLOW.
Disallow	Do not allow the contact to be synchronized. In a rule, this value would be TC_DISALLOW. In its base configuration, BillingCenter never sets a contact to Disallow.
Suspended	The contact was synchronized in the past, but synchronizing is not currently allowed. In a rule, this value would be TC_SUSPENDED. For example, this value could be set for all contacts during a ContactManager outage.

The previous code descriptions provide values you can set for AutoSync in a rule. To see a rule that sets AutoSync, open BillingCenter Studio. Then, in the Project window, navigate to configuration → config → Rule Sets → Preupdate → ContactPreupdate → All Contacts sync by default.

BillingCenter does not link or synchronize contacts of producers. To control this aspect of synchronization, BillingCenter adds a property to Contact entities, ShouldSendToContactSystem, that determines if the contact is

synchronized. This boolean property is set in a get method in `gw.api.address.ContactEnhancement`. The code for the method is:

```
property get ShouldSendToContactSystem() : boolean {  
    return this.AutoSync == AutoSync.TC_ALLOW  
    and not this.ID.Temporary  
    and (isOnAccount() or isOnPolicyPeriod())  
}
```

For a contact to be sent to ContactManager, it must have an `AutoSync` value of `AutoSync.TC_ALLOW` and it must be connected to either an account or a policy period.

## Synchronizing BillingCenter Contact Fields

When BillingCenter synchronizes contact data with ContactManager, it updates the fields in the BillingCenter contact with the centralized ContactManager data. The system overwrites all the fields defined in the `ContactMapper` class. This class also defines the fields that BillingCenter sends to ContactManager. Use this class to control specifics of synchronizing contact data between BillingCenter and ContactManager.

For more information, see “Working with Contact Mapping Files” on page 140.

## Synchronizing ClaimCenter and ContactManager Contacts

To enable ContactManager to send contact updates to ClaimCenter, ClaimCenter implements the `ContactManager` interface `ABCClientAPI` in the class `gw.webservice.cc.cc800.contact.ContactAPI`. ContactManager requires that all Guidewire core applications implement the `ABCClientAPI` interface. Implementing this interface ensures that ContactManager can broadcast contact changes to the core applications by using standard web services, and that the core applications have logic to handle these updates.

ClaimCenter synchronizes contacts as follows:

- If ContactManager sends a contact change, ClaimCenter saves only the contact ID and then uses the contact automatic synchronization mechanism to update all local instances of the contact. For a description of this mechanism, see “Configuring Automatic Synchronization with ClaimCenter” on page 201.  
This behavior is different from the other Guidewire core applications. Those applications have just one local instance of a linked contact and apply all changes sent from ContactManager to their linked contacts. ClaimCenter, however, can have multiple local instances of any contact, one instance for each claim. ClaimCenter uses its automatic synchronization mechanism to ensure that all instances are synchronized.
- Like the other core applications, ClaimCenter automatically sends updates to ContactManager for linked contacts. In the default configuration, for a non-vendor contact, ClaimCenter calls ContactManager and specifies that the change is to be applied to the ContactManager contact. For a vendor contact, when the ClaimCenter user updates the contact changes, ClaimCenter determines if the user has contact permissions to perform the action. ClaimCenter then calls ContactManager appropriately:
  - If the user has permissions, ClaimCenter makes a call to ContactManager specifying that the vendor contact update is to be applied to the ContactManager contact.
  - If the user does not have permissions, ClaimCenter makes a call to ContactManager specifying that the vendor contact update is to be pending. A pending update has to be reviewed by a ContactManager contact administrator.

**Note:** For information on defining this vendor contact behavior in the class `ContactSystemApprovalUtil`, see “Linking in ClaimCenter” on page 194.

If the update succeeds, ClaimCenter then updates all local instances of the contact by using the automatic synchronization mechanism. A pending update succeeds if the update is approved in ContactManager and ContactManager notifies ClaimCenter.

- When ContactManager applies a contact change or create that originated from ClaimCenter, ContactManager sends the update to any other Guidewire applications that are integrated with ContactManager. It does not send the update back to ClaimCenter.

This topic includes:

- “Configuring Automatic Synchronization with ClaimCenter” on page 201
- “ClaimCenter Synchronizing Controls” on page 201
- “Rules that Affect Contact Synchronization” on page 201
- “Initiating a Manual Synchronization from ClaimCenter” on page 203
- “Synchronizing ClaimCenter Contact Fields” on page 203

## Configuring Automatic Synchronization with ClaimCenter

ClaimCenter contacts that are linked to ContactManager contacts can be synchronized automatically. The synchronizing mechanism ensures data consistency between ClaimCenter and ContactManager. This mechanism updates centrally managed contacts appropriately when a contact changes. You can configure automatic synchronization in ClaimCenter, as described in the next topic.

### ClaimCenter Synchronizing Controls

There are a number of places where you can configure synchronization of contacts in ClaimCenter.

#### AutoSync Attribute

The ClaimCenter Contact entity has a typekey attribute called AutoSync that controls whether contacts are automatically synchronized. By default, all contacts are synchronized, an AutoSync setting of Allow.

For any Contact instance, the AutoSync value can be one of three codes:

Allow	Allow the contact to be synchronized automatically. In a rule, this value would be TC_ALLOW.
Disallow	Do not allow the contact to be synchronized. In a rule, this value would be TC_DISALLOW.
Suspended	The contact was synchronized in the past, but synchronizing is not currently allowed. In a rule, this value would be TC_SUSPENDED. ClaimCenter typically sets AutoSync to this value when a claim is closed. If the claim is reopened, ClaimCenter sets the value of AutoSync to Allow.

#### Rules that Affect Contact Synchronization

In the default configurations of ClaimCenter and ContactManager, all contacts are set to allow automatic synchronization. However, for automatic synchronization to run, you must schedule the batch process. For more information, see “Running and Scheduling the Work Queue” on page 202.

You can determine which contacts you want your users to be able to synchronize. For example, you might not 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 ClaimCenter to set automatic synchronization values, determine how addresses update, and set the minimum required tags.

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

1. Start ClaimCenter Studio.

At a command prompt, navigate to the ClaimCenter bin directory and enter the following command:  
gwcc studio

2. In the Project window, navigate to configuration → config → Rule Sets → Preupdate → ContactPreupdate.

3. The predefined rule sets are:

- COP01000 - Update Check Address – Updates addresses that have changed for synchronized contacts.

- **COP02500 - Set all Vendors to auto sync** – Ensures that all contacts added to a claim are set to synchronize. For all contacts, AutoSync is set to Allow
- **COP03000 - Add default tags** – Ensures that the tags for a contact are the minimum required tags for ClaimCenter. The default tags are defined in the class `gw.api.contact.CCContactMinimalTagsImpl`. By default, all contacts get the Claim Party tag, and all vendor types, such as PersonVendor and CompanyVendor and their subtypes, also get the Vendor tag.

### Setting Up the Work Queue

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 Project window, navigate to `configuration → config → workqueue`, and then double-click `work-queue.xml` to open it in the 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.cc.domain.contact.CCContactAutoSyncWorkQueue"  
    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 Distributable Work Queues” on page 128 in the *System Administration Guide*.

### Running and Scheduling the Work Queue

You can have your work queues process changes on a schedule or in real time as they happen. The `InstantaneousContactAutoSync` parameter in the `config.xml` file controls this behavior. This parameter determines how ClaimCenter updates copies of a contact instance that has changed, either by a user action in ClaimCenter or by a change notification from ContactManager.

To open this file, start ClaimCenter Studio. Then, in the Project window, navigate to `configuration → config` and double-click `config.xml`. After changing this parameter, you must rebuild and redeploy ClaimCenter for the changes to take effect.

By default, the `InstantaneousContactAutoSync` attribute is set to `true`:

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

**IMPORTANT** With this parameter set to `true`, updating local instances of a contact can affect system performance if you have a large number of local instances. Additionally, if ClaimCenter receives a contact change notification from ContactManager, ClaimCenter updates all local instances of the contact, even instances open for edit. If a ClaimCenter user is editing a local instance of that contact, the user will be unable to save the changes made in the editing session. The user will have to exit the editing session and start over with the newly updated contact data received from ContactManager.

Setting `InstantaneousContactAutoSync` to `false` causes ClaimCenter to perform synchronized updates on the schedule set for the `ContactAutoSync` batch process in the `scheduler-config.xml` file. To open this file in the editor, start ClaimCenter Studio, and in the Projects window, navigate to `configuration → config → scheduler` and double-click `scheduler-config.xml`. You must rebuild and redeploy ClaimCenter for changes to take effect.

By default, `ContactAutoSync` is set to run at 5:00 am and 5:00 pm every day. It is on this schedule to ensure that all contacts are synchronized before the Financials Escalation process, `FinancialsEsc`, runs every day at 6:05 am and 6:05 pm. Synchronizing contacts before running the Financials Escalation process prevents the Financials Escalation process from having validation errors for contacts that are not synchronized. If you change the time `ContactAutoSync` times, also schedule the Financials Escalation process to run at an appropriate time afterwards.

The default `ContactAutoSync` setting looks like this:

```
<!-- Contact Auto Sync batch process should run every day at 5 am and 5 pm.  
It needs to run before the financials escalations to prevent those  
from triggering validation errors from out of sync contacts.-->  
<ProcessSchedule process="ContactAutoSync">  
  <CronSchedule hours="5,17"/>  
</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 `true` and you have `ContactAutoSync` scheduled, changes process immediately, and then the scheduled `ContactAutoSync` process also runs as scheduled.

For general information on administering and configuring scheduled batch processes, see “Scheduling Batch Processes Sequentially to Avoid Problems” on page 145 in the *System Administration Guide*.

### Initiating a Manual Synchronization from ClaimCenter

You can initiate a manual synchronization from ClaimCenter. Centrally managed contacts have a status message that informs you when they are not synchronized. For example, you have a claim open and on the `Parties Involved` → `Contacts` screen, the `Basics` card for the contact says:

**This contact is linked to the Address Book but is out of sync**

When you see this status message, you can click **Copy from Address Book** to synchronize the Address Book data for this one contact. Clicking this button copies the contact data from ContactManager to ClaimCenter.

### Synchronizing ClaimCenter Contact Fields

When ClaimCenter synchronizes contact data with ContactManager, it updates the fields in the ClaimCenter contact with the centralized ContactManager data.

The system overwrites the fields defined in the `ClaimCenter ContactMapper` class. This class also defines the fields that ClaimCenter sends to ContactManager. Use this class to control which fields are synchronized between ClaimCenter and ContactManager.

In addition, you can edit the `RelationshipSyncConfig` class in ClaimCenter Studio to specify the contact relationships to include or exclude in a synchronization.

### Setting Properties to Ignore in Determining a Contact’s Synchronization Status

In general, to determine which properties to consider in determining if a contact is synchronized, in most cases you use the `ContactMapper` class. With the exception of contact relationships, `ContactMapper` must list all contact fields to be transferred between ClaimCenter and ContactManager. If you omit a property from this class, it is not transferred or considered for synchronization.

By default, all properties listed in `ContactMapper` are considered in determining synchronization status.

If there is a property in `ContactMapper` that you want excluded from the synchronization check, use the `withAffectsSync` method and set its parameter to `false`. For example, the following code excludes the property `CategoryScores`:

```
arrayMapping(Contact#CategoryScores)  
  .withMappingDirection(TO_BEAN)  
  .withAffectsSync(false),
```

For more information, see “Excluding Contact Fields from ClaimCenter Contact Synchronization” on page 292.

## Setting Relationships to Be Included or Excluded

To include or exclude a contact relationship in the contact synchronization, use the `includeRelationship` and `excludeRelationship` methods of the `RelationshipSyncConfig` class.

**IMPORTANT** Typically, you include only relationships that appear on the **Contact Basics** panel of the **ClaimCenter Parties Involved** screen. 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 **ContactManager**.

To open the `RelationshipSyncConfig` file:

1. Start ClaimCenter Studio.

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

```
gwcc studio
```

2. Press `Ctrl+N` and enter `RelationshipSyncConfig`, and then double-click the class name in the search results.

The `RelationshipSyncConfig` class has two methods that support including or excluding relationships when synchronizing a Contact subtype:

- `includeRelationship` – Specifies relationships to include in synchronizing the specified Contact or Contact subtype.
- `excludeRelationship` – Specifies relationships to exclude in synchronizing the specified Contact or Contact subtype.

These two methods have the same parameters:

- `contactType` – The Contact subtype for which the syncable relationship will be added or excluded.
- `contactBidiRel` – The relationship `ContactBidiRel`, a valid typecode in the `ContactRel` typelist. The typecode must be specified as an enumerated typelist value, such as `TC_EMPLOYER` or `TC_PRIMARYCONTACT`.
- `appliesToSubtype` – If `true`, the relationship exclusion or inclusion applies to all subtypes of `contactType` unless overridden by an `includeRelationship` or `excludeRelationship` call. If `false`, the method applies only to the Contact subtype specified in `contactType`.

**Note:** You use a coding pattern called *method chaining* to add the method calls to the `init` method of the `RelationshipSyncConfig` class. With method chaining, the object returned by one method becomes the object from which you call the next method in the chain. See “Chaining Query Builder Methods” on page 174 in the *Gosu Reference Guide*.

The `init` method can have multiple `includeRelationship` or `excludeRelationship` method calls chained to it. Add the methods before the final `create` method in the chain.

For example, you might include a relationship for one contact type but exclude it for its child subtypes. The following method call, which is included in the `RelationshipSyncConfig` class in the base configuration, includes the `guardian` relationship for a `Person` entity, but not for its subtypes:

```
.includeRelationship(Person, TC_GUARDIAN, false)
```

The default behavior for an entity’s relationships is that the system ignores them. Only in certain cases do you need to use `excludeRelationship`, such as overriding a setting of `includeRelationship` in a parent type. For example, the default `Contact` definition includes the relationship `primarycontact` for all `Contact` entities and subentities. However, the `Person` definition overrides the `Contact` definition and excludes the `primarycontact` relationship only for `Person` entities, as shown in the following code snippet:

```
return new RelationshipSyncConfigBuilder<RelationshipSyncConfig>(
    new RelationshipSyncConfig())
    .includeRelationship(Contact, TC_PRIMARYCONTACT, true)
    .excludeRelationship(Person, TC_PRIMARYCONTACT, false)
    <!-- ... -->
    .create()
```

**See also**

- “Linking a Contact” on page 191
- “Synchronizing with ContactManager” on page 196



# ContactManager Rules

This topic provides an overview of the rules that Guidewire ContactManager uses to handle business processes.

**Note:** Before reading this topic, it would be useful to read the following topics, which introduce rules and the Studio Rules editor and describe how to write Gosu code for rules:

- “Rules: A Background” on page 13 in the *Rules Guide*
- “Rules Overview” on page 19 in the *Rules Guide*
- “Using the Rules Editor” on page 25 in the *Rules Guide*
- “Writing Rules: Testing and Debugging” on page 31 in the *Rules Guide*
- “Writing Rules: Examples” on page 33 in the *Rules Guide*

This topic includes:

- “ContactManager Rule Sets” on page 207
- “About Preupdate and Validation” on page 212
- “Overview of ContactManager Validation” on page 212
- “The Validation Graph in Guidewire ContactManager” on page 213
- “Top-level ContactManager Entities That Trigger Validation” on page 213

## ContactManager Rule Sets

This topic describes the sample rules that Guidewire provides as part of the base ContactManager application. You access these rule sets in ContactManager Studio by navigating in the Project window to **configuration** → **config** → **Rule Sets**.

ContactManager provides the following rule sets in the base product:

Rule set category	Description
EventMessage	Rules that handle communication with integrated external applications. See “EventMessage Rule Sets” on page 208. For information on events and event messaging, see “Messaging and Events” on page 299 in the <i>Integration Guide</i> .
Preupdate	Rules triggered any time that a contact or other high-level entity changes. See “Preupdate Rule Sets” on page 210.
Validation	Rules that check for missing information or invalid data on ABContact and Region entities. See “Validation Rule Sets” on page 212.

## EventMessage Rule Sets

In the base configuration, there is a single rule set, **EventFired**, in the EventMessage rule category. The rules in this rule set:

- Perform event processing.
- Generate messages about events that have occurred.

ContactManager calls the Event Fired rules if an entity involved in a bundle commit triggers an event for which a message destination has registered interest. As part of the event processing, ContactManager:

- Runs the rules in the Event Fired rule set once for every event for which a message destination has registered interest.
- Runs the Event Fired rule set once for each destination that is listening for that particular event. It is possible for ContactManager to run the Event Fired rule sets multiple times for each event, once for each destination interested in that event.

This topic includes:

- “Messaging Events” on page 208
- “Message Destinations and Message Events” on page 209
- “Message Destinations and Message Plugins” on page 209
- “Generating Messages” on page 209
- “Exercising Caution With Event Fired Rules and Messaging Plugin Implementations” on page 210

## Messaging Events

ContactManager automatically generates certain events, called *standard* events, for most top-level objects. In general, events are generated for any addition, modification, removal, or retirement of a top-level entity.

For example, ContactManager automatically generates the following events on the ABContact object:

- ABContactAdded
- ABContactChanged
- ABContactRemoved
- ABContactResync
- ABContactPendingChangeRejected

It is also possible to create a custom event on an entity by using the `addEvent` method. This method takes a single parameter, `eventName`, which is a `String` value that sets the name of the event.

```
entity.addEvent(eventName)
```

For more information, see:

- “ContactManager Messaging Events by Entity” on page 280
- “Messaging and Events” on page 299 in the *Integration Guide*
- “Triggering Custom Event Names” on page 325 in the *Integration Guide* for custom events

## Message Destinations and Message Events

You can use the **Messaging** editor to link an event to a message destinations. Open ContactManager Studio, and in the Project window, navigate to **configuration** → **config** → **Messaging**, and then double-click **messaging-config.xml**.

- A *message destination* is an external system to which it is possible to send messages.
- A *message event* is an abstract notification of a change in ContactManager that is of possible interest to an external system. For example, events can be adding, changing, or removing a Guidewire entity.

By using the **Messaging** editor, you can associate one or more events with a particular message destination.

For more information, see:

- “Using the Messaging Editor” on page 137 in the *Configuration Guide*
- “Message Destination Overview” on page 311 in the *Integration Guide*
- “Implementing Messaging Plugins” on page 343 in the *Integration Guide*
- “Messaging and Events” on page 299 in the *Integration Guide*

## Message Destinations and Message Plugins

Each message destination encapsulates all the necessary behavior for an external system, but uses three different plugin interfaces to implement the destination. Each of the following plugins handles different parts of what a destination does:

- The message request plugin handles message pre-processing.
- The message transport plugin handles message transport.
- The message reply plugin handles message replies.

You register new messaging plugins in Studio first in the Plugins editor. After you create a new implementation, Studio prompts you for a plugin interface name, and, in some cases, a plugin name. Use that plugin name in the Messaging editor in Studio to register each destination. Remember that you need to register your plugin in two different editors in Studio, first in the Plugins editor, and then in the Messaging editor.

---

**IMPORTANT** After the ContactManager application server starts, ContactManager initializes all message destinations. ContactManager saves a list of events for which each destination requested notifications. As this happens at system start-up, you must restart the ContactManager application if you change the list of events or destinations.

---

For more information, see:

- “Using the Messaging Editor” on page 137 in the *Configuration Guide*
- “Implementing Messaging Plugins” on page 343 in the *Integration Guide*

## Generating Messages

Use the method `createMessage` to create the message text, which can be either a simple text message or a more involved constructed message. The following code is an example of a simple text message that uses the Gosu in-line dynamic template functionality to construct the message. Gosu in-line dynamic templates combine static text with values from variables or other calculations that Gosu evaluates at run time. For example, the following `createMessage` method call creates a message that lists the event name and the entity that triggered this rule set.

```
messageContext.createMessage("${messageContext.EventName} - ${messageContext.Root}")
```

For more information, see:

- “Generating New Messages in Event Fired Rules” on page 329 in the *Integration Guide*

## Exercising Caution With Event Fired Rules and Messaging Plugin Implementations

Be extremely careful about modifying entity data in Event Fired rules and messaging plugin implementations. Use these rules to perform only the minimal data changes necessary for integration code. Entity changes in these code locations do not cause the application to run or re-run validation or preupdate rules. Therefore, do not change fields that might require those rules to run again. Only change fields that are not modifiable from the user interface. For example, you might set custom data model extension flags only used by messaging code.

**WARNING** Guidewire does not support the following:

- Guidewire does not support and does not recommend adding or deleting business entities from Event Fired rules or messaging plugins, even indirectly through other APIs.
- From within rule sets, you must never call any message acknowledgment or any skipping methods such as the Message methods `reportAck`, `reportError`, or `skip`. Use those methods only within messaging plugins. This prohibition also applies to Event Fired rule set execution.
- Guidewire does not support creating messages outside the Event Message rule set.

### See Also

- “Messaging Overview” on page 300 in the *Integration Guide*
- “Using the Messaging Editor” on page 137 in the *Configuration Guide*

## Preupdate Rule Sets

**Note:** See “About Preupdate and Validation” on page 212 for a discussion of how the validation and preupdate rules work together in performing validation.

Use the preupdate rule sets to perform domain logic or validation that must be committed before the entity in question is committed. Only a change to an entity marked as validatable in its definition file can trigger a preupdate rule.

Typically, these rules execute before the validation rule set. Preupdate rules can perform a wide variety of actions as needed. The primary use of the preupdate rules in the base configuration is to keep a history of changes to contacts and addresses.

Additionally, there is `PendingContactChangePreupdate` rule set with rules that handle rejection of either a pending contact creation or a pending contact update.

This topic includes:

- “Triggering Preupdate Rules” on page 210
- “Preupdate Rules and Custom Entities” on page 211
- “ABContactContact Preupdate” on page 211
- “ABContact Preupdate” on page 211
- “Address Preupdate” on page 211
- “PendingContactChange Preupdate” on page 212

### Triggering Preupdate Rules

The following entities trigger preupdate rules in the ContactManager base configuration:

- `ABContactContact`
- `ABContact`
- `Address`
- `PendingContactChange`

For an entity to trigger the preupdate rules, it must implement the `Validatable` delegate. All entities that implement the `Validatable` delegate trigger the validation rules as well.

ContactManager runs the preupdate and validation rules whenever:

- An instance of a `Validatable` entity is created, modified, removed, or retired.
- A subentity of a `Validatable` entity is created, modified, removed, or retired, and the `Validatable` entity is connected to the subentity through a `triggersValidation="true"` link.

---

**IMPORTANT** In running the preupdate rule set, ContactManager first computes the set of objects on which to run the preupdate rules. It then runs the rules on this set of objects. If a preupdate rule modifies an entity, the preupdate rules for that entity do not fire unless the schedule for preupdate rules already includes this entity. In other words, changing an object in a preupdate rule does not cause the preupdate rules to run on that object as well. Additionally, creating a new entity in a preupdate rule does not cause the preupdate rules to run on that entity.

---

## Preupdate Rules and Custom Entities

You can create preupdate rules for custom entities—entities that you create yourself and are not part of the base Guidewire ContactManager configuration.

### To create an extension entity that triggers preupdate rules

1. The entity must the `Validatable` delegate with an `implementsEntity` element that has its `Name` set to `Validatable`.
2. The preupdate and validation rule sets that you want the custom entity to trigger must conform to the following naming convention:
  - Put your preupdate rules in the `Preupdate` rule set category and name the rule set `entity-namePreupdate`.
  - Put your validation rules in the `Validation` rule set category and name the rule set `entity-nameValidationRules`.

You must create a rule set category with the same name as the extension entity and add the word `Preupdate` to it as a suffix. You then put your rules in this rule set.

For example, if you create an extension entity named `NewEntityExt`, you need to create a rule set to hold your preupdate rules and name it `NewEntityExtPreupdate`.

For information on creating new rule sets, see “Working with Rules” on page 25 in the *Rules Guide*.

## ABContactContact Preupdate

Use the ABContactContact Preupdate rules to modify `ABContactContact` and related entities. This rule set in the base configuration has rules that track the history on changes to a contact’s related contacts.

## ABContact Preupdate

Use the ABContact Preupdate rules to modify `ABContact` and related entities. This rule set in the base configuration has rules that:

- Set the `BatchGeocode` field for a vendor contact.
- Track history for creation of a contact, removal of a related contact, and changes to a contact’s name, phone numbers, addresses, and so on.

## Address Preupdate

Use the Address Preupdate rules to modify `Address` and related entities. This rule set in the base configuration has rules that track the history on changes to a contact’s primary and secondary addresses.

## PendingContactChange Preupdate

These rules handle rejection of either a pending contact creation or a pending contact update. Pending contacts are reviewed by a contact administrator and can be either accepted or rejected. Rejected changes trigger an event that sends a message to the core application, which notifies the user who made the change that the change was rejected. See “Reviewing Pending Changes to Contacts” on page 252.

## Validation Rule Sets

ContactManager uses validation rules to ensure that:

- A region zone is available in the zone lookup for the organization.
- The date of birth of each ABContact instance is in the past.
- The country code portion of all non-null phone numbers for each ABContact instance is correctly formatted.

## About Preupdate and Validation

ContactManager runs preupdate and validation rules every time it does a *database bundle commit*—commits data to the database. The validation rules execute after ContactManager runs all preupdate callbacks and preupdate rules. ContactManager runs the validation rules as the last step before writing data to the database.

Before applying rules during the bundle commit operation, ContactManager builds a validation object graph according to configuration settings. The graph contains possible targets for rule execution and is used by both the preupdate and the validation rules.

Preupdate rules differ from validation rules in that preupdate rules can modify additional objects during execution. Therefore, there can be additional objects that need to be constructed after the preupdate rules run and before the validation rules run. Following are some examples:

- The preupdate rules modify an entity that is not in the commit bundle.  
ContactManager can run additional validation rules if the entity being modified also triggers validation.
- The preupdate rules modify only entities that are in the commit bundle.  
ContactManager runs the validation rules for the same set of entities on which the preupdate rules ran.
- The preupdate rules modify an entity that simply triggers validation for one of the top-level entities.  
There is no noticeable difference from those validation rules because they were already going to run.

Therefore, the set of entities to be validated can be a superset of the entities for which preupdate rules are run.

**Note:** Preupdate rules are not recursive. They do not run a second time on objects modified during execution of the preupdate rules. ContactManager simply adds any objects modified by the preupdate rules to the list of objects needing validation, as described by the validation graph.

## Overview of ContactManager Validation

For an entity to have preupdate or validation rules associated with it, the entity must be validatable. To be validatable, the entity must implement the `Validatable` delegate with an `implementsEntity` element that has its `Name` set to `Validatable`. In the base configuration, Guidewire ContactManager comes preconfigured with the following high-level entities that can trigger validation:

- `ABContact`
- `ABContactContact`
- `Activity`
- `Address`
- `Group`

- PendingContactChange
- Region
- User

ContactManager validates these entities in no particular order.

#### See Also

- “<implementsEntity>” on page 200 in the *Configuration Guide*

## The Validation Graph in Guidewire ContactManager

During database commit, ContactManager performs validation on the following entities:

- Any validatable entity that is itself either updated or inserted.
- Any validatable entity that refers to an entity that has been updated, inserted, or removed.

ContactManager gathers all the entities that reference a changed entity into a virtual graph. This graph maps all paths from each type of entity to the top-level validatable entity, such as ABContact. These paths are queried in the database or in memory to determine which validatable entities, if any, refer to the entity that was inserted, updated, or removed.

ContactManager determines the validation graph by traversing the set of foreign keys and arrays that trigger validation. For example, suppose that the data model marks the Address array on ABContact as triggering validation. Therefore, any changes made to an address causes the Rules engine to validate the contact as well.

ContactManager follows foreign keys and arrays that trigger validation through any links and arrays on the referenced entities down the object tree. For example, you might end up with a path like ABContact → ABContactAddress → Address. To actually trigger validation, each link in the chain—Address and ABContactAddress—must be marked as triggering validation, and ABContact must be marked as validatable.

ContactManager stores this path in reverse order in the validation graph. Thus, if an address changes, ContactManager follows the path to find the contact that references a changed address. ContactManager traverses the tree from address to contact address, and finally to the contact—Address → ABContactAddress → ABContact.

## Top-level ContactManager Entities That Trigger Validation

To be validatable, an entity, subtype, delegate, or base object must implement the `Validatable` delegate with an `implementsEntity` element that has its `Name` set to `Validatable`. If an entity has a foreign key or array for which `triggersValidation` is `true`, the referenced entities must also be validatable.

The following table lists those entities that trigger validation in the base ContactManager configuration.

Validatable entity	Foreign key	Array	Comments
ABContact	PrimaryAddress	ContactAddresses SourceRelatedContacts TargetRelatedContacts	In the base configuration, ContactManager runs the preupdate and validation rules on the ABContact object during database commit if any objects with <code>triggersValidation="true"</code> have been modified.
User	Contact Credential UserSettings	Attributes	In the base configuration, ContactManager does not have any preupdate or validation rules for the User entity.



# ClaimCenter Service Provider Performance Reviews

ClaimCenter enables you to resolve losses in a claim by using service providers such as body shops, assessors, attorneys, and medical clinics. These service providers are vendor contacts. In addition, ClaimCenter enables you to evaluate your carrier's service providers by gathering review information on them. Having this information helps in selecting the best providers, controlling your claim costs, increasing customer satisfaction, and increasing claim processing efficiency.

## Important Notes

- This feature is available only if ClaimCenter is integrated with ContactManager, as described in “Integrating ContactManager with Guidewire Core Applications” on page 45.
- This feature aggregates performance scores from questionnaires that evaluate performance of vendors. It does not aggregate service request metrics, like cycle time. For information on service requests, see:
  - “Vendor Services” on page 181
  - “Services” on page 375 in the *Application Guide*

In particular, you can:

- Conduct post-service reviews on any type of vendor subtype of Contact.
- Score each review as part of the claim associated with the vendor's work.
- Score each vendor by combining its individual review scores.

After you have created reviews for your vendors, you can:

- Define lists of preferred vendors based on their past performance, as quantified by their reviews.
- Search for nearby vendors with high review scores.
- Assign nearby and high-rated vendors to provide services.
- Remove poorly performing vendors and steer business to high performers.
- Negotiate contracts with vendors for future services based on objective past performance standards.

This topic includes:

- “Service Provider Performance Reviews” on page 216
- “Using Service Provider Performance Reviews” on page 217
- “Data Model for Service Provider Performance Reviews” on page 222
- “Configuring Service Provider Performance Reviews” on page 224
- “Service Provider Review Plugin Reference” on page 235

## Service Provider Performance Reviews

The central element of the Service Provider Performance Reviews feature is the review. A review consists of sets of scorable questions in a questionnaire evaluating a vendor for work performed on a specific claim. You use the review's score to rank vendors and determine which vendors you will use.

This topic includes:

- “Service Providers” on page 216
- “Reviews of Service Providers” on page 216

### Service Providers

A service provider is a vendor, a contact unrelated to the carrier that provides a service during the resolution of a claim. For example

- Auto repair shop, auto glass shop, auto body shop
- Property assessor, auto damage assessor
- Attorney, law firm
- Medical clinic, home care worker, nurse, physician, physical therapist

**Note:** The sample data available in the base product includes a review for auto repair shops.

Typically, a carrier contacts the service provider and requests services, but the insured or other claim contact can also contact a service provider directly.

ClaimCenter considers all service providers to be vendor contacts and stores them in ContactManager. You can categorize your service providers. For example, you can create different types for auto glass repair, auto body repair, and transmission repair, and then create reviews containing sets of questions using the appropriate vendor types.

### Reviews of Service Providers

ClaimCenter scores each review and stores it with the associated claim. When a review is completed, ClaimCenter sends the results to ContactManager. ContactManager averages the review's overall and category scores with the scores of all reviews for the service provider and saves the result with the service provider's contact information.

Searching for service providers can return results from ContactManager that include these average scores. For example, you might specify Auto Repair Shop contacts and an accident site location in your search. ClaimCenter then provides a list of auto repair shops close to the accident site ranked by average review score.

**Note:** In the base configuration, ClaimCenter is not set up to send review scores to ContactManager, and ContactManager is not set up to process reviews. To configure this behavior, you enable two separate batch process to run, one in each application. See “Scheduling Reviews for Processing by ContactManager” on page 219.

**See also**

- “Using Service Provider Performance Reviews” on page 217
- “Data Model for Service Provider Performance Reviews” on page 222
- “Configuring Service Provider Performance Reviews” on page 224
- “Service Provider Review Plugin Reference” on page 235

## Using Service Provider Performance Reviews

You use the features of service provider performance reviews in ClaimCenter. From the ClaimCenter **Contacts** screen, you can work directly with reviews on claims. You can view, edit, complete, or delete them. You can use the ClaimCenter **Search** tab to search ContactManager for contacts with review scores above a minimum value that you set. You can also view the full **ReviewType** review form and see each contact’s score averages. The average scores include both review scores and scores by category for all review types associated with the contact.

This topic includes:

- “Working With Reviews” on page 217
- “Scheduling Reviews for Processing by ContactManager” on page 219
- “Manually Processing Reviews” on page 221

### Working With Reviews

The sample data includes a review for an auto repair shop contact. You can extend ClaimCenter and ContactManager to provide reviews for other types of service providers. For more information, see “Configuring Service Provider Performance Reviews” on page 224.

The ClaimCenter **Reviews** card is the starting point for working with reviews. You can access it by opening a claim and navigating to **Parties Involved** → **Contacts**.

This topic includes:

- “Beginning a Review” on page 217
- “Viewing or Editing a Review in ClaimCenter” on page 218
- “Viewing a Contact’s Review Scores in ContactManager” on page 218
- “Completing a Review” on page 218
- “Deleting a Review” on page 219
- “Searching for a Contact by Minimum Score” on page 219

#### Beginning a Review

You start a review of a service provider in ClaimCenter by selecting a review type for the review. On the **Contacts** screen, select the contact and then, on the **Reviews** tab, click **Edit** and then **Add New Review** to see all review types available for the contact. The review types are based on the service provider’s **Contact subtype**.

**In ClaimCenter, start a review from a claim, as follows:**

1. Navigate to a claim.
2. Select **Parties Involved** → **Contacts**.
3. Select the contact you want to review, and then click the **Reviews** tab and click **Edit**.
4. If there are review types defined for the contact’s type, you can click **Add New Review**.
5. Select a review type from the drop-down list.

After you have started a review, you can answer the review questions at any time. If you navigate away from the review, ClaimCenter saves the review with its current answers.

You can return to a review, and, if it is not complete, you can reopen the review and answer additional questions or edit your answers. For more information on reopening a review, see “Viewing or Editing a Review in ClaimCenter” on page 218.

## [Viewing or Editing a Review in ClaimCenter](#)

Use the following procedure to view or edit a review in ClaimCenter.

**Note:** After a review is complete, you can view it, but you cannot edit it.

1. Select a claim and click **Parties Involved** → **Contacts**.
2. Select an auto repair shop from the list and then click **Reviews** in the contact details screen below the list to show a list of reviews.

**Note:** The **Reviews** tab appears only if a review type is defined for that type of contact.
3. Select a review.
4. You can see the details of the review below the list of reviews.
5. If the review is not complete, you can click **Edit** to continue filling it out.

## [Viewing a Contact’s Review Scores in ContactManager](#)

1. On the **Contacts** tab, click **Search** in the sidebar, and then search for a contact that has been reviewed in ClaimCenter.

To see a review in ContactManager, at least one review has to have been completed in ClaimCenter and sent to ContactManager, and ContactManager must have processed the review. For more information, see “Scheduling Reviews for Processing by ContactManager” on page 219.

2. Select the contact and click the **Reviews** tab.
3. Select a review.
4. You can see information on the review below the list of reviews.

## [Completing a Review](#)

After you finish editing a review in ClaimCenter, you must *complete* it. Completing a review closes it for further editing and saves it in ContactManager, which updates the average scores for the contact.

### **To complete a review**

1. Display a list of reviews as described in see “Viewing or Editing a Review in ClaimCenter” on page 218.
2. Check the check box for each review you want to complete.
3. Click **Complete Selected** at the top of the list.

ClaimCenter calculates the review’s scores and sends the result to ContactManager, which uses them to calculate the vendor contact’s average scores. ClaimCenter does not immediately change the vendor’s scores to reflect the new scores.

## Deleting a Review

You can delete both complete and incomplete reviews in ClaimCenter.

**Note:** There are separate permissions for deleting incomplete reviews and completed reviews. You can use these permissions, for example, to allow adjusters to delete started, but not scored, reviews. For a list of permissions, see “Review Permissions” on page 234.

### To delete reviews

1. Display a list of reviews as described in see “Viewing or Editing a Review in ClaimCenter” on page 218.
2. Click the check box for each review you want to delete.
3. Click **Delete Selected**.

## Searching for a Contact by Minimum Score

You can search for contacts by score in either ClaimCenter or ContactManager. On the search screen, you select a **Minimum Score** value from a drop-down list with choices of 0, 10, 20, 30, and so on. You can configure the drop-down list by changing the PCF files.

The results are review values that are equal to or greater than the search value. Search results include a column of overall scores. You can sort the list by clicking the column’s title. Scores are not used in the default sort order.

### To Search for a Contact in ClaimCenter

1. Select **Address Book** tab → **Search**.
2. Select a **Contact Type**.
3. If there is a review type for the contact type, you can then select a **Minimum Score**. For example, if you have imported the sample data, selecting an Auto Repair Shop shows the **Minimum Score** list.
4. Enter the minimum data required to search for the contact, such as company name, tax ID, or postal code.
5. Click **Search**.

### To Search for a Contact in ContactManager

1. Select **Contacts** tab → **Search**.
2. Select a **Contact Type**.
3. If there is a review type for the contact type, you see the **Minimum Score** field. For example, if you have imported the sample data, selecting an Auto Repair Shop shows the **Minimum Score** drop-down list.
4. Enter additional data required to search for the contact, such as company name, tax ID, or postal code.
5. Click **Search**.

## Scheduling Reviews for Processing by ContactManager

By default, ClaimCenter does not send reviews to ContactManager, and ContactManager does not compute average scores and save them with the associated contact information. To get both sides of this feature working, you set up ClaimCenter and ContactManager to each run a batch process. As described in the topic that follows, you can optionally configure the work queues defined in the ClaimCenter and ContactManager `work-queue.xml` files.

You can also run the processes manually, as described in “Manually Processing Reviews” on page 221.

**Note:** The following topics talk about scheduling asynchronous batch processes in the `scheduler-config.xml` file. For more information on scheduling batch processes see “Defining a Schedule Specification” on page 144 in the *System Administration Guide*. That topic also describes in detail how to set the time and date attributes that determine when and how often a batch process runs.

This topic includes:

- “Configuring the Work Queues” on page 220
- “Scheduling the ClaimCenter Review Sync Batch Process” on page 220
- “Scheduling the ContactManager ABContact Scoring Batch Process” on page 221

## Configuring the Work Queues

Service provider reviews use the Guidewire work queue infrastructure and the process scheduler for asynchronous processing. The work queues are enabled by default in each application’s `work-queue.xml` file and do not necessarily require configuration. To run the workers asynchronously, you must schedule them as described in:

- “Scheduling the ClaimCenter Review Sync Batch Process” on page 220
- “Scheduling the ContactManager ABContact Scoring Batch Process” on page 221

### ClaimCenter uses the following writer, worker, and work queue configuration

- The ClaimCenter ReviewSync writer collects reviews that have been completed but not sent to ContactManager, and then creates work items for them. The ClaimCenter ReviewSync worker processes each review indicated by a work item and sends it to ContactManager.
- The ClaimCenter `work-queue.xml` file configures one `ReviewSyncWorkQueue`. You can open this file in ClaimCenter Studio by navigating in the Project window to `configuration → config → workqueue` and double-clicking `work-queue.xml`. The entry in this file for `ReviewSyncWorkQueue` is as follows:

```
<work-queue
    workQueueClass="com.guidewire.cc.domain.contact.ReviewSyncWorkQueue"
    progressInterval="600000">
    <worker instances="1"/>
</work-queue>
```

### ContactManager uses the following writer, worker, and work queue configuration

- The ContactManager ABContactScoring writer collects contacts that need to be updated and creates work items for them. The ContactManager ABContactScoring worker processes each contact indicated by a work item and updates the scores for each contact.
- The ContactManager `work-queue.xml` file configures one `ABContactScoringWorkQueue`. You can open this file in ContactManager Studio by navigating in the Project window to `configuration → config → workqueue` and double-clicking `work-queue.xml`. The entry in this file for `ABContactScoringWorkQueue` is as follows:

```
<work-queue
    progressInterval="600000"
    workQueueClass="com.guidewire.ab.domain.contact.ABContactScoringWorkQueue">
    <worker instances="1"/>
</work-queue>
```

For more information on work queues, see:

- “Distributable Work Queues” on page 124 in the *System Administration Guide*
- “Configuring Distributable Work Queues” on page 128 in the *System Administration Guide*

## Scheduling the ClaimCenter Review Sync Batch Process

To schedule the ClaimCenter batch process that sends reviews to ContactManager:

1. Start ClaimCenter Studio.

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

```
gwcc studio
```

2. In the Project window, navigate to `configuration → config → scheduler`.
3. Double-click `scheduler-config.xml` to open the file in an editor.
4. Find the following the `ReviewSync` entry and remove the comment markers:

```
<!--  
  <ProcessSchedule process="ReviewSync">  
    <CronSchedule minutes="20"/>  
  </ProcessSchedule>  
-->
```

This line causes the `ReviewSync` batch process to run every hour at 20 minutes after the hour.

## Scheduling the ContactManager ABContact Scoring Batch Process

To schedule the ContactManager batch process that processes reviews from ClaimCenter:

1. Start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:

```
gwab studio
```

2. In the Project window, navigate to `configuration → config → scheduler`.
3. Double-click `scheduler-config.xml` to open the file in an editor.
4. Find the following `ABContactScoring` entry and remove the comment markers:

```
<!-- <ProcessSchedule process="ABContactScoring">  
  <CronSchedule hours="0"/>  
</ProcessSchedule> -->
```

This entry causes the `ABContactScoring` batch process to run at midnight every day.

### See also

- “Defining a Schedule Specification” on page 144 in the *System Administration Guide*
- “Distributable Work Queues” on page 124 in the *System Administration Guide*
- “Configuring Distributable Work Queues” on page 128 in the *System Administration Guide*

## Manually Processing Reviews

As described in “Scheduling Reviews for Processing by ContactManager” on page 219, you can schedule automatic runs of the ClaimCenter and ContactManager processes that handle reviews. You can also run the processes manually, for example, to test a change you have made.

### To manually run the Completed Review Sync batch process in ClaimCenter

1. Log in to ClaimCenter as an administrator, such as username `su` and password `gw`.
2. Press `Alt+Shift+T` and then click the `Server Tools` tab.
3. Click `Batch Process Info` in the sidebar.
4. Locate the batch process `ClaimCenter (SPM) Completed Review Sync` and click `Run` in the column on the right.

### To manually run the ABContact Score Aggregator batch process in a ContactManager

1. Log in to ContactManager as an administrator, such as username `su` and password `gw`.
2. Press `Alt+Shift+T` and then click the `Server Tools` tab.
3. Click `Batch Process Info` in the sidebar.
4. Locate the batch process `AB Contact Score Aggregator` and click `Run` in the column on the right.

**See also**

- “Service Provider Performance Reviews” on page 216
- “Data Model for Service Provider Performance Reviews” on page 222
- “Configuring Service Provider Performance Reviews” on page 224
- “Service Provider Review Plugin Reference” on page 235

## Data Model for Service Provider Performance Reviews

All reviews consist of the following parts:

- **Header** – Contains information common to all `ReviewTypes`.
- **Question** – The review is a set of *questions*. The questions can be divided into groups called *categories*.
- **Answer** – After the review has been given, each question has an answer.
- **Score** – After the review is complete, it is scored. There is a score for the entire review and a score for each category if the review has categories.

Service providers of any contact type or subtype can have review scores, both overall and category scores, associated with them. Additionally, each claim can store a review’s answers and scores. In general, questions, sometimes with answer choices, are grouped into sets of questions, which are further grouped into categories, which are associated with a review type. Finally, review types are associated with defined contact subtypes.

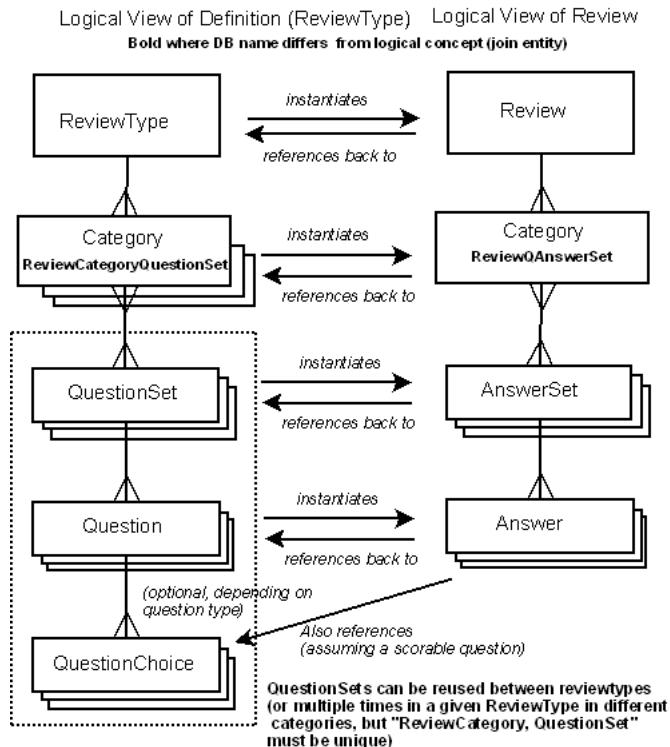
This topic includes:

- “Review Related Entities” on page 222
- “Typelists” on page 224
- “ContactManager Performance Review Data Model” on page 224

### Review Related Entities

Two entities, `Review` and `ReviewType`, form the basis for the ClaimCenter Service Provider Performance Reviews data model. On the left side of this diagram are abstract entities used to create objects shown on the

right side of the diagram. ReviewType and Review entities are related in the same way that Activity Patterns and Activities are related:



- **ReviewType** – An electronic questionnaire type, associated with a particular contact subtype and all its child subtypes, that you use to collect feedback about contacts. An example, included with each release, is an Auto Repair Evaluation.
- **ReviewCategoryQuestionSet** – A join table that relates a ReviewType to a QuestionSet and assigns a ReviewCategory to the QuestionSet. You can divide a questionnaire into one or more sections, or Categories. Categories of this form might include Quality of Work, Delivery, and General Satisfaction. Typecodes in the ReviewCategory typelist define Categories to enable ClaimCenter and ContactManager to calculate scores for them.
- **QuestionSet** – A group of questions.
- **Question** – A member of a QuestionSet.
- **QuestionChoice** – Optionally, potential answers to a particular question, such as true or false or a list of choices.

After someone creates a review, the system creates a parallel set of entities:

- **Review** – One entire questionnaire, containing one or more AnswerSets
- **ReviewAnswerSets** – Pointers to filled-in QuestionSets, each of which is an AnswerSet
- **AnswerSet** – A set of answers for a QuestionSet
- **Answer** – One reply to a QuestionSet

For more details on QuestionSets and their related entities, see “Question Sets” on page 269 in the *Application Guide*.

## TypeLists

Two typeLists, which must be present and identical in both ClaimCenter and ContactManager, affect Service Provider Performance Reviews:

- **ReviewCategory** – The set of categories to which you can assign groups of questions, and from which ClaimCenter generates category scores.
- **ReviewServiceType** – The set of possible service types to which the user can assign individual Reviews. Service types are a reporting category and appear in a drop-down list in the Review header.

## ContactManager Performance Review Data Model

Service provider performance reviews use the entity **ReviewSummary** and extend ContactManager's **ABContact** entity.

### ReviewSummary Entity

**ReviewSummary** is an entity that captures a summary of a **Review** object's information, passed from ClaimCenter. It contains the **Review** object's header information, including the overall score, and a list of category scores. However, it does not include the actual questions and answers.

Each **ReviewSummary** belongs to a particular **ABContact** instance and uses a claim number to associate it with the claim from which it was obtained.

### ABContact Entity

Service provider performance reviews extend the **ABContact** entity in ContactManager in two ways:

- **Score** – The overall score for this **ABContact**, as calculated from **ReviewSummary** entities by the **ABContactScoring** batch job
- **CategoryScores** – An array containing the category scores for this **ABContact** instance, similarly calculated from **ReviewSummary** entities by the **ABContactScoring** batch job

### See also

- “Service Provider Performance Reviews” on page 216
- “Using Service Provider Performance Reviews” on page 217
- “Configuring Service Provider Performance Reviews” on page 224
- “Service Provider Review Plugin Reference” on page 235

## Configuring Service Provider Performance Reviews

ClaimCenter provides one **ReviewType** in its sample data, which is available if you have imported sample data as described under “Installing Sample Data” on page 53 in the *Installation Guide*. To work with review types and their associated entities, you must log in as an administrator and export **Questions** data. The exported XML file includes sample entities for **ReviewCategoryQuestionSet**, **QuestionSet**, **Question**, **QuestionChoice**, **QuestionFilter**, and **ReviewType**. You can make your changes in the exported XML file and import the changed file.

In addition to working with reviews, you can also create or reuse categories and associate them with **ReviewTypes** and **QuestionSets**.

This topic includes:

- “Exporting and Importing Questions and Review Types” on page 225
- “Configuring Review Types” on page 225

- “Configuring Categories” on page 226
- “Configuring Review Service Type and Review Category Typecodes” on page 227
- “Configuring Question Sets, Questions, Question Choices, and Question Filters” on page 229
- “About Scoring” on page 233
- “Review Permissions” on page 234

## Exporting and Importing Questions and Review Types

Review types are grouped with questions in the administration **Import/Export Data** screen. To work with questions, categories and review types, you must export them as a unit and then import them.

### To export a review type and a set of questions and categories

1. Log in to ClaimCenter as an administrator. For example, log in with username **su** and password **gw**.
2. Click the **Administration** tab, and then click **Utilities → Export Data** in the sidebar.
3. At the top of the screen under **Import Administrative Data**, click **Export**.
4. Click **Data to Export** and choose **Questions** from the drop-down list, and then click **Export**.
5. If there is a prompt to save **questions.xml**, choose **Save**, and then choose a directory to save it to.  
You must import the file from this directory if you make changes to the file.
6. Open the saved file in an editor. The sample Auto Repair Service review type is at the bottom of the file, after the question and category definitions. The XML code for this element is:  

```
<ReviewType public-id="SPMReviewType:1">
```

### To import a review type and sets of questions and categories

1. Log in to ClaimCenter as an administrator. For example, log in with username **su** and password **gw**.
2. Click the **Administration** tab, and then click **Utilities → Import Data** in the sidebar.
3. On the **Import Administrative Data** screen, click **Browse** and navigate to the **questions.xml** file that you previously exported, and then click **Open**.
4. Click **Finish** to import the file.

## Configuring Review Types

After you export the **questions.xml** file as described in “Exporting and Importing Questions and Review Types” on page 225, you can add new review types to the file.

**Note:** If you want to change an existing review type, you must retire it and add a new review type.

### Adding a Review Type

1. Find a **ReviewType** in the **questions.xml** file and copy it. For example, the following review type is the sample Auto Repair Service review type definition:

```
<ReviewType public-id="SPMReviewType:1">
  <ContactSubtype>AutoRepairShop</ContactSubtype>
  <Description>Sample Review Type for Auto Repair Shops</Description>
  <Name>Auto Repair Service</Name>
  <ShouldRetireFromImportXML>false</ShouldRetireFromImportXML>
</ReviewType>
```

2. Give the **ReviewType** a public ID, a name, and a description, and specify the vendor contact subtypes to which it applies. For example, the following review type defines a review for an auto towing company:

```
<ReviewType public-id="SPMReviewType:2">
  <ContactSubtype>AutoTowingAgcy</ContactSubtype>
```

```

<Description>Review Type for Auto Towing Agencies</Description>
<Name>Auto Towing Agency</Name>
<ShouldRetireFromImportXML>false</ShouldRetireFromImportXML>
</ReviewType>

```

3. Open ClaimCenter Studio and ContactManager studio, and add new ReviewCategory or ReviewServiceType typecodes or both, as needed. For more information on adding these typecodes, see “Configuring Review Service Type and Review Category Typecodes” on page 227.
4. Use administrative file import to add the XML file containing the ReviewType to your installation. See “To import a review type and sets of questions and categories” on page 225.

For a list of vendor contact subtypes in the ClaimCenter base application, see “Guidewire Core Application and ContactManager Contact Entity Hierarchy” on page 137.

## Retiring a Review Type

If you no longer need a review type, or if you want to change an existing one, you must retire it. You cannot delete or change an existing review type because there might be existing reviews that use it.

To change a review type, you must retire it and add a new review type with a new public ID. After importing the XML file containing the retired review type, you can no longer use it to create new reviews in ClaimCenter. However, existing reviews that use the type continue to work.

---

**IMPORTANT** You can retire only review types. Do not retire QuestionSet, Question, QuestionChoice, or QuestionFilter entities because existing reviews might still be using them.

---

### To retire an existing review type:

1. Open the exported questions.xml file.
2. Find the review type you want to retire. For example, the following review type is the sample Auto Repair Service review type definition:

```

<ReviewType public-id="SPMReviewType:1">
  <ContactSubtype>AutoRepairShop</ContactSubtype>
  <Description>Sample Review Type for Auto Repair Shops</Description>
  <Name>Auto Repair Service</Name>
  <ShouldRetireFromImportXML>false</ShouldRetireFromImportXML>
</ReviewType>

```

3. Change the setting of <ShouldRetireFromImportXML> to true.

```

<ReviewType public-id="SPMReviewType:1">
  <ContactSubtype>AutoRepairShop</ContactSubtype>
  <Description>Sample Review Type for Auto Repair Shops</Description>
  <Name>Auto Repair Service</Name>
  <ShouldRetireFromImportXML>true</ShouldRetireFromImportXML>
</ReviewType>

```

## Configuring Categories

A ReviewCategoryQuestionSet entity connects a review type to a question set and applies a review category typecode to the question set. The typecode, defined in the ReviewCategory typelist, enables you to associate sets of questions with a category. ClaimCenter can then produce category scores for these sets of questions.

You can apply the same ReviewCategory typecode to more than one set of questions by defining multiple ReviewCategoryQuestionSet entities. You cannot apply more than one ReviewCategory typecode to any single question set, however. If you want to use the same question set in another review, you must duplicate the question set and rename it. Then create a ReviewCategoryQuestionSet that specifies the review type, the review category, and the new question set.

For more information on defining ReviewCategory typecodes, see “Configuring Review Service Type and Review Category Typecodes” on page 227.

A `ReviewCategoryQuestionSet` entity has the following elements:

- `ReviewType` – The review type that uses this review category and question set. A `ReviewCategoryQuestionSet` can reference one review type. If you want to use a `ReviewCategory` with more than one `ReviewType`, define separate `ReviewCategoryQuestionSet` entities.
- `QuestionSet` – The question set to which the review category applies. A `ReviewCategoryQuestionSet` can reference one question set.
- `ReviewCategory` – A typecode defined in the `ReviewCategory` typelist. Assigning a review category to a question set enables the question set to be scored and the category score to be averaged.
- `ElementOrder` – An integer that determines the order in which the categories appear in the `ReviewType`. Numbering starts at 0. You do not have to number categories sequentially, but doing so makes them easier to read and maintain.

The following code sample shows the XML definition of a `ReviewCategoryQuestionSet` entity used in the sample review type definition. This entity applies the `ReviewCategory` typecode `timeliness` to the question set `QuestionSetSPM:1` and adds it to the review type `SPMReviewType:1`. Additionally, the category has an element order of 1, making it the first category in the list.

