Guidewire BillingCenter®

BillingCenter Rules Guide

RELEASE 8.0.4

Guidewire®
Deliver insurance your way™

Product Name: Guidewire BillingCenter
Product Release: 8.0.4
Document Name: *BillingCenter Rules Guide*
Document Revision: 02-July-2015

Contents

About BillingCenter Documentation

The following table lists the documents in BillingCenter documentation.

Document	Purpose
InsuranceSuite Guide	If you are new to Guidewire InsuranceSuite applications, read the *InsuranceSuite Guide* for information on the architecture of Guidewire InsuranceSuite and application integrations. The intended readers are everyone who works with Guidewire applications.
Application Guide	If you are new to BillingCenter or want to understand a feature, read the *Application Guide*. This guide describes features from a business perspective and provides links to other books as needed. The intended readers are everyone who works with BillingCenter.
Upgrade Guide	Describes how to upgrade BillingCenter from a previous major version. The intended readers are system administrators and implementation engineers who must merge base application changes into existing BillingCenter application extensions and integrations.
New and Changed Guide	Describes new features and changes from prior BillingCenter 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.
Installation Guide	Describes how to install BillingCenter. The intended readers are everyone who installs the application for development or for production.
System Administration Guide	Describes how to manage a BillingCenter system. The intended readers are system administrators responsible for managing security, backups, logging, importing user data, or application monitoring.
Configuration Guide	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.
Globalization Guide	Describes how to configure BillingCenter for a global environment. Covers globalization topics such as global regions, languages, date and number formats, names, currencies, addresses, and phone numbers. The intended readers are configuration engineers who localize BillingCenter.
Rules Guide	Describes business rule methodology and the rule sets in BillingCenter Studio. The intended readers are business analysts who define business processes, as well as programmers who write business rules in Gosu.
Contact Management Guide	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 BillingCenter implementation engineers and ContactManager administrators.
Best Practices Guide	A reference of recommended design patterns for data model extensions, user interface, business rules, and Gosu programming. The intended readers are configuration engineers.
Integration Guide	Describes the integration architecture, concepts, and procedures for integrating BillingCenter 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.
Gosu Reference Guide	Describes the Gosu programming language. The intended readers are anyone who uses the Gosu language, including for rules and PCF configuration.
Glossary	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
italic	Emphasis, special terminology, or a book title.	A *destination* sends messages to an external system.
bold	Strong emphasis within standard text or table text.	You **must** define this property.
narrow bold	The name of a user interface element, such as a button name, a menu item name, or a tab name.	Next, click **Submit**.
`monospaced`	Literal text that you can type into code, computer output, class names, URLs, code examples, parameter names, string literals, and other objects that might appear in programming code. In code blocks, bold formatting highlights relevant sections to notice or to configure.	Get the field from the `Address` object.
`monospaced italic`	Parameter names or other variable placeholder text within URLs or other code snippets.	Use `getName(`*`first`*`, `*`last`*`)`. `http://`*`SERVERNAME`*`/a.html`.

Support

For assistance, visit the Guidewire Resource Portal – `http://guidewire.custhelp.com` part I

Gosu Business Rules

chapter 1

Rules: A Background

This topic provides an overview of rules and discusses some basic terminology associated with rules and rule sets. It also gives a high-level view of the BillingCenter rule set categories.

This topic includes:

- "Introduction to Business Rules" on page 11
- "Business Rule Terminology" on page 13
- "Overview of BillingCenter Rule Set Categories" on page 13

Introduction to Business Rules

In general, Guidewire strongly recommends that you develop and document the business logic of rules before attempting to turn that logic into rules within BillingCenter. In a large implementation, there can be a large number of rules, so it is extremely beneficial to organize the basic structure of the rules in advance. Use this guide to understand how BillingCenter rules work. It can also help you make decisions about changing your rules as your use of BillingCenter evolves over time.

Rule Hierarchy

A rule set can be thought of as a logical grouping of rules that are specific to a business function within BillingCenter. You typically organize these rules sets into a hierarchy that fits your business model. Guidewire strongly recommends that you implement a rule-naming scheme as you create rules and organize these rules into a hierarchy. This can be similar to that described in "Generating Rule Debugging Information" on page 27.

Prior to implementing rules, it is important to first understand the rule hierarchy that groups the rules. The rule hierarchy is the context in which BillingCenter groups all rules. You can implement a rule hierarchy in several formats, depending on the needs of your organization. However, it is important to outline this hierarchy up-front before creating the individualized rules to reduce potential duplicates or unnecessary rules. You can create multiple hierarchies within BillingCenter. However, make each specific to the rule set to which it belongs.

Rule Execution

The hierarchy of rules in BillingCenter mirrors a decision tree that you might diagram on paper. BillingCenter considers the rules in a very specific order, starting with the first direct child of the root. (The first direct child is the first rule immediately below the line of the rule set.) BillingCenter moves through the hierarchy according to the following algorithm. Recursively navigating the rule tree, it processes the parent and then its children, before continuing to the next peer of the parent.

- Start with the first rule
- Evaluate the rule's conditions. If true…
  - Perform the rule's actions
  - Start evaluating the rule's children, starting with the first one in the list.

  You are done with the rule if a) its conditions are false or b) its conditions are true and you processed its actions and all its child rules.
- Move to the next peer rule in the list. If there are no more peers, then the rules at the current level in the tree are complete. After running all the rules at the top level, rule execution is complete for that rule set.

See also

- "Generating a Profiler Rule Execution Report" on page 42

Rule Management

Guidewire strongly recommends that you tightly control access to changing rules within your organization. Editing rules can be complicated. Because BillingCenter uses rules to make automated decisions regarding many important business objects, you need to be careful to verify rule changes before moving them into production use.