```
<ReviewCategoryQuestionSet public-id="ReviewCatQSet:1">
  <ElementOrder>1</ElementOrder>
  <QuestionSet public-id="QuestionSetSPM:1" />
  <ReviewCategory>timeliness</ReviewCategory>
  <ReviewType public-id="SPMReviewType:1" />
</ReviewCategoryQuestionSet>
```

## Configuring Review Service Type and Review Category Typecodes

There are two typelists in ClaimCenter and ContactManager in which you define typecodes that affect service provider reviews, `ReviewServiceType` and `ReviewCategory`.

- `ReviewCategory` – Typecodes assigned by the `ReviewCategoryQuestionSet` entity to a question set, enabling the system to calculate category scores for that question set.
- `ReviewServiceType` – Typecodes used in a drop-down list on the review screen to enable you to choose the kind of service being provided by the service provider.

**Note:** You must define the same typecodes in both ClaimCenter and ContactManager.

### Configuring Review Category Typecodes

`ReviewCategory` typecodes are categories that you assign to question sets to enable category scores to be calculated for reviews. You assign a typecode to a question set in a `ReviewCategoryQuestionSet` XML element, as described in “Configuring Categories” on page 226. Before importing a review that uses a new typecode, add the typecode to ClaimCenter and ContactManager.

#### To define a `ReviewCategory` typecode:

1. Start ClaimCenter studio.

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

```
gwcc studio
```

2. In the Project window, navigate to `configuration` → `config` → `Extensions` → `Typelist` and double-click `ReviewCategory.ttx` to open it in the Typelist editor.

3. Right-click an existing typecode and choose `Add new` → `typecode`.

4. Give the typecode a `code`, which you use for the `ReviewCategory` in a `ReviewCategoryQuestionSet`.

For example, there is a typecode for `timeliness`. The sample auto repair shop review associates this category with a question set containing questions that ask how quickly the auto repair shop was able to schedule and complete repairs.

5. Give the typecode a **name** and a **desc**, a description. ClaimCenter uses the **name** in screens that show category scores. The description is useful in deciding which categories to apply to question sets.

6. Start ContactManager studio.

At a command prompt, navigate to the `ContactManager/bin` directory and enter:

```
gwab studio
```

7. Add the same typecode you added for ClaimCenter, as described previously in step 2 through step 5.

8. Restart ClaimCenter and ContactManager.

**Note:** Adding or changing a typecode is a data model change, and therefore requires a restart of the applications.

a. If necessary, stop ClaimCenter.

Open a command prompt in the `ClaimCenter/bin` directory and then enter the following command:

```
gwcc dev-stop
```

b. Regenerate the ClaimCenter data dictionary to ensure that your changes are correctly formatted:

```
gwcc regen-dictionary
```

c. Open a command prompt in the `ContactManager/bin` directory and then enter the following command:

```
gwab dev-stop
```

d. Regenerate the ContactManager data dictionary to ensure that your changes are correctly formatted:

```
gwab regen-dictionary
```

e. Start the ContactManager application:

```
gwab dev-start
```

f. In ClaimCenter Studio, refresh the WSDL for the ContactManager review summary web service as follows:

In the Project window, navigate to **configuration** → **config** → **RPC-Encoded Web Services** and double-click `ABReviewSummary.rws`.

Next, click the **Refresh** button near the top of the editor, to the right of the **URL** field.

g. Start the ClaimCenter application:

```
gwcc dev-start
```

## Configuring Review Service Type Typecodes

`ReviewServiceType` typecodes display in the **Service Type** drop-down list on the ClaimCenter review screen for new or incomplete reviews. This drop-down list enables the user to choose the kind of service performed by the service provider. If a service provider performs only one type of service, there might be no need to set up these typecodes for your review.

Part of defining one of these typecodes is specifying the contact subtype that the typecode applies to. You specify the contact subtype in the typecode's **Category**. The category is a typecode from the `Contact` typelist, and it matches the `ContactSubtype` defined in the associated `ReviewType`. The application then uses this typecode category to determine which typecodes to show in the **Service Type** list for a specific review. For a sample review type definition showing a `ContactSubtype`, see “Configuring Review Types” on page 225.

For example, when you start an auto repair shop review, the **Service Type** list shows six service types, **Body**, **Brakes**, **Other**, **Paint**, **Suspension**, and **Transmission and Engine**. All these service types are defined as typecodes in the `ReviewServiceType` typelist, and the **Category** for each of them is the `Contact` typecode **AutoRepairShop**.

### To define a Review Service Type typecode:

1. Start ClaimCenter studio.

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

```
gwcc studio
```

2. In the Project window, navigate to configuration → config → Extensions → Typelist and double-click ReviewServiceType.ttx.
3. Right-click an existing typecode and choose Add new → typecode.
4. Give the typecode a code, used internally to reference the typecode.  
For example, the service is wheel service, so the code can be wheels.
5. Give the typecode a name and a desc, a description. ClaimCenter shows the name in the drop-down list of service types on the review screen. The description is useful for documenting the typecode.  
For example, name the service Wheels. and give it the description Alignment, balancing, tires.
6. Right-click the new typecode and choose Add new → Add Categories.
7. In the Add Categories dialog box, choose the typecode you just added and click Next.  
For example, choose wheels and click Next.
8. For Typelist, choose Contact.
9. Choose a contact subtype that matches the ContactSubtype defined in the associated ReviewType.  
For example, choose AutoRepairShop.
10. Click Finish.
11. Start ContactManager studio.  
At a command prompt, navigate to the ContactManager/bin directory and enter:  

```
gwab studio
```
12. Add the same typecode you added for ClaimCenter, as described previously in step 2 through step 5.  
**Note:** You do not need to add categories for these typecodes in ContactManager, so step 6 through step 10 are not necessary.
13. Generate the data dictionaries for ClaimCenter and ContactManager to ensure that the changes you made are correct.
14. Restart ContactManager and ClaimCenter, and then refresh the WSDL for ABReviewSummary in ClaimCenter studio. For details, see step 8, “Restart ClaimCenter and ContactManager.” on page 228

## Configuring Question Sets, Questions, Question Choices, and Question Filters

For a full descriptions of question sets, questions, question choices, and question filters, see “Question Sets” on page 269 in the *Application Guide*. That topic provides a full, generic description of these entities.

This topic presents these entities from the viewpoint of service provider reviews and provides examples from the auto repair shop sample review.

### Configuring Question Sets

A QuestionSet provides a way to group Question entities and is part of the mechanism for setting up review categories. A question set does not point to its member questions. Instead, each question in the set designates the question set it belongs to.

To enable a QuestionSet to work with service provider reviews, you must set its QuestionSetType to spmreview. This type is not the only one—ClaimCenter also uses other question set types in other areas.

Additionally, to improve readability, you can:

- Make the Name match the ReviewCategory in the corresponding ReviewCategoryQuestionSet.

- Set the Priority to be the same as the ElementOrder defined in the corresponding ReviewCategoryQuestionSet.

**Note:** The following code samples are in the exported file `questions.xml`. See “Exporting and Importing Questions and Review Types” on page 225.

For example, the following question set is in the sample auto repair shop review:

```
<QuestionSet public-id="QuestionSetSPM:1">
  <Name>Timeliness</Name>
  <Priority>1</Priority>
  <QuestionSetType>spmreview</QuestionSetType>
  <ShouldRetireFromImportXML>false</ShouldRetireFromImportXML>
</QuestionSet>
```

The ReviewCategoryQuestionSet example ReviewCatQSet:1 references this question set and sets the question set's review category to the typecode `timeliness`.

```
<ReviewCategoryQuestionSet public-id="ReviewCatQSet:1">
  <ElementOrder>1</ElementOrder>
  <QuestionSet public-id="QuestionSetSPM:1" />
  <ReviewCategory>timeliness</ReviewCategory>
  <ReviewType public-id="SPMReviewType:1" />
</ReviewCategoryQuestionSet>
```

For this example, see “Configuring Categories” on page 226.

**Note:** If you want to use the same question set in more than one review, you must duplicate the question set and rename it. Then define a ReviewCategoryQuestionSet that specifies the review type, the review category, and the new question set.

## Configuring Questions and Question Choices

Service provider reviews do not require special characteristics for defining questions and question choices. However, you typically use certain features of these entities for service provider reviews.

Service provider reviews are typically scored. Therefore, each question belongs to a question set and designates the question set it belongs to. Additionally, to enable scoring, each question typically has a question type of choice, which requires a set of QuestionChoice answers. Each QuestionChoice designates which question it answers.

Naming conventions can make it clear that the questions and question choices are for service provider reviews. For example, you might give the questions IDs like `QuestionSPM:1`, the choices IDs like `QuestionChoiceSPM:1`, and so on.

The question set from the previous topic, `QuestionSetSPM:1`, has a set of questions associated with it that ask about the timeliness of an auto repair.

**Note:** The following code sample is in the exported file `questions.xml`. See “Exporting and Importing Questions and Review Types” on page 225.

The following code defines the first question in this set and its answers, its QuestionChoices:

```
<Question public-id="QuestionSPM:1">
  <DefaultAnswer/>
  <Indent>0</Indent>
  <Priority>0</Priority>
  <QuestionFormat/>
  <QuestionSet public-id="QuestionSetSPM:1"/>
  <QuestionType>Choice</QuestionType>
  <Required>false</Required>
  <ShouldRetireFromImportXML>false</ShouldRetireFromImportXML>
  <Text>How quickly was the service provider able to schedule the repair?</Text>
</Question>
<QuestionChoice public-id="QuestionChoiceSPM:1">
  <Code>win1day</Code>
  <Description>Within 1 day of initial contact</Description>
  <Name>Within 1 day of initial contact</Name>
  <Priority>1</Priority>
  <Question public-id="QuestionSPM:1"/>
  <Score>100</Score>
```

```
</QuestionChoice>
<QuestionChoice public-id="QuestionChoiceSPM:2">
  <Code>win3day</Code>
  <Description>Within 3 days of initial contact</Description>
  <Name>Within 3 days of initial contact</Name>
  <Priority>2</Priority>
  <Question public-id="QuestionSPM:1"/>
  <Score>50</Score>
</QuestionChoice>
<QuestionChoice public-id="QuestionChoiceSPM:3">
  <Code>more3days</Code>
  <Description>More than 3 days from initial contact</Description>
  <Name>More than 3 days from initial contact</Name>
  <Priority>3</Priority>
  <Question public-id="QuestionSPM:1"/>
  <Score>0</Score>
</QuestionChoice>
```

This question, How quickly was the service provider able to schedule the repair?, has a Priority of 0. This priority causes ClaimCenter to put the question at the top of the list when it shows the question set. Since its QuestionType is Choice, the question can be scored and has a series of QuestionChoice answers defined for it that have the scores 100, 50, and 0. The answers display in order according to their Priority values.

## Configuring Question Filters

Question filters are a way to conditionally show questions based on the answer to another question.

In the sample auto repair shop review, there are two filters that show additional questions if the answer to QuestionSPM:11 is yes. That question asks if there were any requotes from the auto repair shop for an auto repair. If the answer is no, this question's score is 100 and no further questions are shown. If the answer is yes, this question's score is 0, and the user sees QuestionSPM:12 and QuestionSPM:13, in that order. These questions ask how many requotes there were and the percentage of additional cost. The answers are choices, with better scores for a lower number of requotes and for a lower percentage of additional cost.

**Note:** The following code sample is in the exported file `questions.xml`. See “Exporting and Importing Questions and Review Types” on page 225.

The following XML code shows the two filters followed by the three questions and their answers:

```
<QuestionFilter public-id="SPMQFilter:1">
  <Answer>yes</Answer>
  <FilterQuestion public-id="QuestionSPM:11"/>
  <Question public-id="QuestionSPM:12"/>
</QuestionFilter>
<QuestionFilter public-id="SPMQFilter:2">
  <Answer>yes</Answer>
  <FilterQuestion public-id="QuestionSPM:11"/>
  <Question public-id="QuestionSPM:13"/>
</QuestionFilter>
<!-- ... -->
<Question public-id="QuestionSPM:11">
  <DefaultAnswer/>
  <Indent>0</Indent>
  <Priority>0</Priority>
  <QuestionFormat/>
  <QuestionSet public-id="QuestionSetSPM:5"/>
  <QuestionType>Choice</QuestionType>
  <Required>false</Required>
  <ShouldRetireFromImportXML>false</ShouldRetireFromImportXML>
  <Text>Were there any requotes?</Text>
</Question>
<QuestionChoice public-id="QuestionChoiceSPM:34">
  <Code>no</Code>
  <Description>No</Description>
  <Name>No</Name>
  <Priority>1</Priority>
  <Question public-id="QuestionSPM:11"/>
  <Score>100</Score>
</QuestionChoice>
<QuestionChoice public-id="QuestionChoiceSPM:35">
  <Code>yes</Code>
  <Description>Yes</Description>
  <Name>Yes</Name>
  <Priority>2</Priority>
```

```
<Question public-id="QuestionSPM:11"/>
<Score>0</Score>
</QuestionChoice>
<!-- ... -->
<Question public-id="QuestionSPM:12">
<DefaultAnswer/>
<Indent>0</Indent>
<Priority>1</Priority>
<QuestionFormat/>
<QuestionSet public-id="QuestionSetSPM:5"/>
<QuestionType>Choice</QuestionType>
<Required>false</Required>
<ShouldRetireFromImportXML>false</ShouldRetireFromImportXML>
<Text>How many quotes?</Text>
</Question>
<QuestionChoice public-id="QuestionChoiceSPM:36">
<Code>one</Code>
<Description>1</Description>
<Name>1</Name>
<Priority>1</Priority>
<Question public-id="QuestionSPM:12"/>
<Score>25</Score>
</QuestionChoice>
<QuestionChoice public-id="QuestionChoiceSPM:37">
<Code>twothree</Code>
<Description>2 to 3</Description>
<Name>2 to 3</Name>
<Priority>2</Priority>
<Question public-id="QuestionSPM:12"/>
<Score>10</Score>
</QuestionChoice>
<QuestionChoice public-id="QuestionChoiceSPM:38">
<Code>morethan3</Code>
<Description>More than 3</Description>
<Name>More than 3</Name>
<Priority>3</Priority>
<Question public-id="QuestionSPM:12"/>
<Score>0</Score>
</QuestionChoice>
<!-- ... -->
<Question public-id="QuestionSPM:13">
<DefaultAnswer/>
<Indent>0</Indent>
<Priority>2</Priority>
<QuestionFormat/>
<QuestionSet public-id="QuestionSetSPM:5"/>
<QuestionType>Choice</QuestionType>
<Required>false</Required>
<ShouldRetireFromImportXML>false</ShouldRetireFromImportXML>
<Text>% of difference between initial quote and final payment?</Text>
</Question>
<QuestionChoice public-id="QuestionChoiceSPM:39">
<Code>atmost5</Code>
<Description>Within 5% of original quote</Description>
<Name>Within 5% of original quote</Name>
<Priority>1</Priority>
<Question public-id="QuestionSPM:13"/>
<Score>25</Score>
</QuestionChoice>
<QuestionChoice public-id="QuestionChoiceSPM:40">
<Code>fiveten</Code>
<Description>Between 5% and 10%</Description>
<Name>Between 5% and 10%</Name>
<Priority>2</Priority>
<Question public-id="QuestionSPM:13"/>
<Score>10</Score>
</QuestionChoice>
<QuestionChoice public-id="QuestionChoiceSPM:41">
<Code>tentwenty</Code>
<Description>Between 10% and 20%</Description>
<Name>Between 10% and 20%</Name>
<Priority>3</Priority>
<Question public-id="QuestionSPM:13"/>
<Score>5</Score>
</QuestionChoice>
<QuestionChoice public-id="QuestionChoiceSPM:42">
<Code>over20</Code>
<Description>Greater than 20%</Description>
<Name>Greater than 20%</Name>
<Priority>4</Priority>
<Question public-id="QuestionSPM:13"/>
```

```
<Score>0</Score>
</QuestionChoice>
```

## About Scoring

After the completion of a service, typically, the insured and the claimant provide information to the claims adjuster or another user, who answers the questions in the performance review. This process results in a scored review that evaluates the services provided by a vendor on a single claim. Each review can have several scores. There is a total score and a score for each category of questions it contains. ClaimCenter scores the review and attaches it to the claim. A claim can have multiple reviews attached to it.

If you have set up the ClaimCenter scheduler to run the **ReviewSync** batch process, ClaimCenter queues the review's scores to send to ContactManager when the review is complete. Additionally, if you have set up the ContactManager scheduler to run the **ABCContactScoring** batch process, ContactManager processes reviews it receives. See “[Scheduling Reviews for Processing by ContactManager](#)” on page 219.

ContactManager creates vendor scores consisting of the average total score and average score for each category for all reviews from all claims for which the vendor provided services. These vendor scores are the main tool for managing categories of service providers.

The questions in the sample review have answers that range from zero to one hundred points. Scores, simple averages of the answered questions, are also in this range.

## Review Scoring

ClaimCenter provides two mechanism to score individual reviews, one to score sets of questions—categories—and another to score the entire review. These scoring mechanisms are similar. Both are simple arithmetic averages, with each answered question given equal weight.

The overall score of an individual review is the arithmetic average of all answered questions. Unanswered questions do not affect this average. To make particular questions always affect the score, you can define those questions as mandatory and provide default answers for them. Similarly, each category score of a single review is the average of all answered questions in that category.

ClaimCenter scores an incomplete review each time it saves the review. After you complete and submit the review by clicking **Complete Selected**, ClaimCenter scores the review and sends it to ContactManager. ClaimCenter also attaches the completed review to the claim and never rescores it, and the review becomes uneditable.

## Contact Scoring

Like reviews themselves, ContactManager can give each vendor two kinds of scores, an *overall* score and a set of *category* scores.

- **Overall score** – The arithmetic average of all overall scores from reviews pertaining to that contact. To be included, a review must be associated with the individual contact.
- **Category scores** – The average of all scores from the same category in individual reviews that pertain to that contact.

Since different review types can reuse the same categories, this average can include category scores taken from all review types that include the category. For example, you might have review types for auto glass repair and for auto repair, both of which have the category of glass installation. A vendor's category scores are the averages of the same categories from all reviews.

Unlike review scores, which ClaimCenter attaches to the claim when the user makes the review complete, aggregated vendor scores update asynchronously through either a batch process or an API call.

## Configuring Scores

By default, ClaimCenter scores are arithmetic means with each question given an equal weight. If you want to weight questions differently, you can either include unanswered questions in the scoring or calculate overall or category scores other than as arithmetic averages.

To configure scoring differently, you must write your own implementation of the `IContactReviewPlugin` plugin interface in ClaimCenter. The base configuration provides the plugin implementation `gw.plugin.spm.ab800.ContactReviewPlugin`. For more information, see “Service Provider Review Plugin Reference” on page 235.

## Using Gosu to Configure Service Provider Performance Reviews

ClaimCenter provides an API that enables you to use Gosu in rules to customize this feature. For example, you can write rules implementing logic such as: *If the overall vendor score for a particular review type is greater than a threshold, set its Preferred Vendor field.*

## Review Permissions

The following permissions are the primary ones associated with service provider reviews in ClaimCenter:

- `reviewviewlist` – Permission to view the list of reviews in the **Address Book** tab
- `reviewviewdetail` – Permission to view the details of reviews in ClaimCenter
- `reviewedit` – Permission to edit the **Review** screen that shows the scores for each claim
- `reviewcreate` – Permission to create a new review
- `reviewdeletecompleted` – Permission to delete a completed review that has been sent to ContactManager for calculation
- `reviewdeleteincomplete` – Permission to delete a review that has not been completed

ContactManager has the following review summary permissions:

- `revsumviewdetail` – Permission to view the **Review Summary** screen and see category scores for each summarized review.
- `revsumviewlist` – Permission to view the list of review summaries and to see the **Reviews** card for a contact.

## Permissions for Contacts

If you need more granular control over who gets to view, edit, create, and delete contacts, do not use the simple view and edit permissions. For example, you can designate users to be contact managers that manage certain subtypes of contacts. For this purpose, you would want the system to enforce permissions at the contact subtype level. You can also enforce permissions at the contact tag level. For more information, see “Securing Access to Contact Information” on page 109.

### See also

- “Securing Access to Contact Information” on page 109
- “Service Provider Performance Reviews” on page 216
- “Using Service Provider Performance Reviews” on page 217
- “Data Model for Service Provider Performance Reviews” on page 222
- “Service Provider Review Plugin Reference” on page 235

# Service Provider Review Plugin Reference

There are two plugin interfaces and two plugins that implement these interfaces that support service provider reviews.

For general information on plugins, see “Plugin Overview” on page 163 in the *Integration Guide*.

This topic includes:

- “ClaimCenter Contact Review Plugin” on page 235
- “ContactManager Scoring Plugin” on page 236

## ClaimCenter Contact Review Plugin

The ClaimCenter plugin `gw.plugin.spm.ab800.ContactReviewPlugin` implements the `IContactReviewPlugin` interface.

The `IContactReviewPlugin` interface defines the following functionality:

- A hook for overriding the default scoring functionality.
- The ClaimCenter side of the integration with ContactManager.

The `ContactReviewPlugin` class implements the methods that follow. You can see these methods in ClaimCenter Studio by opening the class.

- `java.lang.Integer scoreReview(gw.cc.contact.entity.Review review)`

This method scores the review for a given set of answers. If the review has category scores, the method creates or updates those scores on the `Review` object as necessary. It returns the overall score for the review.

- `java.lang.String submitReview(gw.cc.contact.entity.Review review)`

This method submits a summary of the completed review to ContactManager. It returns the `addressBookUID` for the submitted `ReviewSummary`.

Never call this method directly. If you want to extend the header information in `ReviewSummary`, you must edit this method to pass that information to the `ABVendorEvaluationAPIReviewSummary` web service. The method to call is `toReviewSummary(bundle : Bundle) : ReviewSummary`.

For information on `ABVendorEvaluationAPIReviewSummary` web service, open the class in ContactManager Studio. In the Project window, press `Ctrl+N` and enter `ABVendorEvaluationAPIReviewSummary`, and then double-click the class name in the search results.

- `void deleteReview(gw.cc.contact.entity.Review review)`

This method deletes the summary corresponding to a completed review from ContactManager. Do not call this method manually.

- `public void updateScores(entity.Contact)`

This method enables ClaimCenter to trigger a score update on an `ABCContact` instance corresponding to the given local `Contact` without waiting for a run of the `ABCContactScoring` batch job. The local `Contact` must be linked to ContactManager. You can use the method in extensions or rules, but you cannot call it directly.

If you implement your own contact review plugin, you must register your new plugin as follows:

1. Start ClaimCenter studio.

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

```
gwcc studio
```

2. In the Project window, navigate to `configuration → config → Plugins → registry` and double-click `IContactReviewPlugin.gwp`.

3. In the Registry editor, you can see that the registered implementation is the Gosu class `gw.plugin.spm.ab800.ContactReviewPlugin`.

4. In the Registry editor, click Remove Plugin .
5. Click Add Plugin  and, in the drop-down menu, choose either **Add Gosu Plugin** or **Add Java Plugin**, depending on your implementation language.
6. In the **Gosu Class** or **Java Class** field, either search for your class or enter the class with the full package name.
7. Add the password and username parameters you expect to use with the plugin. The default settings are:

Name	Value
password	gw
username	ClientAppCC

## ContactManager Scoring Plugin

The ContactManager plugin implementation `gw.plugin.spm.impl.ABContactScoringPlugin` implements the interface `IABContactScoringPlugin`. The plugin class implements the methods that follow.

- `java.lang.Integer scoreABContact(gw.ab.addressbook.entity.ABContact abContact)`  
This method calculates the average score and average category scores for a contact. It returns the overall average review score. ClaimCenter scripting calls it through `ContactReviewPlugin` and `ABVendorEvaluationAPIReviewSummary`. The `ABContactScoring` work queue worker also calls this method. You can implement a different version of this method to change how scoring works. For example, you might want to weight scores in some manner.
- `java.util.Iterator<gw.pl.persistence.core.Key> findContactsToScore()`  
This method finds the set of ABContacts to be scored for the `ABContactScoring` work queue writer. It returns the iterator key for the contacts it finds.  
This method is called by the work queue writer. You can reimplement this method to change how scoring works. The method is useful if, in your scoring system, some contacts need to be rescored without having received new reviews.  
For example, you might want to weight scores by recency of reviews or to cancel scores that are based on old reviews.

If you implement your own contact scoring plugin, you must register your new plugin as follows:

1. Start ClaimCenter studio.  
At a command prompt, navigate to the `ContactManager/bin` directory and enter:  
`gwab studio`
2. In the Project window, navigate to `configuration → config → Plugins → registry` and double-click `IABContactScoringPlugin.gwp`.
3. In the Registry editor, you can see that the default implementation is `gw.plugin.spm.impl.ABContactScoringPlugin`.
4. In the Registry editor, click Remove Plugin .
5. Click Add Plugin  and, in the drop-down menu, choose either **Add Gosu Plugin** or **Add Java Plugin**, depending on your implementation language.
6. In the **Gosu Class** or **Java Class** field, either search for your class or enter the class with the full package name.

# Working Directly in ContactManager

This topic provides information about working directly with contact data in ContactManager and describes the screens you use to manage ContactManager.

This topic includes:

- “Logging in to ContactManager” on page 238
- “Changing Your Password in ContactManager” on page 239
- “Changing Your User Preferences in ContactManager” on page 239
- “Selecting International Settings in ContactManager” on page 239
- “ContactManager User Interface” on page 240
- “Importing and Exporting Administrative Data” on page 243
- “Managing Contact Data” on page 245
- “Vendor Services Onboarding” on page 254

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

- For information on directory structure and key directories like `module` and `configuration`, see “ClaimCenter Configuration Files” on page 93 in the *Configuration Guide*.
- For information on configuration settings, see “The config.xml File” on page 13 in the *System Administration Guide*.
- For information on defining the server environment, see “Defining the Application Server Environment” on page 14 in the *System Administration Guide*.
- For information on setting up logging, see “Configuring Logging” on page 25 in the *System Administration Guide*.
- For a list of command-line administrative tools, such as the `maintenance_tools` command, see “ClaimCenter Administrative Commands” on page 181 in the *System Administration Guide*.

**Note:** To enter these commands, at a command prompt, navigate to `ContactManager/admin/bin`.

Some commonly used administrative tools described in the topic include:

- “Maintenance Tools Command” on page 183 in the *System Administration Guide*

- “System Tools Command” on page 187 in the *System Administration Guide*
- “Table Import Command” on page 191 in the *System Administration Guide*
- For descriptions of command-line tools you can use to operate ContactManager in the development and configuration environment see the *Installation Guide* topics that follow.
  - “QuickStart Command Tools” on page 106 in the *Installation Guide*
  - “Build Tools” on page 106 in the *Installation Guide*

**Note:** To enter these commands, at a command prompt navigate to `ContactManager/bin`. These topics are written for ClaimCenter and therefore say to use `gwcc` to start each command. Substitute `gwab` in these commands. For example, use `gwab studio` to start ContactManager Studio and `gwab dev-start` to start ContactManager in development mode.

## Logging in to ContactManager

Logging in to ContactManager requires the following:

- **A web browser** – Guidewire supports Firefox, Google Chrome, and Microsoft Internet Explorer.
- **The URL (web address) for connecting to ContactManager** – In the base configuration, the URL is:

`http://localhost:8280/ab`

You can set up a **Favorite** link to the URL or a create a shortcut on your computer desktop that starts your web browser with that URL. For details on installation and the web address, see “Installing ContactManager” on page 39.

- **A user name and password** – You must have one or more roles assigned to your user name by a system administrator. Roles determine the screens you can access and what you can do in ContactManager.

### To log in to ContactManager

1. Launch ContactManager by running a web browser and using the appropriate web address, such as:

`http://localhost:8280/ab`

2. Enter your **User Name** and **Password** on the login screen.

If successful, ContactManager shows your startup view, or landing page.

If you first click **Keep me logged in** and then log in, you are logged in automatically for the next seven days whenever you navigate to the login page. To support this feature, the application must be hosted by the same application server each time you return to the page. If you log out of ContactManager, you will need to log in the next time you navigate to the login page.

When you click **Keep me logged in**, ContactManager writes a cookie to your machine. As is the case with all web browser cookies, writing this cookie can expose a security risk if other people get access to the cookie. For example, someone could copy the cookie to another machine and then be able to log in without entering a user name or password. If this security issue is a concern, you can remove this field from the login page in your ContactManager implementation.

When you log in, ContactManager displays your startup view, or landing page.

**Notes:** Because ContactManager generates screens dynamically:

- You cannot create **Favorites** to screens other than the login screen,
- The **Back** button of the browser is not supported.

In the default configuration, ContactManager initially opens the **Search** screen on the **Contacts** tab.

### See also

- “Working Directly in ContactManager” on page 237

## Changing Your Password in ContactManager

### To change your password

1. Log in as described at “Logging in to ContactManager” on page 238.
2. On the Options menu , click Preferences.  
The Preferences worksheet opens below the main work area.
3. Enter your old password.
4. Enter the new password.
5. Enter the new password again in the Confirm New Password field.
6. Click Update.

### See also

- “Working Directly in ContactManager” on page 237

## Changing Your User Preferences in ContactManager

### To change your user preferences

1. Log in as described at “Logging in to ContactManager” on page 238.
2. On the Options menu , click Preferences.  
The Preferences worksheet opens below the main work area.

In the Preferences worksheet, you can change your password. You can also set your regional formats, your default country, and your default phone region. You set the last three if you want your personal settings to be different from the ones set for all users of ContactManager.

- **Password** – Reset your password. See “Changing Your Password in ContactManager” on page 239.
- **Regional Formats** – Set the regional formats that ContactManager uses to enter and display dates, times, numbers, monetary amounts, and names.
- **Default Country** – Determines the settings for names and addresses.
- **Default Phone Region** – Determines how phone number entries are handled, especially the country code setting.

### See also

- “Working Directly in ContactManager” on page 237

## Selecting International Settings in ContactManager

In ContactManager, each user can set the following:

- The language that ContactManager uses to display labels and drop-down menu choices
- The regional formats that ContactManager uses to enter and display dates, times, numbers, monetary amounts, and names.

You set your personal preferences for display language and for regional formats by using the Options menu  at the top, right-hand side of the ContactManager screen. On that menu, click International, and then select one of the following:

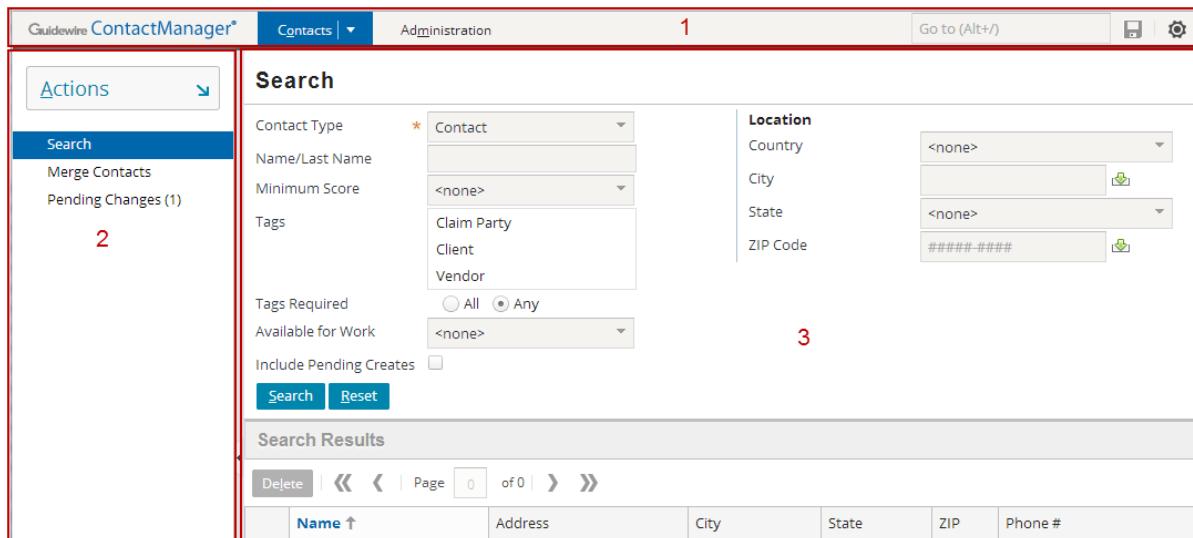
- Language

- Regional Formats

For more information, see “International Settings in ContactManager” on page 242.

## ContactManager User Interface

Like the Guidewire core applications, ContactManager has tabs at the top of the screen and an **Actions** button and a menu on the left for navigation.



The ContactManager main user interface contains the following areas:

Area	Description
1	<p>The Tab Bar contains:</p> <ul style="list-style-type: none"> <li>Tabs, which group major areas of work for you, providing main screens, actions on the <b>Actions</b> menu, and sidebar menu links. The tabs you see depend on your user permissions. All users can see the <b>Contacts</b> tab.</li> <li>QuickJump box. The text box that displays <b>Go to (Alt+)</b>. For more information, see “QuickJump” on page 71 in the <i>Application Guide</i>.</li> <li>Unsaved Work button . This button is active if you have work you have not saved. You can click the button and go to any screen in which you have not saved your changes. This feature is useful if you must navigate away from a screen in which you are making changes, but you intend to return to it later. After you complete and save your work, ContactManager removes that item from the Unsaved Work drop-down menu.</li> <li>The Options  menu. The selections available on this menu depend on where you are in the ContactManager user interface. On the screen shown in the previous figure, you see International, Help, About, Preferences, Clear Layout Preferences, and Log Out Username.</li> </ul> <p>For a description of the <b>International</b> menu and its submenu selections, <b>Regional Settings</b> and <b>Language</b>, see “International Settings in ContactManager” on page 242.</p>
2	The sidebar contains menu links and the <b>Actions</b> menu. Use the sidebar menu links to navigate to screens where you can do your work. The items in the sidebar are contextual and change depending on tab and menu selections.
3	The Screen Area shows most of your business information and is where you interact with that information.

### Contacts Tab

The **Contacts** tab enables you to:

- Click **Actions** to create a new person, company, or place to store in ContactManager, if you have permissions to do so.
- Click **Search** to find contacts by using search criteria, such as contact subtype and location.

- Edit and delete any of these entities, if you have permissions to do so.
- Click **Merge Contacts** to merge duplicate contacts that have been discovered by the **Duplicate Contacts Finder** batch process, if you have permissions to do so. For more information, see “[Detecting and Merging Duplicate Contacts](#)” on page 245.
- Click **Pending Changes** to review changes to contacts that were made by core application users who did not have permissions to make the changes. You need the correct permissions to do so. For more information, see “[Reviewing Pending Changes to Contacts](#)” on page 252.

## Administration Tab

By using the ContactManager **Administration** tab, you can manage roles, permissions, login name and password, and group membership for your ContactManager users. These users log in directly to ContactManager and are not necessarily the same as the Guidewire core application users who you might have authorized to perform ContactManager 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 ContactManager users and groups.
- Also on the left in the sidebar is a hierarchical list of all your groups and users.

Additionally, you can choose the following items from the sidebar:

- **Users and Security** – Groups the following administrative tasks:
  - **Users** – Search for users by role, location, username, and first and last name.
  - **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 ContactManager users. For more information, see “[Securing Access to Contact Information](#)” on page 109.
  - **Regions** – Add, delete, and edit the current regions for ContactManager contacts. *Regions* are geographical areas you can use to define groups’ areas of responsibility. A region can contain one or more states, ZIP codes, counties, or other address elements, such as Canadian provinces. You can assign users and groups to cover one or more regions, and then define business rules to provide location-based assignment. For more information on regions, see “[Managing Regions](#)” on page 479 in the *Application Guide*.
- **Monitoring** – Groups the following administrative tasks:
  - **Message Queues** – Suspend and resume event messages sent by ContactManager and restart the messaging engine. One use of the **Message Queues** screen is to restart message queues that have been suspended. For example, the message queue for a core application might have been suspended because the core application was not running when ContactManager tried to send contact data. For more information, see the topic for your core application:
    - “[Troubleshooting the ClaimCenter Connection with ContactManager](#)” on page 51
    - “[Troubleshooting the PolicyCenter Connection with ContactManager](#)” on page 57
    - “[Troubleshooting the BillingCenter Connection with ContactManager](#)” on page 63
  - For more information on messaging, see “[Monitoring and Managing Event Messages](#)” on page 64 in the *System Administration Guide*.
- **Utilities** – Groups the following administrative tasks:
  - **Import Data** – Import certain types of data through the ContactManager interface.
  - **Export Data** – Export certain types of data through the ContactManager interface.
  - **Script Parameters** – View existing ContactManager script parameters. To create new script parameters, you must use ContactManager Studio. For more information, see “[Script Parameters](#)” on page 105 in the *Configuration Guide*.

## Internal Tools and Server Tools Tabs

As described in “Step 2: Load Sample Data” on page 40, you can log in to ContactManager as the super user. You can then press **Alt+Shift+T**, click the **Internal Tools** tab, and load sample data.

You can also click the **Server Tools** tab and perform administrative tasks, such as starting batch jobs manually, as described at “Starting Batch Processes” on page 253.

For more information on using these tabs, see “Using Server and Internal Tools” on page 159 in the *System Administration Guide*.

### See also

- “Working Directly in ContactManager” on page 237

## International Settings in ContactManager

You can set the following international settings:

- The language in which ContactManager displays labels and drop-down menu choices
- The regional formats that ContactManager uses to enter and display dates, times, numbers, monetary amounts, and names

You set your personal preferences for display language and for regional formats by using the Options  menu at the top, right-hand side of the ClaimCenter screen. On that menu, click **International**, and then select one of the following:

- **Language**
- **Regional Formats**

To set international settings in the application, you must configure ContactManager with more than one region. Your configuration of regions and languages determines what you see on this menu, as follows:

- ContactManager hides the **Language** submenu if only one language is installed.
- ContactManager hides the **Regional Formats** submenu if only one region is configured.
- ContactManager hides the **International** menu option entirely if a single language is installed and ContactManager is configured for a single locale.

ContactManager indicates the current selections for **Language** and **Regional Formats** by placing a check mark to the left of the selected options and displaying them in gray.

### Options for Language

In the base configuration, Guidewire has a single display language, English. To view another language in ContactManager, you must install a language pack and configure ContactManager for that language. If your installation has more than one language, you can select among them from the **Language** submenu. The **LanguageType** typelist defines the set of language choices that the menu displays.

If you do not select a display language from the **Language** submenu, ContactManager uses the default language. The configuration parameter **DefaultApplicationLanguage** specifies the default language. In the base configuration, the default language is English.

### Options for Regional Formats

If your installation contains more than one configured region, you can select a regional format for that region from the **Regional Formats** submenu. At the time you configure a region, you define regional formats for it.

Regional formats specify the visual layout of the following kinds of data:

- Date
- Time

- Number
- Monetary amounts
- Names of people and companies

The **LocaleType** typelist defines the names of regional formats that you can select from the **Regional Formats** submenu. The base configuration defines the following locale types:

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>• Australia (English)</li><li>• Canada (English)</li><li>• Canada (French)</li><li>• France (French)</li></ul> | <ul style="list-style-type: none"><li>• Germany (German)</li><li>• Great Britain (English)</li><li>• Japan (Japanese)</li><li>• United States (English)</li></ul> |
|--|---|

If you do not select a regional format from the **Regional Formats** menu, ClaimCenter uses the regional formats of the default region. The configuration parameter **DefaultApplicationLocale** specifies the default region. In the base configuration, the default region is **en\_US**, United States (English). If you select your preference for region from the **Regional Formats** menu, you can later use the default region again only by selecting it from the **Regional Formats** menu.

#### See also

- “Understanding Globalization” on page 11 in the *Globalization Guide*
- “Working with Regional Formats” on page 67 in the *Globalization Guide*
- “Working with Languages” on page 21 in the *Globalization Guide*

## Importing and Exporting Administrative Data

While much administrative data is entered directly into ContactManager, there are times when it is necessary or convenient to transfer this information in bulk. The **Import Data** and **Export Data** screens available on the **Administration** tab provide a convenient way of moving administrative data, role definitions, and vendor service trees as XML files. In the case of vendor service trees, administrative import and export is required to add new services.

Additionally, you can export the security dictionary as either HTML or XML.

You can also import or export other types of data, in either XML or CSV format, by using an API. Also, there is a command-line command to import files in either format, but not to export them.

Method	Import	Export	File Formats
user interface	admin.xml, vendorservicetree.xml	admin.xml, roles.xml, vendorservicetree.xml securitydictionarynumber.zip	XML, XML or HTML in compressed ZIP
command line	any	no	XML, CSV
API	any	any	XML, CSV

#### See also

- “Administration Tab” on page 241

This topic includes:

- “Importing Administrative Data and Vendor Service Trees” on page 244
- “Exporting Data in the Administration Tab” on page 244
- “Importing and Exporting with Gosu and from the Command Line” on page 244

## Importing Administrative Data and Vendor Service Trees

### To import administrative data and vendor service trees

1. Navigate to Administration tab → Utilities → Import Data to select a file to import.

You can click the **Browse** button to find the file. For example, you have created a file of vendor services called `vendorservicetree.xml`. This file must be either in XML or zipped XML format, with an XSD compatible with the XML files you can import. If you want to import administrative data, you can instead import any subset of this type of data, such as users or regions.

2. Click **Next**, and follow the prompts to resolve differences between the data in the imported file and data already in the database.

Data not yet in the database is imported without question. If the imported data differs from what is already in the database, these prompts enable you either to accept the imported data or to keep what is in the database.

3. Click **Finish** to complete the import.

For more information, see “Importing Data From the User Interface” on page 116 in the *System Administration Guide*.

## Exporting Data in the Administration Tab

Navigate to Administration tab → Utilities → Export Data to see the following categories. Each category has one or more types of data you can export. Each file contains all the data of a certain type in your installation. These export categories are:

- **Export Administrative Data**
  - **Admin** – Exports all administrative data to `admin.xml`, including data of the following types:  
`Group`, and `GroupRegion`, `GroupRuleSet`, and `GroupUser`  
`Region`  
`Role`, `Privileges`, `RolePrivilege`, and `Permission`  
`User`, including `AttributeUser`, `UserRole`, and `UserSettings`  
`UserPreference`
  - **Roles** – Exports all data that maps system permissions to roles to the file `roles.xml`. If you choose **Admin** as the export type, the same role data is exported with the other administrative data. The file has data of the following types:  
`Role`  
`Privileges`  
`RolePrivilege`  
`Permission`
  - **Vendor Service Tree** – Exports the entire vendor service hierarchy to the XML file `vendorservicetree.xml`.
  - **Export Security Dictionary** – Exports the security dictionary as a zipped XML or HTML file.

## Importing and Exporting with Gosu and from the Command Line

You might want to import or export other types of data or use files in formats other than XML. For example, if you receive new information from an external system, you might want to import this new data into ContactManager in a single step. Gosu classes and command-line functions are your two alternatives to the user interface described previously. Gosu classes enable you both to import and to export, but the command line commands support only importing, not exporting.

## Importing from the Command Line

There are command-line commands for importing, but not exporting, XML and CSV files containing any kind of data, not just the types of data supported by the **Administration** tab. See “Importing and Exporting Administrative Data from ClaimCenter” on page 115 in the *System Administration Guide* for more details.

# Managing Contact Data

You can use Guidewire core application screens to manage contacts stored in ContactManager.

- **ClaimCenter** – In the ClaimCenter **Address Book** tab, you can search for and view, but not edit, existing contacts. You can change ContactManager data in ClaimCenter by using screens for managing or adding claim contacts, such as the **New Claim** wizard or a claim’s **Parties Involved** screen.
- **PolicyCenter** – In the PolicyCenter **Contact** tab, you can create new contacts, search for existing contacts, select a recently viewed contact, and change contact information. You can also create an account for the contact. PolicyCenter additionally enables you to work with contacts in its **Account** and **Policy** screens, where you can add, remove, and update contacts in various roles.
- **BillingCenter** – In the BillingCenter **Account** and **Policy** tabs, you can click **Contacts** to open the **Contacts** screen. On this screen you can add new contacts, search for existing contacts, select a recently viewed contact, and change contact information. Additionally, you can search for contacts by clicking the **Search** tab and choosing **Contacts** from the drop-list.

To work with ContactManager from a Guidewire core application, you must install both ContactManager and the Guidewire core application and integrate them. For more information, see:

- “Installing ContactManager” on page 39
- “Integrating ContactManager with Guidewire Core Applications” on page 45.

You can also manage contacts stored in ContactManager by logging in directly to ContactManager as a user with the appropriate role. For example, you must log in to ContactManager to merge contacts or approve pending contact changes. Additionally, if you want to add contacts that are not on a claim, such as vendor contacts, you must do so in ContactManager.

You log in as a user with the **Contact Manager** role, which has permissions supporting viewing, searching for, merging, adding, editing, and deleting contacts and reviewing pending contacts. These permissions include:

- **View merge** permission, with code `abviewmerge`, which enables you to see **Merge Contacts** screens
- **View pending** permission, with code `abviewpending`, which enables you to see **Pending Contacts** screens

Other reasons to log in to ContactManager are to manage its users or to manage the server. For example, you can assign the **User Admin** role to a user. That user can then log in to ContactManager and manage its users.

This topic includes:

- “Detecting and Merging Duplicate Contacts” on page 245
- “Reviewing Pending Changes to Contacts” on page 252
- “Starting Batch Processes” on page 253

### See also

- “Vendor Services Onboarding” on page 254

## Detecting and Merging Duplicate Contacts

Because creating contacts can result in duplicate contact records, there are features in ContactManager to detect duplicate contacts and enable merging them. A user with appropriate privileges in ContactManager can run the

Duplicate Contacts Finder batch process to detect duplicate contacts, and then merge each duplicate contact in the Merge Contacts screen.

**IMPORTANT** The first time you run the Duplicate Contacts Finder batch process, it can perform a large number of comparisons requiring considerable application resources. You can limit the number of contacts it processes by setting a processing time and date before which contacts are ignored. For more information, see “Configuring the Duplicate Contacts Finder Batch Process” on page 246. Additionally, Guidewire recommends that you set this batch process to run initially at a time when general access to your server is restricted. You might also need to allocate time for a ContactManager user to resolve what is likely to be a large number of duplicate contacts.

This topic includes:

- “Configuring the Duplicate Contacts Finder Batch Process” on page 246
- “Running the Batch Process Manually” on page 247
- “Setting the Batch Process and Work Queue to Run Automatically” on page 248
- “Merging Duplicate Contacts” on page 249
- “Merging Contacts and Notifying Core Applications” on page 251

## Configuring the Duplicate Contacts Finder Batch Process

The Duplicate Contacts Finder batch process compares a set of contacts that have recently changed against the rest of the contacts in the ContactManager database. *Recently* is defined by either of the following:

- Contact changes that have occurred since the last time the batch process ran
- Contact changes that occurred since the date and time set in the configuration parameter `DuplicateContactsEarliestModificationDate`

For example:

1. The Duplicate Contacts Finder batch process ran today at 1:00 am. There were 100 contacts in the database at that time.
2. The batch process found 5 possible duplicate contacts.
3. The contact administrator checked today for duplicate contacts and merged 5 of them, leaving 95 contacts in the database.
4. Since the batch process last ran at 1:00 am today, 10 new contacts were created.
5. The batch process runs at 1:00 am tomorrow. It compares each of those 10 new contacts against 104 contacts—the other 9 new contacts and the 95 contacts that were already in the database.

The batch process checks the value of `DuplicateContactsEarliestModificationDate`, which you can set in `config.xml`. The batch process ignores any contacts created or modified before the date and time in this configuration parameter. You can use this setting to avoid processing every contact in the database the first time you run the batch process.

For example, you imported a large number of contacts into the database on November 18, 2011, and you set `DuplicateContactsEarliestModificationDate` to 11/19/2011 12:00 AM. The batch process does not process any contacts from your November 18th import unless you modified them after 12:00 am on November 19, 2011. The batch process does process contacts modified after that date and time.

The date and time for this entry must have the following format:

mm/dd/yyyy hh:mm am

### To set this configuration parameter

1. Start ContactManager Studio.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:

```
gwab studio
```

2. Navigate in the Project window to `configuration → config` and double-click `config.xml` to open this file in the editor.
3. Press `CTRL+F` and enter `DuplicateContactsEarliestModificationDate` to find this parameter setting in the file.
4. Enter a date and time value. For example:  

```
DuplicateContactsEarliestModificationDate="11/19/2011 12:00 AM"
```
5. Save your changes.
6. If `ContactManager` is running, stop `ContactManager` and restart it to pick up the configuration change.
7. Run the batch process, or set it up to run automatically, as described at:
  - “Running the Batch Process Manually” on page 247
  - “Setting the Batch Process and Work Queue to Run Automatically” on page 248

## Running the Batch Process Manually

You can run the `Duplicate Contacts Finder` batch process manually, for example, for testing purposes.

1. Log in as a user with the `Tools View` role. This role has the permission `View BatchProcess tools screen`, which has `code toolsBatchProcessview` and enables you to work with batch processes. For example, log in with user-name `su` and password `gw`.
2. Press `Alt+Shift+T` to open the screen showing the `Server Tools` tab and the `Internal Tools` tab.
3. Click the `Server Tools` tab.
4. From the sidebar on the left under `Actions`, choose `Batch Process Info`.
5. In the `Batch Process Info` screen, scroll down to the `Duplicate Contacts Finder` batch process and click the `Run` button in that row. When the process completes, the `Last Run Status` changes to `Completed` and the `Last Run` column updates to show the run date and time.

Batch Process	Description	Action	Last Run	Last Run Status
AB Contact Score Aggregator	Calculates aggregate review scores for ABContacts from submitted ReviewSummaries.	Run Stop Download History		Not available
AB Geocode Writer	Geocoding ABAddresses queue writer.	Run Stop Download History	07/25/2013 04:38:36...	Completed
Duplicate Contacts Finder	Finds new duplicate contacts and creates a browsable list of said contacts.	Run Stop Download History	07/25/2013 04:47:16...	Completed
Phone number normalizer	Performs a normalization of phone numbers contact	Run Stop Download History	07/25/2013 04:46:21...	Completed

## Running the Batch Process from the Command Line

You can also run the batch process from the command line by using the `maintenance_tools` command.

1. Open a command prompt and navigate to `ContactManager/admin/bin`.

2. Enter the following command:

```
maintenance_tools -startprocess duplicatecontacts
```

**See also**

- For information on the `maintenance_tools` command, see “Maintenance Tools Command” on page 183 in the *System Administration Guide*.

## Setting the Batch Process and Work Queue to Run Automatically

The **Duplicate Contact Finder** batch process is set to run by default once a week on Sunday at 12:00 pm. You can:

- Change this setting to run it when you like. See “Changing Batch Process Settings” on page 248.
- Set up the work queue itself. See “Changing Work Queue Settings” on page 248.
- Set the number of results indicating that the search parameters are not sufficiently specific and return too many results. See “Changing Match Results and Search Scope Settings” on page 249.
- Set the scope of a search. See “Changing Match Results and Search Scope Settings” on page 249.

**See also**

- “Batch Processes and Work Queues” on page 123 in the *System Administration Guide*
- “Distributable Work Queues” on page 124 in the *System Administration Guide*

### Changing Batch Process Settings

In ContactManager Studio, navigate in the **Project** window to **configuration** → **config** → **scheduler** and double-click **scheduler-config.xml** to open the file in the editor. In this XML file, you can change the settings for the **ProcessSchedule** named **DuplicateContacts**. The default entry sets the batch process to run at noon every Sunday.