Two kinds of people need to be involved in managing your business rules:

- **Business Analysts** – First, you need one or more business analysts who own decision-making for making the necessary rules. Business analysts must understand the normal business process flow and must understand the needs of your business organization and how to support these needs through rules in BillingCenter.
- **Technical Rule Writers** – Second, you need one or more rule writers. Generally, these are more technical people. Possibly, this can be someone on the business side with a good technical aptitude. Or possibly, this can be someone within IT with a good understanding of the business entities that are important to your business. Rule writers are responsible for encoding rules and editing the existing set of rules to implement the logic described by business analysts. The rule writers work with the business analysts to create feasible business rules. These are rules that you can actually implement with the information available to BillingCenter.

Sample Rules

Guidewire provides a set of sample rules as examples and for use in testing. These are sample rules only, and Guidewire provides these rules merely as a starting point for designing your own rules and rule sets. You access sample rules (and other Studio resources) through the Studio interface.

See also

- "Overview of BillingCenter Rule Set Categories" on page 13
- "Generating a Rule Repository Report" on page 41
- "Generating a Profiler Rule Execution Report" on page 42

Business Rule Terminology

This guide and other BillingCenter documents use the following terminology throughout the discussion of business rules:

Term	Definition
assignment engine	The assignment engine handles assignment of trouble tickets and activities to users by repeatedly asking BillingCenter to evaluate assignment rules for the item. Guidewire structures the assignment rule sets so that the assignment engine first asks to which group the assignment goes. It then asks to which person within the group (with each of these decisions governed by a different rule set).
entity	An entity is a type of business data configured in the data model configuration files, for example `Account`. You can use the Data Dictionary to review the entity types and their properties. For an entity type, this documentation refers to an instance as an *entity instance* or, for brevity, *object*.
Guidewire Studio	Guidewire Studio is the Guidewire administration tool for managing BillingCenter resources, such as PCF pages, business rules, and Gosu classes.
library	A library is a collection of functions (methods) that you can call from within your Gosu programs. Guidewire provides a number of standard library functions (in the `gw.api.*` packages).
object	An object refers to any of the following: • an instance of an *entity type*, such as `Account` or `Activity`. For an entity type, an object is also called an *entity instance*. • an instance of an *Gosu class* • an instance of an *Java class, such as* `java.util.ArrayList`.

rule	A rule is a single decision in the following form:
	`If {some conditions}`
	`Then {take some action}`

rule set	A rule set combines many individual rules into a useful set to consider as a group.
workflow	A workflow is a Guidewire mechanism to run custom business processes asynchronously, optionally with multiple states that transition over time. Studio stores individual workflows as XML files. However, you manage workflow directly through Guidewire Studio user interface.

Overview of BillingCenter Rule Set Categories

A rule set category is the highest level of logical grouping of rules that are specific to a business function in BillingCenter. A rule set category contains rule sets. In the base configuration, BillingCenter provides the following rule sets.

Rule set category	Description
Assignment	Determine the responsible party for an activity or trouble ticket. See “Assignment” on page 34.
Event Message	Handle communication with integrated external applications. See “Messaging and Events” on page 303 in the *Integration Guide*for information about events and event messaging. See “Event Message” on page 34.
Exception	Specify an action to take if an Activity is overdue and is in the escalated state. See “Exception” on page 37.
Preupdate	Trigger automatic actions any time that an account or other high-level entity changes.See “Preupdate” on page 38.
Validation	Validate a small subset of the BillingCenter entities.See “Validation” on page 40.

chapter 2

Rules Overview

This topic describes the basic structure of a Gosu rule and how to create a rule in Studio. It also describes how to exit a rule and how to call a Gosu rule from Gosu code.

This topic includes:

Rule Structure

The basic structure of every rule has the following syntax:

`IF` {some conditions}

`THEN` {do some actions}

The following table summarizes each of the parts of the rule.

Pane	Requirements

Rule Conditions	A Boolean expression (that is, a series of questions connected by AND and OR logic that evaluates to TRUE or FALSE). For example:
	{This is a new payment plan}
	It is also possible to insert a statement list, instead of a simple expression. However, the statement list must contain a `return` statement. For example:
	`uses java.util.HashSet`
	`uses gw.lang.reflect.IType`
	
	`var o = new HashSet<IType>() {A, B, C, ...}`
	`return o.contains(typeof(...))`

Rule Actions	A list of actions to take. For example: {Create a history event}

The best way to add rules to the rule set structure is to right-click in the Studio rule pane. After you do this, Studio opens a window opens containing a tree of options from which to select. As you use the right-click menu, BillingCenter gives you access only to conditions and actions that are appropriate for the type of your current rule.

Rule Syntax

Guidewire Studio uses a programming language called Gosu, which is a high-level language tailored for expressing business rule logic. The syntax supports rule creation with business terms, while also using common programming methods for ease of use. Gosu syntax can be thought of in terms of both statements and expressions. Before you begin to write rules, Guidewire strongly recommends that you make yourself completely acquainted with the Gosu programming language.

Statements are merely phrases that perform tasks within Studio. Examples of statements include the following:

- Assignment statements
- `If` statements
- Iteration statements

IMPORTANT Use only Gosu expressions and statements to create BillingCenter business rules. For example, do not attempt to define a Gosu function in a business rule. Instead, define a Gosu function in a Gosu class or enhancement, then call that function from a Gosu rule. Guidewire expressly does not support the definition of functions—and especially nested functions—in Gosu rules.

Rule Members

As described previously, a rule consists of a set of conditions and actions to perform if all the conditions evaluate to `true`. It typically references the important business entities and objects (accounts, for example).

- **Rule Conditions** – A collection of expressions that provide true/false analysis of an entity. If the condition evaluates to `true`, then Studio runs the activities in the **Rule Actions** pane.
- **Rule Actions** – A collection of action expressions that perform business activities such as making an assignment or marking a field as invalid. These actions occur only if the corresponding condition is `true`.
- **APIs** – A collection of Gosu methods accessed through `gw.api.*`. They include many standard math, date, string, and financial methods.
- **Entities** – The collection of business object entities supported by BillingCenter. Studio objects use the familiar “dot” notation to reference fields and objects.