```
<ProcessSchedule process="DuplicateContacts">
  <CronSchedule dayofmonth="?" dayofweek="SUN" hours="12"/>
</ProcessSchedule>
```

For example, the following entry would cause the process to run at midnight every night:

```
<ProcessSchedule process="DuplicateContacts">
  <CronSchedule hours="0"/>
</ProcessSchedule>
```

For complete information on the scheduling settings for batch processes see “Defining a Schedule Specification” on page 144 in the *System Administration Guide*.

### Changing Work Queue Settings

In ContactManager Studio, navigate in the **Projects** window to **configuration** → **config** → **workqueue** and double-click **work-queue.xml** to open it in the editor. In this XML file, you can change the settings for the **workQueueClass** named **com.guidewire.ab.domain.contact.DuplicateContactsFinderWorkQueue**:

```
<work-queue
  progressinterval="600000"
  workQueueClass="com.guidewire.ab.domain.contact.DuplicateContactsFinderWorkQueue">
  <worker batchsize="100" instances="1"/>
</work-queue>
```

The settings are described in the comments at the beginning of the **work-queue.xml** file. The base configuration settings for **com.guidewire.ab.domain.contact.DuplicateContactsFinderWorkQueue** are:

- **minpollinterval="0"** – The worker does not sleep after completing a work item.
- **maxpollinterval="60000"** – The worker wakes up and polls for work every 60,000 milliseconds, or one minute.
- **progressinterval="600000"** – The amount of time the system gives the worker to do the work before assuming that it ran into a problem and reassigning the work to another worker. By default this time is 600,000 milliseconds, or 10 minutes.
- **batchsize="100"** – Each worker takes 100 new contact names at a time and compares each one to the contacts in the database.
- **instances="1"** – The number of workers for the process.

For more information on work queues, see:

- “Distributable Work Queues” on page 124 in the *System Administration Guide*
- “Configuring Distributable Work Queues” on page 128 in the *System Administration Guide*

### Changing Match Results and Search Scope Settings

You can set the maximum number of matches that can be found for a contact and how wide the search is. In ContactManager Studio, navigate in the Project window to **configuration** → **config** and double-click **config.xml**. Change the following parameters under the heading **DuplicateContactsFinderWorkQueue**:

- **MaxDuplicateContactsFinderWorkQueueResults** – By default, each **DuplicateContactsFinderWorkQueue** worker finds up to one thousand potential duplicates for a single contact. This parameter is intended to catch searches that have too loose a set of search fields and, therefore, match too many contacts in the database. If the worker finds more than the maximum allowed number of duplicates, it does not create the list of duplicates in its results list. Instead, it logs an exception that indicates the **LinkID** of the new contact. In addition, the new contact has an exception entry in the results list. The default setting is:  
`<param name="MaxDuplicateContactsFinderWorkQueueResults" value="1000"/>`
- **DuplicateContactsWideSearch** – By default, this parameter is **true**, causing the batch process to perform a wide search using the match settings for **ABPerson**, **ABCompany**, and **ABPlace**. The default setting is:  
`<param name="DuplicateContactsWideSearch" value="true"/>`

**Note:** This parameter applies only to the duplicate finder batch process. It does not set general searching behavior in the plugin **FindDuplicatesPlugin**.

A wide search can return matches for a larger number of subtypes than standard matching by using the contact’s supertypes. This type of search aids in finding a contact that was added erroneously to the database as more than one subtype. For example, if the subtype of the contact being compared against the database is **ABDoctor**, a normal search returns only matches for **ABDoctor** subtypes. The wide search in the base configuration returns matches for all **ABPerson** subtypes, not just **ABDoctor**.

The plugin **FindDuplicatesPlugin** defines the **ABContact** subtypes used in this search in the variable **WIDE\_MAP**. For more information, see “**IFindDuplicatesPlugin** Plugin Interface” on page 298.

### Merging Duplicate Contacts

After the **Duplicate Contacts Finder** batch process has run, you can see any potential duplicate contacts this process found by clicking **Merge Contacts** on the **Contacts** tab. To see **Merge Contacts**, log in as a user with the **Contact Manager** role. This role includes the permission **View merge**, which has code **abviewmerge** and enables you to use the **Merge Contacts** screens.

For information the **Duplicate Contacts Finder** batch process, see:

- “Configuring the Duplicate Contacts Finder Batch Process” on page 246
- “Running the Batch Process Manually” on page 247
- “Setting the Batch Process and Work Queue to Run Automatically” on page 248

#### To merge duplicate contacts

1. Log in as a user with the **Contact Manager** role. For example, log in with username **su** and password **gw**.

**Note:** The **Contact Manager** role includes the permission to view the merge screen, **abviewmerge**. To merge contacts, there are additional permissions to edit and delete contacts that are also part of the **Contact Manager** role, such as **abedit**, **abdelete**, and **anytagedit**.

2. Click the **Contacts** tab.
3. In the sidebar, click **Merge Contacts**.

4. In the **Merge Contacts** screen, you see all the duplicate contacts detected by the last run of the **Duplicate Contacts Finder** batch process. The duplicates are grouped into pairs. You typically resolve one pair at a time, but you can mark multiple contact pairs in this screen and click **Ignore** for all of them.

- You can select the check box next to one or more pairs that you determine are not duplicates and click **Ignore** to ignore all the checked pairs. Clicking **Ignore** saves both contacts in each pair and then removes each duplicate contact entry. These duplicate entries will not show up in future runs of the batch process unless a future edit makes them duplicates again.
- You can search for specific duplicate contacts. If you do a search, you can use the **Match Type** list to filter the results by exact match, potential match, or all matches.
- You might search for a specific contact's duplicates on this screen and then want to see all the duplicates again. To do so, click **Reset** and then click **Search**.

5. If you do not see any potential duplicate contacts listed, it is possible that the batch process has not run. It could also have run more than once since the last time you viewed the list.

- If you uncheck the **Last Run Only** checkbox, you can see all duplicates that remain from any run of the batch process.
- If that does not work, you need to run the **Duplicate Contacts Finder** batch process. For more information, see “Running the Batch Process Manually” on page 247.

6. For any pair of contacts, if you click the **Review** button, you can compare the two contacts on the **Review Contacts for Merging** screen.

The **Review Contacts for Merging** screen has four actions you can perform on the two contacts you are comparing:

- **Merge** – The contacts are duplicates. After you have completed all the comparisons on all tabs and the data is correct for both versions of the contact, click **Merge** to save the data in one contact. Clicking **Merge** removes the contact that is a duplicate and this duplicate contact entry. This duplicate entry will not show up in future runs of the batch process.
- **Merge Then Edit** – The same as **Merge**, except that the merged contact opens in an editor after ContactManager completes the merge.
- **Ignore** – The contacts are not duplicates. Clicking **Ignore** saves both contacts and removes this duplicate contact entry. This particular duplicate entry will not show up in future runs of the batch process unless a future edit makes them duplicates again.
- **Cancel** – You do not want to decide if these two contacts are or are not duplicates. Clicking **Cancel** preserves this duplicate contact entry for this run of the batch process. However, the next run of the batch process will not pick up these duplicates. To see them in the **Merge Contacts** screens after any subsequent run of the batch process, you uncheck the **Last Run Only** check box, as described previously in step 5.

7. On the **Review Contacts for Merging** screen, there are multiple tabs. Complete your work on all tabs before clicking **Merge** or **Merge Then Edit** at the top of the screen.

- Initially you see the **Contact Detail** tab with four columns. The columns show the field names, the data for the contact to be kept, the data for the contact to be retired, and the data resulting from the merge.
- The **Addresses** tab enables you to choose replacement addresses and to choose more than one address for the merged contact. You can also change the primary address and set the address type for each address. Check **Include** for any address that you want to keep that is not the primary address. If you do not include an address, you must indicate which address duplicates it in the address's **Duplicate Address** list. You can choose **None** from this list if the address is not duplicated by another address.
- The **Related Contacts** tab enables you to choose which related contacts to keep.
- The **EFT Information** tab enables you to choose which electronic funds transfer accounts to keep.
- The **Vendor Data** tab is visible if one of the contacts has a vendor tag. This tab enables you to determine which tags, availability settings, and services to keep.
- If there are any review scores for the contacts, the scores merge and update the next time the AB Contact Score Aggregator batch process runs.

Review scores are a ClaimCenter feature described in “ClaimCenter Service Provider Performance Reviews” on page 215.

8. When the **Updated Contact** column has all the correct data, you can click **Merge** or **Merge Then Edit**.

- **Merge** – Save the contact with the information you have chosen and exit the merge screens for this contact. ContactManager saves the **Kept** contact and retires the **Retired** contact. ContactManager then notifies all integrated core applications that the contact has changed. On receiving this notification, the core applications can change references to the retired contact and make them references to the kept contact. See “Merging Contacts and Notifying Core Applications” on page 251.
- **Merge Then Edit** – Save the contact with the information you have chosen, and then open the **Kept** contact in the editor. ContactManager first saves the **Kept** contact and retires the **Retired** contact. When the editor opens, you see the newly merged data for the **Kept** contact in the editor. You can make further changes and save your changes.

ContactManager notifies all integrated core applications that the contact has changed. On receiving this notification, the core applications can change references to the retired contact and make them references to the kept contact. See “Merging Contacts and Notifying Core Applications” on page 251.

**Note:** If you merge vendor contacts that have vendor review scores, you do not see the updated average scores until the AB Contact Score Aggregator batch job runs. If you use vendor review scores, it is likely that you have scheduled this job to run regularly. If you want to see the updated average scores right away, you can run this batch process as described under “Starting Batch Processes” on page 253.

### Merging Contacts and Notifying Core Applications

When you merge duplicate contacts, as described in the preceding topic, ContactManager keeps one contact and retires the other. As with any contact change, ContactManager then notifies the integrated core applications that a contact has been merged. ContactManager calls the core application implementation of the **ABClientAPI** interface method **mergeContacts** to notify the application of the kept contact and retired contact **AddressBookUID** values.

The application can then update the contact’s **AddressBookUID** with that of the kept contact. Each core application handles the change appropriately for the application. For example, the core application updates local copies of the retired contact to use the **AddressBookUID** of the kept contact. It then retrieves the data for the kept contact from ContactManager.

ContactManager provides two **ABContactAPI** web service methods, **getReplacementAddress** and **getReplacementContact**, that the core application can call to update the contact data by using the **AddressBookUID** of the kept contact.

In the base configuration, the core applications implement the **ABClientAPI** interface in the class **ContactAPI**. You can see how each application implements **mergeContacts** by opening the class in Studio for that application. Each application uses a different path for the class:

- **BillingCenter** – `gw.webservice.bc.bc801.contact.ContactAPI`
- **ClaimCenter** – `gw.webservice.cc.cc800.contact.ContactAPI`
- **BillingCenter** – `gw.webservice.pc.pc800.contact.ContactAPI`

#### See also

- For a description of the method **ABClientAPI.mergeContacts**, see “**ABClientAPI Interface**” on page 282.
- For descriptions of the methods **ABContactAPI.getReplacementAddress** and **ABContactAPI.getReplacementContact**, see “**ABContactAPI Methods**” on page 276.

## Reviewing Pending Changes to Contacts

Pending changes are saved in ContactManager when a ClaimCenter user who does not have `abcreate` or `abedit` permissions creates or edits a vendor contact. For a pending create or pending update to become a new contact or to update a contact, an authorized user must log in to ContactManager and approve it.

### Notes:

- For a pending create, ContactManager first creates the contact and then flags it as Pending Create. Approving the create simply removes the flag. Disapproving the create removes the pending contact.
- For a pending update, ContactManager stores the update information, but does not apply it to the contact until you approve it. If you disapprove the update, this information is deleted and is never applied to the contact.

### To review pending changes to contacts

1. Log in to ContactManager in a role that has permissions to create, update, view, delete, and search for contacts and to view pending contacts. For example, log in as a user with the Contact Manager role, such as the sample user `aapplegate/gw`.

For information on contact permissions and security, see “ContactManager Contact Security” on page 110.

2. Navigate to **Contacts** tab → **Pending Changes**.

3. On the **Pending Changes** screen, click either the **Updates** card to work with pending contact changes or the **Creates** card to work with new pending contacts.

4. Review each pending Update or Create request.

- **Updates** – There is a list of contacts with information on the contact name, the ClaimCenter user who made the change, and the associated claim. When you select a contact, then, under **Entity or Property**, you can navigate to the fields that were changed. For each field, you can see the **Current Value** and the submitted pending change, the **New Value**.
- **Creates** – There is a list of contacts with information on the contact name, the type of contact, the tags, the requesting user, and the claim number. When you select a contact, the contact details display below on the **Basics**, **Addresses**, and **Related Contacts** cards.

5. You can **Approve**, **Reject**, or **Approve Then Edit** the entire pending update or create for a contact. You can also click **Find Duplicates** for a pending create.

- **Reject** – If you reject the pending change or create, you see a screen asking for a reason and a text explanation. A reason is required. After selecting a reason and optionally entering an explanation, click **Reject Change** to complete the rejection.

ContactManager sends the rejection to ClaimCenter, which creates an activity for the user with the information you chose. ClaimCenter also changes the status message for the contact to indicate that the change did not go through.

- **Approve** – If you approve the change, the contact is updated or created, as needed.
- **Approve Then Edit** – If you want to make further changes to the contact data, click **Approve Then Edit**. The contact is updated or created, as needed, and then ContactManager opens the contact in an editor. At this point, the approved data is saved with the contact, and the contact is open in an editor so you can make further changes.

If the contact approval was for a pending update, ContactManager shows you both the old data and the new data in a worksheet below the **Edit** screen.

**Note:** If you cancel the edit or click **Return to Pending Updates**, you are effectively deciding not to edit the contact data. Cancelling the edit is the same as simply approving the contact update. The old data is no longer available on the worksheet, but you can still access it. To see the old data, search for the contact and open its detail screen, and then click the **History** card to see the list of changes. For a particular change, click **Changes** to see the specifics.

- **Find Duplicates** – This button is available for a pending create only. If you want to check for duplicate contacts for the current contact that is pending creation, click **Find Duplicates**. If there are exact or potential matches, you see a list of the contacts in a worksheet. If one of the contacts is a duplicate of the pending create, you can click **Accept** for that contact.

ContactManager keeps the selected duplicate contact and retires the pending contact. ContactManager then sends the same message to ClaimCenter that it sends for any other duplicate contact merge. The message indicates the **AddressBookUID** of the retained contact and the **AddressBookUID** of the retired contact. ClaimCenter uses this information to update the link for its contact to the retained contact's **AddressBookUID**.

**Note:** If you do not check for duplicates and you approve creation of this contact, ContactManager does not automatically check for duplicate contacts for you. The new contact is created. If the new contact is a duplicate of an existing contact, you can make the correction in the **Merge Contacts** screens after the **Duplicate Contacts Finder** batch process runs. See “[Merging Duplicate Contacts](#)” on page 249.

## Starting Batch Processes

To start a batch process manually, you first log in to ContactManager as a user with an administrative role. The role must have the `toolsBatchProcessview` permission.

You then:

1. Press **Alt+Shift+T**.
2. Click the **Server Tools** tab.
3. In the sidebar on the left, click **Batch Process Info**.

For example, the following figure shows the **Batch Process Info** screen with the **Duplicate Contacts Finder** batch process selected:

Batch Process	Description	Action	Last Run	Last Run Status
AB Contact Score Aggregator	Calculates aggregate review scores for ABContacts from submitted ReviewSummaries.	Run Stop Download History		Not available
AB Geocode Writer	Geocoding ABAddresses queue writer.	Run Stop Download History	07/25/2013 04:38:36...	Completed
Duplicate Contacts Finder	Finds new duplicate contacts and creates a browsable list of said contacts.	Run Stop Download History	07/25/2013 04:47:16...	Completed
Phone number normalizer	Performs a normalization of phone numbers contact	Run Stop Download History	07/25/2013 04:46:21...	Completed

The **Batch Process Info** screen enables you to run batch processes manually, without having to wait for the scheduled processes to run. You can start ContactManager batch processes and view information about them, 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, especially during development:

Batch Process	Description
AB Contact Score Aggregator	Processes service provider review scores sent by ClaimCenter. For more information, see “ClaimCenter Service Provider Performance Reviews” on page 215.
AB Geocode Writer	Geocodes addresses. For more information, see “Geocoding and Proximity Searches for Vendor Contacts” on page 99.
Duplicate Contacts Finder	<p>Compares new contacts to the existing contacts in the database to see if there are entries that might be duplicates of the contact. After this batch job runs, all new contacts that have been processed are marked to prevent them from being processed again.</p> <p>Possible duplicate contacts are saved to a list that a user with the appropriate permissions can process in the Merge Duplicates screen. For more information, see “Detecting and Merging Duplicate Contacts” on page 245.</p>

**Note:** These batch process jobs have entries in the file `scheduler-config.xml` that you can uncomment and set to run at specific times. For examples, see:

- “Changing Batch Process Settings” on page 248
- “Step 4: Schedule Geocoding” on page 103.

#### See also

- “Batch Processes and Work Queues” on page 123 in the *System Administration Guide*

## Large Batch Jobs and 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 44 in the *System Administration Guide*.

## Vendor Services Onboarding

By using the **Vendor Services Onboarding** screen, you can add vendor services to your vendor contacts. This process is useful for initially updating your database of contacts that do not yet have vendor services. For example, you might perform this process after upgrading from a previous release to the current version of ContactManager. Or you might perform this process as part of an initial implementation after migrating contacts from your legacy system to the ContactManager database.

The vendor services you associate with your contacts are described at “Vendor Services” on page 181. Vendor services enable you to specify the services provided by vendor contacts in greater detail than is supported by contact subtypes. In ContactManager, after you create your set of vendor services, you can add services to each

contact by editing the contact. However, if you have a large number of contacts that need services added to them, adding services to each contact, one at a time, might not be practical.

You can use vendor services onboarding to process all your vendor contacts and add services to them.

**IMPORTANT:** Before you can use this feature, you must:

- Create and load your vendor services tree. See “Vendor Services” on page 181.
- Ensure that all your contacts are in a current ContactManager database.
  - If you are upgrading from a previous release, you must complete the upgrade to this release before starting the process described in this topic.
  - If you are implementing a new installation of ContactManager, you must have transferred all your legacy contacts to the ContactManager database, preferably as appropriate ABContact subtypes. In the base configuration, vendor services onboarding uses the contact subtype to determine appropriate vendor services to apply. You can configure this process to use other criteria.
- Ensure that all contacts to which you want to add services have a Vendor tag. The standard ContactManager upgrade process can tag all contacts that are vendor subtypes with the Vendor tag. If you have previously worked with vendor contacts in ClaimCenter 7.0, ClaimCenter will have tagged your contacts for you. See “Contact Tags” on page 177.
- Set up your vendor service mappings in the ServiceMappings class. See “Adding Service Map Settings to the ServiceMappings Class” on page 255.
- Optionally define settings for the Vendor Services Load Status column, which you can use to track your work on each contact. See “Configuring the Vendor Services Load Status Column” on page 257.

This topic includes:

- “Initial Configuration for Vendor Services Onboarding” on page 255
- “Working with Vendor Services Onboarding” on page 259
- “Customizing Vendor Services Onboarding” on page 265

## Initial Configuration for Vendor Services Onboarding

Before you can use the **Vendor Services Onboarding** screen, you must set up your vendor service mappings in a Gosu class. Additionally, you might want to configure an optional tracking field supplied in the base configuration, which is saved as the column Vendor Services Load Status in the exported CSV files.

This topic includes:

- “Adding Service Map Settings to the ServiceMappings Class” on page 255
- “Configuring the Vendor Services Load Status Column” on page 257

### Adding Service Map Settings to the ServiceMappings Class

You must edit the ServiceMappings class to define mappings between sets of contact field values and services. ContactManager uses these mappings to apply services to contacts.

To avoid having to stop and restart ContactManager, you can set this class up before starting the vendor services onboarding process in the user interface. Alternatively, you can generate the initial CSV files first, Step 1 in the user interface, to see what the values of the Key column are for each contact. You can then enter the key values and their associated vendor service codes in this class.

**Note:** If you edit the ServiceMappings class after generating the initial CSV files, you must restart ContactManager.

This class defines pairs of mappings between contact keys defined in the ExportImportVendorServicesUtil class and vendor service codes. These mapping pairs determine which vendor services are assigned to a contact in the mapping stage, “Step 2: Mapping Services to Vendor Contacts” on page 260.

To edit this class, open ContactManager Studio and press **Ctrl+N**. Then enter **ServiceMappings** and, after you see the **ServiceMappings** class in the search results, press **Enter**. The **ServiceMappings** class opens in the editor. In this class, you enter your actual key value and service code pairs, as described later in this topic. When you are done, restart ContactManager.

In the base configuration, the **ServiceMappings** class comes with four sample **HashMap String** pairs. Each pair is a **Key** value from the CSV file associated with one or more vendor service codes:

```
"Auto Repair Shop United States" -> "autoothercourtesycar",
"Adjudicator United States" -> "autoappraise, autoadjudicate, propinspectadjudicate"
"Doctor United States" -> "medicalassess",
"Vendor (Company) United States" -> "contrepairgarden, contreparkitchen"
```

With these default mappings, performing the mapping step in the user interface assigns the following services to the following contacts:

- Vendor contacts of subtype **AutoRepairShop** with a primary address in the United States are assigned the service **Provide Courtesy Car**.
- Vendor contacts of subtype **Adjudicator** with a primary address in the United States are assigned the services **Auto - Appraisal, Adjudicate Claim, and Property - Inspection - Adjudicate Claim**.
- Vendor contacts of subtype **Doctor** with a primary address in the United States are assigned the service **Medical - Medical Assessment**.
- Vendor contacts of subtype **CompanyVendor** with a primary address in the United States are assigned the services **Contents - Repair - Garden and Contents - Repair - Kitchen Appliances**.

### Key Value

The first value of the pair is generated for each contact when you initially generate the CSV files. Each CSV file has a **Key** column. The values stored in the **Key** column are defined in the class

**ExportImportVendorServicesUtil** in its **keys** property. In the base configuration, the default key value is a concatenation of the **ABContact** properties **Subtype** and **PrimaryAddress.Country**:

```
private static var keys = {"Subtype", "PrimaryAddress.Country"}
```

In the base configuration, for each contact added to the CSV file, ContactManager reads the values for the contact's **Subtype** and **PrimaryAddress.Country** fields. It then concatenates these values into a single string and populates the **Key** column for that contact. For example, for sample contact European Autoworks located in San Francisco, California, USA, the **Key** column value would be:

```
Auto Repair Shop United States
```

You do not have to use these two **ABContact** fields to generate the **Key** column. In **ExportImportVendorServicesUtil.keys**, you can specify any **ABContact** field or fields that you want to use to map contacts to services. You would then need to set up the **ServiceMappings** definitions to use the actual values in the **Key** columns generated for your CSV files. See the description of the **key** property in “Properties of **ExportImportVendorServicesUtil**” on page 266.

### Vendor Service Codes

The second value of the pair is one or more vendor service codes that you want ContactManager to assign to a contact with that specific key value.

**Note:** To see your vendor service codes, you can export the file **vendorservicetree.xml** as follows:

1. Navigate to **Administration** tab → **Utilities** → **Export Data**.
2. For the **Data to Export** column, select **Vendor Service Tree** and click **Export**.

The base configuration mapping definitions in **ServiceMappings** use vendor service codes from the sample **vendorservicetree.xml** file in *ContactManager/modules/configuration/config/sampledatal*. For example, the default code defined for "Auto Repair Shop United States", "autoothercourtesycar", is the **<code>** value for the **Provide courtesy car** service:

```
<SpecialistService public-id="svc:aut_oth_cou">
<Active>true</Active>
```

```
<Code>autoothercourtesycar</Code>
<Description>Auto - Other - Provide courtesy car</Description>
<Description_L10N_ARRAY>
<Name>Provide courtesy car</Name>
<Name_L10N_ARRAY>
<Parent public-id="svc:aut_oth"/>
</SpecialistService>
```

## Configuring the Vendor Services Load Status Column

In the base configuration, there are five contact data columns defined in `ExportImportVendorServicesUtil` that become the first five columns of the CSV file generated from the **Vendor Services Onboarding** screen. These five columns are Name, LinkID, Address, Key, and Vendor Services Load Status. You can configure different contact data columns as needed, as described in the table of “Properties of `ExportImportVendorServicesUtil`” on page 266.

The first three columns correspond to standard fields on `ABContact`: Name or `FirstName + LastName`, `LinkID`, and `PrimaryAddress`.

The fourth column, Key, is defined in the property `ExportImportVendorServicesUtil.keys`, and is used as part of the vendor service mapping process. See “Key Value” on page 256.

The fifth column, and the subject of this topic, Vendor Services Load Status, might require configuration before you can make use of it. You can use this column to track your progress in assigning vendor services to a contact.

### To configure Vendor Services Load Status

1. Check `ABContact.etx` to see if it has the typekey `VendorServicesLoadStatus`.
  - a. In ContactManager Studio, press `Ctrl+Shift+N` and enter `ABContact`.
  - b. In the search results, double-click `ABContact.etx` to open it in the editor.
  - a. Look for the typekey `VendorServicesLoadStatus`. If it is on this entity, it is likely to be listed at the end.
2. If the typekey does not exist, add it.
  - a. At the top of the tree on the left, right-click entity (extension) `ABContact` and choose `Add new → typekey`.
  - b. Enter the following values:  
`name – VendorServicesLoadStatus`  
`typelist – VendorServicesLoadStatus`  
`nullok – true`
3. If needed for your tracking purposes, extend the `VendorServicesLoadStatus` typelist to have more values than `Final` and `Draft`.
  - a. Navigate to `configuration → config → Metadata → Typelist` and right-click `VendorServicesLoadStatus.tti`.
  - b. Choose `New → Typelist extension`, and then click `OK` in the `Typelist Extension` dialog.
  - c. Studio opens a new `VendorServicesLoadStatus.ttx` file in the directory at `configuration → config → Extensions → Typelist`.
  - d. To add each typecode in the Typecode editor, right-click a row in the panel on the left and choose `New → typecode`. Then enter a code, name, and description for the new typecode.
4. Restart ContactManager to pick up these changes.

### To use Vendor Services Load Status

1. Export your CSV files. See “Step 1: Exporting the Initial Set of CSV Files” on page 259.
2. Open an exported CSV file in Microsoft Excel and select the row for a contact that you want to track.

The value of the Vendor Services Load Status field for the contact by default is initially `null`. It is an empty string, a blank, in the spreadsheet cell.

3. In the Vendor Services Load Status cell for that contact, enter a valid tracking value for the contact, such as `Draft`.
- Valid values are typecodes from the `VendorServicesLoadStatus` typelist, which by default are `null`, `Draft`, and `Final`.
4. Save the file.
5. The next time you import the CSV file, ContactManager copies the value in the Vendor Services Load Status cell and saves it the contact's `VendorServicesLoadStatus` typekey field.

#### To remove the Vendor Services Load Status field

You might not want to use Vendor Services Load Status to track your work on contacts. If not, you can remove the field, as follows.

**Note:** These instructions assume that you are working with the default set of fields. If you have added or removed fields, your setting for `numberOfNonServiceColumns` will differ from the instructions.

1. In ContactManager Studio, press `Ctrl+N` and enter `ExportImportVendorServicesUtil`.
2. In the search results, double-click `ExportImportVendorServicesUtil` to open this class in the editor.
3. Find the property `columnHeader2` and delete the following code from the property definition:  
`, "Vendor Services Load Status"`
4. Comment out the following line of code:  
`private static var vendorServicesLoadStatusIndex = 4`
5. Find the property `numberOfNonServiceColumns` and change its value from 5 to 4.
6. In the method `createAndAddNextRow`, comment out the following line of code:  
`var vendorLoadStatus = contact.VendorServicesLoadStatus != null ? contact.VendorServicesLoadStatus.toString() : ""`
7. In the same method, the `nextRow` property is defined as follows:  
`var nextRow = {contactName, contact.LinkID.remove(",")}, contactAddress, keyValues.substring(0, keyValues.lastIndexOf(" ")), vendorLoadStatus}`
8. Remove the following part of that definition:  
`, vendorLoadStatus`
9. In the method `updateVendorServicesInDB`, comment out the following lines of code:  

```
if (cells.length > vendorServicesLoadStatusIndex) {
    var loadStatus = cells[vendorServicesLoadStatusIndex]
    if (VendorServicesLoadStatus.get(loadStatus) == null && loadStatus != "") {
        var cellList = cells.toList() as ArrayList<String>
        while (cellList.size() < header2.size()) {
            cellList.add("")
        }
        cellList.add(cellList.size(), "Bad VendorServiceLoadStatus value")
        recordErrorMessage("Row with name '" + cells[nameColumnIndex] +
            "' and id '" + cells[idColumnIndex] +
            "' has bad VendorServiceLoadStatus value")
        errors.add(cellList as String[])
    } else {
        vendor.VendorServicesLoadStatus =
            VendorServicesLoadStatus.get(cells[vendorServicesLoadStatusIndex])
    }
}
```
10. Remove the typekey `VendorServicesLoadStatus` from `ABContact.etcx`.
11. Restart ContactManager to pick up these changes.

## Working with Vendor Services Onboarding

Adding vendor services to your vendor contacts is a multi-step process that you perform in the ContactManager user interface by navigating to **Administration tab** → **Utilities** → **Vendor Services Onboarding**.

After completing the configuration steps described in “Initial Configuration for Vendor Services Onboarding” on page 255, you use the **Vendor Services Onboarding** screen to apply vendor services to contacts.

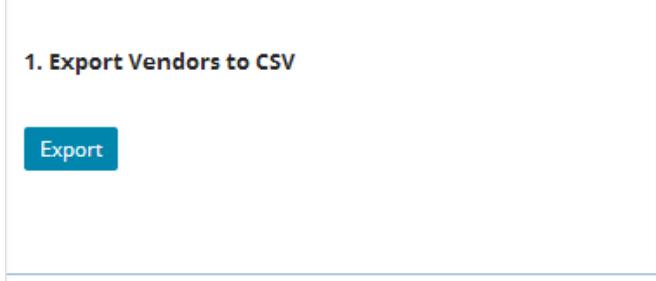
Use the following steps to associate vendor services with contacts:

1. In the ContactManager user interface, export the initial set of CSV files for all your vendor contacts. See “Step 1: Exporting the Initial Set of CSV Files” on page 259.
2. In the ContactManager user interface, run the mapping process to associate vendors with services they provide. See “Step 2: Mapping Services to Vendor Contacts” on page 260.
3. Open the generated CSV files and verify that they have the information you want. Optionally, edit specific contacts to add or remove vendor services, as needed. See “Editing the Generated CSV Files” on page 261.
4. In the ContactManager user interface, import the CSV files containing contacts with their associated services. See “Step 3: Importing Your Mapped Contacts” on page 262.
5. If you receive any errors during the import, you can fix mistakes and re-import the error files containing the contact entries that caused the errors. See “Troubleshooting Errors and Making Corrections” on page 262.

### Step 1: Exporting the Initial Set of CSV Files

The first user interface step in vendor services onboarding is to generate a set of CSV files. Each file has one contact subtype in it, with one row for each vendor contact and columns representing each possible vendor service they might provide. These files are used later to map contacts to vendor services, and then to import those mappings into ContactManager.

1. Navigate to **Administration tab** → **Utilities** → **Vendor Services Onboarding**.
2. In the **1. Export Vendors to CSV** section, click **Export**.



#### 1. Export Vendors to CSV

**Export**

3. ContactManager generates a set of CSV files, each of which contains one subtype of **ABContact**. In the base configuration, this step produces a directory structure and set of files similar to the following:

```
c:\outputfiles\  
  VendorServicesLoad-Mon Mar 03 16.15.50 PST 2014\  
    Adjudicator-0.csv  
    Attorney-0.csv  
    Auto Repair Shop-0.csv  
    Doctor-0.csv  
    Law Firm-0.csv  
    Medical Care Organization-0.csv  
    Vendor (Company)-0.csv
```

- Each CSV file is specific to one subtype of contact, as its name indicates. ContactManager creates a set of these files in a directory that you can specify. Part of the immediate directory name is a timestamp, enabling you to create separate sets of files as you test your output and refine the process.
- The top level directory is defined in `ExportImportVendorServicesUtil.outputFolder`.

- The time-stamped directory is defined in the `timeStampedFolderName` property of the `ExportImportVendorServicesUtil.exportVendors` method.
- In the base configuration, each CSV file contains up to 500 contacts of the subtype specified. The contacts are the data rows of the spreadsheet. The number of contacts per CSV file is defined in `ExportImportVendorServicesUtil.maxRowsPerSpreadsheet`.
- If there are more than 500 contacts of that subtype, ContactManager creates additional CSV files with that subtype name. ContactManager increments the number in the file name by one for each additional file it creates.

For example, one additional file is added for `Doctor-0.csv`. This file is named `Doctor-1.csv`.

- In the base configuration, for each contact, there are five columns of general data, followed by columns for all your services. The default contact data columns are Name, LinkID, Address, Key, and Vendor Services Load Status.

These data column names are defined in `ExportImportVendorServicesUtil.columnHeader2`.

The number of data columns is defined in

`ExportImportVendorServicesUtil.numberOfNonServiceColumns`.

Four of the data columns have database index properties defined in `ExportImportVendorServicesUtil` to improve database access performance. For example, the `LinkID` column has the index property `idColumnIndex`.

The value of the `Key` column is defined in `ExportImportVendorServicesUtil.keys` and is used by the `ServiceMappings` class. See “Adding Service Map Settings to the `ServiceMappings` Class” on page 255.

The Vendor Service Load Status column is described in “Configuring the Vendor Services Load Status Column” on page 257.

4. If the export is successful, you see the following message in the **Process Log** section of the screen. This area is also where you see error messages.

**Export complete. Files located at C:\outputfiles\**

The directory listed in this message depends on your setting in `ExportImportVendorServicesUtil.outputFolder`. In the base configuration, this top level directory is `C:\outputfiles\`.

#### See also

- “Customizing Vendor Services Onboarding” on page 265

### Step 2: Mapping Services to Vendor Contacts

The second step in the user interface process is mapping services to contacts. This step requires that you have previously:

- Created the CSV files containing rows for all your contacts and columns for all your services. See “Step 1: Exporting the Initial Set of CSV Files” on page 259.
- Set up the `ServiceMappings` class with your own key and vendor service code pairs. See “Adding Service Map Settings to the `ServiceMappings` Class” on page 255.

#### To map your vendor services to your vendor contacts:

1. If you previously opened any of the CSV files in an editor like Excel, be sure to close them before you perform this step.
2. Navigate to **Administration tab → Utilities → Vendor Services Onboarding**.

3. In the **2. Map Services to Vendors in CSV** section, enter the path to your time-stamped directory and click **Map**.

**2. Map Services to Vendors in CSV**

File/Directory to Map:  
C:\outputfiles\VendorServicesLoad-Mon Mar 03 15.16.50 PST 2014

Include Subdirectories

**Map**

**Note:** In this screen, you can also enter the path to a specific file. Additionally, you can enter a directory path that has subdirectories and use the **Include Subdirectories** check box to process all subdirectories.

4. If the mapping is successful, you see the message **Mapping Complete** in the **Process Log** section of the screen. This area is also where you see error messages.

After the mapping is complete, the CSV files have `_mapped` added to their names. Any subsequent runs of this step adds another `_mapped` to the file names.

If a file's name does not change when you perform this step, that file did not get mapped. Check to see if the file is open in an editor and close it if it is. You can run the process again without causing any problems in the files that have already been mapped. For example, after you run the process for the second time, the files have the suffix `_mapped_mapped`.

To diagnose other possible errors, see “[Troubleshooting Errors and Making Corrections](#)” on page 262.

### Editing the Generated CSV Files

After generating the CSV files, you can open the mapped CSV files in a spreadsheet program like Microsoft Excel and edit them. For example, you might want to manually edit vendor service assignments for some contacts.

**Note:** When you save the file in Microsoft Excel, you are likely to see a dialog asking if you want to save the file in the current format, comma-delimited CSV. You do. Click **Yes** to save the file in that format.

There are a number of things you need to be careful about when editing these files:

- In the base configuration, the default value for a vendor service that is mapped to a contact is the `String` value "On". For unmapped vendor services, the default value is the empty string, "". These values are defined in the `onValue` and `offValue` properties of `ExportImportVendorServicesUtil`. Be sure to use the values you have defined when manually assigning vendor services to and removing them from a contact.
- In general, avoid editing the data columns for contacts, especially `LinkID` and `Key`. The `Key` field is necessary for vendor service mapping to work properly. The `LinkID` field is necessary for importing and saving the contact data with the correct contact.
- In general, do not manually add or remove vendor service columns in a generated CSV file.

**Note:** If you remove a service column that is not used in the mapping, you see an error message, but the file still gets processed.

- If you are still refining your service tree, wait until the tree is complete before starting the vendor service onboarding process.
- If you change your service tree, you can perform the entire vendor service onboarding process again. For example, you can set up the `ServiceMappings` class to process only your new services. Any vendor services that you have already imported and applied to contacts are preserved for those contacts when `ContactManager` generates the CSV files.
- Do not change any of the header rows, especially the second header row for vendor service columns, which contains vendor service codes.

- While you can delete a contact from a CSV file, which prevents that contact from being processed, do not manually add a contact. Instead, set things up so the file export in Step 1 can add the contact. For example, some contacts do not get added to the CSV file because they do not have vendor tags. You can edit the contacts in ContactManager and add Vendor tags to them, and then regenerate the CSV files.
- Do not delete the generated files, especially if you have edited them, until you have performed the import part of the process.

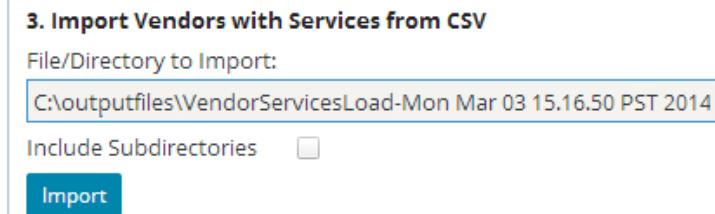
### Step 3: Importing Your Mapped Contacts

When you are satisfied that the mappings are correct for your contacts, you perform the third step in the user interface, importing the contacts with their associated services.

**Note:** If there are any errors during the import, you see error messages in the **Process Log**. ContactManager continues to process the file and saves any unsuccessful imports in a set of CSV error files that you correct and re-import. See “Errors with Importing Contacts” on page 264.

#### To import your mapped contacts:

1. Navigate to Administration tab → Utilities → Vendor Services Onboarding.
2. In the 3. Import Vendors with Services from CSV section, enter the path to your time-stamped directory and click Import.



3. Import Vendors with Services from CSV

File/Directory to Import:

C:\outputfiles\VendorServicesLoad-Mon Mar 03 15.16.50 PST 2014

Include Subdirectories

Import

**Note:** In this screen, you can also enter the path to a specific file. Additionally, you can enter a directory path that has subdirectories and use the **Include Subdirectories** check box to process all subdirectories.

3. If the import is successful, you see the message **Import Complete** in the **Process Log** section of the screen. This area is also where you see error messages.

### Troubleshooting Errors and Making Corrections

ContactManager can encounter errors when exporting files, mapping files, and importing files.

**Note:** Removing a contact row does not cause an error. The contact is simply not processed.

Because you can edit the CSV files, errors most often occur during mapping and import. You see these errors in the **Process Log** section of the **Vendor Service Onboarding** screen, below the sections with the steps.

For example, in **2. Map Services to Vendors in CSV**, you get a series of key value errors.



**Note:** These errors occurred because ContactManager exported some contact subtypes that were not defined in the ServiceMappings class. See “Adding Service Map Settings to the ServiceMappings Class” on page 255. See also “Row errors” on page 264.

### Errors with Files and Directories

Because mapping and importing require that files exist, you can get file or directory errors when you do Step 2 or Step 3 of vendor services onboarding. Some typical errors are:

- Filename is not a valid csv file** – The named file either does not exist (a mapping error) or it has a file extension other than .csv. You might get this error if you renamed a file with the wrong extension, or if you saved a file in the directory that was not a CSV file. ContactManager cannot process the file unless it is a comma-delimited CSV file.
- Directory is not a valid csv file** – You probably entered the wrong directory name, and ContactManager cannot find it.
- IllegalArgumentException** – One cause of this exception is that ContactManager attempted to read a file with a .csv extensions that was not a CSV file. For example, you opened one of the generated CSV files in a text editor and attempted to update it manually. Instead, open these files in a spreadsheet editor and save them as comma-delimited CSV files.

### Errors with Mapping Contacts

The following errors can occur when you map service to vendors in the **2. Map Services to Vendors in CSV** section:

- Key KeyValue found in spreadsheet CSVFileName but not in map** – This contact has a value in its Keys cell that is not defined in the ServiceMappings class.

If you want to map the contact, you must:

1. Add a matching entry to ServiceMappings. See “Adding Service Map Settings to the ServiceMappings Class” on page 255.
2. After adding one or more additional mapping pairs to the ServiceMappings class, you must restart ContactManager and then map your files again. See “Step 2: Mapping Services to Vendor Contacts” on page 260.

**Note:** This condition might not be an error. For example, during initial generation of your CSV files, you generate Key values for Attorney subtypes, but there are no services you want to map to attorneys. In this case, the message is similar to the following:

Key 'Attorney United States' found in spreadsheet 'C:\outputfiles\VendorServicesLoad-Mon Mar 03 15.16.50 PST 2014\Attorney-0.csv' but not in map.

- **Column *VendorServiceID* was improperly added to spreadsheet headers in file: *Filename*** – Each generated vendor service column has two headers, the service name and the service code. There is an invalid service code in the header for one of your vendor service columns. You might either have edited the code value of an existing vendor service or added a vendor service column manually. In general, avoid adding vendor service columns manually and do not edit the header rows. If you did change a code value to an invalid value, you can correct it in the file and remap the file.
- **A service column was deleted from *FileOrDirectoryName*** – If you delete a service column from a CSV file, that service cannot be mapped. If the service was not going to be mapped anyway, no harm is done in deleting the column, and you can ignore this message. If you want the service to be mapped, you must add the column back in or regenerate the file and then map it.
- **IndexOutOfBoundsException** – One cause of this exception is that you removed one of the contact data columns from a CSV file, such as the `LinkID` or the `Key` column. The CSV files are not usable without these contact data columns. You must export your files again to recover from this error.

### Errors with Importing Contacts

If ContactManager encounters any errors during the import process, it displays error messages in the **Process Log** section of the **Vendor Service Onboarding** screen. ContactManager processes all the data it can. If an error prevents a row from being processed, ContactManager copies that row and saves it in a CSV file in the **Errors** directory. It continues processing the rest of the file. If you see errors during import, especially the row errors listed later, check the **Errors** directory for files that have an `error-` prefix added to the original file name. See “To correct row errors” on page 264.

Column errors might not prevent a contact from being processed, but they might prevent a vendor service from being added to a contact. See “Column errors” on page 265.

ContactManager does not alter any rows in the original exported CSV files, so those files will continue to have erroneous data in them unless you change it.

### Row errors

The following row errors can prevent a contact from being imported. ContactManager imports all contacts it can from the file and saves the ones that have errors in an error CSV file. You can edit an error file and re-import it.

- **Row with name: *Name and id: ID has bad id value*** – The `LinkID` field is the unique identifier for a contact.
  - The most likely cause of this error is that someone edited this field in the CSV file.
  - Another cause could be that someone manually added a contact to the file with an invalid `LinkID`. You can correct the `LinkID` value in the error file and then re-import the file.
  - The most severe cause of this error is that the entire `LinkID` column was removed, in which case you see an error message for every contact in the file. In this last case, the file is virtually unusable and must be exported and mapped again.
- **Row with name: *Name and id: ID has bad on/off value*** – At least one of the `String` values indicating if a vendor service applies or does not apply to a contact has been entered incorrectly. These values are defined in the `onValue` and `offValue` properties of `ExportImportVendorServicesUtil`. Be sure to use the defined values when manually assigning vendor services to and removing them from a contact. You can correct the values in the error file and then re-import the file.

### To correct row errors

Edit and correct the error files. After you fix the data in these error files and save the files, you can re-import the error files. After you re-import an error file, ContactManager removes any contacts that were successfully imported from the file. If there are still contacts with errors in the file, ContactManager adds another `error-` prefix to the file name, and saves the file.

If you get more errors after re-importing error files with fixes in them, continue fixing data in the error files and re-importing them until you get no more errors. When there are no more contacts with errors in an error file, ContactManager deletes the file.

For example, ContactManager is unable to process a contact in the file `Auto Repair Shop-0_mapped_mapped.csv`. You see the error in the **Process Log** section while importing the file.

1. The contact is saved in a CSV file in the `Errors` directory. This directory is at the same level as the time-stamped directory you are importing from. The name of the CSV file is `errors-Auto Repair Shop-0_mapped_mapped.csv`. The following directory structure is an example:

```
c:\outputfiles\  
  Errors\  
    errors-Auto Repair Shop-0_mapped_mapped.csv  
    VendorServicesLoad-Mon Mar 03 16.15.50 PST 2014\  
      Adjudicator-0_mapped_mapped.csv  
      Attorney-0_mapped_mapped.csv  
      Auto Repair Shop-0_mapped_mapped.csv  
      Doctor-0_mapped_mapped.csv  
      Law Firm-0_mapped_mapped.csv  
      Medical Care Organization-0_mapped_mapped.csv  
      Vendor (Company)-0_mapped_mapped.csv
```

2. Open `errors-Auto Repair Shop-0_mapped_mapped.csv` in Microsoft Excel, make corrections, and save the file.
3. Navigate to **Administration** tab → **Utilities** → **Vendor Services Onboarding**.
4. In the **3. Import Vendors with Services from CSV** section, enter the path to the error file, and then click **Import**.
5. If there are more errors reported during this re-import, check the `Errors` directory for files with an additional `errors-` prefix, such as `errors-errors-Auto Repair Shop-0_mapped_mapped.csv`. Open these files and fix the errors in the remaining contacts in the files.
6. Continue fixing errors in the error files and re-importing them until there are no more errors reported.

If there are no more errors in an error file, ContactManager deletes the file after you re-import it.

#### Column errors

The following column errors can indicate that a single vendor service was not processed in the CSV file. However, all the other vendor services in the file are processed for all the contacts in the file. These errors do not prevent a contact from being processed, and no error files are saved.

To correct column errors, after the import finishes, you can export the files again. All the data you imported is added to the newly exported files, and the columns are restored. You can then re-import these newly exported files to pick up any column data that was not originally processed.

- **A service column was deleted from `FileOrDirectoryName`** – If you delete a service column from a CSV file, that service cannot be associated with the contacts in this file. This message can also indicate that you deleted a nonessential contact data column, such as `Address`. This error does not prevent ContactManager from processing the file.
- **Column with ID: `VendorServiceID` cannot be added** – Each generated vendor service column has two headers, the service name and the service code. ContactManager was unable to process a vendor service column because it has an invalid code value in its service code header. This error does not prevent ContactManager from processing the file.

## Customizing Vendor Services Onboarding

The Gosu class `gw.exportimport.ExportImportVendorServicesUtil` has optional settings and methods that determine how the Vendor Services Onboarding feature works. In this class, you specify values of properties that determine settings like:

- The directory where your CSV files are stored.
- The number of contacts saved in each CSV file.

- The number and name of the contact data columns that precede the service columns in your spreadsheet.
- The number of query results returned for each database query.
- The Vendor Services Load Status column. See “Configuring the Vendor Services Load Status Column” on page 257.
- The ABContact fields that are used to map a vendor service to a contact. See:
  - The keys property in “Properties of ExportImportVendorServicesUtil” on page 266
  - “Key Value” on page 256

Additionally, you must configure the mappings between contact properties and vendor service codes to enable mapping for your contacts and services. You configure these mappings in the class `ServiceMappings`. See “Adding Service Map Settings to the `ServiceMappings` Class” on page 255

This topic includes:

- “Properties of ExportImportVendorServicesUtil” on page 266
- “Methods of ExportImportVendorServicesUtil” on page 268

## [Properties of ExportImportVendorServicesUtil](#)

The following table lists the properties that you typically customize in `ExportImportVendorServicesUtil`.

Property	Description	Import/Export Step
<code>outputFolder</code>	The main folder that ContactManager uses to store the CSV files it exports and imports. Default value is "C:\\outputfiles\\\".	All three steps
<code>queryPageSize</code>	The number of results returned from a query to the database, by default 100. Affects performance. You can adjust this number if performance is too slow.	All three steps
<code>maxRowsPerSpreadsheet</code>	The number of contacts that can be added to a CSV file, by default, 500. When this limit is reached, ContactManager creates a new file and increments the number in the file name.	<ul style="list-style-type: none"> <li>• “Step 1: Exporting the Initial Set of CSV Files” on page 259</li> </ul>
<code>exportVendors.timeStampedFolderName</code>	The time-stamped subfolder that ContactManager uses to store the CSV files it exports and imports. Default value is <code>outputFolder + "VendorServicesLoad-" + DateUtil.currentDate().toString().replace(".", ".")</code> .	<ul style="list-style-type: none"> <li>• “Step 1: Exporting the Initial Set of CSV Files” on page 259</li> </ul>
	<b>Note:</b> The time stamp is an important part of the folder name that supports iterative passes through the contacts.	

Property	Description	Import/Export Step
columnHeader2	<p>Defines the names of the first set of columns in the spreadsheet, the contact data columns that show information other than vendor service associations. For example, these columns indicate the Name, LinkID, and Key of the contact. These columns precede the columns of services, which are defined by the <code>arraylist</code> in <code>columnHeader1</code>. The actual values of this first set of columns are populated for each contact by the method <code>createAndAddNextRow</code>.</p>	<ul style="list-style-type: none"> <li>“Step 1: Exporting the Initial Set of CSV Files” on page 259</li> </ul>
	<b>Notes:</b>	
	<ul style="list-style-type: none"> <li>The Key column is used to map the contact to one or more services. Its value is defined in the <code>keys</code> property. The specific value for a contact is used during the mapping stage if it is part of a <code>HashMap</code> pair defined in the <code>ServicesMapping</code> class. See “Adding Service Map Settings to the <code>ServiceMappings</code> Class” on page 255.</li> <li>Vendor Services Load Status is a column you can use to indicate if you have previously loaded services for a contact. It can be used for tracking purposes when you are doing multiple reloads. Some configuration is required to use this column. See “Configuring the Vendor Services Load Status Column” on page 257.</li> <li>If you add a new contact data column, also add an <code>index</code> property for it.</li> </ul>	
numberOfNonServiceColumns	<p>The number of columns specified in <code>columnHeader2</code>. If you change the number of contact data columns, also change this value. See also “To remove the Vendor Services Load Status field” on page 258.</p>	<ul style="list-style-type: none"> <li>“Step 1: Exporting the Initial Set of CSV Files” on page 259</li> </ul>
keys	<p>Properties of the contact, saved in the Key column of the CSV file. Default key values are <code>subtype</code> and <code>address.country</code>. These values are used in the vendor mapping step to associate the contact with a vendor, as defined by <code>HashMap</code> pairs in the <code>ServicesMapping</code> class. See “Adding Service Map Settings to the <code>ServiceMappings</code> Class” on page 255.</p>	<ul style="list-style-type: none"> <li>“Step 1: Exporting the Initial Set of CSV Files” on page 259</li> <li>“Step 2: Mapping Services to Vendor Contacts” on page 260</li> </ul>
onValue	<p>The value stored in the spreadsheet for a service that is associated with a specific contact. By default, this value is “On”.</p>	<ul style="list-style-type: none"> <li>“Step 2: Mapping Services to Vendor Contacts” on page 260</li> </ul>
offValue	<p>The value stored in the spreadsheet for a service that is not associated with a specific contact. By default, this value is the empty string, “”.</p>	<ul style="list-style-type: none"> <li>“Step 2: Mapping Services to Vendor Contacts” on page 260</li> </ul>
nameColumnIndex	<p>An index used for mapping and importing the Name column defined in <code>columnHeader2</code>. The index numbers for the <code>columnHeader2</code> columns are zero-relative. This index is index 0.</p>	<ul style="list-style-type: none"> <li>“Step 2: Mapping Services to Vendor Contacts” on page 260</li> <li>“Step 3: Importing Your Mapped Contacts” on page 262</li> </ul>
idColumnIndex	<p>An index used for mapping and importing the LinkID column defined in <code>columnHeader2</code>. The index numbers for the <code>columnHeader2</code> columns are zero-relative. This index is index 1.</p>	<ul style="list-style-type: none"> <li>“Step 2: Mapping Services to Vendor Contacts” on page 260</li> <li>“Step 3: Importing Your Mapped Contacts” on page 262</li> </ul>

Property	Description	Import/Export Step
keyColumnIndex	An index used for mapping and importing the Key column defined in columnHeader2. The index numbers for the columnHeader2 columns are zero-relative. This index is index 3.	<ul style="list-style-type: none"> <li>“Step 2: Mapping Services to Vendor Contacts” on page 260</li> <li>“Step 3: Importing Your Mapped Contacts” on page 262</li> </ul>
vendorServicesLoadStatusIndex	An index used for mapping and importing the Vendor Services Load Status column defined in columnHeader2. The index numbers for the columnHeader2 columns are zero-relative. This index is index 4.	<ul style="list-style-type: none"> <li>“Step 2: Mapping Services to Vendor Contacts” on page 260</li> <li>“Step 3: Importing Your Mapped Contacts” on page 262</li> </ul>

## Methods of ExportImportVendorServicesUtil

The following table lists the methods that you typically customize in `ExportImportVendorServicesUtil`.

Method	Description	Import/Export Step
<code>exportVendors</code>	Does the initial export of vendor contacts and their data and service columns to the CSV files. Creates the directories and files as needed.  This method calls <code>writeCurrentGrid</code> to create new files with the current contact subtype in the name. If you want to modify this file naming convention, create a new method similar to <code>writeCurrentGrid</code> that uses your file naming convention. Then change the code in <code>exportVendors</code> that calls <code>witeCurrentGrid</code> and have it call your new method.	“Step 1: Exporting the Initial Set of CSV Files” on page 259
<code>mapServices</code>	Maps services to contacts as specified. Uses the settings in the <code>ServiceMappings</code> class to do this mapping.	“Step 2: Mapping Services to Vendor Contacts” on page 260
<code>importVendors</code>	Reads the CSV file or files and saves mapped vendor service data with each contact in the file.	“Step 3: Importing Your Mapped Contacts” on page 262
<code>createAndAddNextRow</code>	Adds a contact row to the CSV file. You must edit this method if you want to change any of the contact data columns defined in <code>columnHeader2</code> and <code>numberOfNonServiceColumns</code> .	“Step 1: Exporting the Initial Set of CSV Files” on page 259

# ContactManager Integration Reference

This topic describes ContactManager web services, plugins, high level entities, mapping classes, and messaging events. It also supplies reference information for some of the more important classes and files that enable ContactManager to communicate with the Guidewire core applications.

This topic includes:

- “ContactManager Integration Overview” on page 269
- “ContactManager Entities” on page 270
- “ContactManager Web Services” on page 272
- “ContactManager Messaging Events by Entity” on page 280
- “ABClientAPI Interface” on page 282
- “ContactMapper Class” on page 285
- “ContactManager Plugins” on page 293
- “IFindDuplicatesPlugin Plugin Interface” on page 298
- “ValidateABCContactCreationPlugin Plugin Interface” on page 306
- “Testing Clock Plugin Interface—Only For Non-Production Servers” on page 308

## ContactManager Integration Overview

ContactManager, like the Guidewire core applications, is a complete application built on the Guidewire platform. ContactManager has its own versions of:

- Plugin interfaces
- Web service (SOAP) APIs published from the ContactManager application
- Messaging events
- Destination plugins
- Java API Reference Javadoc
- SOAP API Reference Javadoc

If you have integrated ContactManager with core applications from the InsuranceSuite—ClaimCenter, PolicyCenter, and BillingCenter—they work together to implement centralized contact management.

- In the ClaimCenter application, you can click the **Address Book** tab to search for and view contacts stored in ContactManager. You can add and edit contacts on a claim's **Contacts** screen and in the **New Claim** wizard.
- In PolicyCenter, you can click the **Contact** tab and create new contacts, search for existing contacts, select a recently viewed contact, and change contact information.
- In BillingCenter, you can click **Search → Contacts** to find contacts. You can also add, edit, and delete centrally managed contacts on **Contacts** screens of the **Account** and **Policy** tabs.

The Guidewire core applications send information to ContactManager across the network by using *plugins* to call ContactManager's web services. ContactManager itself has plugins that it uses to call the core applications' web services and to perform internal tasks. For more information, see “ContactManager Plugins” on page 293.

Additionally, each core application implements a **ContactAPI** web service that ContactManager can use to send contact changes to the application. For more information, see “ABClientAPI Interface” on page 282.

To change or add properties to contact-related entities, you can extend the data model in both the Guidewire core applications and ContactManager. Alternatively, you can add tags for any contact, or you can specify services to use with vendor contacts. Whichever approach you take, you must ensure that information flows correctly between the applications. For information on extending the data model and translating properties, see “Extending the Contact Data Model” on page 133.

The following areas of Guidewire core application functionality are not present in any form in ContactManager:

- Notes
- Documents
- Administrative groups
- Activities
- Assignment

There are no ContactManager entities, plugins, or APIs associated with these objects and features.

#### See also

- “Integrating ContactManager with Guidewire Core Applications” on page 45
- “ContactManager Entities” on page 270
- “ContactManager Web Services” on page 272
- “ContactManager Messaging Events by Entity” on page 280
- “ContactManager Plugins” on page 293

## ContactManager Entities

ContactManager has entities that are similar to entities in ClaimCenter, PolicyCenter, and BillingCenter. For example, the ClaimCenter, PolicyCenter, and BillingCenter **Contact** entity has a corresponding ContactManager **ABContact** entity.

**ABContact** is an important entity for ContactManager. It is the primary entity that ContactManager uses to manage contacts. The **ABContact** entity has the subtypes **ABCompany**, **ABPerson**, and **ABPlace**, and each of those three entities has specialized versions (for example, **ABCompany** has a vendor subtype called **ABCompanyVendor**). This subtype hierarchy parallels the **Contact** subtype hierarchy in Guidewire core applications.

**Note:** ContactManager also has entities like **User** and **Group** to support ContactManager's own users and groups. These entities are not involved in integrating with core applications.

Some important ContactManager entities are listed in the following table:

Entity or class	Description
ABContact	The ContactManager version of the Contact entity that stores name, phone number, address, and so on for the contact.
ABContactContact	The ContactManager version of the ContactContact entity that connects contacts that have a relationship.
ABContactTag	An entity connecting an ABContact entity to a tag, a typecode in the ContactTagType typelist. ABContact has an array of ABContactTag entities.
ABContactSpecialistService	An entity connecting an ABContact to a service. The ABContact entity has an array reference to this entity.
Address	An address associated with a contact.
History	An entity that captures the history of actions performed on the contact, such as when it was created and what changes were made to the contact data. Additionally, this object can record which user and application performed the actions.
ReviewSummary	An entity that captures a summary of a review's information, passed from ClaimCenter. For more information, see "ContactManager Performance Review Data Model" on page 224.

#### See also

- For information on the ABContact hierarchy and entity relationships, see "ABContact Data Model" on page 134.
- For information on the Contact hierarchy and entity relationships, see "Contact Data Model" on page 135.

## ContactManager Link IDs and Comparison to Other IDs

ContactManager entities that implement the ABLinkable delegate, such as ABContactTag, ABContactAddress, and ABContact and its subentities, have a LinkID property. This property uniquely identifies an ABLinkable entity instance for integration use, such as with Guidewire core applications. It is similar to the PublicID property in Guidewire core applications, which those applications can use as a primary key value for entities in external systems. However, because ContactManager is the system of record for contacts across the core applications, ContactManager must ensure that a contact or address be uniquely identifiable across all Guidewire applications. Therefore, unlike a PublicID, a LinkID cannot be changed.

If the core application does not specify an External\_UniqueID when it calls ContactManager to create a new contact, ContactManager creates the LinkID for a new entity. If the core application does specify the unique ID when it sends a create request, ContactManager populates the LinkID with the External\_UniqueID specified in the XmlBackedInstance data. See the `createContact` method in the table at "ABContactAPI Methods" on page 276.

ContactManager passes the LinkID back to the core applications. A core application can use the return value either to identify the local version of the contact already created or to populate the equivalent AddressBookUID property.

**Note:** The unique ID passed to ContactManager for any ABLinkable entity in the ABContact graph might already exist on an entity of that type. If ContactManager detects that one of these entities, including retired entities, already use this ID, ContactManager throws a `DuplicateKeyException` for the call to `createContact`. In this case, the contact is not created in ContactManager. It is up to the calling application to recover from this state, either by providing a new unique ID or allowing ContactManager to create a unique ID.

The `AddressBookUID` property is the core application version of the `ContactManager LinkID` property. `AddressBookUID` properties and `LinkID` properties are mapped in the `ContactMapper` classes described at “`ContactMapper Class`” on page 285.

For specific examples of mapping between `AddressBookUID` and `LinkID`, see:

- “`Mapping Fields of a ContactManager Contact`” on page 286
- “`Mapping Fields of a Core Application Contact`” on page 289

For more information about the `PublicID` property, see “`Public IDs and Integration Code`” on page 27 in the *Integration Guide*.

The following table compares the various types of IDs related to contacts:

Type of ID	Description
<code>LinkID</code>	The name for the internal ID that <code>ContactManager</code> uses for each <code>ABContact</code> entity and subentity
<code>PublicID</code>	The standard Guidewire public record ID that can be changed as needed. For more information on these IDs, see “ <code>Public IDs and Integration Code</code> ” on page 27 in the <i>Integration Guide</i> .
<code>AddressBookUID</code>	In Guidewire core applications, this property of the <code>Contact</code> entity is the internal ID for a contact entity that is stored in <code>ContactManager</code> . If you are using a Guidewire core application and <code>ContactManager</code> together, an address book UID and a link ID have the same value. If you integrate with a different external contact management system, the address book UID has the same value as the internal ID of an object in that external application.

## ContactManager Web Services

Web services provide a language-neutral, platform-neutral mechanism for invoking actions or requesting data from other applications across a network. Guidewire applications provide web services that are intended to be used both by other Guidewire applications and by external applications. You can also write your own web services. For more information on Guidewire web services, see “`Web Services Introduction`” on page 31 in the *Integration Guide*.

**IMPORTANT** `ContactManager` has two types of web services, `WS-I` and `RPCE`. `RPCE` is an older style of web service and in most cases has been deprecated. If there is a `ContactManager` web service that replaces `RPCE`, you are strongly encouraged to move to that web service. For more information, see “`RPCE Web Services Deprecated in 8.0`” on page 49 in the *New and Changed Guide*.

### Web Services Provided by `ContactManager`

`ContactManager` provides the following web services. The first one, `ABContactAPI`, is the primary web service used by Guidewire core applications to communicate with `ContactManager`. The second one, `ABVendorEvaluationAPI`, supports vendor evaluations. The remaining web services are standard services available in all Guidewire applications.

For a complete list of web services, see “Reference of All Built-in Web Services” on page 33 in the *Integration Guide*

Web Service	Description
ABContactAPI	The primary web service used by Guidewire core applications to search for, create, update, and delete contacts. This WS-I compliant web service's WSDL is retrieved by every Guidewire core application to make it available to the application plugin that communicates with ContactManager. The class package is <code>gw.webservice.ab.ab801.abcontactapi</code> . See “ABContactAPI Web Service” on page 275.
ABVendorEvaluationAPI	<p>A WS-I compliant web service that enables ClaimCenter to send and receive review summary information. The class package is <code>gw.webservice.ab.ab800.abvendorevaluationapi</code>.</p> <p>For more information on vendor contact performance reviews, see “ClaimCenter Service Provider Performance Reviews” on page 215.</p>
ImportToolsAPI	Imports administrative data from an XML file. Use this web service only with administrative database tables, such as entities of type User. The system does not perform complete data validation tests on any other type of imported data. See “Importing Administrative Data” on page 145 in the <i>Integration Guide</i> .
LoginAPI	WS-I authentication happens with each API call. However, if you want to explicitly test specific authentication credentials in your web service client code, ContactManager publishes the built-in LoginAPI web service. Call this web service's <code>Login</code> method, which takes a user name as a <code>String</code> and a password as a <code>String</code> . If authentication fails, the API throws an exception. You can also use LoginAPI to purposely leave a user session open for logging purposes. See “Login WS-I Authentication Confirmation” on page 65 in the <i>Integration Guide</i> .
MaintenanceToolsAPI	Provides a set of tools that start and manage various background processes. The methods of this web service are available only when the server run level is <code>maintenance</code> or higher. See “Maintenance Web Services” on page 146 in the <i>Integration Guide</i> .
MessagingToolsAPI	Provides messaging methods for managing the messaging system remotely for message acknowledgements error recovery. The methods of this web service are available only when the server run level is <code>multiuser</code> . See “Web Services for Handling Messaging Errors” on page 356 in the <i>Integration Guide</i> .
ProfilerAPI	Sends information to the built-in system profiler. See “Profiling Web Services” on page 149 in the <i>Integration Guide</i> .
SystemToolsAPI	<p>Provides a set of tools that are always available, even if the server is set to <code>DBMaintenance</code> run level. For servers in clusters, system tools API methods execute only on the server that receives the request. Some uses of this web service include:</p> <ul style="list-style-type: none"> <li>• Getting the version of the server, including application version and schema version</li> <li>• Checking the consistency of the underlying physical database</li> <li>• Getting and setting the run level</li> </ul> <p>See “System Tools Web Services” on page 147 in the <i>Integration Guide</i>.</p>
TableImportAPI	Provides a table-based import interface for high volume data import. This web service is typically used for large-scale data conversions, particularly for migrating records from a legacy system into ContactManager prior to bringing ContactManager into production. See “Importing from Database Staging Tables” on page 415 in the <i>Integration Guide</i> .
TypeListToolsAPI	Provides tools for getting the list of valid typecodes for a typelist and for mapping between ContactManager internal codes and codes in external systems. See “Mapping Typecodes to External System Codes” on page 143 in the <i>Integration Guide</i> .

Web Service	Description
WorkflowAPI	Performs various actions on a workflow, such as suspending and resuming workflows and invoking workflow triggers. See “Workflow Web Services” on page 149 in the <i>Integration Guide</i> .
ZoneImportAPI	Imports geographic zone data from a comma separated value (CSV) file into a staging table, in preparation for loading zone data into the operational table. See “Importing from Database Staging Tables” on page 415 in the <i>Integration Guide</i> .

## Support Classes for ContactManager Web Services

ContactManager provides a number of Gosu classes that are used by its web services. The following classes are ones that you can edit to support your extensions to ContactManager, such as support for new search criteria.

**Note:** The ABContactAPI methods referenced in the following table are described in “ABContactAPI Methods” on page 276.

Gosu Class	Description
ABContactAPIFindDuplicatesResult	A Gosu class used to define duplicate ABContact entities found by the ABContactAPI.findDuplicates method. You can edit this class to add your contact extensions so they can be returned by the findDuplicates method.
ABContactAPIPendingContactChange	An instance of this Gosu class is the second parameter to the method ABContactAPI.createContactPendingApproval. This class contains the metadata for the change that ContactManager will send back to the calling application if the change is rejected. The calling application can then notify the user of the rejection.
ABContactAPIProximitySearchParameters	A Gosu class that defines the search parameters for a proximity search. You can modify this class. For general information on performing proximity searches, see “Geocoding and Proximity Searches for Vendor Contacts” on page 99.
ABContactAPISearchCriteria	A Gosu class that specifies the contact search criteria that the Guidewire core applications can use. The class package is gw.webservice.ab.ab801.abcontactapi. This class is used by the web service ABContactAPI. You can edit this class to add new search criteria for contacts, as described in “Step B: Adding Search Support in ContactManager for Guidewire Core Applications” on page 89.
ABContactAPISearchResult	A Gosu class that specifies the fields included in the contact search results that ContactManager sends back to the Guidewire core application. The class package is gw.webservice.ab.ab801.abcontactapi. This class is used by the web service ABContactAPI. For more information, see “Step B: Adding Search Support in ContactManager for Guidewire Core Applications” on page 89.
ABContactAPISpecialistService	A Gosu class that defines a SpecialistService entity to be returned by the method ABContactAPI.getSpecialistService. You can edit this class if you have made extensions to how Services work.
ABVendorEvaluationAPIReviewSummary	A Gosu class that enables ClaimCenter to update the review summary of a vendor evaluation with additional, new information. The class package is gw.webservice.ab.ab800.abvendorevaluationapi. This class supports the web service ABVendorEvaluationAPI.
AddressInfo	A Gosu class that defines Address objects to be passed in ABContactAPI method calls. If you have extended the Address entity, you can edit this class to add your extensions, such as new address fields.

There are additional read-only classes that support this web service in the class package `gw.webservice.ab.ab801.abcontactapi`. You can navigate to these read-only classes in Studio as follows:

1. Open ContactManager Studio.
2. In the Project window, navigate to External Libraries → <ContactManager SDK>.
3. In `ab-1.0-SNAPSHOT.jar`, navigate to `gw.webservice.ab.ab801.abcontactapi`.

## ABContactAPI Web Service

The ABContactAPI web service is the primary web service used by Guidewire core applications to search for, create, update, and delete contacts. This web service also supports specifying services in search criteria used in searches for contacts. This WS-I compliant web service's WSDL is retrieved by every Guidewire core application to make it available to the application's plugin that communicates with ContactManager.

- In ContactManager, to see the source code for ABContactAPI, open Studio. In the Project window, press `Ctrl+N` twice to include non-project classes and enter ABContactAPI. In the search results, double-click `ABContactAPI (gw.webservice.ab.ab801.abcontactapi)` to open this class in the editor.
- In ClaimCenter, PolicyCenter, and BillingCenter, the WSDL for ABContactAPI is in the same web collection for all three applications. Open Studio and then open the Project window. Navigate to configuration → gsrc and then to `wsi/remote/gw/webservice/ab/ab801.wsc`. Double-click `ab801.wsc` to open the editor.

You can use this web service to load data from other data sources, to query contacts, to query services, or to update contacts. For example, you could write a command-line tool that enables an external system to add new contacts based on new business data from another part of the company.

## ABContactAPI and Typelists

The ABContactAPI web service exposes the typecodes in contact-related typelists as strings, which simplifies the code needed to access these typecodes from external or Guidewire core applications. For example, states are defined in the typelist State. The following annotation in ABContactAPI exposes this typelist as a string:

```
@WsiExposeEnumAsString(typekey.State)
```

Exposing this typelist as a string makes it possible for a Guidewire core application to do a simple assignment like the following in the ClaimCenter ContactSearchMapper class:

```
searchCriteriaInfo.ServiceState = searchCriteria.ServiceState.Code
```

## ABContactAPI Methods

The ABContactAPI web service provides the following methods.

Method	Parameters	Description
createContact	abContactXML – Contact information in XmlBackedInstance format.	<p>Creates a new contact and returns an AddressBookUIDContainer containing IDs for the Contact and child objects.</p> <p>Contact information is expected to be in XmlBackedInstance format.</p> <p>If the abContactXML parameter includes an External_UniqueID field and that field has a value, ContactManager uses this value to populate the LinkID field. In this manner, the calling application specifies the ultimate value of the LinkID. If the External_UniqueID field is missing or is null, ContactManager generates its own LinkID and passes it back to the calling application.</p> <p>ContactManager determines if the unique ID passed to ContactManager for any ABLINKable entity in the ABContact graph already exists on an entity of that type, including retired entities. If so, ContactManager throws a DuplicateKeyException for the call to createContact. In this case, the contact is not created in ContactManager. It is up to the calling application to recover from this state, either by providing a new unique ID or allowing ContactManager to create a unique ID.</p> <p>Calls ValidateABContactCreationPlugin to ensure that there is enough data to create the contact. If not, the method returns RequiredFieldException to the calling application.</p> <p>If there is enough data, the method creates a new ABContact of the subtype specified by abContactXML. The method then populates the new entity with data it retrieves by calling ContactIntegrationMapper.populateABContactFromXML.</p>
createContactPendingApproval	abContactXML – Contact information in XmlBackedInstance format. updateContext – User, entity, and application information sent by core application. An instance of ABContactAPIPendingContactChange.	<p>Creates a new contact of the type specified and sets its status to PENDING_APPROVAL. Returns an AddressBookUIDContainer containing IDs for the Contact and child objects and the update context and transaction ID.</p> <p>This method is called by the core application because the core application user creating the contact does not have permission to create a contact.</p> <p>Contact information is expected to be in XmlBackedInstance format.</p> <p>Calls ValidateABContactCreationPlugin to ensure that there is enough data to create the contact. If not, the method returns RequiredFieldException to the calling application.</p> <p>If there is enough data and no other exceptions are thrown, the method creates a new ABContact of the subtype specified by abContactXML. The method then populates the new entity with data it retrieves by calling ContactIntegrationMapper.populateABContactFromXML and sets its status to PENDING_APPROVAL.</p>

Method	Parameters	Description
findDuplicates	abContactXML – XmlBackedInstance that contains the contact data for which duplicates are being found.  abContactAPISearchSpec – Specifies how the search is to be returned.	Finds contacts that match the specified contact.  Returns an ABContactAPIFindDuplicatesResultContainer containing summary information about each match.
getReplacementAddress	addressLinkID – linkID of an address that has been replaced because of a merge	Gets the address that has replaced the address passed in the parameter. An address can be replaced as a part of a merge.  Returns the linkID of the address that replaces the address denoted by the argument.
getReplacementContact	contactLinkID – linkID of a contact that has been replaced because of a merge	Gets the contact that has replaced the contact passed in the parameter. A contact can be replaced as a result of a merge.  Returns the linkID of the contact that replaces the contact denoted by the argument, or null if the contact has not been replaced by another contact.
getSpecialistServices	contactLinkID – linkID of the contact that has specialist services	Gets the specialist services associated with the contact passed in the parameter.  Returns an array of ABContactAPISpecialistService objects, or null if the contact has no specialist services.
removeContact	abContactXML – XmlBackedInstance that contains the linkID of the contact to be removed and miscellaneous information.	Removes the contact with the matching linkID if the contact is not being used by any remote application. If a remote application is not running when this method runs, the contact is assumed to be in use and is not removed.  Returns a boolean indicating whether the contact was successfully removed.
retrieveContact	linkID – ID uniquely associated with this contact. In a core application, this value is in the AddressBookUID.	Retrieves information about the contact uniquely specified by the linkID.  Returns contact information in XmlBackedInstance format.
retrieveRelatedContacts	linkID – ID uniquely associated with this contact  relationshipTypes – Array of ContactBidiRel relationship types of the related contacts to return information on. If null, then no restriction is enforced on the related contacts returned.	Retrieves information about the contact's related contacts.  Returns a RelatedContactInfoContainer containing information about the contact's related contacts.
searchContact	abContactAPISearchCriteria – Criteria for the search. This parameter must not be null.  abContactAPISearchSpec – Specifies how the results are to be returned.	Searches for all contacts that match the given search criteria.  Return an ABContactAPISearchResultContainer object containing the search results.

Method	Parameters	Description
updateContact	abContactXML – Contact information in XmlBackedInstance format.	<p>Updates an existing contact and returns an AddressBookUIDContainer object containing IDs for the Contact and child objects.</p> <p>Contact information is expected to be in XmlBackedInstance format.</p> <p>An existing ABContact is selected based on the abContactXML.LinkID. If none is found, the method throws BadIdentifierException.</p> <p>Then, origValue attributes for all fields and foreign keys being updated are checked against those same values in the selected ABContact. If discrepancies are found, the method throws EntityStateException. The purpose is to manage versioning of contact-related changes across multiple client applications.</p> <p>If no exceptions have yet been thrown, the method calls ContactIntegrationMapper.populateABContactFromXML to update the data for local entities from abContactXML.</p>
updateContactPendingApproval	abContactXML – Contact information in XmlBackedInstance format. updateContext – User, entity, and application information sent by core application.	<p>Submits for approval an update to an existing contact that is pending until approved.</p> <p>The core application calling this method has determined that the user updating the contact does not have permission to do so.</p> <p>Contact information is expected to be in XmlBackedInstance format.</p> <p>If no existing ABContact can be found based on the abContactXML.LinkID, the method throws BadIdentifierException.</p> <p>If an ABContact entity is found with this LinkID, this method creates a PendingUpdate entity for the contact.</p>
validateCreateContact	abContactXML – Contact information in XmlBackedInstance format.	<p>Determines if the specified contact can be created.</p> <p>Calls ValidateABContactCreationPlugin.validateCreate to see if abContactXML has the minimum data required to create an ABContact entity.</p> <p>Returns an ABContactAPIValidateCreateContactResult object indicating whether validation passed and, if validation failed, an error message. For more information, see “ValidateABContactCreationPlugin Plugin Interface” on page 306.</p>

## Retrieving Contacts and Their Relationships

If you want to integrate ContactManager with an application other than a Guidewire core application, you can write custom web services to do so. For example, to retrieve properties from contact records, you can write a custom web service (SOAP API) that extracts the subset of entity contact information that you want. Using an AddressBookUID, the API could extract properties from an ABContact record and return it from ContactManager to the SOAP client.

For typical cases, return only certain properties or subobjects, such as the properties needed to display or edit the record. Design your integration point accordingly to return only the necessary data, which reduces bandwidth and server resources.

In some cases, you might want to create new entities or classes to encapsulate your custom data.

If you are retrieving contact records, you can optionally retrieve related contacts, as you can do with ClaimCenter. Related contacts are referred to generally in ClaimCenter and ContactManager as *relationships*. For example, to retrieve a contact's parent or guardian or employer contact information, specify one of these types of relationships. Finally, design web services for each integration point to return that information if it exists.

To use relationship retrieval, you must understand the difference between *source* and *target* relationships. These terms are ways of using the relationship identification codes for relationships such as *parent/guardian* and *employer*, but also specify the directionality of the relationship.

Relationship codes in Gosu are defined in the `ContactRel` typelist in Studio. For example, this Gosu enumeration includes relationship code `TC_employer` as a reference to the employer typecode in the typelist. Suppose you want the record for someone named John Smith. If you want to return John Smith's employers, specify `TC_employer` as a *target relationship*. If you want to return John Smith's employees, specify `TC_employer` as a *source relationship*.

A *target relationship* describes the relationship if the `ContactRel` typecode description fits it into the sentence with the following structure:

“I want to retrieve a contact and the \_\_\_\_\_ of the contact, if any is found.”

For example, for the employee relationship (`employer`), the sentence would be:

“I want to retrieve a contact and the **employer** of the contact, if any is found.”

If you want your request to specify the opposite of that relationship, such as employee instead of employer, specify the relationship as a *source relationship*.

While retrieving a contact record, you might want both directions of relationships. For example, you might want to simultaneously retrieve all employees of a company and the company's primary contact. Use the `ContactBidiRel.EMPLOYER` source relationship and the `ContactBidiRel.PRIMARYCONTACT` target relationship.

---

**IMPORTANT** If extracting contact information by using custom SOAP APIs, write your API to return only the required properties and the required relationships. If you return an entire `ABContact` or unneeded related contacts, large amounts of data created by SOAP serialization can trigger server or client memory problems.

---

## ABVendorEvaluationAPI Web Service

This WS-I compliant web service provides methods used by ClaimCenter to communicate with ContactManager about service provider performance review information.

Service provider performance reviews are described at “ClaimCenter Service Provider Performance Reviews” on page 215.

This web service provides the following methods:

Method	Parameters	Description
addNewReviewSummary	reviewInfo – The new review summary, an ABVendorEvaluationAPIReviewSummary object. The review summary sent to ContactManager must not have a LinkID set.	Adds a new review summary based on the ABVendorEvaluationAPIReviewSummary object passed in.  Returns the created review summary, an ABVendorEvaluationAPIReviewSummary object, complete with LinkID.
deleteReviewSummary	linkID – The LinkID of the review summary as a String.	Deletes the review summary that has the passed-in LinkID.  Returns a Boolean indicating if the review summary was successfully deleted
updateReviewScoresForContact	linkID – The LinkID of the associated vendor contact, as a String.	Updates the scores for all the reviews for the contact whose LinkID is passed in.  Returns an int indicating the current score.

## ContactManager Messaging Events by Entity

ContactManager sends messages to integrated Guidewire core applications after changes to certain entities, such as adding or deleting a contact or changing data for a contact. These changes trigger events, which trigger code that sends messages. For example, a user of a core application changes the address of a contact stored in ContactManager. ContactManager receives that change and updates the contact, triggering an event, ABContactChanged. This event triggers event message rules that cause ContactManager to send the contact change to the integrated core applications.

- For information on the messaging system, see “Messaging and Events” on page 299 in the *Integration Guide*.
- For information on ContactManager event messaging rules, see “EventMessage Rule Sets” on page 208.

You can see the messaging events that are generated for each entity in the Data Dictionary. To build the data dictionary for your system, open a command prompt and navigate to ContactManager/bin, and then enter the command gwab regen-dictionary. The command builds the security and data dictionaries. It saves the Data Dictionary in *ContactManager/build/dictionary/data/datamodel1.html*.

The following table lists the messaging events for each ContactManager entity for which ContactManager sends messages:

Entity	Events	Description
ABContact	ABContactAdded ABContactChanged ABContactRemoved	Contact entities exist only as subtypes of ABContact, such as ABPerson. Those subtypes generate standard A/C/R events for root entity ABContact.
	ABContactPendingChangeRejected	An ABContact pending change request has been rejected.
	ABContactResync	An ABContact has been resynchronized. Re-send all related messages to external systems as appropriate.
		<b>See also</b> <ul style="list-style-type: none"> <li>• “Message Ordering and Multi-Threaded Sending” on page 334 in the <i>Integration Guide</i></li> <li>• “Resynchronizing Messages for a Primary Object” on page 350 in the <i>Integration Guide</i>.</li> </ul>
ABAdjudicator	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.

Entity	Events	Description
ABAttorney	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABAutoRepairShop	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABAutoTowingAgcy	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABCompany	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABCompanyVendor	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABDoctor	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABLawFirm	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABLegalVenue	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejectedAB ContactResync	Standard A/C/R events for root entity ABContact.
ABMedicalCareOrg	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABPerson	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABPersonVendor	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABPlace	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABPolicyCompany	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.

Entity	Events	Description
ABPolicyPerson	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.
ABUserContact	ABContactAdded ABContactChanged ABContactRemoved ABContactPendingChangeRejected ABContactResync	Standard A/C/R events for root entity ABContact.

## ABContact Message Safe Ordering

ContactManager supports safe ordering of **Message** entity instances associated with **ABContact** entity instances. Messages associated with an **ABContact** instance are sent ordered by **ABcontact** instance for each messaging destination. Safe ordering allows only one message per contact-destination pair to be *in flight*—sent but not acknowledged at any given time. Messages of this type are referred to as *safe-ordered messages*.

If an event generates two safe-ordered messages related to one **ABContact** entity instance, ContactManager sends the first message immediately but does not send the second message. After ContactManager receives an acknowledgement from the destination about the first message, it sends the second message.

Because ContactManager manages events for many **ABContact** entity instances, a Guidewire core application can have many messages in flight at a time, even multiple messages to one destination. However, there can be only one message in flight for each contact-destination pair.

For more information about safe ordering as it relates to claims, see “Message Ordering and Multi-Threaded Sending” on page 334 in the *Integration Guide*.

## ABContact Message Resynchronization

ContactManager supports resynchronizing a contact (the **ABContact** entity) with an external system. If an **ABContact** entity resynchronizes, a destination could listen for the **ABContactResync** event. If that event triggers, the Event Fired rules that process it could resend important messages to external systems for this entity to synchronize with the external system.

For more information about message resynchronizing as it relates to claims, see “Resynchronizing Messages for a Primary Object” on page 350 in the *Integration Guide*.

## ABCClientAPI Interface

ContactManager requires that each Guidewire core application implement the **ABCClientAPI** interface and expose it as a web service.

- ClaimCenter implements this interface in `gw.webservice.cc.cc800.contact.ContactAPI`.
- PolicyCenter implements this interface in `gw.webservice.pc.pc800.contact.ContactAPI`.
- BillingCenter implements this interface in `gw.webservice.bc.bc800.contact.ContactAPI`.

See “Contact Web Service APIs” on page 508 in the *Integration Guide*.

ContactManager then calls the web service methods in each application when it needs to broadcast changes in contacts to the applications. Each Guidewire core application can determine what to do in response to the call from ContactManager.

This ContactManager interface is in the package `gw.webservice.contactapi`. To access it, do a class search that includes non-project classes by pressing **Ctrl+N** twice and entering **ABCClientAPI**. Then double-click the class in the search results.

This interface provides the following method signatures:

Method	Parameters	Description
isContactDeletable	addressBookUID – The address book unique ID of the contact. For ContactManager, this is the linkID.	Return true either if the contact associated with the AddressBookUID can be deleted or if no contact is associated with AddressBookUID. Return false if the contact cannot be deleted.
mergeContacts	keptContactABUID – The addressBookUID of the contact to be kept.  deletedContactABUID – The addressBookUID of the contact to be deleted.	Merge two contacts that were merged in ContactManager. ContactManager does not send contact data with this method call. It is up to the Guidewire core application to retrieve the data for the kept contact.
pendingCreateApproved	context – An ABCClientAPIPendingChangeContext object providing information on the user requesting this change.	Notifies the client system that a pending contact creation it submitted has been approved by ContactManager. The client application can use the information in the context parameter to inform the user who submitted the request that the contact was created. Additionally, the core application can update the sync status of the contact and post an appropriate message.
pendingUpdateApproved	context – An ABCClientAPIPendingChangeContext object providing information on the user requesting this change.	Notifies the client system that a pending update it submitted has been approved by ContactManager. The client application can use the information in the context parameter to inform the user who submitted the change that the change was approved. Additionally, the core application can update the sync status of the contact and post an appropriate message.
pendingCreateRejected	context – An ABCClientAPIPendingChangeContext object providing information on the user requesting this change.	Notifies the client system that a pending contact creation it submitted has been rejected by ContactManager. The client application can use the information in the context parameter to inform the user who submitted the contact to be created that the creation was rejected. Additionally, the core application can update the sync status of the contact and post an appropriate message.
pendingUpdateRejected	context – An ABCClientAPIPendingChangeContext object providing information on the user requesting this change.	Notifies the client system that a pending update it submitted has been rejected by ContactManager. The client application can use the information in the context parameter to inform the user who submitted the change that the change was rejected. Additionally, the core application can update the sync status of the contact and post an appropriate message.
removeContact	addressBookUID – Unique ID of a contact that has been removed in ContactManager.	The contact with this AddressBookUID has been removed in ContactManager. This call does not require that the contact be removed by the Guidewire core application. For example, if the contact is in use by an account, PolicyCenter does not retire it.
updateContact	contactXML – XmlBackedInstance that contains the addressBookUID of the contact and the updates to be made.	Data for this contact has changed in ContactManager. Update the contact by using the data in contactXML.

## Deleting Contacts in ClaimCenter

As described in “Synchronizing ClaimCenter and ContactManager Contacts” on page 200, ClaimCenter, unlike the other core applications, can have multiple local instances of any contact, an instance for each claim. Therefore, ClaimCenter might have a lot of work to do in deleting a contact. ClaimCenter has to check every local instance of the contact. For each local instance, ClaimCenter must ensure that there are no entities that reference

the instance that would prevent it from being deleted. For example, if there are any claims that have a contact as an involved party, the contact cannot be deleted.

## ClaimCenter ContactAPI Web Service Methods for Removing Contacts

The ClaimCenter implementation of `ABCClientAPI` is the web service `gw.webservice.cc.cc800.contact.ContactAPI`. This web service provides implementations of all the methods, two of which are used by `ContactManager` in requesting that ClaimCenter delete a contact:

- `isContactDeletable(addressBookUID : String)` – If there are less than ten local instances of the contact, this method finds all the contacts linked to the `addressBookUID`. It then calls the `ContactRetireHelper.retireContact(Contact)` method on each one and returns `true` if they can all be successfully retired. If any contact cannot be successfully retired, the method returns `false` to indicate that the contact cannot be deleted. If there are more than ten contacts, the method returns `false`, and then it generates work items to check all the local contacts as part of the work queue. The work queue calls `ContactRetireHelper.retireContact` to retire each contact instance.
- `removeContact(addressBookUID : String)` – Calls `ContactRetireHelper.retireContact` for each ClaimCenter contact found with the `addressBookUID`.

## ClaimCenter Classes for Removing Contacts

ClaimCenter provides two classes you can use in configuring how ClaimCenter handles any data model extensions you have added that might affect the removal, or *retiring*, of contacts.:

- `IRetireContactPlugin` – This plugin interface has a method to return a set of safe properties. These properties represent foreign key links to a contact that can be retired without any further checking when a contact is retired. You can add to the list any extension foreign keys that you deem safe. You can use the implementation of this interface, `gw.plugin.contact.RetireContactPlugin`, to add properties to the safe list.
- `ContactRetireBean` – This interface can be implemented by entities that have an unsafe foreign key link to a contact. The implementation can determine if an entity instance will prevent a contact instance from being retired. If not, the contact can be retired, and then the implementation can also determine if any other contacts or entities can be retired along with the contact. See the class `gw.api.contact.ContactContactRetireBeanImpl` for an example of checking to see if a `ContactContact`, and the associated `Contact`, can be retired.

## Implementation Details for Retiring Contacts in ClaimCenter

ClaimCenter provides the `gw.api.contact.ContactRetireHelper` class to determine if a contact can be retired. This class provides two methods used in retiring contacts:

- `public static boolean retireContact(Contact contact)` – Attempts to retire the passed in `Contact`, returning `true` if it was successful in doing so. This method is called by the `Contact Retire` work item.
- `public static boolean computeCanRetireContact(Contact contact, ContactRetireContext retireContext)` – Available to be used when implementing the `ContactRetireBean` interface to see if a contact can be retired. You might call this method if, while using `ContactRetireBean.computeCanRetireBeanForContactProperty`, you encounter another contact and need to retire it as well.

The `retireContact` method splits the foreign key references for the `Contact` into two components, those properties deemed safe from the `IRetireContactPlugin` implementation and the other properties. The safe properties for the contact do not block the retirement of the contact. Additionally, if there is a bean connected to the contact through this property, it is retired along with the contact.

In the base implementation, the safe references are:

Entity	Array field
ContactAddress	Contact
ContactCategoryScore	Contact
ContactTag	Contact
EFTData	Contact
OfficialID	Contact
Review	Contact

The other properties by default will block the retirement of the contact unless they implement the `ContactRetireBean` interface. If an entity implements that interface, the `computeCanRetireBeanForContactProperty` method of the interface can be called to see if the entity can be retired.

In the base configuration, the only entity requiring this interface to be implemented is `ContactContact`. This entity has the dual role of being an array on `Contact` and also having a foreign key to a contact. `ContactContact` represents a join table to two contacts, so the other contact is a foreign key reference. The class that implements the `ContactRetireBean` interface is `gw.api.contact.ContactContactRetireBeanImpl`.

## ContactMapper Class

Guidewire core applications send contact information to `ContactManager` by using an XML SOAP object. They use the `ContactMapper` class both to generate the SOAP object and to interpret it when they receive an XML SOAP object from `ContactManager`.

`ContactManager` and the Guidewire core applications each have their own version of the `ContactMapper` class. This class enables the applications to convert `Contact` entities and subentities to XML and send them to `ContactManager`. The class also enables Guidewire core applications to receive XML representations of the entities from `ContactManager` and convert them to `Contact` entities. `ContactManager` also uses its version of this class to generate XML to send to the applications and to interpret the XML it receives from the applications. The class file in each core application and in `ContactManager` explicitly specifies every field of the entities to send or receive and how to handle them in that application.

In the base configuration of each Guidewire core application, the `ContactMapper` class is configured to handle all the `Contact` entities and subentities. The class handles the entities and fields of those entities that are to be sent to and received from `ContactManager`.

In the base configuration, there are differences in entity names, typecodes, and contact entity field names between the core applications and `ContactManager`. Each Guidewire core application has its own domain namespace, but the `ContactManager` web services require the `ContactManager` domain namespace. The core applications use `ContactManager` web services to communicate with `ContactManager`. Therefore, when there are differences in names of fields, datatypes, and typecodes between the core application and `ContactManager`, each core application is responsible for doing the name mapping.

- To map differing entity field names, a core application uses a method on the `ContactMapper` class.
- To map names of datatypes and typecodes, a core application uses the `ContactMapper` class in conjunction with a mapping class that specifies these name mappings. For example, a `Contact` in the core application maps to an `ABContact` in `ContactManager`. See “Core Application Mapping” on page 141.

The `ContactManager` `ContactMapper` class handles only `ABContact` entities and subentities. Because `ContactManager` must be able to communicate with all the Guidewire core applications, its `ContactMapper` class must contain all fields of each type of contact that each core application needs. In a Guidewire core application, you can map just the contact fields that you want to send to `ContactManager` and receive from it.

The `ContactMapper` class creates a SOAP object called `XMLBackedInstance` to pass the data between a Guidewire core application and `ContactManager`. This object contains the fields and data of contacts and their related arrays and foreign keys that are defined in the `ContactMapper` class. `XmlBackedInstance` has a representation both as an object and as an XML string.

If you are working only with the entities supplied by Guidewire in the base configuration, you do not need to make any changes to the `ContactMapper` class. For information on the contact data model in the base configuration, see “Overview of Contact Entities” on page 133.

You can extend your Guidewire core application’s `Contact` data model, as described in “Extending the Contact Data Model” on page 133. You make the extension in `ContactManager` as well, and then map the entities to each other in `ContactMapper` in both the Guidewire core application and in `ContactManager`. Additionally, you use the application’s mapping class to map differing entity names and any different typecodes to and from `ContactManager`.

This topic includes:

- “`ContactManager ContactMapper Class`” on page 286
- “`Core Application ContactMapper Class`” on page 289

## ContactManager ContactMapper Class

In `ContactManager`, you use the class `gw.contactmapper.ab800.ContactMapper` to map `ABContact` entities and subentities. The class maps them to an XML object to send to Guidewire core applications and maps them from the XML objects received from Guidewire core applications. The class in the base `ContactManager` configuration works only with `ABContact` entities and subentities. `ContactManager` leaves it to the applications to map `ABContact` entities and subentities to and from `Contact` entities and subentities.

You can access this class in `ContactManager` Studio in the `Project` window. Navigate to `configuration → gsrc` and then to `gw.contactmapper.ab800.ContactMapper`.

The following methods map the fields, foreign keys, and arrays of the `ContactManager` `ABContact` entities and subentities:

- `fieldMapping` – Maps fields of an entity
- `fkMapping` – Maps foreign keys of an entity
- `arrayMapping` – Maps array references of an entity

This topic includes:

- “`Mapping Fields of a ContactManager Contact`” on page 286
- “`Mapping Foreign Keys of a ContactManager Contact`” on page 287
- “`Mapping Array References of a ContactManager Contact`” on page 288
- “`Special Handling in ContactManager for Addresses`” on page 288

## Mapping Fields of a ContactManager Contact

The method `fieldMapping(Entity#Field)` by default maps `ABContact` entity and subentity fields in both directions.

**Note:** Parameters to methods in `ContactMapper` use Gosu *feature literals* syntax to statically refer to an entity’s fields. See “Feature Literals” on page 365 in the *Gosu Reference Guide*.

The mapping directions are:

- **To a Guidewire core application** – From a `ContactManager` entity to an XML object to be sent to a Guidewire core application
- **From a Guidewire core application** – From an XML object sent by a core application to a `ContactManager` entity

You also use this method to map fields of an entity that the contact references with either an array reference or a foreign key, as described later. To map a foreign key or array reference itself, you use different methods, also described later.

Each complete method call must be followed by a comma unless it is the last method call in the block.

### Specifying a Single Direction

You specify a single direction by using `.withMappingDirection`, as follows:

- `.withMappingDirection(TO_XML)` – Maps the field from ContactManager to the core application.

For example, use this method call to map a `LinkID` field to a core application. This mapping is one-way because ContactManager sets and maintains the `LinkID` value. For example:

```
fieldMapping(ABContact#LinkID)
    .withMappingDirection(TO_XML),
```

- `.withMappingDirection(TO_BEAN)` – Maps the field from the core application to ContactManager.

### Mapping Externally Specified Unique IDs of a ContactManager Contact

ContactManager supports creation of a contact with an external unique ID specified by the core application.

- If an external unique ID is specified in the `XmlBackedInstance` data sent in the `createContact` method call, ContactManager populates the `LinkID` of the new contact with that value. The field that ContactManager checks for in the `createContact` call is `External_UniqueID`.
- If the `createContact` method call does not specify an external unique ID, ContactManager generates its own unique ID and stores that value in the contact's `LinkID`.

For example, the following lines of mapping code from various parts of the `construct` method in the `ContactManager` class `ContactMapper` support this mechanism:

```
fieldMapping(ABContact#External_UniqueID),
fieldMapping(ABContactAddress#External_UniqueID),
fieldMapping(Address#External_UniqueID),
fieldMapping(ABContactTag#External_UniqueID),
fieldMapping(EFTData#External_UniqueID),
fieldMapping(ABContactCategoryScore#External_UniqueID),
```

### See also

- “PolicyCenter Mapping of Externally Specified Unique IDs” on page 291
- “Creating and Linking a Contact” on page 192
- “ContactManager Link IDs and Comparison to Other IDs” on page 271
- “ABContactAPI Methods” on page 276

### Mapping Foreign Keys of a ContactManager Contact

The method `fkMapping(Entity#ForeignKey)` maps foreign keys for `ABContact` entities, subentities, and join tables. You can use the following additional qualifying methods as well:

- `.withMappingDirection(TO_XML)` – Specifies that the foreign key is mapped to the core application.
- `.withMappingDirection(TO_BEAN)` – Specifies that the foreign key is mapped from the core application.

Use the `fieldMapping` method to map all the fields of the entity to which a foreign key refers.

For example, the foreign key on `ABContactAddress` that points to an `Address` object is defined as follows:

```
fkMapping(ABContactAddress#Address),
```

Additionally, the fields for the `Address` object must also be defined by using `fieldMapping` method calls. The code for the `Address` entity referenced by the foreign key is:

```
fieldMapping(Address#LinkID)
    .withMappingDirection(TO_XML),
```

```

fieldMapping(Address#External_PublicID),
fieldMapping(Address#AddressLine1),
fieldMapping(Address#AddressLine1Kanji),
fieldMapping(Address#AddressLine2),
fieldMapping(Address#AddressLine2Kanji),
fieldMapping(Address#AddressLine3),
fieldMapping(Address#AddressType),
fieldMapping(Address#City),
fieldMapping(Address#CityKanji),
fieldMapping(Address#Country),
fieldMapping(Address#County),
fieldMapping(Address#Description),
fieldMapping(Address#GeocodeStatus),
fieldMapping(Address#PostalCode),
fieldMapping(Address#State),
fieldMapping(Address#ValidUntil),
fieldMapping(Address#CEDEX),
fieldMapping(Address#CEDEXBureau),

```

## Mapping Array References of a ContactManager Contact

The method `arrayMapping(Entity#ArrayKey)` maps array references for ABContact entities and subentities. You can use the following additional qualifying methods as well:

- `withMappingDirection(TO_XML)` – Specifies that the array reference is mapped to the core application.
- `withMappingDirection(TO_BEAN)` – Specifies that the array reference is mapped from the core application.

Use the `fieldMapping` method to map all the fields of the entity to which an array reference refers.

For example, the array extension example “Extending Contacts with an Array” on page 158 adds a new entity named `ContactServiceState`. This entity represents a state in which a vendor contact operates. Each vendor can operate in more than one state, so the example adds an array reference to ABContact named `ContactServiceArea` that references an array of `ContactServiceState` entities.

**Note:** The example also creates a ClaimCenter `ContactServiceState` entity and a `ContactServiceArea` array reference on the ClaimCenter `Contact` entity.

To transfer this data into and out of ContactManager, you define the array reference and map the fields of the entity to which it refers in `ContactMapper`, as follows:

```

arrayMapping(ABContact#ContactServiceArea),
fieldMapping(ContactServiceState#LinkID)
    .withMappingDirection(TO_XML),
fieldMapping(ContactServiceState#External_PublicID),
fieldMapping(ContactServiceState#ServiceState)

```

## Special Handling in ContactManager for Addresses

Addresses sent from core applications require special handling in ContactManager to handle scenarios like swapping a primary address for a secondary address. Turning address data into XML does not require any special handling, so addresses being sent to core applications use the normal `ContactMapper` code. For these reasons, the following foreign key and array references are defined as one-way, from ContactManager to the core application:

```

fkMapping(ABContact#PrimaryAddress)
    .withMappingDirection(TO_XML),
arrayMapping(ABContact#ContactAddresses)
    .withMappingDirection(TO_XML),

```

When a change to a primary or secondary address for a contact comes in from a core application, ContactManager calls `ABUpdateBeanPopulator.populateAddresses` instead of the regular mapping code.

While the address foreign key and array reference require special handling, the fields of an address do not. You can add or delete address fields as needed by using the `fieldMapping` method.

## Core Application ContactMapper Class

In the Guidewire core applications, you use the class `gw.contactmapper.ab800.ContactMapper` to map entities between the core application and ContactManager. The class maps entities to an XML object to send to ContactManager and maps entities from the XML objects received from ContactManager.

You can access this class in Guidewire Studio from the Project window. Navigate to `configuration → gsrc` and then to `gw.contactmapper.ab800.ContactMapper`.

**Note:** ContactManager also has a `ContactMapper` class. If you want ContactManager to handle changes you make in the `ClaimCenter` class, you must update the `ContactMapper` class as well. For example, you might add a new field to a `Contact` subtype and want ContactManager to use that new field with the `ABContact` subtype. For more information on the `ContactMapper` class, see “[ContactManager ContactMapper Class](#)” on page 286.

As described in “[ContactMapper Class](#)” on page 285, each core application uses `ContactMapper` to map differing field names used by the core application and ContactManager. To map differing contact entity names and type-codes used by `ClaimCenter` and ContactManager, each core application uses a separate `XXNameMapper` Gosu class. You can access this class for each core application in Guidewire Studio from the Project window. Navigate to `configuration → gsrc` and then to `gw.contactmapper.ab800`. Double-click the following file for your application to open it in an editor:

- `ClaimCenter – CCNameMapper`
- `PolicyCenter – PCNameMapper`
- `BillingCenter – BCNameMapper`

There is no equivalent `ContactMapper` class for mapping differing entity and typecode names. All the mapping of differing names is done on the core application side.

For an example that shows how to map entity names from `ClaimCenter` to ContactManager, see “[Step 3: Map the Subtype Names](#)” on page 149.

This topic includes:

- “[Mapping Fields of a Core Application Contact](#)” on page 289
- “[Mapping Foreign Keys of a Core Application Contact](#)” on page 290
- “[Mapping Array References of a Core Application Contact](#)” on page 290
- “[PolicyCenter Mapping of Externally Specified Unique IDs](#)” on page 291
- “[Special Handling in ContactMapper in Core Applications](#)” on page 291

### Mapping Fields of a Core Application Contact

The method `fieldMapping(Entity#Field)` by default maps `Contact` entity and subentity fields in both directions:

- **To ContactManager** – From a core application entity to an XML object to be sent to ContactManager
- **From ContactManager** – From an XML object sent by ContactManager to a core application

You also use this method to map fields of an entity that the contact references with either an array reference or a foreign key. To map a foreign key or array reference itself, you use different methods, as described later.

Each method call with qualifying methods, if any, is an entry in the set of return values. If you add a method to the set, add a comma after it if your method is not the last method in the set.

#### Specifying a Single Mapping Direction for a Field

You specify a single direction for the field mapping by using the qualifying method `withMappingDirection`, as follows:

- `.withMappingDirection(TO_XML)` – Maps the field from the core application to ContactManager.

For example, use the following method call to map a `PublicID` field to the `ContactManager` `External_PublicID` field. This mapping is one-way because the core application sets and maintains this value.

- ```
fieldMapping(Contact#PublicID)
    .withMappingDirection(TO_XML)
    .withABName(MappingConstants.EXTERNAL_PUBLIC_ID),
• .withMappingDirection(TO_BEAN) – Maps the field from ContactManager to the core application.
```

### Mapping Fields with Differing Names

If a field for a contact has a different name in the core application from the name in `ContactManager`, you map it by using the `withABName` method. For example, the `AddressBookUID` property on a `ClaimCenter` `Contact` entity is called `LinkID` on an `ABCContact` in `ContactManager`. The following code maps these different field names for a `Contact` entity:

```
fieldMapping(Contact#AddressBookUID)
    .withABName(MappingConstants.LINK_ID),
```

### Mapping Foreign Keys of a Core Application Contact

The method `fkMapping(Entity#ForeignKey)` maps foreign keys for `Contact` entities, subentities, and join tables. The method by default operates in both directions. You can use the following additional qualifying methods as well:

- `withMappingDirection(TO_XML)` – Specifies that the foreign key is mapped to `ContactManager`.
- `withMappingDirection(TO_BEAN)` – Specifies that the foreign key is mapped from `ContactManager`.
- `withABName(name:String)` – Specifies a different name for the foreign key on the entity in `ContactManager`

Use the `fieldMapping` method to map all the fields of the entity to which a foreign key refers.

For example, the foreign key on `ContactAddress` that points to an `Address` object is defined as follows:

```
fkMapping(ContactAddress#Address),
```

Additionally, the fields for the `Address` object must all be defined by using `fieldMapping` method calls. The code for the `Address` entity referenced by the foreign key is:

```
fieldMapping(Address#AddressBookUID)
    .withABName(MappingConstants.LINK_ID),
fieldMapping(Address#PublicID)
    .withMappingDirection(TO_XML)
    .withABName(MappingConstants.EXTERNAL_PUBLIC_ID),
fieldMapping(Address#AddressLine1),
fieldMapping(Address#AddressLine2),
fieldMapping(Address#AddressLine3),
fieldMapping(Address#AddressType),
fieldMapping(Address#City),
fieldMapping(Address#County),
fieldMapping(Address#Country),
fieldMapping(Address#Description),
fieldMapping(Address#GeocodeStatus),
fieldMapping(Address#PostalCode),
fieldMapping(Address#State),
fieldMapping(Address#ValidUntil),
fieldMapping(Address#AddressLine1Kanji),
fieldMapping(Address#AddressLine2Kanji),
fieldMapping(Address#CityKanji),
fieldMapping(Address#CEDEX),
fieldMapping(Address#CEDEXBureau),
```

### Mapping Array References of a Core Application Contact

The method `arrayMapping(Entity#ArrayKey)` maps array references for `Contact` entities and subentities. The method by default operates in both directions. You can use the following additional qualifying methods as well:

- `withMappingDirection(TO_XML)` – Specifies that the array reference is mapped to `ContactManager`.
- `withMappingDirection(TO_BEAN)` – Specifies that the array reference is mapped from `ContactManager`.

For example, in ClaimCenter, the array of category scores from a vendor performance review is maintained by ContactManager. Therefore, the direction of the array reference is one-way, from ContactManager:

- ```
arrayMapping(Contact#CategoryScores)
    .withMappingDirection(TO_BEAN),
• withABName(name:String) – Specifies a different name for the array reference on the entity in ContactManager
```

Use the `fieldMapping` method to map all the fields of the entity to which an array reference refers.

For example, the array extension example “Extending Contacts with an Array” on page 158 adds a new entity named `ContactServiceState`. This entity represents a state in which a vendor contact operates. Each vendor can operate in more than one state, so the example adds an array reference to `Contact` named `ContactServiceArea` that references an array of `ContactServiceState` entities.

**Note:** The example also creates a ContactManager `ContactServiceState` entity and a `ContactServiceArea` array reference on the ContactManager `ABContact` entity.

To transfer this data to and from a core application, you define the array reference and map the fields of the entity to which it refers in `ContactMapper`, as follows:

```
arrayMapping(Contact#ContactServiceArea),
    fieldMapping(ContactServiceState#AddressBookUID)
        .withABName(MappingConstants.LINK_ID),
    fieldMapping(ContactServiceState#PublicID)
        .withMappingDirection(TO_XML)
        .withABName(MappingConstants.EXTERNAL_PUBLIC_ID),
    fieldMapping(ContactServiceState#ServiceState)
```

## PolicyCenter Mapping of Externally Specified Unique IDs

As described in “PolicyCenter Contact Creation with External Unique IDs” on page 192, PolicyCenter specifies unique IDs for the contacts it creates and sends to ContactManager. ContactManager uses these unique IDs for the new contact’s `LinkID` value rather than creating its own unique ID.

The PolicyCenter `ContactMapping` class defines the following mapping for the unique ID of a contact that is to be created in ContactManager:

```
fieldMapping(Contact#ExternalID)
    .withMappingDirection(TO_XML)
    .withABName(EXTERNAL_UNIQUE_ID),
```

### See also

- “Mapping Externally Specified Unique IDs of a ContactManager Contact” on page 287
- “Creating and Linking a Contact” on page 192
- “ContactManager Link IDs and Comparison to Other IDs” on page 271
- “ABContactAPI Methods” on page 276

## Special Handling in ContactMapper in Core Applications

In some cases, it is not possible to use `ContactMapper` code to handle the transfer of contact data. In those cases, the core application uses methods that handle more complex cases, like related contacts and addresses.

### Special Handling in One Direction

Address handling is an example of special handling of data sent from ContactManager. Addresses sent from ContactManager require special handling in core applications to support scenarios like swapping a primary address for a secondary address. Turning address data into XML does not require any special handling, so addresses being sent to ContactManager use the normal `ContactMapper` code. For these reasons, the following foreign key and array references are defined as one-way, from the core application to ContactManager:

```
fkMapping(Contact#PrimaryAddress)
    .withMappingDirection(TO_XML),
```

```
arrayMapping(Contact#ContactAddresses)
    .withMappingDirection(TO_XML),
```

When a change to a primary or secondary address for a contact comes in from a core application, the core application calls `populateAddresses` instead of the regular mapping code. The method is defined in `ContactIntegrationXMLMapperAppBase`.

This method call is explicit in `ClaimCenter`:

```
arrayMapping(Contact#ContactAddresses)
    .withMappingDirection(TO_BEAN)
    .withArrayBeanBlock( \ am, bp ->
        populateAddresses(bp.Bean as Contact, bp.XmlBackedInstance)),
```

`PolicyCenter` and `BillingCenter` also call `populateAddresses` for incoming addresses, but the method call is not explicit in `ContactMapper`.

While the address foreign key and address array reference require special handling, the fields of an address do not. You can add or delete address fields as needed by using the `fieldMapping` method.

### Special Handling in Both Directions

In `ClaimCenter`, related contacts use a parameter of the `withMappingDirection` method not discussed so far:

```
.withMappingDirection(BOTH)
```

You can use this method to specify one behavior for a field when it is sent to `ContactManager` and another behavior when the field is received from `ContactManager`. For example:

```
fkMapping(ContactContact#RelatedContact)
    .withMappingDirection(BOTH)
    .withABName("RelABCContact")
    .withEntityXMLBlock( \ lm, xp -> populateContactXmlForRelatedContact(lm, xp))
    .withEntityBeanBlock( \ lm, bp -> populateBeanFromXml(bp)),
```

### Excluding Contact Fields from ClaimCenter Contact Synchronization

You can use the `ContactMapper` method `withAffectsSync` to configure fields to be excluded from the set of fields that `ClaimCenter` uses to determine if a contact is synchronized with `ContactManager`. By default, all fields that you add to `ContactMapper` are included in the fields that are checked for synchronization status.

**Note:** You do not use `ContactMapper` for fields that determine contact relationships, such as `contactBidiRelCode="employer"`. The inclusion or exclusion of contact relationship fields is defined in the `RelationshipSyncConfig` class. See “Synchronizing ClaimCenter Contact Fields” on page 203.

If there is a field in `ContactMapper` that you want excluded from the synchronization check, use the `withAffectsSync` method and set its parameter to `false`. For example, the following code excludes the field `CategoryScores`:

```
arrayMapping(Contact#CategoryScores)
    .withMappingDirection(TO_BEAN)
    .withAffectsSync(false),
```

### See also

- “Synchronizing ClaimCenter and ContactManager Contacts” on page 200

### Excluding Contact Fields from Being Saved in the ClaimCenter Database

You can use the `ContactMapper` method `withPersist` to specify that a contact field not be saved in the `ClaimCenter` database. This feature enables `ClaimCenter` to receive the field from `ContactManager` and display it

the ClaimCenter user interface, but not save the field. By default, all contact fields in `ContactMapper` are saved, or *persisted*.

---

**IMPORTANT** Setting a field to not persist means that you want the ClaimCenter user to be able to see the value of the field but not change it. Do not enable editing for the field in screens like `ContactDetails.pcf`, and do not send the field back to `ContactManager`.

---

To set a field to not persist, use the `withPersist` method and set its parameter to `false`. You must also use `withMappingDirection(TO_BEAN)` to map the field only from `ContactManager` to ClaimCenter. Additionally, use `withAffectsSync(false)` to exclude the field from being included in the synchronization check. Otherwise, the contact would always be out of sync. For example:

```
arrayMapping(Contact#ContactManagerOnlyField)
    .withMappingDirection(TO_BEAN)
    .withAffectsSync(false)
    .withPersist(false),
```

## ContactManager Plugins

As described in “Plugin Overview” on page 163 in the *Integration Guide*, *plugins* are software modules that `ContactManager` uses to perform actions or calculate results. Strictly speaking, the term *plugin* refers to an interface, and the term *plugin implementation* is the implementation of a plugin interface. However, in practice, *plugin implementation* is often shortened to just *plugin*.

As described in the *Integration Guide* topic, you can write your own plugin implementations of Guidewire plugin interfaces in Java or, preferably, in Gosu.

Additionally, there are *plugin registry nodes*, which you access in `ContactManager` studio. You use a registry to identify which plugin implementation `ContactManager` is to use.

This topic describes the plugins available in `ContactManager`, but does not cover implementation.

### Plugin Terminology Example

An example of a plugin registry in `ContactManager` is `ClaimSystemPlugin`, which `ContactManager` can use to send changes in contacts to ClaimCenter.

#### To view the plugin registry and the plugin implementation

1. Start `ContactManager` Studio.

At a command prompt, navigate to `ContactManager/bin` and enter the following command:

```
gwab studio
```

2. In the Project window, navigate to `configuration → Plugins` and then to `gw.plugin.ClientSystemPlugin.ClaimSystemPlugin`. The last node, `ClaimSystemPlugin`, is a *plugin registry*.

3. In the Plugin Registry editor on the right, in the `Class` field, the default setting in the base application is `gw.plugin.StandAloneClientSystemPlugin`. This class is a *plugin implementation* that extends the `AbstractClientSystemPlugin` class.

**Note:** You might have set up `ContactManager` to integrate with ClaimCenter, as described in “Integrating `ContactManager` with ClaimCenter in QuickStart” on page 46. In that case, the class you see is `gw.plugin.claim.cc800.CCCClaimSystemPlugin`, which `ContactManager` actually does use to communicate changes to ClaimCenter.

4. If you navigate to **configuration** → **gsrc** and then to `gw.plugin.claim.cc800.CCClaimSystemPlugin`, you can see that this *plugin implementation* extends the `AbstractClientSystemPlugin` class. The `AbstractClientSystemPlugin` class implements `ClientSystemPlugin`.

## ContactManager Plugin Overview

The following table lists most of the ContactManager plugins.

**IMPORTANT** If you want to change how ContactManager communicates with a core application to send contact updates, extend `gw.plugin.AbstractClientSystemPlugin`. Do not implement the plugin interface `gw.plugin.ClientSystemPlugin`. Implementing the `ClientSystemPlugin` interface will prevent ContactManager from starting.

Registry	Description
<code>AuthenticationSourceCreatorPlugin</code>	<ul style="list-style-type: none"> <li>• Registry – <code>configuration</code> → <code>config</code> → <code>Plugins</code> → <code>registry</code> → <code>AuthenticationSourceCreatorPlugin.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.security.AuthenticationSourceCreatorPlugin</code></li> <li>• Default Registered Plugin Implementation – <code>com.guidewire.pl.web.internal.DefaultAuthenticationSourceCreatorPlugin</code></li> </ul> <p>Authenticates the user logging in to ContactManager.</p>
<code>BillingSystemPlugin</code>	<ul style="list-style-type: none"> <li>• Registry – <code>configuration</code> → <code>config</code> → <code>Plugins</code> → <code>registry</code> → <code>BillingSystemPlugin.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.ClientSystemPlugin</code></li> </ul> <p><b>Important:</b> Do not implement this interface.</p> <ul style="list-style-type: none"> <li>• Default Registered Plugin Implementation – <code>gw.plugin.AbstractClientSystemPlugin</code></li> <li>• Parent Class of Working Registered Plugin – <code>gw.plugin.AbstractClientSystemPlugin</code></li> <li>• Working Plugin Class Implementation – <code>gw.plugin.billing.cc800.BCBillingSystemPlugin</code></li> </ul> <p>Sends changes in contacts to BillingCenter. See “Integrating ContactManager with Guidewire Core Applications” on page 45.</p>
<code>ClaimSystemPlugin</code>	<ul style="list-style-type: none"> <li>• Registry – <code>configuration</code> → <code>config</code> → <code>Plugins</code> → <code>registry</code> → <code>ClaimSystemPlugin.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.ClientSystemPlugin</code></li> </ul> <p><b>Do not implement this interface.</b></p> <ul style="list-style-type: none"> <li>• Default Registered Plugin Implementation – <code>gw.plugin.AbstractClientSystemPlugin</code></li> <li>• Parent Class of Working Registered Plugin – <code>gw.plugin.AbstractClientSystemPlugin</code></li> <li>• Working Plugin Class Implementation – <code>gw.plugin.claim.cc800.CCClaimSystemPlugin</code></li> </ul> <p>Sends changes in contacts to ClaimCenter. See “Integrating ContactManager with Guidewire Core Applications” on page 45.</p>
<code>GeocodePlugin</code>	<ul style="list-style-type: none"> <li>• Registry – <code>configuration</code> → <code>config</code> → <code>Plugins</code> → <code>registry</code> → <code>GeocodePlugin.gwp</code></li> <li>• Parent Class – <code>gw.api.geocode.AbstractGeocodePlugin</code></li> <li>• Plugin Class Implementation – <code>gw.plugin.geocode.impl.BingMapsPlugin</code></li> </ul> <p>Connects with a geocoding service to provide geocoding information for addresses. For information on setting up geocoding with this plugin, see “Geocoding and Proximity Searches for Vendor Contacts” on page 99.</p>

Registry	Description
<code>IABContactScoringPlugin</code>	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → <code>IABContactScoringPlugin.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.contact.IABContactScoringPlugin</code></li> <li>• Plugin Class Implementation – <code>gw.plugin.spm.impl.ABContactScoringPlugin</code></li> </ul> <p>Scores provider reviews sent from ClaimCenter. See “ClaimCenter Service Provider Performance Reviews” on page 215.</p>
<code>IAddressAutocompletePlugin</code>	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → <code>IAddressAutocompletePlugin.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.addressautocomplete.IAddressAutocompletePlugin</code></li> <li>• Plugin Class Implementation – <code>gw.api.address.DefaultAddressAutocompletePlugin</code></li> </ul> <p>Use this plugin configure how automatic address completion and fill-in operate. See “The Address Automatic Completion and Fill-in Plugin” on page 113 in the <i>Globalization Guide</i>.</p>
<code>IEncryption</code>	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → <code>IEncryption.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.util.IEncryption</code></li> <li>• Plugin Class Implementation – <code>gw.plugin.encryption.EncryptionByReversePlugin</code></li> </ul> <p>Encodes or decodes a String based on an algorithm you provide to hide important data, such as bank account numbers or private personal data. ContactManager does not provide any encryption algorithm in the product. ContactManager simply calls the <code>EncryptionByReversePlugin</code> implementation, which does nothing. See “Encryption Integration” on page 249 in the <i>Integration Guide</i>.</p>
<code>IFindDuplicatesPlugin</code>	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → <code>IFindDuplicatesPlugin.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.contact.IFindDuplicatesPlugin</code></li> <li>• Plugin Class Implementation – <code>gw.plugin.contact.findduplicates.FindDuplicatesPlugin</code></li> </ul> <p>Finds duplicate contacts. See “<code>IFindDuplicatesPlugin</code> Plugin Interface” on page 298.</p>
<code>IGroupExceptionPlugin</code>	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → <code>IGroupExceptionPlugin.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.exception.IGroupExceptionPlugin</code></li> <li>• Plugin Class Implementation – <code>com.guidewire.pl.domain.escalation.RulesBasedGroupExceptionPlugin</code></li> </ul> <p>Calls the group exception rule set. See “Exception and Escalation Plugins” on page 270 in the <i>Integration Guide</i>.</p>
<code>IPhoneNormalizerPlugin</code>	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → <code>IPhoneNormalizerPlugin.gwp</code></li> <li>• Plugin Interface – <code>gw.plugin.phone.IPhoneNormalizerPlugin</code></li> <li>• Parent Class of Registered Plugin – <code>gw.api.phone.DefaultPhoneNormalizerPlugin</code></li> <li>• Default Registered Plugin Class – <code>gw.api.phone.DefaultABPhoneNormalizerPlugin</code></li> </ul> <p>The default registered plugin class extends the <code>DefaultPhoneNormalizerPlugin</code> class, which implements the interface. <code>DefaultABPhoneNormalizerPlugin</code> provides support for phone numbers to ABContact and its subtypes.</p>

Registry	Description
IUserExceptionPlugin	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → IUserExceptionPlugin.gwp</li> <li>• Plugin Interface – gw.plugin.exception.IUserExceptionPlugin</li> <li>• Plugin Class Implementation – com.guidewire.pl.domain.escalation.RulesBasedUserExceptionPlugin</li> </ul> <p>Calls the user exception rule set. See “Exception and Escalation Plugins” on page 270 in the <i>Integration Guide</i>.</p>
ITestingClock	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → ITestingClock.gwp</li> <li>• Plugin Interface – gw.plugin.system.ITestingClock</li> <li>• Plugin Class Implementation – com.guidewire.pl.plugin.system.internal.OffsetTestingClock</li> </ul> <p>Used for testing complex behavior over a long span of time, such as timeouts that are multiple days or weeks later. This plugin is for development (non-production) use only. It programmatically changes the system time to simulate passage of time in ContactManager.</p> <p><b>WARNING:</b> You must <b>never</b> use the testing clock plugin on a production server. See “Testing Clock Plugin Interface—Only For Non-Production Servers” on page 308.</p>
OfficialIdToTaxIdMappingPlugin	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → OfficialIdToTaxIdMappingPlugin.gwp</li> <li>• Plugin Interface – gw.plugin.contact.OfficialIdToTaxIdMappingPlugin</li> <li>• Plugin Class Implementation in ContactManager, ClaimCenter, and BillingCenter – gw.plugin.contact.internal.AlwaysFalseOfficialIdToTaxIdMappingPlugin</li> <li>• Plugin Class Implementation in PolicyCenter – gw.plugin.contact.impl.PCOOfficialIdToTaxIdMappingPlugin</li> </ul> <p>This plugin is used only by PolicyCenter. However, it is available to all core applications. In the base configurations of ContactManager, ClaimCenter, and BillingCenter, the registered plugin is AlwaysFalseOfficialIDToTaxIDMappingPlugin, which always returns the boolean value false. ContactManager, ClaimCenter, and BillingCenter support only a single TaxID field and do not need to do this mapping.</p> <p>In the base configuration of PolicyCenter, the registered plugin is gw.plugin.contact.impl.PCOOfficialIdToTaxIdMappingPlugin. PolicyCenter supports an array of official IDs for each contact. For the integration with ContactManager to work, PolicyCenter needs to be able to map one of the typecodes from its OfficialIDType typelist to the contact's TaxID field. In PolicyCenter, this field is the contact's primary ID for identification purposes.</p> <p>In the base configuration of PolicyCenter, the plugin implementation overrides the method <code>isTaxId</code>. This method override determines if the tax ID is in the OfficialIDType typelist as a Social Security Number (TC_SSN) or a Federal Employer Identification Number (TC_FEIN). You can extend PolicyCenter with your own official IDs and map the ones appropriate for your locale or country.</p>

Registry	Description
PendingContactChangeConfigurationPlugin	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → PendingContactChangeConfigurationPlugin.gwp</li> <li>• Plugin Interface – gw.plugin.contact.PendingContactChangeConfigurationPlugin</li> <li>• Plugin Class Implementation – gw.plugin.contact.pendingchange.PendingABContactChangeConfigPlugin</li> </ul> <p>Controls the matching used in generating the contact difference view in the Pending Updates screen. Returns a reference to a PendingChangeConfigImpl object, used to configure the graph traversal behavior of the pending changes web UI. To configure how pending updates work, write a plugin that implements the interface.</p>
PolicySystemPlugin	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → PolicySystemPlugin.gwp</li> <li>• Plugin Interface – gw.plugin.ClientSystemPlugin <b>Do not implement this interface.</b></li> <li>• Default Registered Plugin Implementation – gw.plugin.AbstractClientSystemPlugin</li> <li>• Parent Class of Working Registered Plugin – gw.plugin.AbstractClientSystemPlugin</li> <li>• Working Plugin Class Implementation – gw.plugin.policy.cc800.PCPolicySystemPlugin</li> </ul> <p>Sends changes in contacts to PolicyCenter. See “Integrating ContactManager with Guidewire Core Applications” on page 45.</p>
ValidateABContactCreationPlugin	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → ValidateABContactCreationPlugin.gwp</li> <li>• Plugin Interface – gw.plugin.contact.ValidateABContactCreationPlugin</li> <li>• Plugin Class Implementation – gw.plugin.contact.ValidateABContactCreationPluginImpl</li> </ul> <p>Validates that creation criteria have been met before creating a contact. See “ValidateABContactCreationPlugin Plugin Interface” on page 306.</p>
ValidateABContactSearchCriteriaPlugin	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → ValidateABContactSearchCriteriaPlugin.gwp</li> <li>• Plugin Interface – gw.plugin.contact.ValidateABContactSearchCriteriaPlugin</li> <li>• Plugin Class Implementation – gw.plugin.contact.ValidateABContactSearchCriteriaPluginImpl</li> </ul> <p>Validates that search criteria have been met for finding a contact.</p> <p>To specify that a Contact entity field is required in a search, add it to gw.plugin.contact.ValidateABContactSearchCriteriaPluginImpl.</p> <p>The following code snippet from this class shows the default search criteria defined for an ABPerson entity:</p> <pre data-bbox="780 1495 1449 1801"> private class ABPersonLogic extends ABContactSubtypeLogic {     override function execute(bean : Bean) : boolean {         var searchCriteria = (bean as ABContactSearchCriteria)         if (searchCriteria.Keyword == null             and searchCriteria.FirstName == null             and searchCriteria.TaxID == null             and isInLocale(searchCriteria)             and searchCriteria.Address.PostalCode == null             and not searchCriteria.isValidProximitySearch()) {             return false         }         return true     } } </pre>

Registry	Description
WebservicesAuthenticationPlugin	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → WebservicesAuthenticationPlugin.gwp</li> <li>• Plugin Interface – gw.plugin.security.WebservicesAuthenticationPlugin</li> <li>• Plugin Class Implementation – com.guidewire.pl.security.internal.DefaultWebservicesAuthenticationPlugin</li> </ul>
For WS-I web services only, configures custom authentication logic. See “WS-I Web Services Authentication Plugin” on page 56 in the <i>Integration Guide</i> .	
<b>Messaging plugins (see “Messaging and Events” on page 299 in the <i>Integration Guide</i>)</b>	
BCBillingSystemTransport	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → BCBillingSystemTransport.gwp</li> <li>• Interfaces – gw.plugin.messaging.MessageTransport gw.plugin.InitializablePlugin</li> <li>• Plugin Class Implementation – gw.plugin.integration.InitializableClientSystemTransport</li> </ul>
CCClaimSystemTransport	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → CCClaimSystemTransport.gwp</li> <li>• Interfaces – gw.plugin.messaging.MessageTransport gw.plugin.InitializablePlugin</li> <li>• Plugin Class Implementation – gw.plugin.integration.InitializableClientSystemTransport</li> </ul>
PCPolicySystemTransport	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → PCPolicySystemTransport.gwp</li> <li>• Interfaces – gw.plugin.messaging.MessageTransport gw.plugin.InitializablePlugin</li> <li>• Plugin Class Implementation – gw.plugin.integration.InitializableClientSystemTransport</li> </ul>
emailMessageTransport	<ul style="list-style-type: none"> <li>• Registry – configuration → config → Plugins → registry → emailMessageTransport.gwp</li> <li>• Parent Class – gw.api.email.AbstractEmailMessageTransport</li> <li>• Plugin Class Implementation – gw.plugin.email.impl.EmailMessageTransport</li> </ul>

## IFindDuplicatesPlugin Plugin Interface

The FindDuplicatesPlugin plugin implementation implements the IFindDuplicatesPlugin plugin interface in Gosu and is available in ContactManager Studio. To view or edit the plugin implementation, start ContactManager Studio. Then, in the Project window, navigate to configuration → gsrc and then to gw.plugin.contact.findduplicates.FindDuplicatesPlugin.

**Note:** The FindDuplicatesPlugin plugin implementation and its helper classes use *Gosu generics*. For more information, see “Gosu Generics” on page 239 in the *Gosu Reference Guide*.

The plugin attempts to find exact and potential matches for the subtype of ABContact that is passed to it. If no Duplicate Finder exists for the subtype, the plugin traverses the ABContact tree to find the closest parent type it can use in the search.

The plugin uses a set of Duplicate Finder classes, which define potential and exact match logic for a set of ABContact subtypes. The plugin sets up the duplicate finder classes in two Map definitions:

```
static final var WIDE_MAP : Map<typekey.ABContact, Type<DuplicateFinderBase>> = {  
    typekey.ABContact.TC_ABCOMPANY      -> CompanyDuplicateFinder<ABCompany>,  
    typekey.ABContact.TC_ABPERSON       -> PersonDuplicateFinder<ABPerson>,  
    typekey.ABContact.TC_ABPLACE        -> PlaceDuplicateFinder  
}  
  
static final var DEFAULT_MAP : Map<typekey.ABContact, Type<DuplicateFinderBase>> = {  
    typekey.ABContact.TC_ABCOMPANYVENDOR -> CompanyDuplicateFinder<ABCompanyVendor>,  
    typekey.ABContact.TC_ABCOMPANY      -> CompanyDuplicateFinder<ABCompany>,  
    typekey.ABContact.TC_ABPERSONVENDOR -> PersonVendorDuplicateFinder,  
    typekey.ABContact.TC_ABUSERCONTACT -> UserDuplicateFinder,  
    typekey.ABContact.TC_ABPERSON       -> PersonDuplicateFinder<ABPerson>,  
    typekey.ABContact.TC_ABPLACE        -> PlaceDuplicateFinder  
}
```

These two Map definitions are likely to be the only area of the plugin itself that you need to edit. For example, if you add a Duplicate Finder class for the ABAttorney subtype, you could add a typekey definition for that class to the DEFAULT\_MAP definition.

**Note:** You do not have to add a map definition for any subtype, even a subtype extension that you create. Because the plugin traverses the ABContact tree to find the nearest parent type, any subtype of ABContact already has matching logic defined for it. Add a Duplicate Finder class only if you need special matching logic for a subtype.

The WIDE\_MAP definition is used by the ContactManager batch process Duplicate Contact Finder in the configuration parameter `DuplicateContactsWideSearch`. For more information, see:

- “Changing Match Results and Search Scope Settings” on page 249
- “Detecting and Merging Duplicate Contacts” on page 245

This topic includes:

- “Duplicate Finder Classes” on page 299
- “Query Builder Classes” on page 303

## Duplicate Finder Classes

`FindDuplicatesPlugin` uses a set of duplicate finder classes, each of which defines:

- The minimum set of fields required to attempt a match.
- The fields required for exact matches for the ABContact subtype.
- The fields required for potential matches for the ABContact subtype.

Each class checks for the existence of a minimum set of fields for performing a match with this subtype. If there is enough data for a match, the class can perform a search for potential matches and check the potential matches to see if any are an exact match.

If there is not enough data for a match, the `validateMandatoryFields` method throws a `TooLooseContactDuplicateMatchCriteriaException`. These exceptions use display keys, which you can access in ContactManager Studio. In the Project window, navigate to `configuration → Localization → en_US`, and then double click `display.properties` to open this file in the editor. In the editor, press `CTRL+F` and search for `TooLooseContactDuplicateMatchCriteriaException`.

To view or edit the Duplicate Finder classes, start ContactManager Studio. Then, in the Project window, navigate to `configuration → gsrc` and then to `gw.plugin.contact.finnduplicates`. Under this node, you see all the Duplicate Finder classes.

In these classes, you can change the logic required for matching a subtype. For example, you want to add a new field to the potential match logic in one of the Duplicate Finder classes. You must first add appropriate matching

code for the field to the query builder class that supports that duplicate finder class. You then add the query finder method call to the Duplicate Finder class. For an example of a class that adds field matching logic, see “Person-DuplicateFinder” on page 301.

The query builder classes are located one node down in the Project window. Navigate to configuration → gsrc and then to gw.plugin.contact.findduplicates.querybuilder. For more information on these classes, see “Query Builder Classes” on page 303.

You also might want to define new matching for an ABContact subtype that does not have a Duplicate Finder class. In this case, you would create a new Duplicate Finder class and a query builder class for the subtype.

**Note:** You do not have to add a Duplicate Finder class for any subtype, even a subtype extension that you create. Because the plugin traverses the ABContact tree to find the nearest parent type, any subtype of ABContact already has matching logic defined for it. Add a Duplicate Finder class only if you need special matching logic for a subtype.

The Duplicate Finder class descriptions that follow describe the functionality available in the base product. The CompanyDuplicateFinder description includes code for required fields, potential match, and exact match. The code in the other duplicate finders is similar.

This topic includes:

- “CompanyDuplicateFinder Class” on page 300
- “PersonDuplicateFinder” on page 301
- “PersonVendorDuplicateFinder Class” on page 302
- “PlaceDuplicateFinder Class” on page 302
- “UserDuplicateFinder Class” on page 303

## CompanyDuplicateFinder Class

This Gosu class defines matching for the ABCompany subtype. If the subtype is ABCompanyVendor, ABAutoRepairShop, ABAutoTowingAgcy, ABLawFirm, or ABMedicalCareOrg, matching is performed for ABCompany. The class uses the query builder CompanyQueryBuilder, described at “CompanyQueryBuilder Class” on page 304.

To open this class in Studio, navigate in the Project window to configuration → gsrc and then to gw.plugin.contact.findduplicates.CompanyDuplicateFinder.

**Note:** The match criteria descriptions that follow includes the code for required fields, potential match, and exact match. The code in the other duplicate finders is similar.

CompanyDuplicateFinder defines the following match criteria.

- **Fields to match:**
  - Name – Match this field as *starts with* or *equals*, depending on context.
  - PrimaryAddress (AddressLine1, City, State, PostalCode) – Match these PrimaryAddress fields as *equals*.
  - TaxID – Match this field as *equals*.
  - WorkPhone or FaxPhone – Match any single phone field as *equals*.
- **Required fields:** Name and at least one of PrimaryAddress, TaxID, or any phone field: WorkPhone or FaxPhone.

The following code defines the required fields for an ABCompany:

```
override function validateMandatoryFields() {  
    if (_searchContact.Name == null ||  
        (hasNoPrimaryAddress() and  
        hasNoPhoneNumber() and  
        _searchContact.TaxID == null))  
        throwException(_searchContact)  
}
```

- **Potential match types:**

- Starts with Name and equals either PrimaryAddress or a phone field.
- Equals TaxID.

The following code defines the potential match fields for an ABCompany:

```
override function makeQueries() : List<Query<C>> {
    var queries = new ArrayList<Query<C>>()
    //Query: TaxID
    new CompanyQueryBuilder<C>(_searchContact)
        .hasEqualTaxId()//AND
        .buildAndAdd(queries)
    //Query: Name and PhoneNumber
    if (not hasNoPhoneNumber()) {
        new CompanyQueryBuilder<C>(_searchContact)
            .startsWithName()//AND
            .hasEqualPhoneNumbers()
            .buildAndAdd(queries)
    }
    //Query: Name and Address
    if (not hasNoPrimaryAddress()) {
        new CompanyQueryBuilder<C>(_searchContact)
            .startsWithName()//AND
            .hasEqualAddress()
            .buildAndAdd(queries)
    }
    return queries
}
```

- **Exact Match – Equals** both TaxID and Name.

The following code defines the exact match fields for an ABCompany:

```
override function isExactMatch(searchContact : C, resultABContact : C) : boolean {
    return equalsAndNotNull<String>(searchContact.TaxID, resultABContact.TaxID) &&
        equalsAndNotNull<String>(searchContact.Name, resultABContact.Name)
}
```

## PersonDuplicateFinder

This Gosu class defines matching for the ABPerson subtype. If the subtype is ABAdjudicator, matching is performed for ABPerson.

To open this class in Studio, navigate in the Project window to configuration → gsrc → gw and then to gw.plugin.contact.findduplicates.PersonDuplicateFinder.

The class uses the query builder PersonQueryBuilder, described at “PersonQueryBuilder Class” on page 306. That query builder extends PersonQueryBuilderBase, which adds logic for matching the first name, last name, date of birth, and phone numbers of the contacts. You can use that class and this one as an example of how to add matching logic for a field to a query builder and a Duplicate Finder.

PersonDuplicateFinder defines the following match criteria:

- **Fields to match:**

- FirstName – Match this field as *starts with* or *equals*, depending on context.
- LastName – Match this field as *equals*.
- PrimaryAddress (AddressLine1, City, State, PostalCode) – Match these PrimaryAddress fields as *equals*.
- LicenseNumber and LicenseState – Match both Driver’s License fields as *equals*.
- DateOfBirth – Match this field as *equals*.
- TaxID – Match this field as *equals*.
- HomePhone, WorkPhone, FaxPhone, or CellPhone – Match any single phone field as *equals*.
- **Required fields:** FirstName and LastName and at least one of PrimaryAddress, DateOfBirth, TaxID, any phone field, or LicenseNumber and LicenseState.
- **Match types:**

- **Potential** – Starts with FirstName and equals LastName and equals one of PrimaryAddress, DateOfBirth, any phone field, or LicenseNumber and LicenseState.
- **Potential** – Equals LastName and TaxID.
- **Exact** – Equals FirstName and LastName and DateOfBirth and equals one of PrimaryAddress or any phone field.
- **Exact** – Equals FirstName and LastName and TaxID.

## PersonVendorDuplicateFinder Class

This Gosu class defines matching for the ABPersonVendor subtype. If the subtype is ABAttorney or ABDirector, matching is performed for ABPersonVendor.

To open this class in Studio, navigate in the Project window to configuration → gsrc → gw and then to gw.plugin.contact.findduplicates.PersonVendorDuplicateFinder.

The class uses the query builder PersonQueryBuilder, described at “PersonQueryBuilder Class” on page 306.

PersonDuplicateFinder defines the following match criteria:

- **Fields to match:**
  - FirstName – Match this field as *starts with* or *equals*, depending on context.
  - LastName – Match this field as *equals*.
  - PrimaryAddress (AddressLine1, City, State, PostalCode) – Match these PrimaryAddress fields as *equals*.
  - TaxID – Match this field as *equals*.
  - HomePhone, WorkPhone, FaxPhone, or CellPhone – Match any single phone field as *equals*.
- **Required fields:** FirstName and LastName and at least one of PrimaryAddress, TaxID, or any phone field.
- **Match types:**
  - **Potential** – Starts with FirstName and equals LastName and equals one of PrimaryAddress or any phone field.
  - **Potential** – Equals TaxID.
  - **Exact** – Equals FirstName and LastName and TaxID.

## PlaceDuplicateFinder Class

This Gosu class defines matching for the ABPlace subtype. If the subtype is ABLegalVenue, matching is performed for ABPlace.

To open this class in Studio, navigate in the Project window to configuration → gsrc and then to gw.plugin.contact.findduplicates.PlaceDuplicateFinder.

The class uses the query builder PlaceQueryBuilder, described at “PlaceQueryBuilder Class” on page 304.

PlaceDuplicateFinder defines the following match criteria:

- **Fields to match:**
  - Name – Match this field as *starts with* or *equals*, depending on context.
  - PrimaryAddress (AddressLine1, City, State, PostalCode) – Match these PrimaryAddress fields as *equals*.
- **Required fields:** Name and PrimaryAddress.
- **Match types:**
  - **Potential** – Starts with Name.
  - **Potential** – Equals PrimaryAddress.
  - **Exact** – Equals Name and AddressLine1 and City and State and PostalCode.

## UserDuplicateFinder Class

This class defines matching for the ABUserContact subtype.

To open this class in Studio, navigate in the Project window to configuration → gsrc and then to gw.plugin.contact.findduplicates.UserDuplicateFinder.

The class uses the query builder UserContactQueryBuilder, described at “UserContactQueryBuilder Class” on page 306.

UserContactDuplicateFinder defines the following match criteria:

- **Fields to match:**
  - FirstName – Match this field as *starts with* or *equals*, depending on context.
  - LastName – Match this field as *equals*.
  - EmployeeNumber – Match this field as *equals*.
- **Required fields:** FirstName and LastName and at least one of PrimaryAddress, TaxID, or any phone field.
- **Match types:**
  - **Potential** – Starts with FirstName and equals LastName.
  - **Potential** – Equals EmployeeNumber.
  - **Exact** – Equals EmployeeNumber.

## Query Builder Classes

The query builder Gosu classes build queries that the Duplicate Finder classes use to search the database for specific subtypes of ABContact. They define queries that:

- Compare fields to see if they are equal. See hasEqualTaxID under “ContactQueryBuilder Class” on page 303.
- Compare fields to see if they start with the same characters. See startsWithLastName under “ContactQueryBuilder Class” on page 303.
- Compare a set of fields to see if they are all equal. See hasEqualAddress under “ContactQueryBuilder Class” on page 303.
- Compare a set of fields to see if one of them is equal. See hasEqualPhoneNumbers under “ContactQueryBuilder Class” on page 303.

For information on the Duplicate Finder classes, see “Duplicate Finder Classes” on page 299.

This topic includes:

- “ContactQueryBuilder Class” on page 303
- “CompanyQueryBuilder Class” on page 304
- “PlaceQueryBuilder Class” on page 304
- “PersonQueryBuilderBase Class” on page 305
- “PersonQueryBuilder Class” on page 306
- “UserContactQueryBuilder Class” on page 306

## ContactQueryBuilder Class

This Gosu class provides a basic set of ABContact queries for the query builder classes. The classes that extend this one produce a set of queries for specific ABContact subtypes, which in turn are used by the Duplicate Finder classes for those subtypes.

To open this class in Studio, navigate in the Project window to configuration → gsrc and then to gw.plugin.contact.findduplicates.querybuilder.ContactQueryBuilder.

The class builds the following queries:

- `hasEqualTaxId` – Adds an expression to the query. The expression determines if the TaxID field of the ABContact being checked for duplicates is equal to the TaxID field of an ABContact in the database.
 

```
function hasEqualTaxId() : U {
    addExpression(new EqualFieldExpression<T>("TaxID", _contact.TaxID))
    return this as U
}
```
- `hasEqualPhoneNumbers` – Adds an expression to the query. Determines if the HomePhone, WorkPhone, or FaxPhone field of the ABContact being checked for duplicates is equal to the equivalent field of an ABContact in the database. If one of the fields matches, the contacts have equal phone numbers.
 

```
function hasEqualPhoneNumbers() : U {
    var numbers = PhoneNumbers
    var phoneOperators : List<FieldExpression<T>> = {
        new InFieldExpression<T>("HomePhone", numbers),
        new InFieldExpression<T>("WorkPhone", numbers),
        new InFieldExpression<T>("FaxPhone", numbers)
    }
    addExpression(new OrCompositeFieldExpression<T>(
        phoneOperators.toTypedArray() ))
    return this as U
}
```
- `hasEqualAddress` – Adds an expression to the query. Compares fields of the PrimaryAddress of the ABContact being checked for duplicates to the equivalent fields of the PrimaryAddress of an ABContact in the database. If the AddressLine1, State, City, and PostalCode fields are equal for both contacts, the addresses are considered equal.
 

```
function hasEqualAddress() : U {
    addExpression(new AndCompositeFieldExpression<T>({
        new EqualFieldExpression<T>("AddressLine1", "PrimaryAddress",
            _contact.PrimaryAddress.AddressLine1, false),
        new EqualFieldExpression<T>("State", "PrimaryAddress",
            _contact.PrimaryAddress.State, false),
        new EqualFieldExpression<T>("City", "PrimaryAddress",
            _contact.PrimaryAddress.City, false),
        new EqualFieldExpression<T>("PostalCode", "PrimaryAddress",
            _contact.PrimaryAddress.PostalCode, false)
    }))
    return this as U
}
```
- `startsWithName` – Adds an expression to the query. Uses the entire string in the Name field of the ABContact being checked for duplicates. Compares this string to a substring at the start of the Name field of an ABContact in the database. If the string and the substring are equal, the contacts are a *starts with* match. For example, “Asim” would be a *starts with* match with “Asimov”.
 

```
function startsWithName() : U {
    addExpression(new StartsWithFieldExpression<T>("Name", _contact.Name))
    return this as U
}
```

## CompanyQueryBuilder Class

This Gosu class builds queries for an ABCompany entity. It extends `ContactQueryBuilder`, making the query logic defined in that class available to the class `CompanyDuplicateFinder`. For descriptions of those two classes, see:

- “[ContactQueryBuilder Class](#)” on page 303
- “[CompanyDuplicateFinder Class](#)” on page 300

To open this class in Studio, navigate in the Project window to `configuration → gsrc` and then to `gw.plugin.contact.findduplicates.querybuilder.CompanyQueryBuilder`.

## PlaceQueryBuilder Class

This Gosu class builds queries for an ABPlace entity. It extends `ContactQueryBuilder`, making the query logic defined in that class available to the class `PlaceDuplicateFinder`. For descriptions of those two classes, see:

- “[ContactQueryBuilder Class](#)” on page 303
- “[PlaceDuplicateFinder Class](#)” on page 302

To open this class in Studio, navigate in the Project window to configuration → gsrc → gw and then to gw.plugin.contact.findduplicates.querybuilder.PlaceQueryBuilder.

### PersonQueryBuilderBase Class

This Gosu class builds queries for querying an ABPerson entity. It extends ContactQueryBuilder and adds new query logic used by PersonQueryBuilder and UserContactQueryBuilder. You can compare PersonQueryBuilderBase to “ContactQueryBuilder Class” on page 303 to see how to add new query logic for a field, such as LastName or LicenseNumber.

To open this class in Studio, navigate in the Project window to configuration → gsrc and then to gw.plugin.contact.findduplicates.querybuilder.PersonQueryBuilderBase.

The class builds the following queries in addition to, or in place of, those in ContactQueryBuilder:

- **startsWithFirstName** – Adds an expression to the query. Uses the entire string in the FirstName field of the ABPerson being checked for duplicates. Compares this string to a substring at the start of the FirstName field of an ABPerson in the database. If the string and the substring are equal, the contacts are a *starts with* match. For example, “Asim” would be a *starts with* match with “Asimov”.

```
function startsWithFirstName() : U {
    addExpression(new StartsWithFieldExpression<T>("FirstName", Contact.FirstName))
    return this as U
}
```

- **hasEqualLastName** – Adds an expression to the query. The expression determines if the LastName field of the ABPerson being checked for duplicates is equal to the LastName field of an ABPerson in the database.

```
function hasEqualLastName() : U {
    addExpression(new EqualFieldExpression<T>(
        "LastNameDenorm", Contact.LastName, false))
    return this as U
}
```

- **hasEqualBirthDate** – Adds an expression to the query. The expression determines if the DateOfBirth field of the ABPerson being checked for duplicates is equal to the DateOfBirth field of an ABPerson in the database.

```
function hasEqualBirthDate() : U {
    addExpression(new EqualFieldExpression<T>("DateOfBirth", Contact.DateOfBirth))
    return this as U
}
```

- **hasEqualLicenseNumber** – Adds an expression to the query. The expression determines if the LicenseNumber and LicenseState fields of the ABPerson being checked for duplicates are equal to the same fields of an ABPerson in the database.

```
function hasEqualLicenseNumber() : U {
    addExpression(new AndCompositeFieldExpression<T>({
        new EqualFieldExpression<T>("LicenseNumber", Contact.LicenseNumber),
        new EqualFieldExpression<T>("LicenseState", Contact.LicenseState)
    }))
    return this as U
}
```

- **hasEqualPhoneNumbers** – Adds an expression to the query. The method overrides ContactQueryBuilder.hasEqualPhoneNumbers and adds the CellPhone field as one of the phone number fields to check. The method determines if the HomePhone, WorkPhone, FaxPhone, or CellPhone field of the ABPerson being checked for duplicates is equal to the equivalent field of an ABPerson in the database. If one of the fields matches, the contacts have equal phone numbers.

```
override function hasEqualPhoneNumbers() : U {
    var numbers = PhoneNumbers
    var phoneOperators : List<FieldExpression<T>> = {
        new InFieldExpression<T>("HomePhone", numbers),
        new InFieldExpression<T>("WorkPhone", numbers),
        new InFieldExpression<T>("FaxPhone", numbers),
        new InFieldExpression<T>("CellPhone", numbers)
    }
    addExpression(new OrCompositeFieldExpression<T>(
        phoneOperators.toTypedArray() ))
    return this as U
}
```

## PersonQueryBuilder Class

This Gosu class builds queries for an ABPerson entity. It extends PersonQueryBuilderBase, making the query logic defined in that class available to the classes PersonDuplicateFinder and PersonVendorDuplicateFinder.

For descriptions of those classes, see:

- “PersonQueryBuilderBase Class” on page 305
- “PersonDuplicateFinder” on page 301
- “PersonVendorDuplicateFinder Class” on page 302

To open this class in Studio, navigate in the Project window to configuration → gsrc and then to gw.plugin.contact.findduplicates.querybuilder.PersonQueryBuilder.

## UserContactQueryBuilder Class

This Gosu class builds queries for an ABUserContact entity. It extends PersonQueryBuilderBase, making the query logic defined in that class available to the class UserDuplicateFinder. In addition, it adds a method, hasEqualEmployeeNumber for comparing equality of the EmployeeNumber fields of the two contacts.

For descriptions of those classes, see:

- “PersonQueryBuilderBase Class” on page 305
- “UserDuplicateFinder Class” on page 303

To open this class in Studio, navigate in the Project window to configuration → gsrc and then to gw.plugin.contact.findduplicates.querybuilder.UserContactQueryBuilder.

## ValidateABContactCreationPlugin Plugin Interface

This plugin interface is implemented in Gosu as ValidateABContactCreationPluginImpl. The registry file for this plugin is ValidateABContactCreationPlugin.gwp. If you open that registry file, you see that the Gosu class gw.plugin.contact.ValidateABContactCreationPluginImpl is registered.

To open the plugin implementation in the editor, start ContactManager Studio. Then, in the Project window, navigate to configuration → gsrc and then to gw.plugin.contact.ValidateABContactCreationPluginImpl.

This topic includes:

- “Plugin Logic” on page 306
- “Validation Errors and Displaykeys” on page 307

## Plugin Logic

The plugin enables you to provide your own definition of the minimum criteria required for creating new contacts. In the base configuration, all contacts are required to have a contact tag.

In the base configuration, the plugin implements the following minimum creation requirements:

- ABPerson – Last name, contact tag, and one of the following:
  - **Primary address** – Address line 1, city, state, ZIP code
  - **Phone number** – Any of the phone numbers, such as home, cell, fax, or work phone
  - **Tax ID**
  - **Date of birth**
  - **Driver’s license and driver’s license state**
- ABCompany – Name, contact tag, and one of the following:
  - **Primary address** – Address line 1, city, state, ZIP code

- **Phone number** – Any of the phone numbers, such as fax or work phone
- **Tax ID**
- ABCompanyVendor – Name, contact tag, and tax ID
- ABPersonVendor – Last name, contact tag, and tax ID
- ABPlace – Name, contact tag, and primary address, consisting of address line 1, city, state, and ZIP code
- ABUser – Last name, contact tag, and employee number

## Setting the TaxID Requirement

You can set whether or not TaxID is required for creating contacts. In the base configuration, this setting affects only PersonVendor and CompanyVendor subtypes. You set the variable `RequiresTaxID` in the registry file `ValidateABContactCreationPlugin.gwp`.

`ValidateABContactCreationPluginImpl` picks up this value when it makes the following method call:

```
override function setParameters(p0: Map<Object, Object>) {  
    _requiresTaxID = Boolean.valueOf(p0.get("RequiresTaxID") as String)  
}
```

### To set the `RequiresTaxID` variable

1. If necessary, start ContactManager Studio.
2. In the Project window, navigate to **configuration** → **config** → **Plugins** → **registry** and double-click `ValidateABContactCreationPlugin.gwp`.
3. The default registered plugin implementation is in the **Gosu Class** field:  
`gw.plugin.contact.ValidateABContactCreationPluginImpl`.
4. Under the **Gosu Class** field, `RequiresTaxID` is defined in the **Parameters** table. The default value is `true`. If you set it to `false`, TaxID is not required for creating any contact subtype.

## Code Examples for Creating ABContact and ABCompany Contacts

The following code sample shows code defining contact creation requirements for ABContact and ABCompany:

```
protected function abContactIsInvalid(contact : ABContact) : boolean {  
    return contact.Tags == null or contact.Tags.IsEmpty  
}  
  
protected function abCompanyIsInvalid(contact : ABCompany) : boolean {  
    if (abContactIsInvalid(contact))  
        return true  
  
    if (RequiresTaxID) {  
        return contact.Name == null  
        or (isLackingCompleteAddress(contact.PrimaryAddress)  
            and isLackingAnyPhoneNumber(contact)  
            and contact.TaxID == null)  
    } else {  
        return contact.Name == null  
        or (isLackingCompleteAddress(contact.PrimaryAddress)  
            and isLackingAnyPhoneNumber(contact))  
    }  
}
```

## Validation Errors and Displaykeys

If an ABContact instance fails validation, the plugin throws a `TooLooseContactCreateCriteriaException`, which supports a set of display keys based on locale and ABContact subtype.

For general information on working with display keys, see “Display Keys Editor” on page 143 in the *Configuration Guide*.

You can use ContactManager Studio to access the set of display keys provided in the base configuration. In the Project window, navigate to **configuration** → **Localization** → **en\_US** and then double-click **display.properties** to open it in the editor. In the editor, search for **TooLooseContactCreateCriteriaException**.

There is a set of **TooLooseContactCreateCriteriaException** display keys for **ABContact** and some of its subtypes. These display keys are accessible in code. For example:

```
Java.TooLooseContactCreateCriteriaException.ABCompany  
Java.TooLooseContactCreateCriteriaException.ABCompany.ja_JP  
Java.TooLooseContactCreateCriteriaException.ABCompanyVendor  
Java.TooLooseContactCreateCriteriaException.ABCompanyVendor.ja_JP
```

In this example, an **ABContact** that is a **ABCompanyVendor** with locale **us\_EN** fails validation. It throws an exception, using for its message the display key

**Java.TooLooseContactCreateCriteriaException.ABCompanyVendor**. If the Japanese locale is specified, the exception uses the display key **Java.TooLooseContactCreateCriteriaException.ABCompanyVendor.ja\_JP**.

Similar behavior applies to **ABContact** entities that are of class **ABCompany** but not **ABCompanyVendor**. They use the **Java.TooLooseContactCreateCriteriaException.ABCompany** display keys.

## Testing Clock Plugin Interface—Only For Non-Production Servers

**IMPORTANT** The **ITestingClock** plugin interface is supported only for testing on non-production development servers. Do not register an implementation of this plugin on production servers.

To test ContactManager behavior over a simulated long span of time, you can implement the **ITestingClock** plugin interface and programmatically change the system time to simulate the passing of time. For example, you can define a plugin implementation that returns the real time except in special cases in which you artificially increase the time to represent a time delay. The delay could be one week, one month, or one year.

This plugin interface has two methods, **getCurrentTime** and **setCurrentTime**, which get and set the current time using the standard ContactManager format of milliseconds stored in a **long** integer.

If you cannot set the time in the **setCurrentTime** function, throw the exception **java.lang.IllegalArgumentException**. For example, you are using an external time server and temporarily cannot reach it.

Time must always increase, not go back in time. Going back in time is likely to cause unpredictable behavior in ContactManager.

### Notes:

- The plugin method **setCurrentTime** advances the time only for **gw.api.util.DateUtil.currentDate**. The class **java.util.Date** always states the server time. Therefore, you must use **gw.api.util.DateUtil.currentDate** in your implementation code.
- The advanced date does not change in all parts of the product. There are certain areas that intentionally do not use the rolled-over date. For example, the time shown in the next scheduled run for batch processes is not affected.

## Using the Testing Clock Plugin

The base configuration of ContactManager has an implementation of the **ITestingClock** plugin interface. This topic describes how to use that plugin implementation.

1. Start ContactManager Studio.
2. Navigate in the Project window to **configuration** → **config** → **Plugins** → **registry**, and then double-click **ITestingClock.gwp**.

3. In the editor, verify that the following Java class is registered:  
`com.guidewire.pl.plugin.system.internal.OffsetTestingClock`
4. Navigate in the Project window to **configuration → config** and then double-click **config.xml** to open it in the editor.
5. In the **config.xml** file under Environment Parameters, set **EnableInternalDebugTools** to **true**, as follows:  
`<param name="EnableInternalDebugTools" value="true"/>`
6. Save your changes.
7. Start ContactManager, or stop and restart it if necessary.
8. Log in as a user in the role superuser, such as user su with password gw.
9. Press **Alt+Shift+T**, and then click the **Internal Tools** tab.
10. In the sidebar, click **Testing System Clock**.
11. Set the clock as needed, either by clicking one of the **Add** buttons or entering a specific date and time, and then click **Change Date**.

## Using the Testing Clock Plugin in ContactManager Clusters

If you are operating a cluster of ContactManager servers, use the following procedure to change the time.

1. Stop all servers with the exception of the batch server.
2. Advance the testing clock.
3. Restart all the cluster nodes.



# Release Notes Archive

This topic contains the release notes for previous versions of ContactManager. You can use these previous release notes to see which issues have been fixed and how features have changed from one release to the next.

---

**IMPORTANT** This topic contains upgrade information originally provided for earlier ContactManager releases. It might be superseded by later release notes or other upgrade documentation.

---

This topic includes:

- “Guidewire ContactManager 7.0.1 Release Notes” on page 311
- “Guidewire ContactManager 7.0.2 Release Notes” on page 326
- “Guidewire ContactManager 7.0.3 Release Notes” on page 337
- “Guidewire ContactManager 7.0.4 Release Notes” on page 344
- “Guidewire ContactManager 7.0.5 Release Notes” on page 351
- “Guidewire ContactManager 7.0.6 Release Notes” on page 358
- “Guidewire ContactManager 8.0.0 Release Notes” on page 363
- “Guidewire ContactManager 8.0.1 Release Notes” on page 369

## Guidewire ContactManager 7.0.1 Release Notes

### Overview of ContactManager 7.0.1 Release Notes

These release notes contain the following:

- “Release Information for ContactManager 7.0.1” on page 312
- “Installing ContactManager 7.0.1” on page 312
- “Support for ContactManager 7.0.1” on page 312
- “Issues and Major Changes for ContactManager 7.0.1” on page 312
- “Improvements and General Issues for ContactManager 7.0.1” on page 315

- “Known Issues and Limitations for ContactManager 7.0.1” on page 322

## Release Information for ContactManager 7.0.1

These release notes apply only to this release of Guidewire ContactManager.

### Version Number

This release of Guidewire ContactManager is 7.0.1.

- If you are integrating with Guidewire BillingCenter, this version of ContactManager requires BillingCenter 7.0.0 or later.
- If you are integrating with Guidewire PolicyCenter, this version of ContactManager requires PolicyCenter 7.0.1 or later.
- If you are integrating with Guidewire ClaimCenter, this version of ContactManager requires ClaimCenter 7.0.0 or later.

## Installing ContactManager 7.0.1

For general installation information, see “Installing ContactManager” on page 39.

For any versions prior to 7.0.1 that you have skipped, see prior ClaimCenter release notes, which include a section for ContactManager release notes. For example, see “Release Notes Archive” on page 135 in the *New and Changed Guide*.

## Support for ContactManager 7.0.1

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

- At the Guidewire Resource Portal – <http://guidewire.custhelp.com>
- By email – [support@guidewire.com](mailto:support@guidewire.com)
- By phone – +1-650-356-4955

## Issues and Major Changes for ContactManager 7.0.1

This topic contains issues and major changes that might affect your installation.

- “ContactManager Integration with Core Applications” on page 312
- “Geocoding Using Bing and MapPoint (PL-16708)” on page 313
- “Changes in This Release Provided in Upgrade Diff Report” on page 313
- “New Web Services APIs return current AddressBookUID for merged contact (CTC-588)” on page 313
- “Provide web service for validating contacts for creation (CTC-603)” on page 313
- “Make ValidateABContactCreationPlugin return the error message (CTC-611)” on page 314
- “Make ValidateABContactSearchCriteriaPlugin return the error message (CTC-612)” on page 314

### ContactManager Integration with Core Applications

This release of ContactManager provides integrations with Guidewire ClaimCenter, PolicyCenter, and BillingCenter. For more information, see “Integrating ContactManager with Guidewire Core Applications” on page 45.

## Geocoding Using Bing and MapPoint (PL-16708)

Guidewire has added an implementation of the GeocodePlugin that connects to the Microsoft Bing Maps Geocode Service. The Bing Maps plugin implementation replaces the Microsoft MapPoint implementation.

Guidewire has deprecated the MapPoint implementation. Microsoft announced plans to retire the MapPoint web service effective November 18, 2011. If you currently use geocoding features and the MapPoint plugin, you must migrate from MapPoint to Bing Maps. Otherwise, geocoding features in the application will not function.

## Changes in This Release Provided in Upgrade Diff Report

Guidewire provides a report detailing certain differences between the current release and your prior release. This report describes changes in display keys, entities, typelists, and the Gosu API. To obtain your custom Upgrade Diff Report, visit the Guidewire Resource Portal.

## New Web Services APIs return current AddressBookUID for merged contact (CTC-588)

When a contact is retired as part of a merge operation, ContactManager now keeps track of the LinkID of the contact that replaced it. There is no safe ordering on merge messages. Therefore, a subsequent merge might occur before the core application queries ContactManager to retrieve the LinkID for the replacement contact. ContactManager now provides APIs that enable a core application to discover the current LinkID in ContactManager that replaces a LinkID that the application has to a merged contact.

The APIs that ContactManager supplies for this purpose are:

- ABContactAPI.getReplacementContact(contactLinkID : String) : String
- ABContact.getReplacementContact() : ABContact

To access this method, you must fetch updates for the ContactManager web service ABContactAPI in Guidewire Studio for your core application, as follows:

1. Ensure that ContactManager is running.
2. Start Guidewire Studio for your core application.
3. In the Project window, navigate to configuration → Classes → wsi → remote → gw → webservice → ab → ab700.
4. In the editor on the right, click the following resource in the Resources pane:  
 `${ab}/ws/gw/webservice/ab/ab700/abcontactapi/ABContactAPI?wsdl`
5. Click Fetch Updates to get the latest updates to ABContactAPI. If you see a message asking if you want to create a copy of the resource, click Yes.

## Provide web service for validating contacts for creation (CTC-603)

The ContactManager web service ABContactAPI has a new method that determines if the specified contact can be created.

The method, ABContactAPI.validateCreateContact, calls ValidateABContactCreationPlugin to see if the contact has enough data specified to allow it to be created.

The method returns an ABContactAPIValidateCreateContactResult object, which is a Gosu version of the Java object ValidateABContactCreationPluginResult that the ValidateABContactCreationPlugin plugin returns. For more information, see “ValidateABContactCreationPlugin Interface” on page 306.

To use this method, you must fetch updates for the ContactManager web service ABContactAPI in Guidewire Studio for your core application, as described previously for CTC-588. See “New Web Services APIs return current AddressBookUID for merged contact (CTC-588)” on page 313.

## Make ValidateABContactCreationPlugin return the error message (CTC-611)

`ValidateABContactCreationPluginImpl.validateCanCreate` used to return a `Boolean` indicating whether validation passed. Now it returns a `ValidateABContactCreationPluginResult` object that has the following methods:

- `boolean isValid();`
- `String getErrorMessage();`

The code in the class did not change much. If you changed the `validateCanCreate` method itself, you must apply these changes to the new private method `canCreate`. If you changed code elsewhere in the class, you must re-apply those changes.

For more information on this class, see “[ValidateABContactCreationPlugin Plugin Interface](#)” on page 306.

## Make ValidateABContactSearchCriteriaPlugin return the error message (CTC-612)

`ValidateABContactSearchCriteriaPluginImpl.validateCanCreate` used to return a `Boolean` indicating whether validation passed. Now it returns a `ValidateABContactSearchCriteriaPluginResult` object that has the following methods:

- `boolean isValid();`
- `String getErrorMessage();`

The code in the class did not change much. If you changed the `validateCanCreate` method itself, you must apply these changes to the new private method `canCreate`. If you changed code elsewhere in the class, you must re-apply those changes.

## Improvements and General Issues for ContactManager 7.0.1

This topic describes improvements and issues corrected in this release. Guidewire attempts to provide information for issues of primary importance to our customers. This list is not intended to be comprehensive.

ID	Description
<b>ContactManager</b>	
CTC-588	<p>When a contact is retired as part of a merge operation, ContactManager now keeps track of the LinkID of the contact that replaced it. There is no safe ordering on merge messages. Therefore, it is possible that a subsequent merge has occurred by the time the core application queries ContactManager to retrieve the LinkID for the replacement contact. ContactManager now provides APIs to the core applications so they can discover the current LinkID for a LinkID they have to a contact that has been merged.</p> <p>The APIs that ContactManager supplies for this purpose are:</p> <ul style="list-style-type: none"> <li>• ABContactAPI.getReplacementContact(contactLinkID : String) : String</li> <li>• ABContact.getReplacementContact() : ABContact</li> </ul> <p>To access this method, you must fetch updates for the ContactManager web service ABContactAPI in Guidewire Studio for your core application, as follows:</p> <ol style="list-style-type: none"> <li>1. Ensure that ContactManager is running.</li> <li>2. Start Guidewire Studio for your core application.</li> <li>3. In the Resources pane on the left, navigate to configuration → Classes → wsi → remote → gw → webservice → ab → ab700.</li> <li>4. In the editor on the right, click the following resource in the Resources pane: \${ab}/ws/gw/webservice/ab/ab700/abcontactapi/ABContactAPI?wsdl</li> <li>5. Click Fetch Updates to get the latest updates to ABContactAPI. If you see a message asking if you want to create a copy of the resource, click Yes.</li> </ol>
CTC-596	In some cases, the Duplicate Contact Finder batch process was picking the wrong contact as the contact to be kept. It now consistently picks the older of two contacts as the kept contact.
CTC-599	Importing sample data from a locale other than en_US, en_GB, en_AU, and en_NZ, such as ja_JP, was throwing an exception when the withTaxId method was called. This method has been changed to return an empty string for locales other than these four locales.
CTC-603	<p>The ContactManager web service ABContactAPI has a new method that determines if the specified contact can be created.</p> <p>The method, ABContactAPI.validateCreateContact, calls ValidateABContactCreationPlugin to see if the contact has enough data specified to allow it to be created.</p> <p>The method returns an ABCContactAPIValidateCreateContactResult object, which is a Gosu version of the Java object ValidateABContactCreationPluginResult that the ValidateABContactCreationPlugin plugin returns. For more information, see “ValidateABContactCreationPlugin Interface” on page 306.</p> <p>To use this method, you must fetch updates for the ContactManager web service ABContactAPI in Guidewire Studio for your core application, as described previously for CTC-588.</p>
CTC-606	The ContactManager AddressBuilders.withBatchGeocode(true) method was not setting the Address.Geocode field to true for sample code in the database. This method is working correctly now.
CTC-609	Incorrect permission check in ABContactAPI.retrieveRelatedContacts. This method used perm.Contact.viewab. Instead, the correct permission check is perm.ABContact.view because the method deals with ABContact objects and not Contact objects. It now uses the correct permission check expression.
CTC-611	<p>ValidateABContactCreationPluginImpl.validateCanCreate used to return a Boolean indicating whether validation passed. Now it returns a ValidateABContactCreationPluginResult object that has the following methods:</p> <ul style="list-style-type: none"> <li>• boolean isValid();</li> <li>• String getErrorMessage();</li> </ul> <p>The code in the class did not change much. If you changed the validateCanCreate method itself, you must apply these changes to the new private method canCreate. If you changed code elsewhere in the class, you must re-apply those changes.</p> <p>For more information on this class, see “ValidateABContactCreationPlugin Plugin Interface” on page 306.</p>

CTC-612	<p>ValidateABContactSearchCriteriaPluginImpl.validateCanCreate used to return a Boolean indicating whether validation passed. Now it returns a ValidateABContactSearchCriteriaPluginResult object that has the following methods:</p> <ul style="list-style-type: none"> <li>• boolean isValid();</li> <li>• String getErrorMessage();</li> </ul> <p>The code in the class did not change much. If you changed the validateCanCreate method itself, you must apply these changes to the new private method canCreate. If you changed code elsewhere in the class, you must re-apply those changes.</p>
<b>Assignment</b>	
PL-13175	Delegated the UserRoleAssignment entity out to a UserRoleAssignmentDelegate and removed the UserRoleAssignment entity at the platform level. All Guidewire applications now own their own UserRoleAssignment entity and use entity delegation to UserRoleAssignmentDelegate.
PL-18196	Fixed assignGroupByRoundRobin to manage GroupAssignmentState with proper set of keys including intended GroupType to assign a group properly.
PL-18397	Fixed an issue that occurred during the creation of an exposure at the end of FNOL wizard if you were using assignGroupByRoundRobin assignment.
<b>Batch Processes</b>	
PL-13912	Guidewire has added the ability to set an env attribute on the <ProcessSchedule> element in scheduler-config.xml to specify the schedule to run in a certain environment. As a consequence, you can now have different results for batch processing based on environment. By default, Guidewire does not set the env attribute.
PL-15657	<p>Added Last Run Status column to the Processes table of the Batch Process Info screen. This column allows a user to see if the last run of this batch task completed successfully or failed.</p> <p>The possible values are:</p> <ul style="list-style-type: none"> <li>• Completed – The last run has completed successfully.</li> <li>• Not available – This task has not ran yet or running right now.</li> <li>• Failed/Interrupted –The last run has failed or was interrupted. See the history of the process for more details.</li> </ul>
<b>Clustering</b>	
PL-10224	In addition to improvements to JGroups implementation of UNICAST protocol, this fix also introduced a new merge protocol, MERGE4. This is not set in the default protocol stack, in which MERGEFAST is set.
PL-17295	The valid range for multicast address is between 224.0.0.0 and 239.255.255.255. This change verifies the validity of the multicast address before attempting to join the cluster. If the address is invalid, ContactManager throws the following exception:  "Invalid multicast address " + multicastAddress + ". Valid range is between 224.0.0.0 and 239.255.255.255."
PL-17297	Guidewire has upgraded JGroups to the latest version, which is 2.12.1.
PL-17903	Guidewire now requires a unique server ID for each node in a cluster. The application verifies its server ID for uniqueness across all other cluster members as the node starts.
<b>Code Utilities</b>	
PL-18054	If you have used the \${x} substitution in document templates, then you need to revert those uses back to the <%= ... %> syntax.
<b>Configuration Upgrade</b>	
PL-18021	Fixed an issue with how the Upgrade tool automatically handled rule upgrades.
<b>Database</b>	
PL-17368	Database upgrade no longer runs certain statistics commands automatically. Database upgrade no longer includes the following: <ul style="list-style-type: none"> <li>• Upgrade does not update statistics on indexes as both SQL Server and Oracle have options to automatically create statistics on indexes during database creation.</li> <li>• Upgrade does not update statistics on tables and columns for new database.</li> </ul>
PL-17388	Modified the length of the columns created to support linguistic searching, the <i>DENORM</i> columns, reducing them in size from twice the original column's size to exactly the original column's size

PL-18012	Previously, Guidewire prohibited you from connecting a development server to a production database, or connecting a production server to a development database. Guidewire now permits this kind of connection (merely generating a warning) if using the H2 database.
PL-18142	Fixed an issue in the Leap Year Days calculation that caused an infinite loop if times in the start and end of the DateRange did not match.
<b>Database Upgrade</b>	
PL-17732	Fixed performance issues with the upgrade process.
PL-17849	Guidewire has added a DateFromDatetime method to the BeforeUpgradeDBFunction utility class, which is available only in the context of upgrade queries. It is available only in restrictions, not in the query column list. Certain Guidewire applications need this for use in an upgrade trigger. In a BeforeUpgrade query, you can use this function in a SELECT statement, and it can take a nested function as an argument
<b>Datamodel</b>	
PL-17540	It is now possible to add an event to an entity through an extension, even if the base configuration definition of the entity contains one or more events.
PL-7602	Enhanced GosuDoc to catch all throwables thrown when attempting to load a type. Previously, it only caught Runtime Exceptions.
PL-15360	Fixed an issue in which Guidewire allowed @WebServices annotations at the function level in some cases. Now, you can use this annotation at the class level only.
PL-15806	Fixed an issue in which using the == operator in an overridden equals method caused a stack overflow.
<b>Entities/Metadata</b>	
PL-10608	Guidewire now flags as an error in Studio—and as a warning at server start-up—if you set a non-queryable path on the Entity Path field in the Entity Names editor.
PL-17718	Fixed an issue in which a forward slash (/) in a typecode definition prevented the application server from starting, but Studio did not display an error flag for this condition.
PL-18071	Added support for denormalizing columns across tables to enhance the performance of search queries.
<b>Geocoding/Proximity Search</b>	
PL-16708	Guidewire has added an implementation of the GeocodePlugin that connects to the Microsoft Bing Maps Geocode Service. The Bing Maps plugin implementation replaces the Microsoft MapPoint implementation.
PL-17675	Fixed an issue in which the Bing Geocode plugin implementation was not visible in Studio.
PL-17886	The MapPoint implementation is deprecated by Guidewire with this release. Microsoft announced plans to retire the MapPoint web service, effective November 18, 2011. If you currently use geocoding features and the MapPoint plugin, you must migrate from MapPoint to Bing Maps before November 18, 2011. Otherwise, geocoding features in the application cease to function on November 18, 2011, and afterwards.
<b>Gosu</b>	
PL-16935	Fixed an issue in which ContactManager threw a Null Pointer Exception if you attempted to change Gosu code in Studio and it was connected to the application server. (The issue was specific to functions in code blocks that were called from other PCF functions.)
PL-17478	Modified the Gosu Tester parser to handle scriptability modifiers in the same way that Gosu classes and enhancements do. The Gosu Tester no longer shows the following error message for scriptability modifiers:  'The property [or method], "Xxxx", is not visible under the parser's visibility constraints.'
PL-17489	Fixed an issue in which certain corner cases of conditional variable declarations would fail with a Java byte-code VerifyError if the server was running in debug mode. This issue did not affect servers running in production mode.
PL-18033	Fixed an issue that caused incorrect behavior if you called reverse() on an implicitly-declared List in Gosu. (The issue did not occur with an explicitly declared List.) The issue was that the reverse sort on an implicitly-declared List occurred in-place, meaning that the sort occurred on the original list elements. The sort now occurs on a copy of the list, which is the desired behavior.

PL-18299	<p>Added a preload mechanism to support pre-compilation of Gosu classes, as well as other primary classes in the system. The intent is to make the system more responsive the first time requests are made. To support this, Guidewire has added a Studio Other Resources → preload.txt file in which you can add a list of actions to take. The file contains static no-argument method calls, as well as the names of Gosu types to compile to byte-code or the Java types to load.</p> <p>Guidewire also added a new logging category of Server.Preload that provides DEBUG level logging of all actions during server pre-loading of Gosu classes.</p>
<b>GX Tools</b>	
PL-12128	<p>Added the ability for a web service to use types from a different WSDL. For example,</p> <pre>&lt;xs:import namespace="http://example.com/gw/api/test/TheirAPI"     schemaLocation="../../wsdl/local/gw/api/test/TheirAPI.wsdl" /&gt;</pre>
PL-18297	Restored missing eachException() methods on the Guidewire XML Modeler. Guidewire inadvertently removed these methods in a previous 7.x release.
<b>History</b>	
PL-17720	Guidewire no longer marks the HistoryType typelist as final. Therefore, it is now possible to add new values to this typelist. However, Guidewire made this change to allow the removal of the retrieved HistoryType in ClaimCenter. Guidewire recommends that you continue to add custom history types to the CustomHistoryType typelist, rather than the HistoryType typelist.
<b>Integration</b>	
PL-17479	The base configuration no longer contains a suite-config.xml file that contains default URL values of localhost:port/productname. Instead, this file has entries that are commented out. Guidewire considers any product not configured in suite-config.xml, which is the default for all products, to be non-existent in the suite.
PL-16370	The regen-soap-api command now generates RPC-Encoded and WSI artifacts into different directories.
PL-17945	Guidewire has changed the type of the URL field in the suite-config XSD definition file from xs:anyURI to xs:string.
PL-17946	Guidewire now disallows duplicate entries in suite-config.xml. Guidewire defines duplicate entries as those with the same name, URL, and env values.
<b>Logging</b>	
PL-18234	<p>Improved logging of ConcurrentDataChangeException. Now, the log message shows both of the following:</p> <ul style="list-style-type: none"> <li>The username of the current user, who received the ConcurrentDataChangeException</li> <li>The previous user, who made the initial data modification</li> </ul> <p>The log writes the current user's name before the previous user's name in the message.</p>
<b>Messaging</b>	
PL-18211	Fixed an issue in which messages could potentially be sent in incorrect send order if a bundle commit contained multiple event root entities.
<b>Miscellaneous</b>	
PL-13354	Guidewire now certifies Oracle WebLogic for use as an application server for ContactManager. Refer to the Platform Support Matrix for more detail on the exact version that has been certified.
PL-17350	Fixed an issue with regen-soap-api that caused it to not properly generate the template/toolkit-javadoc/gw-cc-plugin directory.
PL-16779	Fixed an issue with regen-soap-api that caused it to fail if specific MQ libraries existed in configuration/plugins/shared/lib or gosu/lib. Executing regen-soap-api with these libraries now logs a warning rather than forcing an outright failure.
PL-17987	Guidewire has removed the extraneous Bundle property on the Bundle object (Bundle.Bundle).
<b>Other - PL Services</b>	

PL-14011	The (Server Tools) Management Beans → CurrentUserSessions information now displays a sorted list of users. If a user is logged in more than once, ContactManager appends the number of sessions to the user name in parentheses. For example:  psmith(2),bsmith  The intent is to preserves all information that is currently available, while providing a view that is easier to scan because of the sorting and the non-repetition of names.
<b>Persistence</b>	
PL-17656	Guidewire has introduced a new class com.guidewire.external.Type to use for Java API methods that require an instance of com.guidewire.external.Type.  You can obtain an instance of this class using the com.guidewire.external.Type#of method by passing in the class representing the desired type. Methods that require an instance of com.guidewire.external.Type need to document what kinds of types are expected.
<b>Plugins</b>	
PL-17162	Guidewire has added interface gw.plugin.SharedBundlePlugin. Any plugin that implements this interface shares the current bundle rather than creating a new one. For example, any plugins that implements IPreUpdateHandler now needs to add gw.plugin.SharedBundlePlugin to their list of implemented interfaces.
<b>Profiling</b>	
PL-17533	Fixed an issue with the (Server Tools) Web Profiler in which the Group Frames result page did not show detail information. The page now shows the detail information.
<b>Queries</b>	
PL-17676	Guidewire has deprecated the RawQuery property. Instead, create a gw.api.filters.StandardQueryFilter and use its filterQuery method or the IQueryBeanResult.addFilter method to apply the filter to a query.
PL-17719	
<b>Rules Infrastructure</b>	
PL-17893	Guidewire has added a new (Server Tools) Info Page that shows the safe persisting order of the Preupdate rules. This page lists the order in which ContactManager runs the Preupdate rules for their root entities.
<b>Security</b>	
PL-10126	Guidewire has added a new Credentials plugin to the base configuration. The CredentialsPlugin is similar to the DBAuthenticationPlugin and consists of the following three components: <ul style="list-style-type: none"> <li>Interface CredentialsPlugin with the following sole method: public UsernamePasswordPairBase retrieveUsernameAndPassword(String key)</li> <li>Class CredentialsUtil with the following sole static method: public static UsernamePasswordPairBase getCredentialsFromPlugin(String key)</li> <li>Class CredentialsPlugin.gs provides a default implementation of the plugin in the base configuration.</li> </ul> Code can call the method CredentialsUtil.getCredentialsFromPlugin, passing in a key to specify for which application the credentials are sought. The static method returns the username/password pair for the desired key. The default implementation currently reads the username/password pair from a Credentials.xml file as follows: <ul style="list-style-type: none"> <li>The code encounters the token \${username} as the value of the username value or the token \${password} as the value of the password value in the properties file. In this case, the code uses the Credentials plugin to obtain the real username and password.</li> <li>The Credentials plugin is not enabled or these tokens are not present. In this case, the code simply uses the values that are present in the properties file for backwards compatibility.</li> </ul> However, you can substitute any mechanism for retrieving username/password pairs in place of the default mechanism. This can include retrieving username/password pairs from a directory service or from encrypted fields in a database.
<b>Studio IDE - Debugger</b>	
PL-17929	Fixed an issue that caused a Null Pointer Exception, causing the interface to freeze. This issue occurred under specific circumstances after connecting Studio to the application server or starting a debugging session on an already-running server.

PL-18106	Fixed an issue in which some actions in Studio, such as setting breakpoints, would block for a short while waiting for a server response, but would not provide any messages.  Guidewire now shows a progress dialog to indicate the delay.
PL-18110	Previously, if the connection between Studio and the application server was lost for some reason, such as terminating the server session, Studio did not terminate the debugging session. Therefore, after server restart, it was possible for a user to believe that debugging was still ongoing, even though, in reality, the debugger needed to be reset.  The new behavior is that Studio properly terminates the debugging session if it loses connection to the server, and then Studio notifies the user. The user must manually reactivate the debugger after connection to the application server is restored.
PL-18112	Previously, a web service error while activating the Server debugger could cause the activation to fail, but Studio could show the debugger as active, even it was not. Studio now notifies you of the failure and properly puts the debugger into an inactive state.  Guidewire has also put into place additional recovery mechanisms from certain types of web services errors, making it less likely for the debugger connection to fail in the first place.
PL-18116	Fixed an issue that caused Studio to hang if you were debugging the server and the Gosu Tester debugger was also active. It also occurred if you had an active Gosu Tester debug session, then opened the Guidewire Studio connection dialog and immediately canceled. In that case Studio terminated the Gosu Tester debug session as well.
PL-18275	Fixed a problem with the remote tester debugger hanging if you disconnected it from the server during an active debug session.
PL-18298	Fixed a problem that could cause the Studio tester window to freeze up. The window could freeze if the user was using the tester debugger while connected to a remote server and modified some code in Studio and saved it.

**Studio IDE - Other**

PL-16798	Modified the Rule Condition parser to accept a statement list, instead of a simple expression only. However, the statement list must contain a return statement. For example:  <pre>uses java.util.HashSet uses gw.lang.reflect.IType  var o = new HashSet&lt;IType&gt;() {A, B, C, ...} return o.contains(typeof(...))</pre>
PL-17386	Added the ability to inspect inner class values in Gosu while debugging in Studio.
PL-17606	Fixed an issue in which Studio incorrectly handled creating a View entity extension.
PL-18107	Studio now shows a broken server connection state (icon) if it loses a server connection, for example, if the user terminates the server process. <ul style="list-style-type: none"> <li>• It detects lost connections very quickly if debugging as there is immediate feedback from the debug API.</li> <li>• It detects lost connections more slowly (but, with not more than 1 minute of delay), if not debugging, as Studio queries the server for status at one minute intervals.</li> <li>• If Studio detects the server has restarted, Studio reconnects automatically.</li> </ul>
PL-18109	Fixed a problem that could occur if Studio was connected to a server that was not responding for a period of time. Multiple threads would pile up in the Studio process waiting for the server to respond.  Guidewire has improved how it handles restoring the server connection with Studio. It now properly handles reconnecting to a server that was previously connected, but became unavailable for a period of time, and then became available again.
PL-18198	Fixed an issue in which that could occur if you set breakpoints in a Studio session and shut down and restarted Studio. Studio would lose the breakpoints between sessions if you activated the Server debugger after starting the application server, then shut down and restarted Studio.
PL-18227	Fixed an issue in which ContactManager stored the initial value of a newly created script parameter in an incorrect location, thus invalidating the application installation checksum. ContactManager did correctly store subsequent entries in the correct location.

**Studio IDE - PCF Editor**

PL-13845	Guidewire has modified the PCF New Folder dialog so that it is no longer possible to create a folder that differs in case only with an existing folder.
----------	---

PL-11489	Guidewire has added a new PCF Verification option <b>Tools → Options → Verification Options</b> . Use it to enable or disable whether ContactManager limits the number of included sections in second-pass PCF verification. Second-pass verification can cause performance issues with combinations of modal PCF files. For details, see “Setting Verification Options” on page 95 in the <i>Configuration Guide</i> .
PL-17576	<p>Fixed an issue that caused Studio to throw an <code>AssertionError</code> while attempting to load a file, such as a PCF file, into Studio. If the file was modified outside Studio and a stale version of that file was already open in Studio, you would see this error on return to Studio. Studio now saves an open file properly on losing focus, and the error does not occur.</p> <p>Before this fix, this error could happen if:</p> <ul style="list-style-type: none"> <li>You opened a file in Studio and shifted focus to another application such as IntelliJ. You then modified and saved the file and shifted focus back to Studio.</li> <li>You opened a file in Studio and shifted focus away from Studio. Then, through a source control management system, you reloaded a modified version of the file into the local file system and shifted focus back to Studio.</li> </ul> <p>In each case, there was a mismatch between the file stored locally and the version of the file contained in Studio memory. The mismatch happened because Studio did not save the open file properly on losing focus.</p>
<b>Utilities</b>	
PL-17920	Guidewire has removed the following JAR files from the application build: <ul style="list-style-type: none"> <li>activation.jar</li> <li>activation-1.1.1.jar</li> </ul>
PL-17922	Guidewire has removed the following JAR files from the Guidewire application builds as the standard Java EE install contains these files: <ul style="list-style-type: none"> <li>mailapi.jar</li> <li>smtp.jar</li> </ul>
<b>Web - Other</b>	
PL-17861	Added a <code>disablePostOnEnter</code> attribute to the Input Group widget.
<b>Web - UI/Runtime</b>	
PL-18233	Fixed an issue that failed to highlight (in yellow) a problematic <code>RadioButtonCell</code> field even if the data value for the <code>RadioButtonCell</code> contained an error.
<b>Web Services - WSI</b>	
PL-12215	When you place a WSDL from a web service publisher anywhere in the Gosu class path, Guidewire generates Gosu types from the WSDL elements. A problem in which these generated Gosu types could not be used successfully has been fixed.
PL-16577	To authenticate using HTTP authentication on WebLogic, add the following inside the <code>security-configuration</code> tag of WebLogic's <code>config.xml</code> :
	<code>&lt;enforce-valid-basic-auth-credentials&gt;false&lt;/enforce-valid-basic-auth-credentials&gt;</code>
PL-17412	Guidewire now provides a new class, <code>WsRequestLocal</code> , for use with webservice request-based data storage. You can use this class to communicate values between various parts of a webservice implementation in the scope of a single request.
PL-17568	Fixed an XML parse exception ( <code>gw.xml.XmlException</code> ) that occurred if you invoked a web service that used a schema with multiple imports
PL-17601	Restored the functionality to extend an interface by a Gosu type. The WSDL was incorrectly overriding the final methods of the <code>Object</code> class ( <code>wait</code> , <code>notify</code> , and similar methods). Eventually, when these WSDLs were transformed into Java classes, the compilation of the Java class failed because the final methods on <code>Object</code> could not be overridden. The fix was to not override these methods in WSDL generation.
PL-17902	Guidewire has added a check for the existence of the generated WSDL directories.
<b>Workflow</b>	

PL-9668	<p>Added new configuration parameter <code>WorkflowLogDebug</code>, which takes a Boolean value:</p> <ul style="list-style-type: none"><li>• If set to <code>true</code>, ContactManager outputs the ordinary verbose system workflow log messages from the Guidewire server to the workflow log.</li><li>• If set to <code>false</code>, ContactManager does not output any of the ordinary system messages.</li></ul> <p>The setting of this parameter does not have any effect on calls to log workflow messages made by customers. Therefore, all customer log messages are output. If customers are experiencing too many workflow messages being written to the <code>cc_workflowlog</code> table, Guidewire recommends that you set this parameter to <code>false</code>.</p>
PL-17556	<p>It is now possible to run the following writer batch processes from either the (Server Tools) <b>Batch Process Info</b> or (Server Tools) <b>Work Queue Info</b> pages:</p> <ul style="list-style-type: none"><li>• Activity Escalation</li><li>• Group Exception</li><li>• User Exception</li></ul>
<b>XML</b>	
PL-17275	The Gosu tester in Studio now properly handles control characters in program output when connected to a running server.
PL-17934	Fixed an issue in which the XML subsystem did not parse complex types correctly if the complex types within XSD files made circular reference each other.

## Known Issues and Limitations for ContactManager 7.0.1

This section describes known issues with this release of Guidewire ContactManager.

- “ContactManager Known Issues” on page 322
- “Studio/Platform Known Issues” on page 323

**Note:** For maintenance releases, Guidewire often defers fixing configuration issues if they require merging files during the upgrade. Workarounds to many of these issues are listed in the following sections. The goal of this policy is to make upgrades as straightforward as possible.

### ContactManager Known Issues

#### **Customizing contact relationship handling to use messaging results in two relationship records being created (CTC-396)**

**Issue** – Changing a contact relationship updates two contact records. However, ContactManager stores the representation of the relationship in one `ContactContact` entity. ContactManager does not check to make sure the relationship does not already exist. Therefore ContactManager can create multiple, redundant `ContactContact` entities for the same relationship if it receives messages changing the relationship between two contacts.

**Workaround** – In the base configurations of d ContactManager, this behavior is not a problem. ClaimCenter makes a direct web service call to ContactManager to update contact relationships and does not use event messaging for related contact updates.

If you customize ClaimCenter and ContactManager to use event messaging to update related contacts between ClaimCenter and ContactManager, your customization must account for the extra message. Updating one relationship changes two contacts, and therefore ClaimCenter generates two messages and sends them to ContactManager. Your customization must ensure that ContactManager creates only one `ContactContact` entity in this case.

## Studio/Platform Known Issues

### Issues with Internet Explorer 9

**Issue** – If you are using the Internet Explorer 9 browser, it is possible to see issues such as screen flickering or an incorrect tab order for fields. According to public reports, Internet Explorer 9 exhibits these and other issues with a variety of web sites and web applications.

**Workaround** – Because this is the behavior of the Internet Explorer 9 rendering engine, Guidewire cannot address these issues. However, there are reports of an Internet Explorer 9 workaround that reduces these issues. Change the new Internet Explorer 9 Accelerated Graphics settings on the Advanced tab of the Internet Options dialog.

### First time you click the arrow of the typekey input, the drop-down menu does not open (PL-10134)

**Issue** – The drop-down menu does not open on the first click of the arrow on a typekey input. Instead, the help text opens.

**Workaround** – Turn off help text on focus by setting InputHelpTextOnFocus to false in the config.xml file. After you do that, the help text shows only if you mouse over the input, and it does not interfere with opening a drop-down menu.

### XML API upgrade feature missing from documentation (PL-10257)

**Issue** – The *Integration Guide* describes a new set of XML APIs based on the XmlElement class. (Legacy APIs are based on the XmlNode class.) You can continue to use the legacy APIs. However, the *Integration Guide* omits mentioning an additional upgrade-specific feature.

**Workaround** – For backwards compatibility only, you can import an XML schema into the Gosu type system using the legacy XML system by following these instructions:

1. Copy:

```
ContactManager/modules/pl/config/registry/compatibility-xsd.xml
```

To:

```
ContactManager/modules/configuration/config/registry/compatibility-xsd.xml
```

2. Add an entry for your schema. Set the value of the namespace attribute to the Gosu package name of the schema. For example, if the schema is in the package location my.package and is called myschema.xsd, set the value of namespace to my.package.myschema.

### ListDetailPanel throws exception (PL-10316)

**Issue** – It is possible for ContactManager to throw an exception if the user cancels out of a ListDetailPanel widget if StartInEditMode is also True.

**Workaround** – Set StartInEditMode to False for the screen that contains the ListDetailPanel. As a consequence, the user must click Edit to modify that screen.

### Countries configured in zone-config.xml still generate a warning during regen-dictionary even when zone data is loaded for all these countries (PL-11947)

**Issue** – Countries configured in zone-config.xml still generate a warning during regen-dictionary even when zone data is loaded for all of these countries.

**Workaround** – Warning message is created in error and can safely be ignored.

### There is a length limitation on entity localization table names (PL-13360)

**Issue** – There is a length limitation on entity localization table names.

**Workaround** – Ensure that the localization tableName property specified in the entity extension file is less than 16 characters. If the localization table name exceeds the maximum length, the error message indicates that 18

characters are allowed. However, the error message does not account for two additional characters added by the application.

**GX models that reference virtual fields and enhancements throw null pointers if null (PL-13560)**

**Issue** – The GX models that reference virtual fields and enhancements throw null pointers when these fields and enhancements are null.

**Workaround** – Include null checks and error handling to prevent referenced virtual fields or enhancements that are null from causing null pointer exceptions.

**Sending email with file attachment with unicode filename is not correctly handed over to the mail server (PL-13582)**

**Issue** – An email with a file attachment that has a unicode file name is not sent to the mail server correctly.

**Workaround** – Use Latin characters for file names on attached files.

**JavaToolkit.gs has incorrectly hard-coded memory, which results in failed regen-java-api Ant task (PL-13663)**

**Issue** – The JavaToolkit.gs file has hard-coded memory, which can result in failed regen-java-api Ant tasks.

**Workaround** – Increase the size of the maximum heap setting on line 161 of JavaToolkit.gs in the Ant module. The default value is 512.

**Problem with regen-java-api command and JAR files (PL-16351)**

**Issue** – If you run the ContactManager/bin/gwab regen-java-api command, ContactManager creates a ContactManager/java-api/lib directory and puts JAR files for Java development in this directory. However, ContactManager does not always perform this task correctly, resulting in some of the generated JAR files' not being copied to this directory.

**Workaround** – For missing library files, copy or import into your Java development environment the missing library files from the following directory:

ContactManager/admin/lib

Copy them into the following directory:

ContactManager/java-api/lib

**Studio Rules do not use correct capitalization for root object's name (PL-10740)**

**Issue** – Rule set root objects are not named with first letter lower-cased.

**Workaround** – The rules engine issues a warning if the correct case for objects is not being used.

**User interface cannot handle starting multiple instances of a batch process (PL-12372)**

**Issue** – The user interface cannot handle starting multiple instances of a batch process.

**Workaround** – To execute multiple instances of a batch process, start them from the command line. Also, to allow multiple instances to be run simultaneously, you must ensure that the BatchProcess.isExclusive() method returns false.

**Type system refresh after PCF page title change does not update corresponding menu label (PL-13057)**

**Issue** – The type system refresh after a PCF page title change does not update the corresponding menu label.

**Workaround** – After updating a page title, restart the server to refresh menu labels and avoid null pointer exceptions due to stale references.

**US-Locations.txt file with the US geodata from GreatData has special characters that cause validation problems with United States Postal Service (USPS) data (PL-13384)**

**Issue** – The US-Locations.txt file contains information that does not conform to United States Postal Service (USPS) standards for bulk mailings.

**Workaround** – The provided US-Locations.txt file is intended only for use in geocoding to identify addresses for a location. You can edit the US-Locations.txt file to conform to your particular address standards, and then import that version of the file instead.

**GX model generated XSD cannot be parsed by JAXB (PL-13598)**

**Issue** – XSD generated by the GX model cannot be parsed by JAXB.

**Workaround** – Add JAXB annotation elements to the XSD to specify the necessary metadata, such as class names, to enable JAXB to generate the Java class files. Contact Guidewire Support for a sample XSD file that is annotated this way.

**Cannot make a field from a delegate into a localized column (PL-13761)**

**Issue** – You cannot make a field from a delegate into a localized column.

**Workaround** – Move the column to be localized off the delegate and onto each of the implementing entities. Then, to make the column appear as though it exists on the delegate, define an enhancement property on the delegate that *delegates* to the appropriate column.

**Studio test functionality not working correctly (PL-15153)**

**Issue** – If you run a test in the Tests folder in the Studio Resources tree, you get the following exception:

```
Using Test Environment Delegate: com.guidewire.testharness.ConfigEnvTestEnvironmentDelegate
gw.internal.gosu.parser.RuntimeExceptionWithNoStackTrace: java.lang.ClassNotFoundException:
    qa.DothisTest
Caused by: java.lang.ClassNotFoundException: qa.DothisTest
at gw.internal.gosu.parser.TypeLoaderAccess.getIntrinsicTypeByFullName(TypeLoaderAccess.java:522)
at gw.internal.gosu.parser.TypeSystemImpl.getByFullName(TypeSystemImpl.java:139)
at gw.lang.reflect.TypeSystem.getByName(TypeSystem.java:116)
at com.guidewire.studio.junit.ui.RunTestCommand.buildTestSuite(RunTestCommand.java:109
...
...
```

**Workaround** – Run the following command before you run a test:

```
.../bin gwcc dev-deploy
```

**Renaming method or property throws ParseResultsException (PL-16633)**

**Issue** – If you rename a property or method or you change a method signature, and a workflow references that property or method in a Gosu field, ContactManager throws ParseResultsException. This is the intended behavior.

**Workaround** – Restart the workflow engine. To do so:

1. Log into ContactManager using an administrative account.
2. Access Internal Tools → Reload.
3. Click Reload Workflow Engine.

**Deploying EAR File on WebSphere 7.0.0.15 Generates Error Message (PL-18613)**

**Issue** – The following steps generate an error message if you attempt to deploy a Guidewire-generated EAR file to WebSphere 7.0.0.15.

1. Generate an EAR file using the following command:

```
gwcc build-websphere-ear
```

2. Deploy the EAR file to WebSphere.

The deployment fails with an error message similar to the following:

```
com.ibm.websphere.management.application.client.AppDeploy  
Exception: com.ibm.websphere.management.application.client.AppDeploymentException:  
ADMA0207E: EE 5 module ab.war in ear file contains unsupported xmi format bindings file.
```

**Workaround** – Add the following at the top of the web.xml file before attempting to generate the EAR file.

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  
"http://java.sun.com/dtd/web-app_2_3.dtd">
```

## Guidewire ContactManager 7.0.2 Release Notes

### Release Notes Update: 04-June-2012

---

**IMPORTANT** These release notes replace the release notes that were included in the official product release. Please disregard the earlier version of the release notes.

---

### Overview of ContactManager 7.0.2 Release Notes

These release notes contain the following:

- “Release Information for ContactManager 7.0.2” on page 326
- “Installing ContactManager 7.0.2” on page 326
- “Support for ContactManager 7.0.2” on page 327
- “Issues and Major Changes for ContactManager 7.0.2” on page 327
- “Improvements and General Issues for ContactManager 7.0.2” on page 328
- “Known Issues and Limitations for ContactManager 7.0.2” on page 334

### Release Information for ContactManager 7.0.2

These release notes apply only to this release of Guidewire ContactManager.

#### Version Number

This release of Guidewire ContactManager is 7.0.2.

- If you are integrating with Guidewire BillingCenter, this version of ContactManager requires BillingCenter 7.0.0 or later.
- If you are integrating with Guidewire PolicyCenter, this version of ContactManager requires PolicyCenter 7.0.1 or later.
- If you are integrating with Guidewire ClaimCenter, this version of ContactManager requires ClaimCenter 7.0.0 or later.

### Installing ContactManager 7.0.2

For general installation information, see “Installing ContactManager” on page 39.

For versions of ContactManager prior to 7.0.2 that you have skipped, see:

- “Guidewire ContactManager 7.0.1 Release Notes” on page 311

For versions prior to 7.0.0, see the ClaimCenter release notes for that version, which include sections for ContactCenter release notes. For example, see “Release Notes Archive” on page 135 in the *New and Changed Guide*.

## Support for ContactManager 7.0.2

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

- At the Guidewire Resource Portal – <http://guidewire.custhelp.com>
- By email – [support@guidewire.com](mailto:support@guidewire.com)
- By phone – +1-650-356-4955

## Issues and Major Changes for ContactManager 7.0.2

This topic contains issues and major changes that might affect your installation.

- “Changes in This Release Provided in Upgrade Diff Report” on page 313

### Changes in This Release Provided in Upgrade Diff Report for ContactManager 7.0.2

Guidewire provides a report detailing certain differences between the current release and your prior release. This report describes changes in display keys, entities, typelists, and the Gosu API. To obtain your custom Upgrade Diff Report, visit the Guidewire Resource Portal.

## Improvements and General Issues for ContactManager 7.0.2

This topic describes improvements and issues corrected in this release. Guidewire attempts to provide information for issues of primary importance to our customers. This list is not intended to be comprehensive.

ID	Description
<b>Batch Process</b>	
PL-18232	It is now possible to extend any WorkQueue subclass exposed in ContactManager.
PL-18404	<p>Removed the BatchServer configuration parameter from config.xml. Instead, use the JVM argument or registry element <code>isBatchServer</code> to set the batch server.</p> <p>Also, added the ability to promote a non-batch server to be the batch server on the ClusterInfo page. If a non-batch server is found in the cluster, you can click PromoteToBatch for that server to make that particular server the batch server.</p> <p>Also, added the <code>promoteToBatch</code> operation to the Cluster Management Bean.</p>
PL-18637	<p>The default ContactManager behavior for query builder result sets is to retrieve all entries from the database into the application server. If the result set is quite large, this can cause problems. Now, you can use the following method to set the query retrieve chunk size.</p> <pre>public final void setChunkingByID(IQueryResult queryResult, int chunkSize)</pre>
<b>Clustering</b>	
PL-11554	<p>Modified how Guidewire applications discover and maintain the batch server in the clustered environment. Prior to this change, the batch server was discovered with the help of a message exchange through the JGroups cluster communication channel. Users starting up their nodes at approximately the same time could end up with two or more batch servers in the cluster if cluster was not completely formed yet.</p> <p>This changes uses the underlying database to discover and to keep track of the batch server currently active in the cluster.</p>
PL-19141	Improved the usability of the Cluster Info screen on the Server Tools tab. The screen now shows Server ID and host name information. Guidewire also changed the menu slightly.
<b>Cognos Integration</b>	
PL-19226	The ab cognos command now builds an application-specific Cognos integration ZIP file, with a name of the form <code>ab-cognos-reporting-integration.zip</code> . Each zip file contains application-specific integration scripts, <code>ab-config</code> and <code>ab-import</code> . These scripts no longer require the namespace/application parameter.
<b>Command Line Tools</b>	
PL-13663	Removed hard-coded Java heap settings that could cause out-of-memory issues when running certain tools.
PL-18225	Added WSI document literal web services support to Administration command-line tools.
PL-18856	Fixed an issue that involved regenerating the Data Dictionary. Running <code>gwab regen-dictionary</code> did not actually produce the Data Dictionary files.
PL-18892	<p>Guidewire has added the following optional argument to the data dictionary generation tool:</p> <pre>maxSPVInclusions</pre> <p>This value of this parameter defines the depth for second pass verification that limits the number of shared sections that are included the verification of PCF types using <code>verify-a11</code>. For example:</p> <pre>gwab regen-dictionary -DmaxSPVInclusions=1000</pre> <p>For this case, the second pass compilation of PCF files stops after 1000 permutation of modal PCF files.</p>
<b>Note</b>	
	<ul style="list-style-type: none"> <li>If you do not include this option, the dictionary generation tool behaves as before.</li> <li>Use only positive integer values for the <code>maxSPVInclusions</code> property.</li> </ul>
<b>Configuration Upgrade</b>	
PL-12804	Fixed an issue in which the upgrade tool did not handle rules upgrade properly.
<b>Consistency Checker</b>	

PL-16199	One-to-one relationships can have at most one entity on each side of the relationship. Under certain conditions, Guidewire is unable to create a unique index to enforce this. In those cases, Guidewire now creates a consistency check to verify the integrity of the data.
<b>Contact Domain</b>	
PL-18245	Fixed an issue with <code>IgnoreProperty</code> used in <code>contact-sync-config.xml</code> that caused it to take effect on all subtypes of <code>Contact</code> , rather than only those specified. This functionality now works properly and allows fields to be ignored on contact synchronization only for the specified subtypes.
<b>ContactManager</b>	
CTC-957	The minimum search criteria for an <code>ABPerson</code> entity have been changed. They now require that there be a last name or partial last name specified if a first name or partial first name has been specified in the search. Minimum search criteria for contacts are defined in the <code>ContactManager</code> class <code>gw.plugin.contact.ValidateABContactSearchCriteriaPluginImpl</code> .
CTC-967	There was a problem with setting the <code>appscale</code> attribute for <code>MoneyDataType</code> . The setting, specified in <code>datatypes.xml</code> , was being ignored when a column of type <code>money</code> was retrieved. This setting is now working. The <code>appscale</code> attribute controls the number of digits shown to the right of the decimal point for money or currency in single currency mode. It is similar to the <code>scale</code> attribute. It must be smaller than the <code>scale</code> setting and overrides the <code>scale</code> attribute if defined.
<b>Database</b>	
PL-1541	ContactManager now explicitly names primary key constraints by using the following conventions: <ul style="list-style-type: none"> <li>Oracle – Primary key constraint on the table <code>PX_CONTACT</code> will be named <code>PK_CONTACT</code>.</li> <li>All other databases – Primary key constraint on the table <code>PX_CONTACT</code> will be named <code>PX_CONTACT_PK</code>.</li> </ul>
PL-16199	One-to-one relationships are required to have at most one entity on each side of the relationship. Under certain conditions, Guidewire is unable to create a unique index to enforce this requirement. In those cases, Guidewire now creates a consistency check to verify the integrity of the data.
PL-17234	ContactManager no longer creates shadow tables—tables whose names start with <code>XXt_</code> and <code>XXtt_</code> —for new databases. If you start the application server in dev mode, ContactManager now creates shadow tables only if you set the <code>server.running.tests</code> system property to true, either explicitly or programmatically. The JVM parameter for starting the application server in dev mode is <code>-Dgw.server.mode=dev</code> .
PL-17286	Guidewire provides a newer version of the Oracle JDBC driver in the Guidewire application distribution. There is a manual copy required if you are using an external connection pool instead of the Guidewire-bundled connection pool. In this case, you must copy the new driver from the distribution to the location from which the application server data source will pick up the Oracle JDBC driver.
PL-17618	ContactManager now logs the reason for generating a certain step during a database upgrade to the server console log at the DEBUG level. This change also adds this information to the <b>Upgrade Info</b> screen on the <b>Server Tools</b> tab.
PL-18637	The default ContactManager behavior for query builder result sets is to retrieve all entries from the database into the application server. If the result set is quite large, this behavior can cause problems. Now you can use the following method to set the query retrieve chunk size.  <code>public final void setChunkingByID(IQueryResult queryResult, int chunkingSize)</code>
PL-18749	With this release, whenever a database upgrade begins, the upgrade process marks the database with the time the upgrade started and the host and batch server from which it began. At the time the database upgrade succeeds, the upgrade process removes the marker. If a second database upgrade begins before the first upgrade finishes, the second upgrade process detects the marker from the first upgrade process and throws an exception.  A second upgrade process can begin if the first upgrade fails and you attempt a restart without first restoring the database. Alternatively, a second upgrade process can begin if a second batch server begins a database upgrade before the first upgrade process completes successfully.
PL-19421	Fixed an issue on SQL Server that caused a stack trace overflow if both of the following were true: <ul style="list-style-type: none"> <li>There was a connection problem during application start-up.</li> <li>Configuration parameter <code>MigrateToLargeIDsAndDatetime2</code> was set to false.</li> </ul>
<b>Document Management</b>	
PL-18360	A new version of the Guidewire Document Assistant ActiveX control will be downloaded to the client browser.
PL-18432	Improved performance for retrieval of extremely large documents through the ActiveX Guidewire Document Assistant.

PL-18653	Guidewire has upgraded the BFO PDF library to version 2.11.20. Refer to the following web site for more details on the changes in this version: <a href="http://bfo.com/viewtext.jsp?url=products/pdf/docs/CHANGELOG.txt&amp;title=Big+Faceless+PDF+Library+Changelog">http://bfo.com/viewtext.jsp?url=products/pdf/docs/CHANGELOG.txt&amp;title=Big+Faceless+PDF+Library+Changelog</a>
----------	--

PL-18704	Microsoft Word field forms—text fill-in fields—are now restored correctly while generating a document from a Microsoft Word Template.
----------	---

#### Entities/Metadata

PL-14982	Modified the Studio Tools → Verify... feature so that it now reports errors and warnings during verification of entity names.
PL-18162	Guidewire has added a validator that prevents you from attempting to denormalize a localized column. Guidewire does not support localized columns for search denorm columns.
PL-18266	Modified the entity type loader to dynamically generate interfaces for the following: <ul style="list-style-type: none"><li>• Entities that do not have a class</li><li>• Entities that are extended with a delegate or Java interface that the compile-time backing class does not extend</li></ul> The type loader now uses this runtime interface to create array instances. The runtime <code>impl</code> class now implements this dynamically generated interface as well.
PL-18469	Added a new attribute called <code>typelistTableName</code> to entity types. Use this attribute for non-final entities to specify the name of the corresponding subtype typelist table. If not specified, ContactManager uses the name of the entity as the subtype typelist table name. This capability is useful if an entity name is too long to become a valid typelist table name.
PL-19655	Fixed a problem with columns defined as <code>oneToOne</code> that disregarded <code>nullOk = false</code> . Now, ContactManager prevents you from committing empty <code>oneToOne</code> columns that do not allow nulls.
PL-20217	There was a problem with setting the <code>appscale</code> attribute for <code>MoneyDataType</code> . The feature worked for <code>currencyamount</code> columns, but not for <code>money</code> columns. The setting, specified in <code>datatypes.xml</code> , was being ignored when a column of type <code>money</code> was retrieved. This setting is now working.  The <code>appscale</code> attribute controls the number of digits shown to the right of the decimal point for money or currency in single currency mode. It is similar to the <code>scale</code> attribute. It must be smaller than the <code>scale</code> setting and overrides the <code>scale</code> attribute if defined.

#### Geocoding/Proximity Search

PL-1461	Two parameters have been added to the plugin registry for the Geocode plugin. The <code>geocodeDirectionsCulture</code> parameter specifies the locale for geocoded addresses and routing instructions returned from a geocoding and routing service. For example, use the locale code <code>ja-JP</code> for addresses and instructions for Japan. The <code>imageryCulture</code> parameter specifies the language for map imagery. For example, use the language code <code>ja</code> for maps labeled in Japanese.
PL-18619	Proximity search works around some bad execution plans by disabling index fast full scan and hash join if executing on Oracle. New configuration parameters <code>DisableIndexFastFullScanForProximitySearch</code> and <code>DisableHashJoinForProximitySearch</code> control the workaround. The default value for these parameters is <code>false</code> . These parameters have no effect on databases other than Oracle.

#### Global Cache

PL-18322	Fixed an issue with method <code>ProximitySearchQueryUtils.filterWithinRadiusLatLong</code> that did not properly return all the requested points. This issue affected radius searches with a smaller radius value than the number of desired results returned.
----------	---

#### Internal Tools/Server Tools Pages

PL-18017	Guidewire has made the Internal Tools tab available in test mode.
PL-18839	Guidewire has significantly improved the performance of the GW Profiler.
PL-18845	Modified the Batch Process Info screen on the Server Tools tab to enable you to download detailed records for any particular batch process for a specific date range. Clicking Download now opens a screen in which you can specify the date range to download for a given batch process.

#### Localization

PL-18490	Modified <code>localization.xml</code> to let the <code>thousandsSymbol</code> attribute of the <code>&lt;NumberFormat&gt;</code> element type accept a non-breaking space character as the thousands separator. For example:  <code>&lt;NumberFormat thousandsSymbol=" "&gt;</code>
----------	--

#### Logging

PL-14722	<p>Added the following new configuration parameters that configure application logging:</p> <ul style="list-style-type: none"> <li>• LoggerCategorySource</li> <li>• LoggersShowLog4j</li> </ul>
PL-18745	<p>Added support for two new Log4j MDC (Mapped Diagnostic Contexts) keys to include information about the current user in log messages. To use them, include a sequence conforming to the following string in your <code>ConversionPattern</code> in <code>logging.properties</code>:</p> <pre>%-&lt;LL&gt;.&lt;HH&gt;X{user   userName   userID}</pre> <p>The parameters have the following meanings:</p> <ul style="list-style-type: none"> <li>• &lt;LL&gt; – The minimum size of the field. If the actual value is shorter, the user name gets padded with spaces on the right.</li> <li>• &lt;HH&gt; – The maximum size of the field. If the actual value is longer, the user name gets truncated from the left.</li> <li>• user – Prints the user's internal numeric ID number, such as 4231341234. This parameter was available in previous releases and remains unchanged for compatibility.</li> <li>• userName – Prints the user's real-world name, such as John Smith.</li> <li>• userID – Prints the user's username in the system, such as jsmith.</li> </ul> <p>For example:</p> <pre>%-16.16X{userName}</pre>
PL-20907	<p>Guidewire updated the logging API between the following releases:</p> <ul style="list-style-type: none"> <li>• BillingCenter 7.0.1 and 7.0.2</li> <li>• ClaimCenter 7.0.1 and 7.0.2</li> <li>• PolicyCenter 7.0.3 and 7.0.4</li> </ul> <p>Guidewire has created the following Knowledge Base article that describes the changes made to the logging API:</p> <p><i>Changes to the logging API during upgrade to BillingCenter 7.0.2, ClaimCenter 7.0.2, and PolicyCenter 7.0.4 or later versions</i></p> <p>Review this article if you are performing an upgrade, in order to update your configuration to the new logging API. If necessary, contact Guidewire Support for a copy.</p>
<b>Messaging</b>	
PL-18678	<p>The <code>MessageSenderRunnable.run</code> method now includes the <code>messageID</code> in the log. The log entry will look similar to the following:</p> <pre>MW.MessageSenderRunnable.run (dest destination_id):     Entering run() for messageId message_id</pre>
PL-19136	Fixed an issue in which a race condition in a non-clustered server could lead to inconsistent results or runtime exceptions or both when modifying entities within a message transport.
PL-19149	After locking out a user who has reached the maximum number of allowed login failures, ContactManager no longer ignores generated events. Instead, ContactManager executes the transaction with a service token having a super user. As a result, ContactManager now generates UserChanged events properly.
<b>Miscellaneous</b>	
PL-3760	<p>Guidewire has made the following modification to <code>config.xml</code>:</p> <ul style="list-style-type: none"> <li>• Removed element <code>&lt;security sessiontimeoutsecs="10800"/&gt;</code></li> <li>• Added configuration parameter <code>&lt;param name="SessionTimeoutSecs" value="10800"/&gt;</code> as its replacement</li> </ul> <p>ContactManager performs an automatic upgrade of <code>config.xml</code> for this change.</p>

PL-15462	<p>Guidewire applications provide several configuration parameters that accept a comma-separated list of values.</p> <p>Now, ContactManager parses a comma-separated list of values according to CSV format and trims each value automatically. This change enables you to use spaces, tabs, and new lines for more readability while specifying such a value.</p> <p>For example:</p> <ul style="list-style-type: none"> <li>• A value that before the fix had to be without spaces – AUTO, PR, GL, TRAV</li> <li>• The same value after the fix can use spaces – AUTO, PR, GL, TRAV</li> </ul> <p>The following is a more technical description of the format:</p> <ul style="list-style-type: none"> <li>• Fields must be separated with commas.</li> <li>• Leading and trailing spaces are ignored unless the field is delimited with double-quotes. In that case, the white spaces are preserved.</li> <li>• Embedded comma – You must delimit the field with double-quotes, with the comma inside the double quotes.</li> <li>• Embedded double-quotes – You must double embed double-quote characters, and you must delimit the field with double-quotes.</li> <li>• Embedded line-breaks – You must surround the field with double-quotes.</li> <li>• Always Delimiting – You can always delimit a field with double-quotes. If not strictly needed, the delimiters will be parsed and discarded by the reading applications.</li> </ul>
PL-15914	Guidewire has added Save and Cancel buttons on the Server Tools → Management Beans → Guidewire Managed Bean Properties screen. You can use these buttons to save or cancel a change to a bean property value. The buttons become active after you edit an editable property.
PL-18245	Fixed an issue with IgnoreProperty used in contact-sync-config.xml that caused it to take effect on all subtypes of Contact, rather than only those specified. This functionality now works properly and allows fields to be ignored on contact synchronization only for the specified subtypes.
PL-19311	Guidewire has upgraded the Joda-Time third-party library version to 2.0.
<b>Page Configuration</b>	
PL-15548	Printing a second-level list view no longer results in missing items, nor does it cause items to print more than once.
PL-18303	Adding multiple entries in list views for custom entity types no longer causes errors.
PL-19602	Added the ability to retain the scroll position after clicking a pull-right menu item. This fix adds the retainScrollPosition property to the MenuItem widgets.
<b>Persistence</b>	
PL-18162	Guidewire has added a validator that prevents you from attempting to denormalize a localized column. Guidewire does not support localized columns for search denorm columns.
<b>PL Services</b>	
PL-11554	<p>Modified how Guidewire applications discover and maintain the batch server in the clustered environment. Prior to this change, the batch server was discovered with the help of a message exchange through the JGroups cluster communication channel. Users starting up their nodes at approximately the same time could end up with two or more batch servers in the cluster if the cluster was not completely formed yet.</p> <p>This change uses the underlying database to discover and keep track of the batch server that is currently active in the cluster.</p>
PL-18232	It is now possible to extend any WorkQueue subclass exposed in ContactManager.
<b>Queries</b>	
PL-17541	Modified GWDBFunctionEnhancement.gsx and added support for DBFunction calls of MIN and MAX on java.util.Date.
PL-19590	Fixed an issue that caused an error in the ContactManager user interface by a query that contained a column definition twice.
<b>Rules Editor</b>	
PL-19466	Fixed a problem that occurred when you dragged a rule to the top of a rule set. Guidewire Studio incorrectly overwrote the first rule instead of inserting the lower order rule above it. Now Studio inserts the lower order rule correctly above the top rule.

PL-19510	Fixed an issue that caused errors in the Guidewire Studio Find in Path functionality when searching rule resources if the resources contained files that were not proper Gosu rule resources. Proper Gosu rule resources contain Gosu code and end with the file extensions .grs or .gr. Now, Find in Path displays non-Gosu rule resources as errors without affecting search results for valid resources.
<b>Studio</b>	
PL-19568	Guidewire Studio now supports 64-bit Java Virtual Machines (JVMs).
<b>Studio IDE</b>	
PL-13352	Fixed an issue in which Studio ignored the formatting settings for String literals set in Studio Tools → Options → Colors and Fonts.
PL-15219	Fixed an issue with the TemplateInputWidget PCF file in which autocomplete inserted a value that appeared to be correct, but failed validation.
PL-15887	Fixed an incorrect message that Studio showed in the Verify Result pane if verifying a valid entity name in the Entity Names editor.
PL-18203	Fixed an issue in which the Studio debugger step-over functionality stopped on Guidewire internal code in between executing Rule Condition and Rule Action code.
PL-18671	Guidewire has modified Studio behavior in regards to readonly mode as follows: <ul style="list-style-type: none"><li>• Studio now shows a padlock button on the status bar that is visible only if it is in readonly mode. If you click the button, Studio displays a modal message box indicating the reasons why it is in readonly mode.</li><li>• Studio disables the Save button any time that it is in readonly mode.</li><li>• Studio changes the Save button tooltip in readonly mode to show the reason that save is not active in this mode. This message is the same one that Studio shows if you click the padlock icon on the status bar.</li></ul>
PL-18905	Fixed an issue in which it was possible to edit configuration files even if Studio was in readonly mode.
<b>Utilities</b>	
PL-11681	The Gosu StringUtil class used many Perl 5, version 3, regular-expression, syntax-compatible functions to search for and replace strings. These functions were implemented by the Apache/Jakarta ORO library, which has been retired. See <a href="http://jakarta.apache.org/oro/">http://jakarta.apache.org/oro/</a> for details.  The same functionality for regular-expression, syntax-compatible search and replace of strings is now available in the java.util.regex package and the java.lang.String class.  Before this change, you used the following coding pattern:  <pre>StringUtil.substitute(inputString, "s/" + regexString + "/" + replacementString + "/g")</pre> Now, instead, you use the following coding pattern:  <pre>var pattern = java.util.regex.Pattern.compile(regexString) var patchedString = pattern.matcher(inputString).replaceAll(replacementString)</pre> Simpler search and replace functions are also available, such as:  <pre>java.lang.String.replace(String, String).</pre>
PL-15733	Guidewire has updated the Google Guava library to release 10.
<b>Web</b>	
PL-15548	Printing a second-level list view no longer results in missing items, nor does it cause items to print more than once.
PL-18303	Removed errors caused while adding multiple entries in list views for custom entity types.
PL-19454	Fixed an issue that occurred while using Internet Explorer 9 in which clicking a drop-down list did not render the list properly.
PL-19602	Added the ability to retain the scroll position after clicking a pull-right menu item. This fix adds the retainScrollPosition property to the MenuItem widgets.
<b>Web Services</b>	
PL-18361	ContactManager now provides the ability to implement a pre-existing WSDL in Gosu for a WSI web service.
PL-18450	ContactManager now consumes MTOM-enabled WS-I web services. MTOM is the W3C Message Transmission Optimization Mechanism that efficiently sends binary data to and from web services. You do not need to do anything to take advantage of this new feature.

PL-18589	Added the logging category XML.Request for WSI web services to log each request. Each request will be logged at the DEBUG level. This logging includes the connecting address and user, if available, as well as the request qname, which is unique for each operation.
<b>Web – UI/Runtime</b>	
PL-19646	Fixed an issue preventing the Guidewire application shortcut keys from working as intended. Previously, the keyboard responded only to the shortcut keys of the browser. Now the keyboard responds to the Guidewire application shortcut keys.
<b>Work Queues</b>	
PL-18675	Fixed several issues that involved the Work Queue Info screen on the Server Tools tab.
PL-19003	You can now start or stop work queues from IMaintenanceToolsAPI.
PL-19004	Added the attribute orphansfirst to a worker, thereby making it possible to specify that ContactManager is to process orphans before new items on this worker.
PL-19059	The Work Queue Info screen on the Server Tools tab now reports the last time that ContactManager processed a work item, showing last Success time or last Exception time. If the time was for an exception, the Work Queue Info screen also reports the number of consecutive work items that resulted in an exception. The screen shows two other dates as well. The Last Notification time is the last time the worker woke up. If it found work, ContactManager also updates the Last Batch Found time.

## Known Issues and Limitations for ContactManager 7.0.2

This section describes known issues with this release of Guidewire ContactManager.

- “ContactManager Known Issues” on page 322
- “Studio/Platform Known Issues” on page 323

**Note:** For maintenance releases, Guidewire often defers fixing configuration issues if they require merging files during the upgrade. Workarounds to many of these issues are listed in the following sections. The goal of this policy is to make upgrades as straightforward as possible.

### ContactManager Known Issues for Release 7.0.2

#### **RelatedABContactSearchCriteria does not work for related contact searches (CTC-960)**

**Issue** – Using the RelatedABContactSearchCriteria entity and the CriteriaDef elements that are defined for it in search-config.xml is not supported in ContactManager 7.0.2.

**Workaround** – Do not use them. Guidewire is aware of this issue.

#### **Customizing contact relationship handling to use messaging results in two relationship records being created (CTC-396)**

**Issue** – Changing a contact relationship updates two contact records. However, ContactManager stores the representation of the relationship in one ContactContact entity. ContactManager does not check to make sure the relationship does not already exist. Therefore, ContactManager can create multiple, redundant ContactContact entities for the same relationship if it receives messages changing the relationship between two contacts.

**Workaround** – In the base configurations of ContactManager, this behavior is not a problem. ClaimCenter makes a direct web service call to ContactManager to update contact relationships and does not use event messaging for related contact updates.

If you customize ClaimCenter and ContactManager to use event messaging to update related contacts between ClaimCenter and ContactManager, your customization must account for the extra message. Updating one relationship changes two contacts, and therefore ClaimCenter generates two messages and sends them to ContactManager. Your customization must ensure that ContactManager creates only one ContactContact entity in this case.

## Studio/Platform Known Issues for ContactManager 7.0.2

### Issues with Internet Explorer 9

**Issue** – If you are using the Internet Explorer 9 browser, it is possible to see issues such as screen flickering or an incorrect tab order for fields. According to public reports, Internet Explorer 9 exhibits these and other issues with a variety of web sites and web applications.

**Workaround** – Because this is the behavior of the Internet Explorer 9 rendering engine, Guidewire cannot address these issues. However, there are reports of an Internet Explorer 9 workaround that reduces these issues. Change the new Internet Explorer 9 Accelerated Graphics settings on the Advanced tab of the Internet Options dialog.

### First time you click the arrow of the typekey input, the drop-down menu does not open (PL-10134)

**Issue** – The drop-down menu does not open on the first click of the arrow on a typekey input. Instead, the help text opens.

**Workaround** – Turn off help text on focus by setting InputHelpTextOnFocus to false in the config.xml file. After you do that, the help text shows only if you mouse over the input, and it does not interfere with opening a drop-down menu.

### XML API upgrade feature missing from documentation (PL-10257)

**Issue** – The *Integration Guide* describes a new set of XML APIs based on the XmlElement class. (Legacy APIs are based on the XmlNode class.) You can continue to use the legacy APIs. However, the *Integration Guide* omits mentioning an additional upgrade-specific feature.

**Workaround** – For backwards compatibility only, you can import an XML schema into the Gosu type system using the legacy XML system by following these instructions:

1. Copy:

```
ContactManager/modules/pl/config/registry/compatibility-xsd.xml
```

To:

```
ContactManager/modules/configuration/config/registry/compatibility-xsd.xml
```

2. Add an entry for your schema. Set the value of the namespace attribute to the Gosu package name of the schema. For example, if the schema is in the package location my.package and is called myschema.xsd, set the value of namespace to my.package.myschema.

### Studio Rules do not use correct capitalization for root object's name (PL-10740)

**Issue** – Rule set root objects are not named with first letter lower-cased.

**Workaround** – The rules engine issues a warning if the correct case for objects is not being used.

### Countries configured in zone-config.xml still generate a warning during regen-dictionary even when zone data is loaded for all these countries (PL-11947)

**Issue** – Countries configured in zone-config.xml still generate a warning during regen-dictionary even when zone data is loaded for all of these countries.

**Workaround** – Warning message is created in error and can safely be ignored.

### User interface cannot handle starting multiple instances of a batch process (PL-12372)

**Issue** – The user interface cannot handle starting multiple instances of a batch process.

**Workaround** – To execute multiple instances of a batch process, start them from the command line. Also, to allow multiple instances to be run simultaneously, you must ensure that the BatchProcess.isExclusive() method returns false.

**Type system refresh after PCF page title change does not update corresponding menu label (PL-13057)**

**Issue** – The type system refresh after a PCF page title change does not update the corresponding menu label.

**Workaround** – After updating a page title, restart the server to refresh menu labels and avoid null pointer exceptions due to stale references.

**Length limitation on entity localization table names (PL-13360)**

**Issue** – Entity localization table names have a shorter, 16-character limit than other entity tables. If the localization table name exceeds the 16-character limit, the error message indicates incorrectly that 18 characters are allowed.

**Workaround** – Ensure that the localization `tableName` property specified in the entity extension file is less than 16 characters.

**US-Locations.txt file with the US geodata from GreatData has special characters that cause validation problems with United States Postal Service (USPS) data (PL-13384)**

**Issue** – The `US-Locations.txt` file contains information that does not conform to United States Postal Service (USPS) standards for bulk mailings.

**Workaround** – The provided `US-Locations.txt` file is intended only for use in geocoding to identify addresses for a location. You can edit the `US-Locations.txt` file to conform to your particular address standards, and then import that version of the file instead.

**GX models that reference virtual fields and enhancements throw null pointers if null (PL-13560)**

**Issue** – The GX models that reference virtual fields and enhancements throw null pointers when these fields and enhancements are null.

**Workaround** – Include null checks and error handling to prevent referenced virtual fields or enhancements that are null from causing null pointer exceptions.

**Sending email with file attachment with unicode filename is not correctly handed over to the mail server (PL-13582)**

**Issue** – An email with a file attachment that has a unicode file name is not sent to the mail server correctly.

**Workaround** – Use Latin characters for file names on attached files.

**GX model generated XSD cannot be parsed by JAXB (PL-13598)**

**Issue** – XSD generated by the GX model cannot be parsed by JAXB.

**Workaround** – Add JAXB annotation elements to the XSD to specify the necessary metadata, such as class names, to enable JAXB to generate the Java class files. Contact Guidewire Support for a sample XSD file that is annotated this way.

**Cannot make a field from a delegate into a localized column (PL-13761)**

**Issue** – You cannot make a field from a delegate into a localized column.

**Workaround** – Move the column to be localized off the delegate and onto each of the implementing entities. Then, to make the column appear as though it exists on the delegate, define an enhancement property on the delegate that *delegates* to the appropriate column.

**Problem with regen-java-api command and JAR files (PL-16351)**

**Issue** – If you run the `ContactManager/bin/gwab regen-java-api` command, ContactManager creates a `ContactManager/java-api/lib` directory and puts JAR files for Java development in this directory. However, ContactManager does not always perform this task correctly, resulting in some of the generated JAR files' not being copied to this directory.

**Workaround** – For missing library files, copy or import into your Java development environment the missing library files from the following directory:

ContactManager/admin/lib

Copy them into the following directory:

ContactManager/java-api/lib

#### Renaming method or property throws ParseResultsException (PL-16633)

**Issue** – If you rename a property or method or you change a method signature, and a workflow references that property or method in a Gosu field, ContactManager throws `ParseResultsException`. This is the intended behavior.

**Workaround** – Restart the workflow engine. To do so:

1. Log into ContactManager using an administrative account.

2. Access **Internal Tools** → **Reload**.

Click **Reload Workflow Engine**.