Defining the Rule Conditions

The simplest kind of condition looks at a single field on the object or business entity. For example:

```
Activity.ActivityPattern.Code == "AuthorityLimitActivity"
```

This example demonstrates some important basics:

1. To refer to an object (for example, an `Activity` object, or, in this case, an `ActivityPattern` object) and its attributes, you begin the reference with a *root object*. While running rules on an activity, you refer to the activity in question as *Activity*. (Other root objects, depending on your Guidewire application, are *Account*, *PolicyPeriod*, *Producer*, *Invoice*, *TroubleTicket*, and *Charge*.)
2. BillingCenter uses *dot* notation to access fields (for example, `Activity.ActivityPattern`) or objects (`Activity.ActivityPattern.Category`, for example), starting from the root object.

Combining Rule Conditions

For more complicated conditions, you can combine simple conditions like the previous one with standard Boolean logic operators (`and`, `or`, `not`). For example:

```
Activity.ActivityPattern.Code == "AuthorityLimitActivity" and not Activity.Approved
```

(See the "Gosu Operators and Expressions" on page 67 in the *Gosu Reference Guide* for details on operator precedence and other syntactical information.)

IMPORTANT The rule condition statement must evaluate to either Boolean `true` or `false`. If you create a condition statement that evaluates to `null`, BillingCenter interprets this `false`. This can happen inadvertently, especially if you create a condition statement with multiple conditions to evaluate. If your condition evaluates to `null` (`false`), then BillingCenter never executes the associated rule actions.

Defining a Statement List as Rule Conditions

It is also possible to use a statement list, instead of a simple expression, in the **Rule Actions** pane. However, the statement list must contain a `return` statement. For example:

```
uses java.util.HashSet
uses gw.lang.reflect.IType

var o = new HashSet<IType>() {A, B, C, ...}
return o.contains(typeof(...))
```

Defining the Rule Actions

Within the **Rule Actions** pane, you create the outcome for the criteria identified in the **Conditions** section. Actions can be single statements or they can be strung together to fulfill multiple criteria. You can add any number of actions to a rule. Studio executes all of these actions in order if the condition evaluates to `true`.

For example, suppose that you want to assign a trouble ticket using round-robin assignment. The following syntax creates this action:

```
if (Activity.CurrentAssignment.assignUserByRoundRobin()) {
  actions.exit()
}
```

Exiting a Rule

At some point in the decision tree, BillingCenter makes the decision that you want. At this point, it is important that BillingCenter not continue. Indeed, if rule checking did continue, if BillingCenter processed the *default* rule, it might overwrite the decision that came earlier. Therefore, you need to be able to instruct BillingCenter at what point to stop considering any further rules.

Guidewire Studio provides several options for this flow control, with the simplest version simply meaning:

Stop everything! I am done with this rule set.

The following list describe the methods that you can add as an action for a rule to tell BillingCenter what to do next.

Flow control action	Description
`actions.exit`	This is the simplest version. BillingCenter stops processing the rule set as soon as it encounters this action.
`actions.exitAfter`	This action causes BillingCenter to stop processing the rule set after processing any child rules.
`actions.exitToNext`	This action causes BillingCenter to stop processing the current rule and immediately go to the next peer rule. The next peer rule is one that is at the same level in the hierarchy. This is likely to be used only rarely within very complicated action sections.
`actions.exitToNextParent`	This action causes BillingCenter to stop processing the current rule and immediately go to the next rule at the level of the parent rule. It thus skips any child rules of the current rule.
`actions.exitToNextRoot`	This action causes BillingCenter to stop processing the current rule and immediately go to the next rule at the top level. It thus skips any other rules in the entire top-level branch containing this rule.

Usually, you list an exit action as the last action in a rule so that it occurs only after all the other actions have been taken.

Gosu Annotations and BillingCenter Business Rules

Guidewire BillingCenter uses the annotation syntax to add metadata to a Gosu class, constructor, method, or property. For example, you can add an annotation to indicate what a method returns or to indicate what kinds of exceptions the method might throw.

Guidewire marks certain Gosu code in the base application with a `@scriptable-ui` annotation. This restricts its usage (or *visibility*) to non-rules code. (The converse of this is the `@scriptable-all` annotation that indicates that the class, method, or property is visible to Gosu code everywhere.)

Within the business rules, the Gosu compiler ignores a class, a property, or a method marked as `@scriptable-ui`. For example, suppose that you attempt to access a property in Studio that has a `@scriptable-ui` annotation on it. Studio tells you that it cannot find a property descriptor for that property. The compiler does recognize the code, however, in other places such as in classes or enhancements. *Thus, you find that some Gosu code appears valid in some situations and not others. This is something of which you need to be aware as it can appear that you are writing incorrect Gosu code. It is the location of the code that is an issue, not the code itself.*

To see a list of the annotations that are valid for a particular piece of Gosu code (a class, constructor, method, or property), do the following:

- Place your cursor at the beginning of the line directly above the affected code.
- Type an `@` sign.

Studio displays a list of validation annotations.

See also

- For information on working with annotation, including creating your own annotations, see “Annotating a Class, Method, Type, Class Variable, or Argument” on page 227 in the *Gosu Reference Guide*.

Invoking a Gosu Rule from Gosu Code

It is possible to invoke the Gosu business rules in a rule set from Gosu code. To do so, use the following syntax:

```
rules.[rule set category].[rule set name].invoke([root entity])
```

It is important to understand that you use the `invoke` method on a rule set to trigger all of the rules in that rule set. You cannot invoke individual rules within a rule set using the `invoke` method. BillingCenter executes all of the rules in the invoked rule set in sequential order. If a given rule's condition expression evaluates to `true`, then BillingCenter executes the Gosu code in **Rule Actions** pane for that rule.

See also

- "Using the Rules Editor" on page 21
- "Working with Rules" on page 21

chapter 3

Using the Rules Editor

This topic describes the Studio Rules editor and how you use it to work with Gosu business rules.

This topic includes:

Working with Rules

BillingCenter organizes and displays rules as a hierarchy in the center pane of Guidewire Studio, with the rule set appearing at the root, the top level, of the hierarchy tree. Studio displays the **Rule** menu only if you first select a rule set category in the **Project** window. Otherwise, it is unavailable.

There are a number of operations involving rules that you can perform in the Studio Rules (Gosu) editor. See the following for details:

- To view or edit a rule
- To change rule order
- To create a new rule set category
- To create a new rule set
- To create a new rule
- To access a rule set from Gosu code
- To change the root entity of a rule

To view or edit a rule

1. In the Studio **Project** window, navigate to **configuration** → **config** → **Rule Sets**.
2. Expand the **Rule Sets** folder, and then expand the rule set category.

3. Select the rule set you want to view or edit. All editing and saving in the tool occurs at the level of a rule set.

To change rule order

If you want to change the order of your rules, you can drag and drop rules within the hierarchy.

1. Click the rule that you want to move, and then hold down the mouse button and move the pointer to the new location for the rule. Studio then displays a line at the insertion point of the rule. Release the mouse button to paste the rule at that location.
2. To make a rule a child of another rule, select the rule you want to be the child, and then choose **Edit** → **Cut**. Click on the rule that you want to be the parent, and then choose **Edit** → **Paste**.
3. To move a rule up a level, drag the rule next to another rule at the desired level in the hierarchy (the reference rule). Notice how far left the insertion line extends.
   - If the line ends before the beginning of the reference rule's name, Studio inserts the rule as a child of the reference rule.
   - If the line extends all the way to the left, Studio inserts the rule as a peer of the reference rule.

   By moving the cursor slightly up or down, you can indicate whether you want to insert the rule as a child or a peer.

If you move rules using drag and drop, BillingCenter moves the chosen rule and all its children as a group. This behavior makes it easy to reposition entire branches of the hierarchy.

BillingCenter also supports standard cut, copy, and paste commands, which can be used to move rules within the hierarchy. If you paste a cut or copied rule, Studio inserts the rule as if you added a new rule. It becomes the last child of the currently selected rule.

To create a new rule set category

Guidewire divides the sample rule sets into categories, or large groupings of rules that center around a certain business process, for example, assignment or validation. In the rules hierarchy, rule set categories consist of rule sets, which, in turn, further subdivide into individual rules.

- Rules sets are logical groupings of rules specific to a business function with Guidewire BillingCenter.
- Rules contain the Gosu code (condition and action) that perform the actual business logic.

It is possible to create new rule set categories through Guidewire Studio.

1. In the Studio **Project** window, navigate to **configuration** → **config** → **Rule Sets**.
2. Right-click **Rule Sets**, and then and click **New** → **Rule Set Category**.
3. Enter a name for the rule set category.

Studio inserts the rule set category in the category list in alphabetic order.

To create a new rule set

In the base configuration, Guidewire provides a number of business rules, divided into rules sets, which organize the business rules by function. It is possible to create new rule sets through Guidewire Studio.

1. In the Studio **Project** window, navigate to **configuration** → **config** → **Rule Sets**. Although the label is **Rule Sets**, the children of this folder are actually rule set categories.
2. Select a rule set category or create a new rule set category.
3. Right-click the category, and then click **New** → **Rule Set**.

4. Enter the following information:

Field	Description
Name	Studio displays the rule name in the middle pane. For the Guidewire recommendations on rule set names, see “Generating Rule Debugging Information” on page 27. In general, however, if you create a rule set for a custom entity named `Entity_Ext`, you must name your rule set `Entity_Ext<RuleSet>`. Thus, if you want the custom entity to invoke the Preupdate rules, then name your rule set `Entity_ExtPreupdate`. There are some variations in how to name a rule set. See the existing rule sets in that category to determine the exact string to append and follow that same pattern with new rule sets in that category.
Description	Studio displays the description in a tab at the right of the Studio if you select the rule set name in the mid-dle pane. Guidewire recommends that you make the description meaningful, especially if you have multiple people working on rule development. In any case, a meaningful rule description is particularly useful as time passes and memories fade.
Entity Type	BillingCenter uses the entity type as the basis on which to trigger the rules in this rule set. For example, suppose that you select a rule set, then a rule within the set. Right-click and select **Complete Code** from the menu. Studio displays the entity type around which you base the rule actions and conditions.

To create a new rule

1. Select the rule set that will contain the new rule in the Studio **Resources** pane.
2. Do one of the following:
   - If the new rule is to be a top-level rule, select the rule set name in the middle pane.
   - If the new rule is to be a child rule, expand the rule set hierarchy in the middle pane and select the parent rule.
3. Select **New Rule** from the **Rule** menu, or right-click and select **New Rule**.
4. Enter a name for the new rule in the **New Rule** dialog box. Studio creates the new rule as the last child rule of the currently selected rule (or rule set).

To access a rule set from Gosu code

You can access a rule set within a rule set category (and thus, all the rules within the rule set) by using the following Gosu `invoke` method. You can use this method to invoke a rule set in any place that you use Gosu code.

```
rules.RuleSetCategory.RuleSet.invoke(entity)
```

You can only invoke a rule set through the Gosu `invoke` method, not individual rules. Invoking the rule set trig-gers evaluation of every rule in that rule set, in sequential order. If the conditions for a rule evaluate to true, then BillingCenter executes the actions for that rule.

Renaming or Deleting a Rule

Use the following menu commands to rename a rule or to delete it entirely from the rule set. You can also access these commands by right-clicking a rule and selecting the command from the drop-down list.

Command	Description	Actions you take
Rule → **Rename Rule**	Renames the currently selected rule	Select the rule to rename in the center pane of Studio, and then select **Rename Rule** from the **Rule** menu, or right-click and select **Rename Rule**. Enter the new name for the rule in the **Input** dialog box. You must save the rule for the change to become permanent.
Edit → **Delete**	Deletes the currently selected rule	Select the rule to delete in the center pane of Studio, then select **Delete** from the **Edit** menu, or right-click and select **Delete**. Note that there is no secondary dialog window that asks you to confirm your selection. You can use the **Delete** command only to delete rules.