## Guidewire ContactManager 7.0.3 Release Notes

### Overview of ContactManager 7.0.3 Release Notes

These release notes contain the following:

- “Release Information for ContactManager 7.0.3” on page 337
- “Installing ContactManager 7.0.3” on page 337
- “Support for ContactManager 7.0.3” on page 338
- “Issues and Major Changes for ContactManager 7.0.3” on page 338
- “Improvements and General Issues for ContactManager 7.0.3” on page 339
- “Known Issues and Limitations for ContactManager 7.0.3” on page 342

### Release Information for ContactManager 7.0.3

These release notes apply only to this release of Guidewire ContactManager.

### Version Number for ContactManager 7.0.3

This release of Guidewire ContactManager is 7.0.3.

- If you are integrating with Guidewire BillingCenter, this version of ContactManager requires BillingCenter 7.0.0 or later.
- If you are integrating with Guidewire PolicyCenter, this version of ContactManager requires PolicyCenter 7.0.1 or later.
- If you are integrating with Guidewire ClaimCenter, this version of ContactManager requires ClaimCenter 6.0.7 or later.

### Installing ContactManager 7.0.3

For general installation information, see “Installing ContactManager” in the *Guidewire Contact Management Guide*.

For versions of ContactManager prior to 7.0.3 that you have skipped, see:

- “Guidewire ContactManager 7.0.1 Release Notes” on page 311

- “Guidewire ContactManager 7.0.2 Release Notes” on page 326

For versions prior to 7.0.0, see the ClaimCenter release notes for that version, which include sections for ContactCenter release notes. For example, see “Release Notes Archive” on page 135 in the *New and Changed Guide*.

## Support for ContactManager 7.0.3

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

- At the Guidewire Resource Portal at <http://guidewire.custhelp.com>
- By email at [support@guidewire.com](mailto:support@guidewire.com)
- By phone at +1-650-356-4955

## Issues and Major Changes for ContactManager 7.0.3

This topic contains issues and major changes that might affect your installation.

- “Base PCF File Changes for ContactManager 7.0.3” on page 338
- “Rules Changes for ContactManager 7.0.3” on page 338
- “Changes in This Release Provided in Upgrade Diff Report for ContactManager 7.0.3” on page 338

### Base PCF File Changes for ContactManager 7.0.3

All links below require the `ReleaseNotes_files` directory for release 7.0.3 on your local disk.

#### ContactManager release 7.0.2 to 7.0.3

- To view a report of the changes in the base PCF files in the `modules/ab` directory, [click this link](#).
- To view a report of the changes in the base PCF files in the `modules/p1` directory, [click this link](#).

### Rules Changes for ContactManager 7.0.3

#### ContactManager release 7.0.2 to 7.0.3

- There are no changes to the base rules in the `modules/ab` directory.

### Changes in This Release Provided in Upgrade Diff Report for ContactManager 7.0.3

Guidewire provides a report detailing certain differences between the current release and your prior release. This report describes changes in display keys, entities, typelists, and the Gosu API. To obtain your custom Upgrade Diff Report, visit the Guidewire Resource Portal.

## Improvements and General Issues for ContactManager 7.0.3

This topic describes improvements and issues corrected in this release. Guidewire attempts to provide information for issues of primary importance to our customers. This list is not intended to be comprehensive.

ID	Description
<b>Archiving</b>	
PL-19035	Added a Download button to the (Server Tools) Archive Info page to more easily enable the collection and sending of archiving error information.
PL-20041	Fixed an issue in which the archiving schema (archiving.XSD) did not match the archiving output XML.
PL-20690	A null pointer exception sometimes occurred during restore of archived claims. A more informative error message is now shown.
PL-21380	The regen-xsd command generated an XSD file with errors. The command has been updated to produce a valid XSD file.
<b>Batch Processes</b>	
PL-20755	Fixed an issue in which worker threads could unexpectedly quit if any database or JDBC error occurred during a time span longer than the maximum wait period.
<b>Command Line Tools</b>	
PL-19768	Fixed an issue that occurred while attempting to execute the verify-types command on a 64-bit JVM, which caused an OutOfMemoryError.
<b>Consistency Checker</b>	
PL-16205	Requesting a run of the database consistency checks by using the <database> element attribute checker="true" in the config.xml file is a deprecated feature. If you do so, ContactManager now prints a warning message in the log warning that this feature is deprecated. The message also instructs you to use the Info Page or command line to run the consistency checks as a batch job.
<b>Contact Domain</b>	
CTC-1138	The SetVendorsPrimaryAddressBatchGeocode rule was incorrectly resetting an ABContact object's status to BatchGeocode if anything in the object changed, even if the change did not affect geocoding. This rule now does not reset an ABContact object's status to BatchGeocode unless a change to the object affects geocoding.
<b>Core</b>	
CTC-618	Removed print statements that were producing unnecessary clutter in the console log messages when starting ContactManager.
<b>Database</b>	
PL-8729	Guidewire removed the configuration parameter TableEstimatePercent from the config.xml file. Instead, use the samplingpercentage attribute of the database statistics element in config.xml, as the following sample XML code shows.  <databasestatistics samplingpercentage="20">
PL-10469	With this release, Guidewire applications do not start if the CurrentEncryptionPlugin parameter in config.xml specifies an encryption plugin implementation that does not exist.
PL-10468	Fixed an issue on the Database Storage Information page regarding the table display whenever switching from Index Physical Statistics to Tables and Indexes.
PL-14490	The data distribution tables now include a column for the internal major version number of the database schema.
PL-17742	On Oracle, changed database upgrade logic by providing additional optimizer hints to help improve upgrade performance.
PL-17835	Behavior has changed with evicting a database connection from the connection pool when an exception occurs. Guidewire applications now determine whether an exception was fatal to a connection before marking it for eviction. In addition, whenever the application marks a connection for eviction, the application logs the reason for the eviction. Constraint violations no longer cause evictions.
PL-20546	In data distribution tables, the size of numeric columns increased to hold larger values.
PL-20584	On Oracle, the Guidewire Profiler changed to help optimize the capture of the range of AWR snapshots.

PL-21145	Replaced the String value on DBNullConstraint Exception with display key Java.Database.DBException.NullConstraintViolation to enable localization of the message.
PL-21448	On SQL Server, ContactManager now provides additional information on database connections, which can then be captured and analyzed in the Server DMV Snapshot screen.
<b>Database Support</b>	
PL-19689	Guidewire has changed the mechanism for creating the database performance reports for Oracle and SQL Server. You no longer have to click a button and wait for the report to be generated and packaged in a ZIP file for immediate download. The report is now generated by a background batch process that stores the completed report in a new table named ab_dbperfreport. This change requires a database upgrade to create this new table.
<b>Database Upgrade</b>	
PL-16977	Previously after a database upgrade, ContactManager logged differences between the metadata configuration and the upgraded database schema at the INFO logging level. Now, ContactManager logs differences at the WARN logging level.
PL-20200	Guidewire has implemented optimization on the database upgrade. This optimization improves the performance of new database creation, as well as database changes during minor and major upgrades.
PL-21310	Fixed an issue that caused database upgrades to fail if the WorkFlowWorkItem table was populated. Upgrade succeeded if the table was empty. The fix involved changing the WorkFlowWorkItem entity attribute from Final to Extendable.
PL-21357	The Upgrade Info page and download now provide more consistent information about the steps that are being performed during an automated database upgrade.
PL-21387	Corrected an issue that caused a null-pointer exception (NPE) in version triggers during a database upgrade.
<b>Email</b>	
PL-13582	A problem prevented files with Unicode file names attached to email from reaching email clients. Now, email clients that handle attachments with Unicode file names, such as Microsoft Outlook, receive the attachments from ContactManager and send them correctly. If your email client does not handle Unicode file names, you must modify the EmailMessageTransport plugin to convert attachment file names to use Latin characters only.
<b>Entities/Metadata</b>	
PL-20127	Guidewire has corrected an issue with setting the appscale attribute for the Money data type. The appscale attribute worked for currencyamount columns, but not for money columns. The setting, specified in datatypes.xml, was being ignored when a column of type money was retrieved. This setting is now working. The appscale attribute controls the number of digits shown to the right of the decimal point for money or currency in single currency mode. It is similar to the scale attribute. The appscale attribute must be smaller than the scale setting. If defined, the appscale attribute overrides the scale attribute.
PL-18068	Fixed a problem with verification of typelist typecodes on server startup. Typelist typecodes that match regardless of case fail validation. For example, the following two typecodes would fail validation: <ul style="list-style-type: none"> <li>• MyCode and Mycode</li> </ul>
PL-20193	When an element of an owned array changes, the parent changes as well. For effdated owned arrays, the wrong parent changed. Now, the correct parent changes. This issue also affects effdated owned one-to-ones, because they are implemented as owned arrays.
<b>File Export</b>	
PL-21546	Exported CSV files no longer contain UTF-8 BOM (byte order mark) characters at the beginnings of files that do not use UTF-8 encoding.
<b>Global Cache</b>	
PL-20332	A new configuration parameter, GlobalCacheDetailedStats, determines whether to collect detailed statistics for the global cache. Detailed statistics are data that ContactManager collects to explain why items are evicted from the cache. Basic statistics, such as miss ratio, are still collected regardless of the value of this parameter.  The GlobalCacheDetailedStats parameter is set to false by default. Set the parameter to true to help tune your cache. At runtime, use the Management Beans page to enable the collection of detailed statistics for the global cache. Whenever the GlobalCacheDetailedStats parameter is disabled, the Evict Information and Type of Cache Misses graphs are not visible.

<b>Gosu</b>	
PL-20961	In Gosu you can now subclass an inner class that a supertype declares.
<b>Messaging</b>	
PL-18715	The Studio Messaging editor contained hard-coded strings for destination names, which made localization of destination names shown in the Admin tab difficult. Now, destination names are display keys. Therefore, destination names are included in the translatable resources of the application. Display keys for destination names are in the Java.MessageDestination level of the display key namespace.
PL-13887	The IMessagingToolsAPI has a new method to retrieve the status of a messaging destination. The method signature is: <pre>getDestinationStatus(destID : int) : String</pre>
<b>Miscellaneous</b>	
CTC-478	Spelling errors in the display keys for Law Firm Specialty and Medical Firm Specialty have been fixed.
<b>Plugins</b>	
PL-21409	In earlier versions, the User Authentication Service plugin AuthenticationServicePlugin threw four exceptions that LoginForm.java caught and showed. There was no standard mechanism to add custom exceptions to the plugin and the login form. Now, you can modify the User Authentication Service plugin and throw DisplayableLoginException.
<b>Staging Tables</b>	
PL-20303	ContactManager provides two new consistency checks: "One-to-one non-null check" and "Edge foreign key non-null check".
<b>Studio IDE</b>	
PL-19308	An issue occurred with Studio because some source control systems require a file or directory in a directory controlled by Studio. Studio interfered with the special, source control files. Now, Studio safely ignores third party files and directories required by some source control systems.
PL-20531	Fixed a problem that could cause Studio to not properly render selected text and caret positions in files containing true tab characters.
PL-21216	Studio can generate a new warning message during PCF verification, (Verify All). It generates the warning if the number of attempted verification passes exceeds the number set for <b>Section Inclusion Limit</b> in Tools → Verification Options → PCF Verification Options.
PL-21296	Modified the Studio PCF Editor to make the PCF Toolbox show subcategories for all Application customized widgets that you can use in widget filtering.
PL-21447	Corrected an issue in which the Studio Gosu editor periodically froze for a brief period of time on large files if you typed quickly.
<b>Web – UI/Runtime</b>	
PL-20150	Modified ContactManager event-processing logic to skip events for removed widgets. For example, if the values of two widgets have been changed in the same form submission, there will be two onChange events in the event queue. If the handler of the first event implicitly removes the widget that is the source of the second event, the second event will no longer be processed. The result is that the associated onChange handler will not be invoked to avoid errors.
<b>Web Services</b>	
PL-21158	The Guidewire Profiler previously did not profile WS-I web services. Now whenever you select a WS-I webservice to profile, ContactManager provides profile information.
<b>Web Services - WSI</b>	
PL-18729	It is now possible to invoke local web services using the wsdl.local mechanism from Studio without a running server.
PL-20174	It is now possible to run WSI Web Services on a server other than the batch server in a cluster.
<b>XML Element - XSD types</b>	
PL-20076	Fixed an XML parsing bug that caused an xs:choice with a zero-width match to not be handled properly.

## Known Issues and Limitations for ContactManager 7.0.3

This section describes known issues with this release of Guidewire ContactManager.

- “ContactManager Known Issues for Release 7.0.3” on page 342
- “Studio/Platform Known Issues for ContactManager 7.0.3” on page 343

**Note:** For maintenance releases, Guidewire often defers fixing configuration issues if they require merging files during the upgrade. Workarounds to many of these issues are listed in the following sections. The goal of this policy is to make upgrades as straightforward as possible.

### ContactManager Known Issues for Release 7.0.3

#### **RelatedABContactSearchCriteria does not work for related contact searches (CTC-960)**

**Issue** – Using the RelatedABContactSearchCriteria entity and the CriteriaDef elements that are defined for it in search-config.xml is not supported in ContactManager 7.0.3.

**Workaround** – Do not use them. Guidewire is aware of this issue.

#### **Customizing contact relationship handling to use messaging results in two relationship records being created (CTC-396)**

**Issue** – Changing a contact relationship updates two contact records. However, ContactManager stores the representation of the relationship in one ContactContact entity. ContactManager does not check to make sure the relationship does not already exist. Therefore ContactManager can create multiple, redundant ContactContact entities for the same relationship if it receives messages changing the relationship between two contacts.

**Workaround** – In the base configurations of ContactManager, this behavior is not a problem. ClaimCenter makes a direct web service call to ContactManager to update contact relationships and does not use event messaging for related contact updates.

If you customize ClaimCenter and ContactManager to use event messaging to update related contacts between ClaimCenter and ContactManager, your customization must account for the extra message. Updating one relationship changes two contacts, and therefore ClaimCenter generates two messages and sends them to ContactManager. Your customization must ensure that ContactManager creates only one ContactContact entity in this case.

#### **Email field validation in ContactManager not the same as in core applications (CTC-1339)**

**Issue** – The entry in fieldvalidators.xml in ContactManager is more restrictive than in the core applications. This difference can cause issues while creating a contact in a core application.

**Workaround** – Change the ContactManager version of this validator to match the validators in the core products. Use Studio to edit the entry in fieldvalidators.xml for Validator.Email, changing its value from the default ".+@.+\\..+" to the value in the core applications, ".+@.+".

#### **ContactManager 7 might not find duplicate contacts when ClaimCenter 6 requests a relink (CTC-1333)**

**Issue** – When ClaimCenter 6 is integrated with ContactManager 7, if a ClaimCenter contact becomes unlinked, the ClaimCenter user can click the Relink button to link the contact again. What happens then is that ContactManager checks first to see if there are duplicate contacts and creates a new contact only if there are no matches. However, when ContactManager receives a link request from ClaimCenter 6, ContactManager uses older matching logic for finding duplicates. This older matching logic might not find a matching contact that the newer matching logic might find after a link request from ClaimCenter 7. Therefore, ContactManager might create a new contact when it receives a link message from ClaimCenter 6 that it would not create if the link request came from ClaimCenter 7. If ContactManager received the same link request from ClaimCenter 7, it might find definitive or potential matches instead, and not create a new contact.

**Workaround** – Guidewire is aware of this issue.

**Duplicate key warning when starting ContactManager server with empty database (CTC-1341)**

**Issue** – When starting the ContactManager server with an empty database you might see a DBDuplicateKeyException in the logs. There is a log message preceding the exception that states, User duplicate key exception is logged, but can be ignored.

**Workaround** – As stated in the log message, you can ignore the exception.

**Studio/Platform Known Issues for ContactManager 7.0.3****Issues with Internet Explorer 9**

**Issue** – If you are using the Internet Explorer 9 browser, it is possible to see issues such as screen flickering or an incorrect tab order for fields. According to public reports, Internet Explorer 9 exhibits these and other issues with a variety of web sites and web applications.

**Workaround** – Because this is the behavior of the Internet Explorer 9 rendering engine, Guidewire cannot address these issues. However, there are reports of an Internet Explorer 9 workaround that reduces these issues. In Internet Explorer 9, you can try changing the use of software or hardware rendering by toggling the Accelerated Graphics option on the Advanced tab of the Internet Options dialog.

**Studio Rules do not use correct capitalization for root object's name (PL-10740)**

**Issue** – Rule set root objects are not named with first letter lower-cased.

**Workaround** – Guidewire is aware of this issue.

**User interface cannot handle starting multiple instances of a batch process (PL-12372)**

**Issue** – The user interface cannot handle starting multiple instances of a batch process.

**Workaround** – To execute multiple instances of a batch process, start them from the command line. Also, to allow multiple instances to be run simultaneously, you must ensure that the BatchProcess.isExclusive() method returns false.

**Type system refresh after PCF page title change does not update corresponding menu label (PL-13057)**

**Issue** – The type system refresh after a PCF page title change does not update the corresponding menu label.

**Workaround** – After updating a page title, restart the server to refresh menu labels and avoid null pointer exceptions due to stale references.

**Length limitation on entity localization table names (PL-13360)**

**Issue** – Entity localization table names have a shorter, 16-character limit than other entity tables. If the localization table name exceeds the 16-character limit, the error message indicates incorrectly that 18 characters are allowed.

**Workaround** – Ensure that the localization tableName property specified in the entity extension file is less than 16 characters.

**US-Locations.txt file with the US geodata from Great Data has special characters that cause validation problems with United States Postal Service (USPS) data (PL-13384)**

**Issue** – The US-Locations.txt file contains information that does not conform to United States Postal Service (USPS) standards for bulk mailings.

**Workaround** – The provided US-Locations.txt file is intended only for use in geocoding to identify addresses for a location. You can edit the US-Locations.txt file to conform to your particular address standards, and then import that version of the file instead.

**GX model generated XSD cannot be parsed by JAXB (PL-13598)**

**Issue** – XSD generated by the GX model cannot be parsed by JAXB.

**Workaround** – Add JAXB annotation elements to the XSD to specify the necessary metadata, such as class names, to enable JAXB to generate the Java class files. Contact Guidewire Support for a sample XSD file that is annotated this way.

**Cannot make a field from a delegate into a localized column (PL-13761)**

**Issue** – You cannot make a field from a delegate into a localized column.

**Workaround** – Move the column to be localized off the delegate and onto each of the implementing entities. Then, define an enhancement property on the delegate that delegates to the appropriate column, depending on the implementing entity. Doing so makes the column appear as though it exists on the delegate.

**Problem with regen-java-api command and JAR files (PL-16351)**

**Issue** – If you run the `ContactManager/bin/gwpc regen-java-api` command, ContactManager creates a `ContactManager/java-api/lib` directory and puts JAR files for Java development in this directory. However, ContactManager does not always perform this task correctly, resulting in some of the generated JAR files not being copied to this directory.

**Workaround** – For missing library files, copy or import into your Java development environment the missing library files from the following directory:

`ContactManager/admin/lib`

Copy them into the following directory:

`ContactManager/java-api/lib`

**Renaming method or property throws ParseResultsException (PL-16633)**

**Issue** – If you rename a property or method or you change a method signature, and a workflow references that property or method in a Gosu field, ContactManager throws `ParseResultsException`. This behavior is intended.

**Workaround** – Restart the workflow engine. To do so:

1. Log into ContactManager using an administrative account.
2. Access Internal Tools → Reload.
3. Click Reload Workflow Engine.

## Guidewire ContactManager 7.0.4 Release Notes

### Overview of ContactManager 7.0.4 Release Notes

These release notes contain the following:

- “Release Information for ContactManager 7.0.4” on page 344
- “Installing ContactManager 7.0.4” on page 345
- “Support for ContactManager 7.0.4” on page 345
- “Issues and Major Changes for ContactManager 7.0.4” on page 345
- “Improvements and General Issues for ContactManager 7.0.4” on page 347
- “Known Issues and Limitations for ContactManager 7.0.4” on page 348

### Release Information for ContactManager 7.0.4

These release notes apply only to this release of Guidewire ContactManager.

## Version Number Information for ContactManager 7.0.4

This release of Guidewire ContactManager is 7.0.4.

- If you are integrating with Guidewire BillingCenter, this version of ContactManager requires BillingCenter 7.0.0 or later.
- If you are integrating with Guidewire PolicyCenter, this version of ContactManager requires PolicyCenter 7.0.1 or later.
- If you are integrating with Guidewire ClaimCenter, this version of ContactManager requires ClaimCenter 6.0.7 or later.

## Installing ContactManager 7.0.4

For general installation information, see “[Installing ContactManager](#)” in the *Guidewire Contact Management Guide*.

For versions of ContactManager prior to 7.0.4 that you have skipped, see:

- “[Guidewire ContactManager 7.0.1 Release Notes](#)” on page 311
- “[Guidewire ContactManager 7.0.2 Release Notes](#)” on page 326
- “[Guidewire ContactManager 7.0.3 Release Notes](#)” on page 337

For versions prior to 7.0.0, see the ClaimCenter release notes for that version, which include sections for ContactCenter release notes. For example, see “[Release Notes Archive](#)” on page 135 in the *New and Changed Guide*.

## Support for ContactManager 7.0.4

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

- At the Guidewire Resource Portal – <http://guidewire.custhelp.com>
- By email – [support@guidewire.com](mailto:support@guidewire.com)
- By phone – +1-650-356-4955

## Issues and Major Changes for ContactManager 7.0.4

This topic contains issues and major changes that might affect your installation.

- “[Base PCF File Changes for ContactManager 7.0.3](#)” on page 338
- “[Rules Changes for ContactManager 7.0.3](#)” on page 338
- “[Changes in This Release Provided in Upgrade Diff Report](#)” on page 313

### Base PCF File Changes for ContactManager 7.0.4

All links below require the `ReleaseNotes_files` directory on your local disk, in the same directory as this release notes file.

#### ContactManager release 7.0.3 to 7.0.4

- To view a report of the changes in the base PCF files in the `modules/ab` directory, [click this link](#).
- To view a report of the changes in the base PCF files in the `modules/p1` directory, [click this link](#).

### Rules Changes for ContactManager 7.0.4

#### ContactManager release 7.0.3 to 7.0.4

- There are no changes to the base rules in the `modules/ab` directory.

## Changes in This Release Provided in Upgrade Diff Report for ContactManager 7.0.4

Guidewire provides a report detailing certain differences between the current release and your prior release. This report describes changes in display keys, entities, typelists, and the Gosu API. To obtain your custom Upgrade Diff Report, visit the Guidewire Resource Portal.

## Improvements and General Issues for ContactManager 7.0.4

This topic describes improvements and issues corrected in this release. Guidewire attempts to provide information for issues of primary importance to our customers. This list is not intended to be comprehensive.

ID	Description
<b>Authentication</b>	
PL-21737	The Gosu implementation of the AuthenticationSourceCreator plugin now supports JBoss.
<b>Build Infrastructure</b>	
PL-15509	In previous versions, Guidewire supported running build scripts only through the gwab command on Windows. With this version you, can run the EAR and WAR build scripts on Unix by invoking Ant directly with the following command:  <pre>ant -f ContactManager/modules/ant/build.xml buildScript</pre> Substitute one of the following values for <i>buildScript</i> : <ul style="list-style-type: none"> <li>• build-jboss-war – Builds a generic WAR file for use with JBoss.</li> <li>• build-tomcat-war – Builds a generic WAR file for use with Tomcat.</li> <li>• build-weblogic-ear – Builds an EAR file for use with WebLogic.</li> <li>• build-websphere-ear – Builds an EAR file for use with WebSphere.</li> </ul> Although you can run many build scripts by using the gwab command on Windows, if you invoke Ant on Unix, Guidewire supports only the build scripts in the previous list.
<b>Command line tools</b>	
PL-20110	The command-line utility gwab regen-soap-api no longer fails on Japanese Windows.
<b>Contact App</b>	
CTC-1646	Fixed an issue in which the default value of the DuplicateContactsWideSearch configuration parameter was not set.
<b>Core</b>	
CTC-1336	Fixed an issue in which the DefinitiveMatchSet definitions were causing erroneous consistency and integrity checks to be run. The deprecated IContactAPI enforces these checks, but the base configuration of ContactManager does not use IContactAPI. The fix was to edit definitive-match-config.xml and comment out these DefinitiveMatchSet settings.  If you are using IContactAPI, you can re-enable these checks by editing definitive-match-config.xml and removing the comments on the DefinitiveMatchSet settings that are commented out.
<b>Database</b>	
PL-21554	In previous versions on SQL Server, tuning queries was difficult because DBAs could not determine what part of the application generated the queries. Now, you can enable the new IdentifyQueriesViaComments parameter in the config.xml file to provide comments with contextual information in certain SQL Select statements sent to the relational database.  The SQL comments are in the format:  <i>ApplicationName:ActionName</i> <i>ApplicationName</i> is ContactManager. <i>ActionName</i> is the name of the PCF file that submitted the SQL Select statement.
<b>Database Upgrade</b>	
PL-22007	In earlier versions, ContactManager ran the trigger AfterUpgradeVersionTrigger on fresh databases, which could cause problems. With this release, ContactManager runs this trigger only on existing databases, not on fresh databases.
PL-22478	In earlier versions, on SQL Server queries failed if they contained a large number of CASE conditions. Now, ClaimCenter breaks up queries with more than 125 CASE conditions into nested CASE clauses or into separate queries.
PL-22686	With this release, the validation SQL run by the upgrade version checks is now included in the downloads that you obtain from the Upgrade Info page.

PL-22934	This release fixes an issue that caused a SQL failure during database upgrade of the <code>InstrumentedBatchJobID</code> column on the <code>instrumentedworkertask</code> table.
PL-23044	With this release, the database upgrader now supports table and index partitioning. Specify which tables and indexes you want to have partitioned, and the upgrader creates any new table and new indexes appropriately. Existing tables and indexes remain unchanged, regardless of the discrepancy between the metadata configuration file and the database schema.
<b>Entities/Metadata</b>	
PL-22866	Calling an API method that attempts to modify a read-only bundle now shows an immediate error.
PL-22597	Previously, if you defined an entity type that implemented a Gosu interface, problems occurred if the interface had a method that returned an array of that entity type. None of the interface features were accessible on the entity that implemented the interface, and class cast exceptions could occur. Now, the interface features are accessible and class cast exceptions do not occur.
<b>Gosu</b>	
PL-22469	When an entity contains an <code>implementsInterface</code> element, that entity type is supposed to implement the interface specified in the <code>implementsInterface</code> element. Previously, methods inherited by the interface could be invoked, but references to the entity could not be assigned to variables of the interface type. This issue fixes the problem so that the entity type truly implements the interface, preserving assignability.
<b>Integration - BC, Integration - CC, Integration - PC</b>	
CTC-1307	The find duplicates code run by the batch process was configurable to use a narrow or wide search in the <code>DuplicateContactsWideSearch</code> configuration parameter. This configuration parameter is now used by the <code>ABContactAPI</code> <code>findDuplicates</code> method as well, so the behavior of the API call and the batch process is now the same.
<b>Localization</b>	
CTC-1429	Updated all messaging destinations to use display keys for the Name field to support translation into different languages.
<b>Miscellaneous</b>	
CTC-1384	The Merge Contacts permission to handle duplicate contacts was added to the Super User role in ContactManager 7.0.0. At the time, no upgrade trigger was added to upgrade existing installations. There is now an upgrade trigger to add the Merge Contacts permission to the Super User role. This trigger has no effect if you have already added this permission to the Super User role. This permission has not been added to any other role.
<b>Web - Other</b>	
PL-18559	Added Alt text, alternative tooltip text, for the following: <ul style="list-style-type: none"> <li>• <code>BooleanRadioInput</code></li> <li>• <code>PickerLink</code></li> <li>• <code>DocumentationCell</code></li> </ul>

## Known Issues and Limitations for ContactManager 7.0.4

This topic describes known issues with this release of Guidewire ContactManager. It has the following topics:

- “ContactManager Known Issues” on page 322
- “Studio/Platform Known Issues” on page 323

**Note:** For maintenance releases, Guidewire often defers fixing configuration issues if they require merging files during the upgrade. Workarounds to many of these issues are listed in the following sections. The goal of this policy is to make upgrades as straightforward as possible.

## ContactManager Known Issues for Release 7.0.4

### **RelatedABContactSearchCriteria does not work for related contact searches (CTC-960)**

**Issue** – Using the RelatedABContactSearchCriteria entity and the CriteriaDef elements that are defined for it in search-config.xml is not supported in this version of ContactManager.

**Workaround** – Do not use them. Guidewire is aware of this issue.

### **Customizing contact relationship handling to use messaging results in two relationship records being created (CTC-396)**

**Issue** – Changing a contact relationship updates two contact records. However, ContactManager stores the representation of the relationship in one ContactContact entity. ContactManager does not check to make sure the relationship does not already exist. Therefore, ContactManager can create multiple, redundant ContactContact entities for the same relationship if it receives messages changing the relationship between two contacts.

**Workaround** – In the base configurations of ContactManager, this behavior is not a problem. ClaimCenter makes a direct web service call to ContactManager to update contact relationships and does not use event messaging for related contact updates.

If you customize ClaimCenter and ContactManager to use event messaging to update related contacts between ClaimCenter and ContactManager, your customization must account for the extra message. Updating one relationship changes two contacts, and therefore ClaimCenter generates two messages and sends them to ContactManager. Your customization must ensure that ContactManager creates only one ContactContact entity in this case.

### **Email field validation in ContactManager not the same as in core applications (CTC-1339)**

**Issue** – The entry in fieldvalidators.xml in ContactManager is more restrictive than in the core applications. This difference can cause issues while creating a contact in a core application.

**Workaround** – Change the ContactManager version of this validator to match the validators in the core products. Use Studio to edit the entry in fieldvalidators.xml for Validator.Email, changing its value from the default ".+@.+\\...+" to the value in the core applications, ".+@.+".

## Studio/Platform Known Issues for ContactManager 7.0.4

### **Issues with Internet Explorer 9**

**Issue** – If you are using the Internet Explorer 9 browser, it is possible to see issues such as screen flickering or an incorrect tab order for fields. According to public reports, Internet Explorer 9 exhibits these and other issues with a variety of web sites and web applications.

**Workaround** – Because this is the behavior of the Internet Explorer 9 rendering engine, Guidewire cannot address these issues. However, there are reports of an Internet Explorer 9 workaround that reduces these issues. In Internet Explorer 9, you can try to change the use of software or hardware rendering. To do so, toggle the Accelerated Graphics option on the Advanced tab of the Internet Options dialog.

### **Studio Rules do not use correct capitalization for root object's name (PL-10740)**

**Issue** – Rule set root objects are not named with first letter lower-cased.

**Workaround** – Guidewire is aware of this issue.

### **User interface cannot handle starting multiple instances of a batch process (PL-12372)**

**Issue** – The user interface cannot handle starting multiple instances of a batch process.

**Workaround** – To execute multiple instances of a batch process, start them from the command line. Also, to allow multiple instances to be run simultaneously, you must ensure that the `BatchProcess.isExclusive()` method returns `false`.

**Type system refresh after PCF page title change does not update corresponding menu label (PL-13057)**

**Issue** – The type system refresh after a PCF page title change does not update the corresponding menu label.

**Workaround** – After updating a page title, restart the server to refresh menu labels and avoid null pointer exceptions due to stale references.

**Length limitation on entity localization table names (PL-13360)**

**Issue** – Entity localization table names have a shorter, 16-character limit than other entity tables. If the localization table name exceeds the 16-character limit, the error message indicates incorrectly that 18 characters are allowed.

**Workaround** – Ensure that the localization `tableName` property specified in the entity extension file is less than 16 characters.

**US-Locations.txt file with the US geodata from Great Data has special characters that cause validation problems with United States Postal Service (USPS) data (PL-13384)**

**Issue** – The `US-Locations.txt` file contains information that does not conform to United States Postal Service (USPS) standards for bulk mailings.

**Workaround** – The provided `US-Locations.txt` file is intended only for use in geocoding to identify addresses for a location. You can edit the `US-Locations.txt` file to conform to your particular address standards, and then import that version of the file instead.

**GX model generated XSD cannot be parsed by JAXB (PL-13598)**

**Issue** – XSD generated by the GX model cannot be parsed by JAXB.

**Workaround** – Add JAXB annotation elements to the XSD to specify the necessary metadata, such as class names, to enable JAXB to generate the Java class files. Contact Guidewire Support for a sample XSD file that is annotated this way.

**Cannot make a field from a delegate into a localized column (PL-13761)**

**Issue** – You cannot make a field from a delegate into a localized column.

**Workaround** – Move the column to be localized off the delegate and onto each of the implementing entities. Then, make the column appear as though it exists on the delegate. To do so, define an enhancement property on the delegate that delegates to the appropriate column, depending on the implementing entity.

**Problem with regen-java-api command and JAR files (PL-16351)**

**Issue** – If you run the `ContactManager/bin/gwpc regen-java-api` command, ContactManager creates a `ContactManager/java-api/lib` directory and puts JAR files for Java development in this directory. However, ContactManager does not always perform this task correctly, resulting in some of the generated JAR files not being copied to this directory.

**Workaround** – For missing library files, copy or import into your Java development environment the missing library files from the following directory:

`ContactManager/admin/lib`

Copy them into the following directory:

`ContactManager/java-api/lib`

**Renaming method or property throws ParseResultsException (PL-16633)**

**Issue** – If you rename a property or a method or change a method signature, and a workflow references that property or method in a Gosu field, ContactManager throws `ParseResultsException`. This behavior is intended.

**Workaround** – Restart the workflow engine. To do so:

1. Log into ContactManager using an administrative account.
2. Access **Internal Tools** → **Reload**.
3. Click **Reload Workflow Engine**.

## Guidewire ContactManager 7.0.5 Release Notes

### Overview of ContactManager 7.0.5 Release Notes

These release notes contain the following:

- “Release Information for ContactManager 7.0.5” on page 351
- “Installing ContactManager 7.0.5” on page 351
- “Support for ContactManager 7.0.5” on page 352
- “Issues and Major Changes for ContactManager 7.0.5” on page 352
- “Improvements and General Issues for ContactManager 7.0.5” on page 353
- “Known Issues and Limitations for ContactManager 7.0.5” on page 355

### Release Information for ContactManager 7.0.5

These release notes apply only to this release of Guidewire ContactManager.

### Version Number Information for ContactManager 7.0.5

This release of Guidewire ContactManager is 7.0.5.

- If you are integrating with Guidewire BillingCenter, this version of ContactManager requires BillingCenter 7.0.0 or later.
- If you are integrating with Guidewire PolicyCenter, this version of ContactManager requires PolicyCenter 7.0.1 or later.
- If you are integrating with Guidewire ClaimCenter, this version of ContactManager requires ClaimCenter 6.0.7 or later.

### Installing ContactManager 7.0.5

For general installation information, see “Installing ContactManager” in the *Guidewire Contact Management Guide*.

For versions of ContactManager prior to 7.0.5 that you have skipped, see:

- “Guidewire ContactManager 7.0.1 Release Notes” on page 311
- “Guidewire ContactManager 7.0.2 Release Notes” on page 326
- “Guidewire ContactManager 7.0.3 Release Notes” on page 337
- “Guidewire ContactManager 7.0.4 Release Notes” on page 344

For versions prior to 7.0.0, see the ClaimCenter release notes for that version, which include sections for ContactCenter release notes. For example, see “Release Notes Archive” on page 135 in the *New and Changed Guide*.

## Support for ContactManager 7.0.5

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

- At the Guidewire Resource Portal – <http://guidewire.custhelp.com>
- By email – [support@guidewire.com](mailto:support@guidewire.com)
- By phone – +1-650-356-4955

## Issues and Major Changes for ContactManager 7.0.5

This topic contains issues and major changes that might affect your installation.

- “Base PCF File Changes for ContactManager 7.0.5” on page 352
- “Rules Changes for ContactManager 7.0.5” on page 352
- “Rules Changes for ContactManager 7.0.5” on page 352

### Base PCF File Changes for ContactManager 7.0.5

All links below require that the `ReleaseNotes_files` directory be on your local disk in the same directory as this release notes file.

#### ContactManager release 7.0.4 to 7.0.5

- To view a report of the changes in the base PCF files in the `modules/ab` directory, [click this link](#).
- To view a report of the changes in the base PCF files in the `modules/p1` directory, [click this link](#).

### Rules Changes for ContactManager 7.0.5

#### ContactManager release 7.0.4 to 7.0.5

- There are no changes to the base rules in the `modules/ab` directory.

### Changes in This Release Provided in Upgrade Diff Report for ContactManager 7.0.5

Guidewire provides a report detailing certain differences between the current release and your prior release. This report describes changes in display keys, entities, typelists, and the Gosu API. To obtain your custom Upgrade Diff Report, visit the Guidewire Resource Portal.

## Improvements and General Issues for ContactManager 7.0.5

This topic describes improvements and issues corrected in this release. Guidewire attempts to provide information for issues of primary importance to our customers. This list is not intended to be comprehensive.

ID	Description
<b>Contact App</b>	
CTC-1747	The <code>LinkableExtensionUpdateLinkIdVersionTrigger</code> upgrade trigger sets <code>LinkID</code> values on all entities that implement <code>ABLinkable</code> before the upgrade process runs. However, the trigger was unable to create a <code>LinkID</code> for tables that did not have a <code>LinkID</code> column. The trigger can now create the <code>LinkID</code> column in a table if the column does not already exist.
<b>Command line tools</b>	
PL-20378	With this release, the <code>verify-types</code> command-line tool does not stop verification if a failure, such as a programming error, occurs while verifying a specific type. The tool reports all errors, warnings and failures.
<b>Database support</b>	
PL-21842	With this release, the <code>system_tools</code> command has additional options for submitting batch jobs that report database performance.
PL-23523	For Oracle, you can configure new LOB columns to use SecureFile LOBs or compressed SecureFile LOBs instead of the default BasicFile LOBs. You can configure the LOB type for the entire database or for specific tables only. For more information, see "Configuring Oracle LOB Types" in the <i>ClaimCenter Installation Guide</i> .
<b>Database upgrade</b>	
PL-23504	With this release, a database upgrade in a development environment records checkpoints of upgrade triggers that complete successfully. You can restart a failed database upgrade, and it resumes with the upgrade trigger that failed. The restart feature helps you test your upgrade with realistically large data sets. You avoid time spent to restore the database and time spent to run upgrade triggers that work successfully.  To restart a test database upgrade from a checkpoint reached in an earlier upgrade, roll back manually any database changes that occurred during the upgrade trigger that failed. In addition, verify that you resolved the problem that caused the trigger to fail before restarting. A test run of your upgrade is successful only when it runs from start to finish without a restart. Never use the restart feature of database upgrade in a production environment.

PL-23686	<p>Generating table statistics during upgrade is now optional for Oracle databases. This change does not affect statistics generation on the Microsoft SQL Server and H2 development databases.</p> <p>You set this new option in the config.xml file. In the &lt;database&gt; block, there is an &lt;upgrade&gt; block that contains configuration information for the overall database upgrade. There is a new attribute in the &lt;upgrade&gt; block, updatestatistics with default value true. If you set this attribute to false, statistics will not be updated for that database during upgrade.</p> <p>If you do not update statistics during upgrade, you are then responsible for updating statistics by using the command-line batch process. Most of the time, the incremental statistics update is sufficient. For example:</p> <pre>maintenance_tools -password password -startprocess incrementaldbstats</pre> <p>If statistics are not updated during the upgrade, you see a warning that recommends that you run the database statistics batch process in incremental mode. Additionally, the UpgradeInfo page shows that statistics were not updated as part of the upgrade. This page also reports the runs of the statistics batch process, and incremental runs are shown.</p> <p>The UpgradeInfo page does not identify the following case: You ran another upgrade with updatestatistics=true since running a previous update with updatestatistics=false, but you did not update statistics first.</p> <p>When you click the Download button on the UpgradeInfo page, you get a more detailed UpgradeInfo report.</p> <ul style="list-style-type: none"> <li>The UpgradeInfo report shows the value of the updatestatistics attribute at the time of upgrade.</li> <li>Additionally, the UpgradeInfo report shows the update statistics SQL statements that were skipped as part of the upgrade. Normally, you do not need to consult this list because running the database statistics batch process in full mode will cover these statements.</li> </ul> <p><b>Note:</b> The incremental mode might not capture all cases that would have been run during an upgrade with updatestatistics=true.</p>
----------	--

**Entities/Metadata**

PL-24640	Resolved an issue with the Gosu type system that led to an infinite loop and a stack overflow exception when constructing an entity type. The issue occurs in earlier releases if an entity uses <implementsInterface> and that interface has a method that returns an array of the same entity.
----------	--

**Gosu**

PL-24177	Resolved an issue that caused the exists method to throw an exception when the where clause included a Guidewire boxed Boolean instead of the Java primitive type boolean.
----------	--

PL-24095, PL-24478	Fixed an issue in which casting errors were thrown when Gosu code called a Java method that expected a specific entity, such as Document, as a parameter. The system would throw an error similar to the following:
-----------------------	---

```
gw.lang.parser.EvaluationException:  
com.guidewire.commons.metadata.proxy._generated.impl.Document cannot be cast to  
com.guidewire.pc.external.entity.Document
```

This error was caused by the way in which Guidewire Java code was making a reflective method call to Gosu and then handling external entity types. Those reflective method calls no longer require casting of the external entity types.

**Integration**

PL-23132	There was a problem with starting a listening thread on JMS when the application server was the latest version of WebSphere. This problem has been fixed for WebSphere 7 and WebLogic 10.3.5 and later. Now you can use a listening thread with a startable plugin on these versions of WebSphere and WebLogic.
----------	---

**Integration – BC, Integration – CC, Integration – PC**

CTC-1700	Updated the ClientSystemPlugin plugin implementations in ContactManager to handle unexpected exceptions. Rather than suspend the message transport, ContactManager now logs the exception and continues running as usual. The affected plugin implementations are BCBillingSystemPlugin, CCCClaimSystemPlugin, and PCPolicySystemPlugin.
----------	--

**Localization**

PL-24462	In previous releases, ContactManager incorrectly handled Imperial dates that equate to Gregorian Dates in the year 1950, as well as other ranges of old dates. The issue occurred because American military bases used daylight saving time in Japan beginning in 1948, which differed from timekeeping used by the general public. With this release, ContactManager handles old Imperial dates correctly.
----------	---

<b>Management plugin</b>	
PL-16543	With this release, the counters NumActiveDBConnections and NumIdleDBConnectionsmanagement are integers instead of strings, so management tools that connect by using JMX can plot them now on graphs.
<b>Performance</b>	
CTC-1828	The ContactManager Message.eix file contains indexes to speed up the messaging subsystem when it searches for messages with a particular type of primary object, such as ABContact. Performance testing found that the existing indexes could be improved. Therefore, improved indexes have been added to Message.eix, resulting in faster query plans and message processing.
<b>Persistence</b>	
PL-23941	For workflow-related distributed work queues, the logging level of the message "WDW processing workitem: ..." changed from INFO to DEBUG to reduce noise in the log files.
<b>Web - UI/Runtime</b>	
PL-23848	PCF input elements now render currency amounts so the scale and appscale parameters are honored. For example, if appscale == 0 and the amount is the integer 1, the input element renders the amount as "1". If appscale == 2 and the amount is the integer 1, the input element renders the amount as "1.00".
<b>Web services – WS-I</b>	
PL-23514	This release introduces a new method, preExecute, for use in the invocation handlers that you write for WS-I web services. In earlier releases, if you wrote an invocation handler, you unavoidably bypassed some important WS-I features. For example, the application: <ul style="list-style-type: none"> <li>• Did not enable profiling for method calls.</li> <li>• Did not check run level annotations, even at the class level.</li> <li>• Did not check web service permission annotations, even at the class level.</li> <li>• Did not check for duplicate external transaction IDs if those SOAP headers were present.</li> </ul> Now, you can call the preExecute method in your invocation handler to assure the application takes the preceding actions, as needed. See the <i>Integration Guide</i> for more information about invocation handlers and how to use the preExecute method.
PL-23721	To detect duplicate operations from external systems, add the new annotation @WsCheckDuplicateExternalTransaction to your WS-I web service implementation class. See the <i>Integration Guide</i> for details for using the annotation and the SOAP header for the transaction ID.

## Known Issues and Limitations for ContactManager 7.0.5

This topic describes known issues with this release of Guidewire ContactManager. It has the following topics:

- “ContactManager Known Issues for Release 7.0.5” on page 355
- “Studio/Platform Known Issues for ContactManager 7.0.5” on page 356

**Note:** For maintenance releases, Guidewire often defers fixing configuration issues if they require merging files during the upgrade. Workarounds to many of these issues are listed in the following sections. The goal of this policy is to make upgrades as straightforward as possible.

### ContactManager Known Issues for Release 7.0.5

#### **RelatedABContactSearchCriteria does not work for related contact searches (CTC-960)**

**Issue** – Using the RelatedABContactSearchCriteria entity and the CriteriaDef elements that are defined for it in search-config.xml is not supported in this version of ContactManager.

**Workaround** – Do not use them. Guidewire is aware of this issue.

**Customizing contact relationship handling to use messaging results in two relationship records being created (CTC-396)**

**Issue** – Changing a contact relationship updates two contact records. However, ContactManager stores the representation of the relationship in one `ContactContact` entity. ContactManager does not check that the relationship does not already exist. Therefore, ContactManager can create multiple, redundant `ContactContact` entities for the same relationship if it receives messages changing the relationship between two contacts.

**Workaround** – In the base configurations of ContactManager, this behavior is not a problem. ClaimCenter makes a direct web service call to ContactManager to update contact relationships and does not use event messaging for related contact updates.

If you customize ClaimCenter and ContactManager to use event messaging to update related contacts between ClaimCenter and ContactManager, your customization must account for the extra message. Updating one relationship changes two contacts, and therefore ClaimCenter generates two messages and sends them to ContactManager. Your customization must ensure that ContactManager creates only one `ContactContact` entity in this case.

**Email field validation in ContactManager not the same as in core applications (CTC-1339)**

**Issue** – The entry in `fieldvalidators.xml` in ContactManager is more restrictive than in the core applications. This difference can cause issues while creating a contact in a core application.

**Workaround** – Change the ContactManager version of this validator to match the validators in the core products. Use Studio to edit the entry in `fieldvalidators.xml` for `Validator.Email`, changing its value from the default ".+@.+\\..+" to the value in the core applications, ".+@.+".

## Studio/Platform Known Issues for ContactManager 7.0.5

**Issues with Internet Explorer 9**

**Issue** – If you are using the Internet Explorer 9 browser, it is possible to see issues such as screen flickering or an incorrect tab order for fields. According to public reports, Internet Explorer 9 exhibits these and other issues with a variety of web sites and web applications.

**Workaround** – Because this is the behavior of the Internet Explorer 9 rendering engine, Guidewire cannot address these issues. However, there are reports of an Internet Explorer 9 workaround that reduces these issues. In Internet Explorer 9, you can change the use of software or hardware rendering by toggling the **Accelerated Graphics** option on the **Advanced** tab of the **Internet Options** dialog.

**Studio Rules do not use correct capitalization for root object's name (PL-10740)**

**Issue** – Rule set root objects are not named with first letter lower-cased.

**Workaround** – Guidewire is aware of this issue.

**User interface cannot handle starting multiple instances of a batch process (PL-12372)**

**Issue** – The user interface cannot handle starting multiple instances of a batch process.

**Workaround** – To execute multiple instances of a batch process, start them from the command line. Also, to allow multiple instances to be run simultaneously, you must ensure that the `BatchProcess.isExclusive()` method returns `false`.

**Type system refresh after PCF page title change does not update corresponding menu label (PL-13057)**

**Issue** – The type system refresh after a PCF page title change does not update the corresponding menu label.

**Workaround** – After updating a page title, restart the server to refresh menu labels and avoid null pointer exceptions due to stale references.

**Length limitation on entity localization table names (PL-13360)**

**Issue** – Entity localization table names have a shorter, 16-character limit than other entity tables. If the localization table name exceeds the 16-character limit, the error message indicates incorrectly that 18 characters are allowed.

**Workaround** – Ensure that the localization `tableName` property specified in the entity extension file is less than 16 characters.

**US-Locations.txt file with the US geodata from Great Data has special characters that cause validation problems with United States Postal Service (USPS) data (PL-13384)**

**Issue** – The `US-Locations.txt` file contains information that does not conform to United States Postal Service (USPS) standards for bulk mailings.

**Workaround** – The provided `US-Locations.txt` file is intended only for use in geocoding to identify addresses for a location. You can edit the `US-Locations.txt` file to conform to your particular address standards, and then import that version of the file instead.

**GX model generated XSD cannot be parsed by JAXB (PL-13598)**

**Issue** – XSD generated by the GX model cannot be parsed by JAXB.

**Workaround** – Add JAXB annotation elements to the XSD to specify the necessary metadata, such as class names, to enable JAXB to generate the Java class files. Contact Guidewire Support for a sample XSD file that is annotated this way.

**Cannot make a field from a delegate into a localized column (PL-13761)**

**Issue** – You cannot make a field from a delegate into a localized column.

**Workaround** – Move the column to be localized off the delegate and onto each of the implementing entities. Then, to make the column appear to be on the delegate, define an enhancement property on the delegate that `delegates` to the appropriate column, depending on the implementing entity.

**Problem with regen-java-api command and JAR files (PL-16351)**

**Issue** – If you run the `ContactManager/bin/gwpc regen-java-api` command, ContactManager creates a `ContactManager/java-api/lib` directory and puts JAR files for Java development in this directory. However, ContactManager does not always perform this task correctly, resulting in some of the generated JAR files not being copied to this directory.

**Workaround** – For missing library files, copy or import into your Java development environment the missing library files from the following directory:

`ContactManager/admin/lib`

Copy them into the following directory:

`ContactManager/java-api/lib`

**Renaming method or property throws ParseResultsException (PL-16633)**

**Issue** – If you rename a property or a method or change a method signature, and a workflow references that property or method in a Gosu field, ContactManager throws `ParseResultsException`. This behavior is intended.

**Workaround** – Restart the workflow engine. To do so:

1. Log into ContactManager by using an administrative account.
2. Access Internal Tools → Reload.
3. Click Reload Workflow Engine.

**In some languages, web browsers render column headers of list views improperly (PL-18027)**

**Issue** – In some languages, web browsers render some column headers of list views improperly if their column widths are specified too narrowly in their PCF definitions. For example, sometimes a numeric column is specified with a variable width of 1%. This narrow setting forces the browser to render the column too narrowly for the text of the translated column heading.

**Workaround** – Edit the PCF file that defines the column and clear the value from the width property. Without a specified value for the column width, browsers render the column widely enough to display the full text of the translated column heading.

**Database upgrade does not handle nullable to non-nullable columns with a default value for subtypes (PL-23104)**

**Issue** – For entity definitions, the automatic database upgrade converts nullable columns to non-nullable with a default value successfully. However, this column type conversion is not possible for columns in subtype definitions. ContactManager implements non-nullable columns on subtype in the database as nullable because that column must have null values for rows that represent instances of other subtypes.

**Workaround** – Write a version trigger to populate the column with the default value for existing rows for the subtype. After you upgrade, ContactManager enforces the column value to be non-nullable with the default value for new rows of the subtype.

**New transport plugin definitions do not show in the list of valid transport plugins (PL-23317)**

**Issue** – Newly implemented transport plugin definitions do not show in the list of valid transport plugins. You display this list by clicking the light-bulb icon next to the Transport Plugin field in the messaging destination editor.

**Workaround** – Restart Studio.

**Find in Resources fails for resources under Data Model Extensions or Web Resources (PL-23320)**

**Issue** – In Studio, the Find in Resources option does not work for resources that are in Data Model Extensions or Web Resources.

**Workaround** – To see the full resource path, hover over the resource name in the tab of the resource editor. Then, navigate in the resource pane on the left to find the resource in the resource hierarchy.

## Guidewire ContactManager 7.0.6 Release Notes

These release notes contain the following:

- “Release Information for ContactManager 7.0.6” on page 358
- “Installing ContactManager 7.0.6” on page 359
- “Support for ContactManager 7.0.6” on page 359
- “Issues and Major Changes for ContactManager 7.0.6” on page 359
- “Improvements and General Issues for ContactManager 7.0.6” on page 360
- “Known Issues and Limitations for ContactManager 7.0.6” on page 360

### Release Information for ContactManager 7.0.6

These release notes apply only to this release of Guidewire ContactManager.

This release of Guidewire ContactManager is 7.0.6.

- If you are integrating with Guidewire BillingCenter, this version of ContactManager requires BillingCenter 7.0.0 or later.

- If you are integrating with Guidewire PolicyCenter, this version of ContactManager requires PolicyCenter 7.0.1 or later.
- If you are integrating with Guidewire ClaimCenter, this version of ContactManager requires ClaimCenter 6.0.7 or later.

## Installing ContactManager 7.0.6

For general installation information, see “[Installing ContactManager](#)” in the *Guidewire Contact Management Guide*.

For versions of ContactManager prior to 7.0.5 that you have skipped, see:

- “[Guidewire ContactManager 7.0.1 Release Notes](#)” on page 311
- “[Guidewire ContactManager 7.0.2 Release Notes](#)” on page 326
- “[Guidewire ContactManager 7.0.3 Release Notes](#)” on page 337
- “[Guidewire ContactManager 7.0.4 Release Notes](#)” on page 344
- “[Guidewire ContactManager 7.0.5 Release Notes](#)” on page 351

For versions prior to 7.0.0, see the ClaimCenter release notes for that version, which include sections for ContactCenter release notes. For example, see “[Release Notes Archive](#)” on page 135 in the *New and Changed Guide*.

## Support for ContactManager 7.0.6

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

- At the Guidewire Resource Portal – <http://guidewire.custhelp.com>
- By email – [support@guidewire.com](mailto:support@guidewire.com)
- By phone – +1-650-356-4955

## Issues and Major Changes for ContactManager 7.0.6

This topic contains issues and major changes that might affect your installation.

- Base PCF File Changes for ContactManager 7.0.6
- Changes in ContactManager 7.0.6 Provided in Upgrade Diff Report

### Base PCF File Changes for ContactManager 7.0.6

All links below require that the `ReleaseNotes_files` directory be on your local disk in the same directory as this release notes file.

#### ContactManager release 7.0.5 to 7.0.6

- To view a report of the changes in the base PCF files in the `modules/ab` directory, [click here](#).
- To view a report of the changes in the base PCF files in the `modules/p1` directory, [click here](#).

### Changes in ContactManager 7.0.6 Provided in Upgrade Diff Report

Guidewire provides a report detailing certain differences between the current release and your prior release. This report describes changes in display keys, entities, typelists, and the Gosu API. To obtain your custom Upgrade Diff Report, visit the Guidewire Resource Portal.

## Improvements and General Issues for ContactManager 7.0.6

This topic describes improvements and issues corrected in this release. Guidewire attempts to provide information for issues of primary importance to our customers. This list is not intended to be comprehensive.

ID	Description
<b>Contact App</b>	
CTC-2032	Previously, editing a contact's PrimaryAddress did not by itself cause the ABContactChanged event to be fired. This problem has been fixed by adding all properties in the Address entity to the history rules for primary and secondary addresses.
CTC-2065	Adding additional criteria to the Address entity in search-config.xml now works correctly.
<b>Core</b>	
CTC-2509	There was a problem with name related constraints for ABPerson entities not being enforced. If an ABPerson subtype had both Name and LastName set, both fields would be saved to the database, causing a consistency check to fail. This issue has been fixed. The Name field is now set to null on commit to the database, and a warning message is logged.
<b>Integration – BC, Integration – CC, Integration – PC</b>	
CTC-1887	When a validation error happened in the commit of an ABContact, ContactManager was throwing a retryable exception. The client application was then retrying the message and ultimately suspending the message transport due to the repeated failures. ContactManager now returns a non-retryable exception in this case, enabling the client application to handle the failure properly.
<b>Integration – CC</b>	
CTC-1922	You can now extend the data returned for related contact searches in ContactManager. The classes RelatedContactInfoContainer and ABContactAPIRelatedContact now have the annotation @Export and can be edited.
<b>Miscellaneous</b>	
CTC-1375	In the base configuration, the default logging directory specified in logging.properties now matches the logging directory setting for the configured log files.
CTC-1753	There was an issue that caused erroneous consistency check failures after merging contacts. Retired contacts were not being filtered from the consistency checks that ensure that addresses are not shared by contacts. This issue has been fixed.