Renaming a Rule

At a structural level, Guidewire BillingCenter stores each rule as a separate Gosu class, with a `.gr` extension. The name of the Gosu class corresponds to the name of the rule that you see in the Studio **Project** window. BillingCenter stores the rule definition classes in the following location:

```
BillingCenter/modules/configuration/config/rules/...
```

If you rename a rule set, BillingCenter renames the class definition file in the directory structure and any internal class names. It also renames the directory name if the rule has children. Thus, BillingCenter ensures that the rule class names and the file names are always synchronized.

Changing the Root Entity of a Rule

BillingCenter bases each Gosu rule on a specific business entity. In general, the rule set name reflects this entity. For example, in the Preupdate rule set category, you have Activity Preupdate rules and Contact Preupdate rules. These rule set names indicate that the root entity for each rule set is—respectively—the `Activity` object and the `Contact` object. You can also determine the root entity for a specific rule by using the Studio code completion functionality:

1. Create a sample line of code similar to the following:
   ```
   var test =
   ```
2. Right-click after the = sign and select **Complete Code** from the menu. The Studio code completion functionality provides the root entity. For example:
   ```
   activity : Activity
   contact  : Contact
   ```
   You can also use CTRL-Space to access the Studio code completion functionality.

BillingCenter provides the ability to change the root entity of a rule through the use of the right-click **Change Root Entity** command on a rule set.

Why Change a Root Entity?

The intent of this command is to enable you to edit a rule that you otherwise cannot open in Studio because the declarations failed to parse.

Do not use this command in other circumstances.

For example, suppose that you have the following sequence of events:

1. You create a new entity in BillingCenter, for example, `TestEntity`. Studio creates a `TestEntity.eti` file and places it in the following location:

   ```
   modules/configuration/config/extensions
   ```

2. You create a new rule set category called **TestEntityRuleSetCategory** in **Rule Sets**, setting `TestEntity` as the root entity. Studio creates a new folder named `TestEntityRuleSetCategory` and places it in the following location:

   ```
   modules/configuration/config/rules/rules
   ```

3. You create a new rule set under **TestEntityRuleSetCategory** named **TestEntityRuleSet**. Folder **TestEntityRuleSetCategory** now contains the rule set definition file named `TestEntityRuleSet.grs`. This file contains the following (simplified) Gosu code:

   ```
   @gw.rules.RuleName("TestEntityRuleSet")
   class TestEntityRuleSet extends gw.rules.RuleSetBase {
     static function invoke(bean : entity.TestEntity) : gw.rules.ExecutionSession {
       return invoke( new gw.rules.ExecutionSession(), bean )
     }
     ...
   }
   ```

   Notice that the rule set definition explicitly invokes the root entity object: `TestEntity`.

4. You create one or more rules in this rule set that use `TestEntity` object, for example, **TestEntityRule**. Studio creates a `TestEntityRule.gr` file that contains the following (simplified) Gosu code:

   ```
   internal class TestEntityRule {
     static function doCondition(testEntity : entity.TestEntity) : boolean {
       return /*start00rule*/true/*end00rule*/
     }
     ...
   }
   ```

   Notice that this definition file also references the `TestEntity` object.

5. Because of upgrade or other reasons, you rename your `TestEntity` object to `TestEntityNew` by changing the file name to `TestEntityNew.eti` and updating the entity name in the XML entity definition:

   ```
   <?xml version="1.0"?>
   <entity xmlns="http://guidewire.com/datamodel"
           entity="TestEntityNew" ... >
   </entity>
   ```

   This action effectively removes the `TestEntity` object from the data model. This action, however, does not remove references to the entity that currently exist in the rules files.

6. You update the database by stopping and restarting the application server.

7. You stop and restart Studio.

As Studio reopens, it presents you with an error message dialog. The message states that Studio could not parse the listed rule set files. It is at this point that you can use the **Change Root Entity** command to shift the root entity in the rule files to the new root entity. After you do so, Studio recognizes the new root entity for these rule files.

To change the root entity of a rule

1. Select a rule set.
2. Right-click and select **Change Root Entity** from the drop-down menu. Studio prompts you for an entity name.
3. Enter the name of the new root entity.

After you supply a name:

- Studio performs a check to ensure that the name you supplied is a valid entity name.
- Studio replaces occurrences of the old entity in the function declarations of all the files in the rule set with the new entity. This replacement only works, however, if the old root type was in fact an entity.
- Studio changes the name of the entity instance passed into the condition and action of each rule.

- Studio does not propagate the root entity change to the body of any affected rule. You must correct any code that references the old entity manually.

The intent of this command is to enable you to edit a rule that you otherwise cannot open in Studio because the declarations failed to parse. Do not use this command in other circumstances.

Making a Rule Active or Inactive

BillingCenter skips any inactive rule, acting as if its conditions are false. (This causes it to skip the child rules of an inactive rule, also.) You can use this mechanism to temporarily disable a rule that is causing incorrect behavior (or that is no longer needed) without deleting it. Sometimes, it is helpful to keep the unused rule around in case you need that rule or something similar to it in the future.

To make a rule active, set the check box next to it. To make a rule inactive, clear the check box next to it.

chapter 4

Writing Rules: Testing and Debugging

This topic describes ways to test and debug your rules.

This topic includes:

- “Generating Rule Debugging Information” on page 27
- “Using Custom Logging Methods to Debug Rule Issues” on page 28

See also

- “Generating a Rule Repository Report” on page 41
- “Generating a Profiler Rule Execution Report” on page 42
- “Viewing Rule Information in the Profiler Chrono Report” on page 43
- “Guidewire Profiler” on page 175 in the *System Administration Guide*

Generating Rule Debugging Information

It is a very useful practice to add printing and logging statements to your rules to identify the currently executing rule, and to provide information useful for debugging purposes. Guidewire recommends that you use all of the following means of providing information extensively:

- Use the `print` statement to provide instant feedback in the server console window.
- Use the `gw.api.util.Logger.*` methods to print out helpful error messages to the application log files.
- Use comments embedded within rules to provide useful information while troubleshooting rules.

Printing Debugging Information

The `print(`*`String`*`)` statement prints the *`String`* text to the server console window. This provides immediate feedback. You can use this method to print out the name of the currently executing rule and any relevant variables or parameters. For example, the following code prints the name of the currently executing rule set to the server console window:

```
print("This is an A N N O U N C E M E N T   M E S S A G E -- I am running the "
  + actions.getRuleSet().DisplayName + " Rule Set")
```

Logging Debugging Information

Use the `gw.api.util.Logger.*` methods to send information to application log files or the console window for debugging purposes. See the following for details:

- "Logging" on page 445 in the *Integration Guide*
- "Configuring Logging" on page 21 in the *System Administration Guide*

Determining the Complete Package Path of a Rule

Use the following Gosu code determine the full package path for an executing rule:

```
gw.rules.ExecutionSession.getCurrent().RunningRuleType.Name
```

For example, you can place the code in a `print` statement in the rule:

```
print(gw.rules.ExecutionSession.getCurrent().RunningRuleType.Name)
```

Placing the above `print` statement in a Preupdate Activity rule generates the following text in the console log:

```
rules.Preupdate.ActivityPreupdate_dir.ACT_1000345
```

Using Custom Logging Methods to Debug Rule Issues

You can also write a custom Gosu method that logs information. You then call this method from the rule to perform the logging operation. The following example uses Gosu class method `logRule` to output information about the rule and rule set currently executing. Notice that you must create a new instance of the class before you can use it.

- See the for information about creating and using Gosu class methods.
- See "Script Parameters" on page 101 in the *Configuration Guide* for information about creating and using script parameters. (Notice that using a script parameter in this fashion enables you to easily turn rule logging on and off.)

Rule Action

```
var log = new util.CustomLogging();
log.logRule( activity, actions.getRuleSet().DisplayName as java.lang.String,
        actions.getRule() as java.lang.String, "Custom Message..." )
```

CustomLogging Class

```
package util;

class CustomLogging {
  public function logRule( act:Activity, ruleSetName: String, ruleName:String,
    logMessage:String ) : Boolean {

    var ruleInfo:String = "Rule Set: " + ruleSetName + "\n Rule Name: " + ruleName;
    var message:String = (logMessage != null) ? ( "\n Message: " + logMessage) : ("");

    if(ScriptParameters.DO_LOGGING) {
      gw.api.util.Logger.logDebug( "\n### BILLINGCENTER RULE EXECUTION LOGGER (Activity: " + act
              + ") ###" + "\n " + ruleInfo + message + "\n " + "Timestamp: "
              + gw.api.util.StringUtil.formatTime(gw.api.util.DateUtil.currentDate(), "full")+ "\n" );
    }
    return true;
  }
}
```

Result

```
### BILLINGCENTER RULE EXECUTION LOGGER (Activity: ApprovalActivity) ###
    Rule Set: Default Group Activity Assignment Rules
    Rule Name: Assign by Round-Robin
    Message: Custom text...
    TimeStamp: 2:27:36 PM PST
```

chapter 5

Writing Rules: Examples

This topic describes ways to perform more complex or sophisticated actions in rules.

This topic includes:

Accessing Fields on Subtypes

Various entities in BillingCenter have subtypes, and a subtype may have fields that apply only to it, and not to other subtypes. For example, a `Contact` object has a `Person` subtype, and that subtype contains a `DateOfBirth` field. However, `DateOfBirth` does not exist on a `Company` subtype. Similarly, only the `Company` subtype has the `Name` (company name) field.

Because these fields apply only to particular subtypes, you cannot reference them in rules by using the primary root object. For example, the following illustrates an invalid way to refer to the primary contact for an account:

```
Account.Accountcontacts.Contact.LastName.compareTo("Smith")  //Invalid
```

To access a field that belongs to a subtype, you must “cast” (or convert) the primary object to the subtype by using the “as” operator. For example, you would cast a contact to the `Person` subtype using the following syntax:

```
(Account.Contacts.PrimaryContact[0] as Person).LastName.compareTo( "Smith" )
```

Looking for One or More Items Meeting Conditions

If you want to check the value of a single field, such as whether a particular activity is overdue, you use a construction similar to the following:

```
Activity.TargetDate < gw.api.util.DateUtil.currentDate()
```

However, if you want to know whether a trouble ticket has any unfinished activities that are overdue, you need to look at all activities on the trouble ticket. Clearly, you need something more complicated than `TroubleTicket.*` to express this condition. BillingCenter provides an `exists(...in...where...)` syntax to describe these tests.

For example:

```
exists ( Activity in TroubleTicket.Activities where Activity.Status == "open"
        and Activity.AssignmentDate < gw.api.util.DateUtil.currentDate() )
```

This condition evaluates to `true` if there are one or more activities connected to an account that meet the conditions described.

See "Existence Testing Expressions" on page 74 in the *Gosu Reference Guide* for more information on Gosu `exists(...in...where)` expressions.

Taking Actions on More Than One Subitem

BillingCenter provides a `for(... in ...)` syntax and an `if(...) {` ***`then do something`*** `}` syntax that you can use to construct fairly complicated actions. In the following example, the rule iterates through the various invoices on an account to determine if the invoice due is before the current date. If it is, the rule sets the invoice due date to the current date.

```
var account1 : Account //Some account

for (var inv in account1.Invoices) {
  if (inv.PaymentDueDate < gw.api.util.DateUtil.currentDate()) {
    inv.PaymentDueDate = gw.api.util.DateUtil.currentDate()
  }
}
```

Notice that the "{" and the "}" curly braces mark the beginning and end of a block of commands.

- The outer set brackets one or more commands to do "for" each line on the policy.
- The inner set brackets the commands to do "if" the specified coverage group is found.

You can use the `Length` method on a subobject to determine how many subobjects exist. For example, if there are five invoices on this account, then the following expression returns the value 5:

```
account1.Invoices.Length
```

See "Gosu Conditional Execution and Looping" on page 96 in the *Gosu Reference Guide* for more information on the `for(... in ...)` Gosu syntax.

Checking Permissions

BillingCenter provides a Gosu mechanism for checking user permission on an object by accessing properties and methods off the object in the `perm` namespace.