## Known Issues and Limitations for ContactManager 7.0.6

This topic describes known issues with this release of Guidewire ContactManager. It has the following topics:

- “ContactManager 7.0.6 Known Issues” on page 360
- “Studio/Platform Known Issues for ContactManager 7.0.6” on page 361

**Note:** For maintenance releases, Guidewire often defers fixing configuration issues if they require merging files during the upgrade. Workarounds to many of these issues are listed in the following sections. The goal of this policy is to make upgrades as straightforward as possible.

### ContactManager 7.0.6 Known Issues

#### RelatedABContactSearchCriteria does not work for related contact searches (CTC-960)

**Issue** – Using the RelatedABContactSearchCriteria entity and the CriteriaDef elements that are defined for it in search-config.xml is not supported in this version of ContactManager.

**Workaround** – Do not use them. Guidewire is aware of this issue.

**Customizing contact relationship handling to use messaging results in two relationship records being created (CTC-396)**

**Issue** – Changing a contact relationship updates two contact records. However, ContactManager stores the representation of the relationship in one `ContactContact` entity. ContactManager does not check to make sure the relationship does not already exist, and therefore can create multiple, redundant `ContactContact` entities for the same relationship if it receives messages changing the relationship between two contacts.

**Workaround** – In the base configurations of ContactManager, this behavior is not a problem. ClaimCenter makes a direct web service call to ContactManager to update contact relationships and does not use event messaging for related contact updates.

If you customize ClaimCenter and ContactManager to use event messaging to update related contacts between ClaimCenter and ContactManager, your customization must account for the extra message. Updating one relationship changes two contacts, and therefore ClaimCenter generates two messages and sends them to ContactManager. Your customization must ensure that ContactManager creates only one `ContactContact` entity in this case.

**Email field validation in ContactManager not the same as in core applications (CTC-1339)**

**Issue** – The entry in `fieldvalidators.xml` in ContactManager is more restrictive than in the core applications. This difference can cause issues while creating a contact in a core application.

**Workaround** – Change the ContactManager version of this validator to match the validators in the core products. Use Studio to edit the entry in `fieldvalidators.xml` for `Validator.Email`, changing its value from the default ".+@.+\\..+" to the value in the core applications, ".+@.+".

## [Studio/Platform Known Issues for ContactManager 7.0.6](#)

**Issues with Internet Explorer 9**

**Issue** – If you are using the Internet Explorer 9 browser, it is possible to see issues such as screen flickering or an incorrect tab order for fields. According to public reports, Internet Explorer 9 exhibits these and other issues with a variety of web sites and web applications.

**Workaround** – Because this is the behavior of the Internet Explorer 9 rendering engine, Guidewire cannot address these issues. However, there are reports of an Internet Explorer 9 workaround that reduces these issues. In Internet Explorer 9, you can try to change the use of software or hardware rendering by toggling the Accelerated Graphics option on the Advanced tab of the Internet Options dialog.

**Studio Rules do not use correct capitalization for root object's name (PL-10740)**

**Issue** – Rule set root objects are not named with first letter lower-cased.

**Workaround** – Guidewire is aware of this issue.

**User interface cannot handle starting multiple instances of a batch process (PL-12372)**

**Issue** – The user interface cannot handle starting multiple instances of a batch process.

**Workaround** – To execute multiple instances of a batch process, start them from the command line. Also, to allow multiple instances to be run simultaneously, you must ensure that the `BatchProcess.isExclusive()` method returns `false`.

**Type system refresh after PCF page title change does not update corresponding menu label (PL-13057)**

**Issue** – The type system refresh after a PCF page title change does not update the corresponding menu label.

**Workaround** – After updating a page title, restart the server to refresh menu labels and avoid null pointer exceptions due to stale references.

**Length limitation on entity localization table names (PL-13360)**

**Issue** – Entity localization table names have a shorter, 16-character limit than other entity tables. If the localization table name exceeds the 16-character limit, the error message indicates incorrectly that 18 characters are allowed.

**Workaround** – Ensure that the localization `tableName` property specified in the entity extension file is less than 16 characters.

**Cannot make a field from a delegate into a localized column (PL-13761)**

**Issue** – You cannot make a field from a delegate into a localized column.

**Workaround** – Move the column to be localized off the delegate and onto each of the implementing entities. Then, to make the column appear as though it exists on the delegate, define an enhancement property on the delegate that *delegates* to the appropriate column, depending on the implementing entity.

**Problem with regen-java-api command and JAR files (PL-16351)**

**Issue** – If you run the `ContactManager/bin/gwpc regen-java-api` command, ContactManager creates a `ContactManager/java-api/lib` directory and puts JAR files for Java development in this directory. However, ContactManager does not always perform this task correctly, resulting in some of the generated JAR files not being copied to this directory.

**Workaround** – For missing library files, copy or import into your Java development environment the missing library files from the following directory:

`ContactManager/admin/lib`

Copy them into the following directory:

`ContactManager/java-api/lib`

**Renaming method or property throws ParseResultsException (PL-16633)**

**Issue** – If you rename a property or a method or you change a method signature, and a workflow references that property or method in a Gosu field, ContactManager throws `ParseResultsException`. This behavior is intended.

**Workaround** – Restart the workflow engine. To do so:

1. Log into ContactManager by using an administrative account.
2. Access Internal Tools → Reload.
3. Click Reload Workflow Engine.

**In some languages, web browsers render column headers of list views improperly (PL-18027)**

**Issue** – In some languages, web browsers render some column headers of list views improperly if their column widths are specified too narrowly in their PCF definitions. For example, sometimes a numeric column is specified with a variable width of 1%. This narrow setting forces the browser to render the column too narrowly for the text of the translated column heading.

**Workaround** – Edit the PCF file that defines the column and clear the value from the width property. Without a specified value for the column width, browsers render the column widely enough to display the full text of the translated column heading.

**Database upgrade does not handle nullable to non-nullable columns with a default value for subtypes (PL-23104)**

**Issue** – For entity definitions, the automatic database upgrade converts nullable columns to non-nullable with a default value successfully. However, this column type conversion is not possible for columns in subtype definitions. ContactManager implements non-nullable columns on subtype in the database as nullable because that column must have null values for rows that represents instances of other subtypes.

**Workaround** – Write a version trigger to populate the column with the default value for existing rows for the subtype. After you upgrade, ContactManager enforces the column value to be non-nullble with the default value for new rows of the subtype.

**New transport plugin definitions do not show in the list of valid transport plugins (PL-23317)**

**Issue** – Newly implemented transport plugin definitions do not show in the list of valid transport plugins displayed by clicking the light-bulb icon next to the Transport Plugin field in the messaging destination editor.

**Workaround** – Restart Studio.

**Find in Resources fails for resources under Data Model Extensions or Web Resources (PL-23320)**

**Issue** – In Studio, the Find in Resources option does not work for resources that are in Data Model Extensions or Web Resources.

**Workaround** – To see the full resource path, hover over the resource name in the tab of the resource editor. Then, navigate in the resource pane on the left to find the resource in the resource hierarchy.

## Guidewire ContactManager 8.0.0 Release Notes

### Overview of ContactManager 8.0.0 Release Notes

These release notes contain the following sections:

- “Release Information for ContactManager 8.0.0” on page 363
- “Installing ContactManager 8.0.0” on page 363
- “Support for ContactManager 8.0.0” on page 364
- “Issues and Major Changes for ContactManager 8.0.0” on page 364
- “Known Issues and Limitations for ContactManager 8.0.0” on page 364

### Release Information for ContactManager 8.0.0

These release notes apply only to this release of Guidewire ContactManager.

**IMPORTANT** If you are upgrading and you skipped one or more releases of ContactManager, be sure to read the release notes for those releases to learn about changes and fixed issues.

### Version Number Information for ContactManager 8.0.0

This release of Guidewire ContactManager is 8.0.0.

- If you are integrating with Guidewire BillingCenter, this version of ContactManager requires BillingCenter 7.0.0 or later. The latest maintenance release is preferred.
- If you are integrating with Guidewire PolicyCenter, this version of ContactManager requires PolicyCenter 7.0.1 or later. The latest maintenance release is preferred.
- If you are integrating with Guidewire ClaimCenter, this version of ContactManager requires ClaimCenter 7.0.0 or later. The latest maintenance release is preferred.

### Installing ContactManager 8.0.0

For general installation information, see “Installing ContactManager” on page 39.

For versions of ContactManager prior to 8.0.0 that you have skipped, see:

- “Guidewire ContactManager 7.0.1 Release Notes” on page 311
- “Guidewire ContactManager 7.0.2 Release Notes” on page 326
- “Guidewire ContactManager 7.0.3 Release Notes” on page 337
- “Guidewire ContactManager 7.0.4 Release Notes” on page 344
- “Guidewire ContactManager 7.0.5 Release Notes” on page 351
- “Guidewire ContactManager 7.0.6 Release Notes” on page 358

For versions prior to 7.0.0, see the ClaimCenter release notes for that version, which include sections for ContactCenter release notes. For example, see “Release Notes Archive” on page 135 in the *New and Changed Guide*.

## Support for ContactManager 8.0.0

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

- At the Guidewire Resource Portal – <http://guidewire.custhelp.com>
- By email – [support@guidewire.com](mailto:support@guidewire.com)
- By phone – +1-650-356-4955

## Issues and Major Changes for ContactManager 8.0.0

This topic contains issues and major changes that might affect your installation.

- “Base PCF File Changes for ContactManager 8.0.0” on page 364
- “Rules Changes for ContactManager 8.0.0” on page 364
- “Changes in ContactManager 8.0.0 Provided in Upgrade Diff Report” on page 364

### Base PCF File Changes for ContactManager 8.0.0

The link that follows requires that the `ReleaseNotes_files` directory be on your local disk in the same directory as this release notes file.

#### ContactManager release 7.0.5 to 8.0.0

To view a report of the changes to the base PCF files, *click this link*.

### Rules Changes for ContactManager 8.0.0

The link that follows requires that the `ReleaseNotes_files` directory be on your local disk in the same directory as this release notes file.

#### ContactManager release 7.0.5 to 8.0.0

To view a report of the changes to the base rules, *click this link*.

### Changes in ContactManager 8.0.0 Provided in Upgrade Diff Report

Guidewire provides a report detailing certain differences between the current release and your prior release. This report describes changes in display keys, entities, typelists, and the Gosu API. To obtain your custom Upgrade Diff Report, visit the Guidewire Resource Portal.

## Known Issues and Limitations for ContactManager 8.0.0

This section describes known issues with this release of Guidewire ContactManager:

- “ContactManager Known Issues for Release 8.0.0” on page 365
- “Platform/Studio Known Issues for ContactManager 8.0.0” on page 366

## ContactManager Known Issues for Release 8.0.0

### **Customizing contact relationship handling to use messaging results in two relationship records being created (CTC-396)**

**Issue** – Changing a contact relationship updates two contact records. However, ContactManager stores the representation of the relationship in one `ContactContact` entity. ContactManager does not verify that the relationship does not already exist. Therefore ContactManager can create multiple, redundant `ContactContact` entities for the same relationship if it receives messages changing the relationship between two contacts.

**Workaround** – In the base configuration of ContactManager, this behavior is not a problem. ClaimCenter makes a direct web service call to ContactManager to update contact relationships and does not use event messaging for related contact updates.

If you customize ClaimCenter and ContactManager to use event messaging to update related contacts between ClaimCenter and ContactManager, your customization must account for the extra message. Updating one relationship changes two contacts, and therefore ClaimCenter generates two messages and sends them to ContactManager. Your customization must ensure that ContactManager creates only one `ContactContact` entity in this case.

### **RelatedABContactSearchCriteria does not work for related contact searches (CTC-960)**

**Issue** – Using the `RelatedABContactSearchCriteria` entity and the `CriteriaDef` elements that are defined for it in `search-config.xml` is not supported in this version of ContactManager.

**Workaround** – Do not use them. Guidewire is aware of this issue.

### **ClaimCenter ignoring ContactMapper persist properties when a contact is initially linked (CTC-2112)**

**Issue** – Setting `.withPersist(false)` for the field of a `Contact` entity or subentity in the `ContactMapper` class in ClaimCenter does not always have the intended effect. It is supposed to prevent the field from being persisted in the ClaimCenter database. However, in the following case, ClaimCenter is ignoring this setting. When you initially add an existing contact from ContactManager to ClaimCenter, all properties on the contact brought over from ContactManager are saved in the ClaimCenter database.

Updates from ContactManager for subsequent sync operations on the linked contact do work correctly. These subsequent sync operations do not persist properties that have been marked `.withPersist(false)`.

**Workaround** – Guidewire is aware of this issue.

### **The DefaultApplicationLanguage property is missing from the config.xml file (CTC-2238)**

**Issue** – In the base configuration, the configuration file `config.xml` is missing the `DefaultApplicationLanguage` property. This issue is a problem only because developers expect to see this parameter in the file and might not think to set it if it is not there. There are other parameters that are not in `config.xml` that can be set simply by adding and setting them, as needed, so a missing parameter is not necessarily a problem.

**Workaround** – Add the property to `config.xml` and set it to the default language. During installation, and before starting the database for the first time, you must provide a value for this configuration parameter. This value must be a typecode that exists in the `LanguageType` typelist.

## Platform/Studio Known Issues for ContactManager 8.0.0

### **Chrome browser cannot display product documentation in HTML format (DOC-7251)**

**Issue** – If you use the Google Chrome browser, to view the HTML Guidewire product documentation, it must be served by an HTTP server using the `http://` protocol. The Chrome browser cannot load HTML product documentation from your local disk by using the `file://` protocol.

**Workaround** – Use a different browser, such as Microsoft Internet Explorer. If you use the HTML documentation only to access the PDF files, you can open them directly in the `pdf` subdirectory of the `doc` directory.

### **GX model generated XSD cannot be parsed by JAXB (PL-13598)**

**Issue** – XSD generated by the GX model cannot be parsed by JAXB.

**Workaround** – Add JAXB annotation elements to the XSD to specify the necessary metadata, such as class names, to enable JAXB to generate the Java class files. Contact Guidewire Support for a sample XSD file that is annotated this way.

### **Problem with regen-java-api command and JAR files (PL-16351)**

**Issue** – If you run the `gwab regen-java-api` command, ContactManager creates a `ContactManager/java-api/lib` directory and puts JAR files for Java development in this directory. However, ContactManager does not always perform this task correctly, resulting in some of the generated JAR files not being copied to this directory.

**Workaround** – For missing library files, copy or import into your Java development environment the missing library files from the following directory:

`ContactManager/admin/lib`

Copy them into the following directory:

`ContactManager/java-api/lib`

### **Renaming method or property throws ParseResultsException (PL-16633)**

**Issue** – If you rename a property or a method or change a method signature, and a workflow references that property or method in a Gosu field, ContactManager throws `ParseResultsException`. This is the intended behavior.

**Workaround** – Restart the workflow engine. To do so:

1. Log in to ContactManager using an administrative account.
2. Access Internal Tools → Reload.
3. Click Reload Workflow Engine.

### **Gosu class can override @InternalAPI methods when a PublishInGosu java class is subclassed (PL-18217)**

**Issue** – When a Gosu class extends a Java class, it is possible for the Gosu class to override methods in the Java class that are marked as `@InternalAPI`. These overrides could lead to unpredictable behavior.

**Workaround** – Do not override methods marked as `@InternalAPI` when creating a Gosu class that subclasses a Java class.

### **Client-side document production scripts cannot be customized in this release (PL-21502)**

**Issue** – In previous releases, you could customize client-side document production scripts downloaded by the ActiveX Document Assistant. You could modify JScript files in the web application and remove the cached copies from a `temp` directory on all user computers. In this release, the ActiveX control was replaced by a signed Java Web Start (JWS) application. Because client-side scripts are encapsulated in the signed JWS application, you cannot change the scripts in this release.

**Workaround** – Guidewire is aware of this issue.

**New inbound integration system requires additional configuration information (PL-25227)**

**Issue** – This release includes a new inbound integration system, which is documented in the ContactManager Integration Guide. Additional configuration information is necessary to use the new API in this release.

**Workaround** – Contact Guidewire Customer Support for details.

**JBoss 6 application server unable to start (PL-27203)**

**Issue** – JBoss 6 generates an exception when it is deployed with ContactManager 8.0.0.

**Workaround** – Remove or comment out the tag <resource-ref> in the file web.xml.

**Multiple rule folders are created during a configuration upgrade (PL-27338)**

**Issue** – Multiple rule folders are created if you repeatedly run the configuration upgrade tool followed by the clean command.

**Workaround** – Restore the innermost rules folder to its proper location.

1. Copy the innermost folder to a temporary location.
2. Remove all the nested folders from the original location.
3. Copy back the innermost folder from the temporary location to its proper location.
4. Make sure the folder is named correctly (ContactManager8\_0\_0Rules).

**List view columns that are initially not visible and then set to visible always appear on the right side (PL-27556)**

**Issue** – In ContactManager 8.0, you can reorder the columns of list views or change their width. These settings are then saved by ContactManager in the web browser for each list view, so the same ordering and width can be used when the page is revisited.

However, a layout of a list view can change due to differences in data or because the server configuration has changed. New columns added to the list view since the last time the user visited show up on the far right side of the list view. They are not in the order specified in the PCF file. This can be confusing, especially when you must scroll the page to the right to see the new columns.

This behavior can also occur when there are two modes of a PCF page containing list views of a similar structure, but with a different ordering of columns. The order and width settings can be applied to the wrong list view in this case, and columns can appear in a different order than intended.

**Workaround** – If list view columns seem to be missing, first scroll to the right to see if they are there. To correct the order of list view columns, you can reset your layout preferences to restore the default list ordering and widths. To do this, select Options → Clear Layout Preference in Guidewire ContactManager.

**Javadoc command does not generate index file in expected location (PL-27679)**

**Issue** – When you run gwab regen-java-api, an index.html file is not created in *ContactManager/java-api/doc*.

**Workaround** – The command now generates Javadoc JAR files in *ContactManager/java-api/doc*. To view the Javadoc, add the JAR files to the Studio project.

**Command to generate data dictionary fails if maxSPVInclusions option is specified (PL-27693)**

**Issue** – The data dictionary is not generated when you run the gwab regen-dictionary command with the maxSPVInclusions option.

**Workaround** – Do not use the maxSPVInclusions option with this command.

**Gosu does not automatically downcast if the left side of the typeis or typeof expression uses deprecated members (PL-27724)**

**Issue** – To improve readability of your Gosu code, Gosu automatically downcasts after a `typeis` expression if the type is a subtype of the original type. This is particularly valuable for `if` statements and similar Gosu structures. For example, if a variable has type `Object`, you can use code such as:

```
if( x typeis String ) {  
    length // NOTE: length is a property on String, but *not* on Object.  
}
```

In this release, Gosu does not automatically downcast if the left side of the `typeis` or `typeof` expression uses deprecated members. This may result in new compilation errors.

**Workaround** – To fix these compiler errors, explicitly downcast with the “as” keyword before you access properties or methods on the subtype but not on the original type. For example, suppose a property called `Dep` is deprecated:

```
if (x.Dep typeis ExampleType) {  
    return (x.Dep as ExampleType).PropertyOnExampleSubtype  
}
```

**Upgrade trigger for postOnChange on PCF widgets is not working in some cases (PL-27755)**

**Issue** – In PCF files for some widgets, the ContactManager 8.0.0 upgrade tool does not upgrade the `postOnChange` property to the new syntax for this property. This problem can occur with any widgets that you have added that use `postOnChange` and with any widgets in the base configuration for which you have set `postOnChange`.

**Workaround** – After running the ContactManager 8.0.0 upgrade tool:

1. Find all instances of widgets that did not have their `postOnChange` properties converted. For example, search the `ContactManager/modules/configuration/config/webpcf` folder and subfolders for occurrences of `postOnChange=`. The instances that you will find are either widgets that need correction or widgets that are commented out (disabled.) There is no need to make the correction on disabled widgets, although there is also no harm in doing so.
2. For each widget that needs correction, open its PCF file in an XML editor and change the following old syntax to the new syntax:
  - Old syntax example:

```
<Input id="abx"  
      postOnChange="true"  
      onChange="someMethod()"  
      disablePostOnEnter="doEvaluation()"/>
```
  - New syntax example:

```
<Input id="abx">  
  <PostOnChange onChange="someMethod()" disablePostOnEnter="doEvaluation()"/>  
</Input>
```
3. To verify that you have corrected all instances, open Guidewire Studio and navigate in the **Project** window to **configuration** → **config** → **Page Configuration**. Then compile all files in the `pcf` folder. If there are no errors relating to `postOnChange`, your corrections are complete.

**New mechanism for reloading Gosu classes (DOC-8218)**

**Issue** – In past releases, you could reload your PCF files and Gosu classes in Studio by using a key combination. To reload the changes into the running server, you pressed **Alt+Shift+L** in the application user interface. This shortcut no longer loads Gosu classes.

**Workaround** – To have the server reload your Gosu classes, in Studio, click **Build** → **Make Project**. When Studio is finished compiling your project, the changes will be loaded. You can also restart your server to load the Gosu classes.

# Guidewire ContactManager 8.0.1 Release Notes

These release notes contain the following topics:

- “Release Information for ContactManager 8.0.1” on page 369
- “Installing ContactManager 8.0.1” on page 369
- “Support for ContactManager 8.0.1” on page 369
- “Issues and Major Changes for ContactManager 8.0.1” on page 370
- “Improvements and General Issues for ContactManager 8.0.1” on page 370
- “Known Issues and Limitations for ContactManager 8.0.1” on page 377

## Release Information for ContactManager 8.0.1

These release notes apply only to this release of Guidewire ContactManager.

**IMPORTANT** If you skipped one or more upgrade releases to ContactManager, be sure to read the release notes for those releases to learn about changes and fixed issues.

This release of Guidewire ContactManager is 8.0.1.

- If you are integrating with Guidewire BillingCenter, this version of ContactManager requires BillingCenter 7.0.0 or later. The latest maintenance release is preferred.
- If you are integrating with Guidewire PolicyCenter, this version of ContactManager requires PolicyCenter 7.0.1 or later. The latest maintenance release is preferred.
- If you are integrating with Guidewire ClaimCenter, this version of ContactManager requires ClaimCenter 7.0.0 or later. The latest maintenance release is preferred.

## Installing ContactManager 8.0.1

For general installation information, see “Installing ContactManager” on page 39.

For versions of ContactManager prior to 8.0.1 that you have skipped, see:

- “Guidewire ContactManager 7.0.1 Release Notes” on page 311
- “Guidewire ContactManager 7.0.2 Release Notes” on page 326
- “Guidewire ContactManager 7.0.3 Release Notes” on page 337
- “Guidewire ContactManager 7.0.4 Release Notes” on page 344
- “Guidewire ContactManager 7.0.5 Release Notes” on page 351
- “Guidewire ContactManager 7.0.6 Release Notes” on page 358
- “Guidewire ContactManager 8.0.0 Release Notes” on page 363

For versions prior to 7.0.0, see the ClaimCenter release notes for that version, which include sections for ContactCenter release notes. For example, see “Release Notes Archive” on page 135 in the *New and Changed Guide*.

## Support for ContactManager 8.0.1

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

- At the Guidewire Resource Portal – <http://guidewire.custhelp.com>
- By email – [support@guidewire.com](mailto:support@guidewire.com)
- By phone – +1-650-356-4955

## Issues and Major Changes for ContactManager 8.0.1

This topic contains the following changes that might affect your installation. For information on new features and major changes, see the topic “New and Changed in ClaimCenter 8.0.1” in the ClaimCenter New and Changed Guide.

- “Base PCF File Changes for ContactManager 8.0.1” on page 370
- “Rules Changes for ContactManager 8.0.1” on page 370
- “Changes in ContactManager 8.0.1 Provided in Upgrade Diff Report” on page 370

### Base PCF File Changes for ContactManager 8.0.1

The link that follows requires that the `ReleaseNotes_files` directory be on your local disk in the same directory as this release notes file.

#### ContactManager release 8.0.0 to 8.0.1

To view a report of the changes to the base PCF files, [click here](#).

### Rules Changes for ContactManager 8.0.1

The link that follows requires that the `ReleaseNotes_files` directory be on your local disk in the same directory as this release notes file.

#### ContactManager release 8.0.0 to 8.0.1

To view a report of the changes to the base rules, [click here](#).

### Changes in ContactManager 8.0.1 Provided in Upgrade Diff Report

Guidewire provides a report detailing certain differences between the current release and your prior release. This report describes changes in display keys, entities, typelists, and the Gosu API. To obtain your custom Upgrade Diff Report, visit the Guidewire Resource Portal.

## Improvements and General Issues for ContactManager 8.0.1

This topic describes improvements and issues corrected in this release. Guidewire attempts to provide information for issues of primary importance to our customers. This list is not intended to be comprehensive.

ID	Description
<b>Application Server</b>	
PL-27203	JBoss 6 deployment now works properly.
<b>Archiving</b>	
PL-26930	Modified the <code>restoreSeveredTransactionOffsetOnsetOnsetLinks</code> method so it can be called from Gosu.
<b>Bundles and Transactions</b>	
PL-22240	The touch method has been re-implemented. Please refer to the <a href="#">Gosudoc</a> for detailed usage.
<b>Cognos Integration</b>	
PL-21153	Thread usage on the LDAP server is now managed more efficiently.
<b>Command Line Tools</b>	
PL-27378	Removed debug-start from the command-line options.
PL-28544	The obsolete command line command copy-theme has been removed.

<b>Configuration Upgrade</b>	
PL-27207	The log file of the upgrade tool now reflects the difference between platform upgrade triggers and application-specific upgrade triggers.
PL-27208	You can now expand all and collapse all in the upgrade tool.
PL-27209	You can now right-click to export the directory tree within the upgrade tool.
PL-27338	Fixed an issue in which running upgrade multiple times created nested rules folders.
PL-27744	Fixed an issue that occurred if you added any <code>ClaimContactInput</code> widgets that use <code>postOnChange</code> , or if you modified a base <code>ClaimContactInput</code> widget by adding <code>postOnChange</code> . In these cases, the <code>postOnChange</code> would not get upgraded to the new syntax.
<b>Core</b>	
CTC-473	In the Merge Contacts screen, the <b>Ignore</b> buttons for each row representing a duplicate contact pair have been replaced with check boxes. There is now an <b>Ignore</b> button in the toolbar for this screen that applies to all checked rows. If you select a check box for any row, clicking <b>Ignore</b> applies to that checked row. When you click <b>Ignore</b> , before the action takes place, ContactManager displays a confirmation message that enables you to cancel the action.
CTC-2361	The Find Duplicates process now varies based on the locale. For example, for the Japanese locale, there is no TaxID involved in the query for the Find Duplicates process, and there is no possibility for an exact match, only a potential match. This behavior matches the Find Duplicates behavior in ContactManager 7.0.
CTC-2399	When the user has not assigned a primary phone number for a contact, ContactManager assigns the first phone field that has a value to the primary phone field. ContactManager now uses the order work phone, then home phone, then cell phone to determine which field to assign. If no phone number is specified in any of these fields, ContactManager does not assign a primary phone number.
<b>Data Distribution</b>	
PL-25702	The Data Distribution page enables <b>Download comparison Zip file</b> and <b>Download Combined Zip file</b> options only if two executions are selected. In prior versions, the options were erroneously enabled even if there were not two executions selected.
<b>Database Configuration</b>	
PL-28656	Fixed a problem that prevented the <code>DBAuthenticationPlugin</code> from working.
<b>Database Support</b>	
PL-24417	The performance requirement to set the <code>action="delete"</code> attribute for <code>cc_message</code> in <code>database-config.xml</code> has been removed.
PL-27438	Added support for Oracle Date interval partitioning.
PL-28323	Fixed a bug which prevented scheduling of the Database Statistics process.
<b>Database Support – Oracle</b>	
PL-26203	Nullable <code>monetaryamount</code> columns will use the Oracle default value feature during upgrade. This is a performance optimization.
PL-28085	The Oracle fast add column feature is used when adding <code>monetaryamount</code> amount columns to an entity or a subtype.
PL-28327	Guidewire added an option to switch off the Oracle adaptive optimization feature for Guidewire applications. See “Configuring Oracle Adaptive Optimization for ClaimCenter” in the <i>ClaimCenter Installation Guide</i> .
<b>Database Upgrade</b>	
PL-27016	Guidewire has added support for using Oracle's parallel DDL execution feature during upgrade. The <code>createIndexInParallel</code> attribute of the <code>&lt;upgrade&gt;</code> element has been replaced with the new <code>degree-parallel-ddl</code> attribute. See “Configuring Parallel DML and DDL Statement Execution” in the <i>Upgrade Guide</i> .
PL-27865	Guidewire resolved a rare issue in which the server would be able to start after an incomplete upgrade but subsequent upgrade attempts would fail.
PL-27918	The Database Upgrader now honors the database statistics configuration.
PL-28041	The <code>updatestatistics</code> attribute in <code>database-config.xml</code> now controls both deletion and collection of database statistics during upgrade.

PL-28122	The DeferredUpgradeTasks process is now profiled and appears under Guidewire Profiler. This process is used when deferring creation of archive indexes until after the upgrade. See “Deferring Creation of Archive Indexes” in the <i>Upgrade Guide</i> .
PL-28449	Fixed a bug that prevented handling of statistics on the ID column properly.
<b>Document Management</b>	
PL-27480	Document templates no longer have size constraints.
PL-27577	You can now call the method <code>DocumentsUtil.createNewDocument</code> without needing the current user to have the <code>doc:create</code> permission. Use this method if you implement the <code>IDocumentMetadataSource</code> plugin.
<b>Email</b>	
PL-27921	Due to potential cross-site scripting vulnerabilities, HTML is disabled as the content of an email. If you would like to continue to have HTML email and accept the risk of this vulnerability, you may remove the escaping of the subject and body in the document template.
<b>Entities/Metadata</b>	
PL-19023	The <i>Data Dictionary</i> has been modified to show references for subtypes
PL-24743, PL-27611	<i>Data Dictionary</i> descriptions for core locale fields have been updated.
PL-25622	Fixed an issue in which MonetaryAmount appeared as two separate fields under Actual Amount in the <i>Data Dictionary Data Entity View</i> .
PL-25809	Fixed an issue in which typelist codes did not include documentation for the typecodes (name and description).
PL-27465	Added upgrade trigger to insert <code>xmlns</code> if missing on <code>.eti</code> files.
PL-27501	Added the ability to override and add <code>keyfilters</code> and <code>typefilters</code> by using <code>extensions.xml</code> .
PL-27819	Added a new <code>overlapTable</code> attribute to the <code>edgeForeignKey</code> and <code>localization</code> entities, which specifies that the entity implements the <code>OverlapTable</code> delegate.
<b>Globalization</b>	
PL-26606	Updated <code>GroupUserSearchDV.pcf</code> to support Japanese kanji fields in the standard way represented elsewhere in the application.
PL-27394	Values substituted in display keys that are of type <code>BigDecimal</code> , <code>Date</code> , and <code>IMoney</code> and its implementing classes, such as <code>MonetaryAmount</code> , are now properly formatted according to the regional formats in effect for users. For example, the substitution value “123456” is formatted as “123,456” for U.S. (English) or “123 456” for France (French).
PL-27609	Fixed an issue in which the display value for locale shifts when an admin user is changing his/her own language/locale when viewing the profile of another user.
PL-28068	The configuration parameter for overriding the default maximum width for labels moved from display key <code>ExtJS.Form.LabelWidth</code> to XML element <code>LabelWidth</code> , with attribute <code>width</code> specified in pixels. Use the <code>LabelWidth</code> element in the <code>language.xml</code> file for the language that you want to configure. For example,
	<pre>&lt;GWLanguge     code="de_DE"     name="German (Germany)"     typecode="de_DE"&gt;     &lt;ExtJsSettings&gt;         &lt;LabelWidth size="190" /&gt;     &lt;/ExtJsSettings&gt; &lt;/GWLanguge&gt;</pre>
<b>Gosu</b>	
PL-18217	A Gosu class now preserves annotations inherited from Java classes in the class hierarchy. Note that this change effectively restricts access by a subclass to features tagged with the <code>@InternalAPI</code> annotation.
PL-25700	Fixed an issue that caused errors on startup under very specific conditions related to the compilation of particular Gosu classes.
PL-27099	Gosu supports annotations on parameters in methods, properties, and constructors.

PL-27428	The Gosu language now provides limited support for the Java annotation <code>@SuppressWarnings</code> , which tells the compiler to suppress warnings. Use this annotation on declarations of a type, function, property, constructor, field, or parameter. Note that local variables do not support this annotation.
----------	---

You must pass a `String` value as an argument to indicate which warnings to suppress. Pass the argument "all" to suppress all warnings. Pass the argument "deprecation" to suppress deprecation warnings. For example, to suppress deprecation warnings in a Gosu class, add the annotation `@SuppressWarnings("deprecation")` on the line before the class declaration.

PL-27651	Gosu now recognizes the annotation <code>@java.lang.Deprecated</code> as a form of deprecation, in addition to <code>@gw.lang.Deprecated</code> and the <code>@deprecated</code> Javadoc tag.
----------	---

#### Integration

PL-28196	This release changed how to configure inbound multi-threaded integrations such as the built-in file and JMS integrations. In previous releases, you added configuration parameters in the Plugins registry in Studio. In this release, you set a single parameter <code>integrationService</code> and then do the rest of the configuration in the new file <code>inbound-integration-config.xml</code> . Also, the API details for file and JMS integrations changed. There is a new plugin interface called <code>InboundIntegrationHandlerPlugin</code> . Also, the file integration now supports processing one file at a time, rather than one line at a time. For details, refer to the <i>Integration Guide</i> .
----------	--

#### Integration - CC

CTC-1918	To extend the related contact Integration in ContactManager, you must modify the classes <code>ABContactAPIRelatedContact</code> and <code>RelatedContactInfoContainer</code> . These classes are now marked <code>@Export</code> and can be modified.
CTC-2028	An intermittent problem was occurring with linking a contact from ClaimCenter. Sometimes the contact was left out of sync after a link request was sent to ContactManager. This problem has been fixed. ClaimCenter now always synchronizes after linking.
CTC-2112	Previously, when you initially added a contact to ClaimCenter that was stored in ContactManager, all properties on the contact brought over from ContactManager were saved in the ClaimCenter database. Even properties that were marked non-persistent were saved in ClaimCenter. This problem has been fixed, and now ClaimCenter does not save non-persistent properties. This fix applies to fields in the ClaimCenter class <code>ContactMapper</code> to which you have added <code>.withAffectsSync(false).withPersist(false).withMappingDirection(TO_BEAN)</code> .
CTC-2286	Previously, the first name was required when creating a contact of type <code>Person</code> or <code>PersonVendor</code> or a subtype of one of these contact types. With this change, the first name is no longer required for creating any contact. You can edit <code>ValidateABContactCreationPluginImpl</code> to change this and other contact creation requirements.
CTC-2323	There was a problem with the Duplicate Finder batch process throwing an exception if there was a contact in Pending Create status. This issue has been fixed.

#### IntelliJ IDE – Compiler

PL-28346	Fixed a compilation error when compiling an entity or its extension if the entity had a subtype.
----------	--

#### IntelliJ IDE – Debugger

PL-27875	When debugging Gosu code, you can now browse the structure of an Entity object and inspect the property values stored within it. To enable this, in Guidewire Studio, click File → Settings, and then in the Guidewire Studio panel set <code>Enhance Entities Visualization</code> .
----------	---

#### IntelliJ IDE – Display Key Editor

PL-27971	Improved typing performance in the Display Key editor.
PL-28677	Fixed a bug in Studio where converting a string into a display key caused exceptions. The behavior has changed slightly, so now the locale folder is required to have a <code>display.properties</code> file already in it before it appears in the Create Display Key dialog or the Step Name Localizations tab in the Workflow editor.

#### IntelliJ IDE – Entity Editor

PL-26336	Column validation has been re-enabled in Studio.
PL-26364	Fixed an exception in the Entity editor that occurred when attempting to override a read-only attribute.
PL-26375	Fixed an issue in the New Entity dialog in which the <code>viewEntity</code> was listing suggestions that were not applicable.
PL-26540	Added additional error notes to the Entity editor to highlight the parent elements if a child is invalid.

PL-27220 In the Entity editor in Studio, when editing the <tag> subelement of the <column> element, there is now a drop-down list showing available values.

PL-27454 Fixed an issue in which creating multiple entity extensions with suffixes would produce an exception.

PL-27715 Fixed an error that would occur when the effDatedBranchType attribute was not correct in an entity of type effdated.

PL-28089 In the Entity editor in Studio, you are now required to specify a value for the nullOk attribute.

PL-28529 Fixed an error in Studio when creating an entity extension if there is a Java class under src.

#### IntelliJ IDE – Gosu Editor

PL-19418 Fixed a compilation error during bytecode generation on certain annotations from Java source types.

PL-26640 Fixed an exception that would occur when entering display in a Gosu class.

PL-26866 The Gosu using clause syntax now has an additional feature for adding additional cleanup code. You can optionally add a finally clause that runs after the statement body, even if exceptions occur in the body of the using clause. See the *Gosu Reference Guide* for details.

PL-27135 Studio now shows additional warnings for improper usages of internal gw classes.

PL-27320 Fixed an issue in which pressing Ctrl+O threw an exception in Gosu.

PL-27724 Fixed an issue with some deprecated methods not being shown in strikethrough text in Studio.

PL-27873 Fixed an issue in which pasting code into Studio caused multiline statements to be concatenated and merged with comments.

PL-27893 Fixed an exception that was thrown when creating a new Gosu template.

PL-27943 Fixed an issue with some deprecated methods not being shown in strikethrough text in Studio.

PL-27944 The Gosu language has two new compound assignment operators, which are operators that apply an operation to a variable then re-assign the variable to the result. The new operator &&= performs the logical AND operation to the previous value. The new operator ||= performs the logical OR to the previous value. Both operators work with the primitive type boolean or the object type Boolean on either side of the operator. For example, suppose you have two boolean variables called needsUpdate and flagTest. The statement needsUpdate ||= flagTest has the meaning of needsUpdate = (needsUpdate OR flagTest). Do not confuse these new operators with the other operators &= and |=, which apply bitwise AND and bitwise OR operations.

PL-28019 Fixed a false compile error in the Gosu editor that manifested when a property getter or setter overrode a getter or setter in a superclass implemented in Java.

PL-28027 Gosu does not support numeric expressions in the for statement after the in keyword. The code:

```
for (x in 10) {...}
```

is illegal and must be upgraded with an interval such as:

```
for (x in 0..|10) {...}
```

using the provided inspection in Studio.

#### IntelliJ IDE – Line of Business Editor

PL-26590 The LOB tab has been removed from .tti files in Studio to prevent missing loss types.

PL-26595 Added the ability to more easily select multiple typekeys in Studio by using the Ctrl or Shift keys.

PL-26620 Retired typecodes are now shown in strikethrough text.

PL-26826 Fixed an issue in which Studio would throw an exception when a categorylist was added to a LossType typecode.

PL-26971 Fixed an issue in Studio in which removing a typecode from its parent also incorrectly removed it from all its other parents.

#### IntelliJ IDE – Other

PL-27198 Fixed an issue in which Run commands in the QuickJump box did not work when the server was started from Studio.

PL-27862 Fixed an issue where Studio would not suggest types defined on XSD files when trying to create an enhancement.

#### IntelliJ IDE – PCF Editor

PL-26516 Improved the PCF editor to highlight the correct panel when selecting widgets in nested files.

PL-27147	The PCFMapping tool has been updated to include fields such as PanelIterator.
<b>IntelliJ IDE – IntelliJ IDE – Plugins, OSGi, Plugins</b>	
PL-27497	You can now implement plugin interfaces in Java using the OSGi standard. OSGi is a Java module system and service platform that helps isolate code modules and any necessary Java libraries. Guidewire recommends OSGi for all new Java plugin development. To simplify OSGi configuration, ContactManager includes IntelliJ IDEA with OSGi Editor, an application separate from Guidewire Studio. For more information, refer to the <i>Integration Guide</i> .
<b>IntelliJ IDE – Refresh</b>	
PL-24108	Fixed an issue in which the server threw an exception and did not handle newly created enhancements.
PL-28174	Fixed an issue where methods added in entity classes were invalid until restarting Studio.
PL-28187	Fixed an error that would occur after renaming an element in an XSD and then navigating to a Gosu type that contained a usage of that element.
<b>IntelliJ IDE – Typelist Editor</b>	
PL-24391	Fixed the typelist editor in Studio to filter out options under the drop-down as you enter text.
PL-26445	Fixed an issue in which creating the first extension of a typelist caused an exception.
PL-26613	Fixed an issue in Studio in which disabled typecodes looked enabled when selected.
PL-26535	Fixed an exception in the text editor of the typelist view.
PL-27174	Fixed an issue in which the default setting for filtering metadata did not apply to the Typelist editor.
PL-27570	Fixed an issue in which clicking on the name attribute in a typelist extension would not allow you to override it.
PL-27584	Fixed the Add To Category dialog so retired typecodes appear in strikethrough text as options for filtering.
PL-28177	In Studio, the Entity editor and Typelist editor are now case-insensitive when resolving references to other metadata.
PL-28515	Fixed an issue that generated multiple errors in the Studio Typelist editor in the localization panel.
<b>IntelliJ IDE – Web Services Editor</b>	
PL-26576	Added a check to the timeout value of web services to insure that it is lower than Studio's maximum integer.
<b>Localization</b>	
CTC-1893	When you are editing a contact who is a Person subtype in ClaimCenter or an ABPerson subtype in ContactManager, there is a Driver's License section on the edit screen. The State field on this screen now uses the Jurisdiction.ttx file to populate the drop-down list for the field. If you want to change the contents of the drop-down list, edit the Jurisdiction.ttx file in Studio and add or remove values from the outgoing category driving_lic. Be sure to make this change in both ClaimCenter and ContactManager.
CTC-1936	Kanji fields require a case-sensitive search. Previously, in the base configuration, search for Kanji contact fields was using a case-insensitive search. Now in the base configuration, searching for these fields is case-sensitive.
<b>Notes:</b>	
	<ul style="list-style-type: none"> <li>With Kanji fields, use startsWithCaseSensitive in search-config.xml to do a case-sensitive search. If you use startsWith, the search is case-insensitive.</li> <li>You can use eq in search-config.xml to do a case-sensitive search if the referenced field has supportsLinguisticSearch set to true in the schema. Otherwise, eq performs a case-insensitive search.</li> </ul>
CTC-2104	The minimum set of fields required for a contact search can vary by country and locale. The error message displayed in response to an invalid contact search now varies based on the country specified in the search and the locale setting of the user making the search.
CTC-2238	The DefaultApplicationLanguage configuration parameter is now visible in the config.xml file.
CTC-2332	Previously, the TaxID field was required when creating a VendorPerson or VendorCompany contact and any subtypes of those entity types, even for locales that did not require this field to create a vendor. This problem has been fixed. The PCF files no longer require this field. Additionally, you can now set whether TaxID is required in the plugin implementation class ValidateABContactCreationPluginImpl, as described in the Contact Management Guide.

CTC-2335	<p>In the search results for a contact search, the <b>Country</b> search criterion now determines the labels for the <b>State</b> and <b>PostalCode</b> fields of the <b>Address</b> entity. For example:</p> <ul style="list-style-type: none"> <li>• <b>US</b> – State and ZIP Code</li> <li>• <b>Japan</b> – Prefecture and Postcode</li> <li>• <b>Canada</b> – Province and Postal Code</li> <li>• <b>Most others</b> – State and Postcode</li> </ul> <p>In ContactManager, the values of these field labels are defined in <b>ContactSearchResultsLV.pcf</b>. In ClaimCenter, the values of these field labels are defined in <b>AddressBookSearchLV.pcf</b>. In both applications, the two field labels are populated by the following code:</p> <pre>• gw.api.address.AddressCountrySettings.getSettings(     searchCriteria.Address.Country).StateLabel • gw.api.address.AddressCountrySettings.getSettings(     searchCriteria.Address.Country).PostalCodeLabel</pre>
<b>Other – Persistence</b>	
PL-25820	Fixed an issue with whitespace not being trimmed by trimming unicode full-width whitespace.
<b>Profiling</b>	
PL-28172	Fixed an issue in which the Purge Profiler Data batch job would throw exceptions when purging web services profiling data.
<b>Queries</b>	
PL-18578	<p>This release adds a new class, <code>gw.api.database.QuerySelectColumns</code>, with static methods that help you specify the columns that you want selected in a row query. Instead of passing a Gosu block to the <code>select</code> method, you pass a list of <code>IQuerySelectColumn</code> objects, which you construct by using static methods on the <code>QuerySelectColumns</code> class. Each <code>IQuerySelectColumn</code> object represents a column in the result.</p> <p>For example, the following Gosu sample code creates a row query, with <b>Address</b> as the primary entity. The result includes only the <b>City</b> column for <b>Address</b> instances that match the query criteria.</p> <pre>uses gw.api.database.Query uses gw.api.database.QuerySelectColumns uses gw.api.path.Paths  var addressQuery = Query.make(Address) ... // Join and condition statements go here. ... var addressResult =     addressQuery.select({QuerySelectColumns.path(Paths.make(Address#City))})</pre> <p>The <code>QuerySelectColumns</code> class includes static methods that represent database functions to help you construct aggregate queries. For example, the following Gosu code returns a count of all <b>Address</b> instances.</p> <pre>uses gw.api.database.DBFunction uses gw.api.database.Query uses gw.api.database.QuerySelectColumns uses gw.api.path.Paths  var addressQuery = Query.make(Address). var addressQuery.withDistinct(true) // Always run aggregate queries with Distinct set to true. addressResult = addressQuery.select({QuerySelectColumns.dbFunction( DBFunction.Count(Paths.make(Address#City))) })</pre>
PL-27474	Fixed an issue in which the <code>Count</code> property on a <code>gw.api.database.Query</code> result was producing an incorrect SQL statement and count result when the <code>withDistinct(true)</code> setting was combined with a <code>Join</code> .
<b>User Interface</b>	
CTC-2047	In the ContactManager user interface, the contact history list view now displays international phone numbers correctly.
CTC-2235	In the ContactManager user interface, PersonVendor detail views are now showing the cell phone field.

Web - ListViews	
PL-10908	Added the groupedOnEnter attribute to many cell-based PCF elements. If true, the ListView is grouped by this cell upon entering the page. Only one grouped cell is allowed at any given time, and it is applicable only when the column is sortable.
PL-27732	A new height attribute has been added to the ListViewPanel PCF element. This attribute sets the vertical size in pixels of the list view. If the list data is taller than the specified height, a vertical scroll bar appears within the list view. The height is calculated from the list view header toolbar. Header rows are fixed, and the footer scrolls with the data. Note that this attribute is currently experimental and might not function properly. You should fully test any use of this attribute.
PL-28466	Fixed an issue with columns reordering when switching between filtering views in a list view.
Web – UI/Runtime	
PL-28749	Fixed a critical security vulnerability to persistent cross-site scripting attacks.
PL-27884	The following new methods were added to the Javascript gw.api.Util class: getValue, setValue, getValues, setValues. These methods enable you to get or set values of input elements. For example, you can use these methods in TemplatePanel and pass Gosu variables as arguments into these methods.
PL-28336	Fixed an issue that would occur when there was an action defined on a range cell.
Work Queues	
PL-27301	Improved server performance when selecting available work queue items.
PL-28696	Fixed an issue with workItem orphan detection during daylight saving time adjustment.
XMLElement (and XSD types)	
PL-27131	Gosu XML/XSD types now support use of circular xs:include references.

## Known Issues and Limitations for ContactManager 8.0.1

This topic describes known issues with this release of Guidewire ContactManager.

**Note:** For maintenance releases, Guidewire often defers fixing configuration issues that require merging files during the upgrade. Workarounds to many of these issues are listed in the following topics. The goal of this policy is to make upgrades as straightforward as possible.

This topic includes:

- “ContactManager 8.0.1 Known Issues” on page 377
- “Platform/Studio Known Issues for ContactManager 8.0.1” on page 378

### ContactManager 8.0.1 Known Issues

#### Customizing contact relationship handling to use messaging results in two relationship records being created (CTC-396)

**Issue** – Changing a contact relationship updates two contact records. However, ContactManager stores the representation of the relationship in one ContactContact entity. ContactManager does not verify that the relationship does not already exist, and therefore can create multiple, redundant ContactContact entities for the same relationship if it receives messages changing the relationship between two contacts.

**Workaround** – In the base configuration of ContactManager, this behavior is not a problem. ClaimCenter makes a direct web service call to ContactManager to update contact relationships and does not use event messaging for related contact updates.

If you customize ClaimCenter and ContactManager to use event messaging to update related contacts between ClaimCenter and ContactManager, your customization must account for the extra message. Updating one relationship changes two contacts, and therefore ClaimCenter generates two messages and sends them to ContactManager. Your customization must ensure that ContactManager creates only one ContactContact entity in this case.

**RelatedABContactSearchCriteria does not work for related contact searches (CTC-960)**

**Issue** – Using the RelatedABContactSearchCriteria entity and the CriteriaDef elements that are defined for it in search-config.xml is not supported in this version of ContactManager.

**Workaround** – Do not use them. Guidewire is aware of this issue.

**Proximity Search Integration between ClaimCenter 7 and ContactManager 8 Is Broken (CTC-2443)**

**Issue** – Contact proximity search from ClaimCenter 7.0 to ContactManager 8.0 does not work. There are two fields missing in the ContactManager 8.0 WSDL that are required by ClaimCenter 7.0 contact proximity search.

**Workaround** – In ContactManager 8.0 Studio, add the following two fields to the ab700 version of the ProximitySearchParametersInfo class:

```
public var RadiusSearchMaxResults: Integer  
public var GID : String
```

This class is in the package gw.webservice.ab.ab700.abcontactapi.

After adding these two fields to the ContactManager class, regenerate the ContactManager 8.0 WSDL and refresh the WSDL in ClaimCenter 7.0, as follows:

1. Restart the ContactManager 8.0 server to regenerate the WSDL.
2. Open ClaimCenter 7.0 Studio.
3. In the Resources pane on the left, navigate to configuration → Classes → wsi → remote → gw → webservice → ab → ab700.
4. In the editor on the right, in the Resources pane, select the following resource:  
 `${ab}/ws/gw/webservice/ab/ab700/abcontactapi/ABContactAPI?wsdl`
5. At the bottom of the pane, click Fetch Updates. Fetching updates for ABContactAPI also pulls in changes to the ContactManager Search web services.
6. Restart the ClaimCenter server.

## Platform/Studio Known Issues for ContactManager 8.0.1

**Problem with regen-java-api command and JAR files (PL-16351)**

**Issue** – If you run the gwab regen-java-api command, ContactManager creates a *ContactManager/java-api/lib* directory and puts JAR files for Java development in this directory. However, ContactManager does not always perform this task correctly, resulting in some of the generated JAR files not being copied to this directory.

**Workaround** – For missing library files, copy or import into your Java development environment the missing library files from the following directory:

*ContactManager/admin/lib*

Copy them into the following directory:

*ContactManager/java-api/lib*

**Renaming method or property throws ParseResultsException (PL-16633)**

**Issue** – If you rename a property or a method or you change a method signature, and a workflow references that property or method in a Gosu field, ContactManager throws ParseResultsException. This is the intended behavior.

**Workaround** – Restart the workflow engine. To do so:

1. Log in to ContactManager using an administrative account.
2. Access Internal Tools → Reload.
3. Click Reload Workflow Engine.

**Javadoc command does not generate index file in expected location (PL-27679)**

**Issue** – When you run gwab regen-java-api, an index.html file is not created in *ContactManager/java-api/doc*.

**Workaround** – The command now generates Javadoc JAR files in *ContactManager/java-api/doc*. To view the Javadoc, add the JAR files to the Studio project.

**Gosu does not automatically downcast if the left side of the typeis or typeof expression uses deprecated members (PL-27724)**

**Issue** – To improve readability of your Gosu code, Gosu automatically downcasts after a typeis expression if the type is a subtype of the original type. This is particularly valuable for if statements and similar Gosu structures. For example, if a variable has type Object, you can use code such as:

```
if( x typeis String ) {  
    length // NOTE: length is a property on String, but *not* on Object.  
}
```

In this release, Gosu does not automatically downcast if the left side of the typeis or typeof expression uses deprecated members. This may result in new compilation errors.

**Workaround** – To fix these compiler errors, explicitly downcast with the as keyword before you access properties or methods on the subtype but not on the original type. For example, suppose a property called Dep is deprecated:

```
if (x.Dep typeis ExampleType) {  
    return (x.Dep as ExampleType).PropertyOnExampleSubtype  
}
```

**Long text in table cells can add white space to the right of the page (PL-28288)**

**Issue** – In an editable list view, extremely long text entered in a single cell can cause additional white space on the right side of the page. Long text is text that occupies approximately the entire width of the screen. This issue occurs primarily in Chrome.

**Workaround** – If you expect users to enter large amounts of text into the cells of a column, configure the column to support text wrapping.

**AddressZoneValue in zone-config.xml files uses display name instead of code (PL-28511)**

**Issue** – Some of the <AddressZoneValue> expressions in the US, AU, CA, and DE zone-config.xml files use the DisplayName, rather than the Code, of the associated State.Abbreviation. Configuring ContactManager to use translated names for the StateAbbreviation typelist can result in failure to find some zones for that country.

**Workaround** – Replace references to State.Abbreviation.DisplayName with State.Abbreviation.Code in <AddressZoneValue> elements.

In each zone-config.xml file, replace the following DisplayName property:

```
<AddressZoneValue>Address.State.Abbreviation.DisplayName</AddressZoneValue>
```

Instead, use the following Code property:

```
<AddressZoneValue>Address.State.Abbreviation.Code</AddressZoneValue>
```

**Problem with running regen-java-api command with deprecated=true flag (PL-28992)**

**Issue** – If you run the gwab regen-java-api command with the flag -Ddeprecated=true, the command can fail.

**Workaround** – Specify dev-deploy as the value of the depends attribute, as follows:

1. Open the build.xml file in *ContactManager/modules/ant* in an editor.

2. Find the entry that starts as follows:

```
<target name="regen-java-api" depends="init"
```

3. Change the value of depends to dev-deploy, as follows:

```
<target name="regen-java-api" depends="dev-deploy"
```

4. Save the file and run the `regen-java-api` command.

#### **Administrative command-line tools cannot refresh WSDL (PL-29021)**

**Issue** – Some administrative command-line tools rely on web service implementation classes such as `MaintenanceToolsAPI.gs`. Source files for these classes use the `@Export` annotation, which allows you to edit the file. In this release, the administrative command-line tools cannot refresh the WSDL for these classes. Any change to the web service implementation class that changes the WSDL can prevent the administrative tools from working. Therefore, the only changes you can make to these classes are changes that do not affect the WSDL. For example, you can add `@WsPermission` annotations to change the permissions without changing the WSDL.

The web services used in the command-line tools for ContactManager are `ImportToolsAPI`, `MaintenanceToolsAPI`, `MessagingToolsAPI`, `SystemToolsAPI`, `TableImportAPI`, `WorkflowAPI`, and `ZoneImportAPI`.

**Workaround** – Guidewire is aware of this issue.

#### **Security Vulnerability - Reflected XSS (PL-29052)**

**Issue** – There is a non-persistent cross-site scripting (reflected XSS) vulnerability. Unlike other XSS types, this vulnerability does not permit privilege escalation and does not propagate easily to other users.

**Workaround** – Guidewire is aware of the issue. Strong email filtering with phishing/malware detection is an effective defense against this type of attack. Contact Customer Support for more information.

#### **Chrome browser cannot display product documentation in HTML format (DOC-7251)**

**Issue** – If you use the Google Chrome browser, you can view the HTML Guidewire product documentation only if it is served by an HTTP server using the `http://` protocol. The Chrome browser cannot load HTML product documentation from your local disk by using the `file://` protocol.

**Workaround** – Use a different browser, such as Microsoft Internet Explorer. If you use the HTML documentation only to access the PDF files, you can open them directly in the `pdf` subdirectory of the `doc` directory.

#### **New mechanism for reloading Gosu classes (DOC-8218)**

**Issue** – In past releases, you could modify your PCF files and Gosu classes in Studio, and then reload the changes into the running server by pressing `Alt+Shift+L` in the application user interface. This shortcut no longer loads Gosu classes.

**Workaround** – To have the server reload your Gosu classes, in Studio, click **Build** → **Make Project**. When Studio is finished compiling your project, the changes will be loaded. You can also restart your server to load the Gosu classes.