- BillingCenter exposes static permissions that are non-object-based (like the permission to create a user) as Boolean properties.
- BillingCenter exposes permissions that take an object (like the permission to edit a claim) as methods that take an entity as their single parameter.
- BillingCenter exposes application interface permissions as typecodes on the `perm.System` object.

All the properties and methods return Boolean values indicating whether or not the user has permission to perform the task. BillingCenter always evaluates permissions relative to the current user unless specifically instructed to do otherwise. You can use permissions anywhere that you can use Gosu (in PCF files, rules, and classes) and there is a current user.

For example:

```
print(perm.Activity.view(Activity))
print(perm.User.create)
print(perm.System.acctbillall)
```

You can also check that any given user has a specific permission, using the following Gosu code:

```
var u : User = User( "SomeUser" /* Valid user name*/ )
var hasPermission = exists (role in u.Roles
        where exists (perm in role.Role.Privileges where perm.Permission=="SomeValidPermission"))
```

If using this code in a development environment, you must connect Studio to a running development application server before Studio recognizes users and permissions.

chapter 6

Rule Set Categories

This topic describes the sample rules that Guidewire provides as part of the BillingCenter base configuration.

This topic includes:

- "Rule Set Summaries" on page 33
- "Assignment" on page 34
- "Event Message" on page 34
- "Exception" on page 37
- "Preupdate" on page 38
- "Validation" on page 40

Rule Set Summaries

BillingCenter Studio contains the following sample rule sets:

Rule set category	Contains rules to
Assignment	Determine the responsible party for an activity or trouble ticket
Event Message	Handle communication with integrated external systems. See the "Messaging and Events" on page 303 in the *Integration Guide* for information on events and event messaging
Exception	Specify an action to take if an Activity is overdue and has entered the escalated state.
Preupdate	Trigger automatic actions anytime that an account or other high-level entity changes.
Validation	Validate a small subset of the BillingCenter entities.

Assignment

Guidewire refers to certain business entities as *assignable* entities. Thus, it is possible to determine—either through the use of assignment business rules or through the use of Gosu assignment methods—the party responsible for that entity. In the BillingCenter base configuration, Guidewire defines the following entities as *assignable*:

- `Activity`
- `AgencyBillWorkflow`
- `BCWorkflow`
- `DelinquencyProcess`
- `DelProcessWorkflow`
- `Hold`
- `TroubleTicket`
- `UserRoleAssignment`

At the minimum, an entity must implement the `Assignable` delegate to be assignable. It is not possible to make an entity that is not assignable in the base configuration to be assignable. You can, however, extend an entity, and make the new entity assignable.

You use Gosu assignment methods to set an assigned group and user for an assignable entity. These assignment methods run either in the context of the Assignment engine (within the Assignment rule set) or simply within Gosu code (within a class or enhancement, for example).

For information on assignment and assignable entities in Guidewire BillingCenter, see “Assignment in BillingCenter” on page 47.

Event Message

IMPORTANT BillingCenter runs the Event Message rules as part of the database bundle commit process. Only use these rules to create integration messages.

In the base configuration, there is a single rule set—Event Fired—in the Event Message rule category. The rules in this rule set:

- Perform event processing
- Generate messages about events that have occurred

BillingCenter 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:

- BillingCenter runs the rules in the Event Fired rule set once for every event for which a message destination has registered interest.
- BillingCenter runs the Event Fired rule set once for each destination that is listening for that particular event. Thus, it is possible for BillingCenter to run the Event Fired rules sets multiple times for each event, once for each destination interested in that event.

In a normal bundle commit any new or updated entity can generate an event. This event can be detected in the event messaging rules and used to create a new message for communicating with an external system. BillingCenter uses bulk updates to maintain many denormalized values. If the only change to an entity is a bulk update change then that entity does not appear as a modified entity in the bundle. Therefore the entity change does not fire an event.

Typically, the update of a denormalized field is not important to business logic because denormalization is used to cache information that is already stored elsewhere. Write rules to detect changes to the original entity rather than side effects of those changes. For example, look for `TransactionAdded` rather than looking for changes in the `AmountDenorm` of a `TAccount`.

Messaging Events

BillingCenter automatically generates certain events for most top-level objects. (Guidewire calls these events *standard* events.) In general, this occurs for any addition, modification, or removal (or retirement) of a top-level entity. BillingCenter automatically generates the following events on the `Activity` object:

- `ActivityAdded`
- `ActivityChanged`
- `ActivityRemoved`

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 information on...

- Base configuration events, see “List of Messaging Events in BillingCenter” on page 323 in the *Integration Guide*.
- Custom events, see “Triggering Custom Event Names” on page 330 in the *Integration Guide*.

Message Destinations and Message Events

You use the Studio Messaging editor to define message destinations and message events.

- 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 BillingCenter that is of possible interest to an external system. For example, this can be adding, changing, or removing a Guidewire entity.

Using the editor, it is possible to associate (register) one or more events with a particular message definition. For example, in the base configuration, BillingCenter associates the following events with the `PAS` (Policy Administration System) message destination (ID=100):

- `PAS_(\w)*`

If you modify, add, or remove a `Contact` object (among others), BillingCenter generates the relevant event. Then, during a bundle commit of the `Contact` object, BillingCenter notifies any Event Message rule that has registered an interest in one of these events that the event has occurred. You can then use this information to generate a message to send to an external system or to the system console for logging purposes, for example.

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 plugin handles different parts of what a destination does. Thus:

- 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.

IMPORTANT After the BillingCenter application server starts, BillingCenter initializes all message destinations. BillingCenter saves a list of events for which each destination requested notifications. As this happens at system start-up, you must restart the BillingCenter application if you change the list of events or destinations.

Generating Messages

Use 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}")
```

The following is an example of a constructed message that provides more detailed information if BillingCenter adds an account successfully.

```
if (messageContext.EventName == "AccountAdded") {
  var account = messageContext.Root as Account
  messageContext.createMessage("Account [" + account.AccountNumber + "] added")
}
```

A Word of Warning

Be extremely careful about modifying entity data in Event Fired rules and messaging plugin implementations. Use these rule 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 re-run. Only change fields that are not modifiable from the user interface. For example, set custom data model extension flags only used by messaging code.

Guidewire does not support the following:

- Guidewire does not support (and, it is dangerous) to add or delete business entities from Event Fired rules or messaging plugins (even indirectly through other APIs).
- Guidewire does not support—even within the Event Message rule set—calling any message acknowledgment or skipping methods such as `reportAck`, `reportError`, or `skip`. Use those methods only within messaging plugins.
- Guidewire does not support creating messages outside of the Event Message rule set.

See also

- “Messaging and Events” on page 303 in the *Integration Guide*
- “List of Messaging Events in BillingCenter” on page 323 in the *Integration Guide*
- “Using the Messaging Editor” on page 131 in the *Configuration Guide*

Detecting Object Changes

It is possible that you want to listen for a change in an object that does not automatically trigger an event if you update it. For example, suppose that you want to listen for a change to the `User` object (an update of the address, perhaps). However, in this case, the `User` object itself does not contain an address. Instead, BillingCenter stores addresses as an array on `UserContact`, which is an extension of `Person`, which is a subtype of `Contact`, which points to `User`. Therefore, updating an address does not directly in itself touch the `User` object.

However, in an Event Message rule, you can listen for the `ContactChanged` event that BillingCenter generates every time that the address changes. The following example illustrates this concept. (It prints out a message to the system console anytime that the address changes. In actual practice, you would use a different message destination, of course.)

Condition

```
//DestID 68 is the Console Message Logger
messageContext.DestID == 68 and messageContext.EventName == "ContactChanged"
```

Action

```
uses gw.api.util.ArrayUtil

var message = messageContext.createMessage( "Event: " + messageContext.EventName )
var contact = message.MessageRoot as Contact
var fields = contact.PrimaryAddress.ChangedFields
print( ArrayUtil.toString( fields ) )
```

Event Fired

In the following example, the rule constructs a message containing the account number any time that the application adds an account.

Rule Conditions

```
// Only fire if adding an account
messageContext.EventName == "AccountAdded"
```

Rule Actions

```
// Create handle to root object.
var account = messageContext.Root as Account

// Print a message to the application log
print("Account [" + account.AccountNumber + "] added")

// Create the message
messageContext.createMessage("Account [" + account.AccountNumber + "] added")
```

Exception

Use the Exception rule sets to specify an action to take if an Activity is overdue and has entered the escalation state. You can use this rule set to send e-mail, create an activity or to set a flag, among other actions.

BillingCenter runs these rule sets at regularly scheduled intervals. See “Batch Processing” on page 107 in the *System Administration Guide* for information on exception batch processes.

Activity Escalation Rules

An activity has two dates associated with it:

Due date	The target date to complete this activity. If the activity is still open after this date, it becomes overdue.
Escalation date	The date at which an open and overdue activity becomes escalated and needs urgent attention.

BillingCenter runs scheduled process `activityesc` every 30 minutes (by default). This system process runs against all open activities in the system to find activities that need to be escalated using the following criteria:

- The activity has an escalation date that is non-null.
- The escalation date has passed.
- The activity has not already been escalated.

BillingCenter marks each activity that meets the criteria as escalated. After the batch process runs, BillingCenter runs the escalation rules for all the activities that have hit their escalation date.

Use this rule set to specify what actions to take anytime that the activity enters the escalated condition. For example, you can use this rule set to reassign escalated activities. The simplest approach to escalated activities is to add an activity for the appropriate supervisor to intervene and check on why the work is not yet complete. You can also decide that there is no follow-up action needed for certain kinds of activities.

In the following example, if an activity associated with a trouble ticket becomes escalated, the rule sets the activity priority to “high”.

Condition

```
Activity.TroubleTicket.Escalated
```

Action

```
Activity.TroubleTicket.Priority ="high"
```

Preupdate

Use the Preupdate rule sets to perform domain logic or validation that you want to commit to the database before BillingCenter commits the entity in question to the database. For example, use these rule sets to handle changes to the delinquency process that occur as a result of other entity changes. Only a change to an entity marked as validatable in its definition file can trigger a preupdate rule.

It is important to understand that the Preupdate rules are non-recursive. The Preupdate rules do not run a second time on objects modified during their initial execution.

Triggering Preupdate Rules

The following BillingCenter base configuration entities all trigger preupdate rules:

- `Account`
- `Activity`
- `AgencyBillPlan`
- `BaseMoneyReceived`
- `BillingInstruction`
- `Charge`
- `Contact`
- `DelinquencyProcess`
- `Disbursement`
- `History`
- `Hold`
- `Invoice`
- `OutgoingPayment`
- `PaymentRequest`
- `Plan`
- `Policy`
- `PolicyPeriod`
- `Policy`
- `Producer`
- `ProducerPayment`
- `ProducerStatement`
- `SuspensePayment`
- `Transaction`
- `TroubleTicket`
- `User`
- `Writeoff`

For an entity to trigger the preupdate rules, it must implement the `Validatable` delegate. This is true for an entity in the base BillingCenter configuration or for a custom entity that you create.

BillingCenter runs the preupdate and validation rules:

- whenever an instance of a validatable entity is created or modified.

- whenever a *subentity* of a validatable entity is created or modified *and* the validatable entity is connected to the subentity through a `triggersValidation="true"` link.

IMPORTANT In running the Preupdate rule set, BillingCenter 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 then 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 then cause the preupdate rules to run on that object as well.

This is also the case for entities newly created within a preupdate rule. For example, if you create a new activity within a preupdate rule, BillingCenter does not then run the Activity preupdate rules on that activity.

See also

- For information on how to create an entity that implements the `Validatable` delegate, see “Delegate Data Objects” on page 158 in the *Configuration Guide*.

Preupdate Rules and Custom Entities

It is possible to create preupdate rules for custom entities, meaning those entities that you create yourself and which are not part of the base Guidewire BillingCenter configuration.

To create an extension entity that triggers preupdate rules

For an entity to trigger a preupdate rule:

1. The entity must implement the `Validatable` delegate. In short, any extension entity that you create that you want to trigger a preupdate rule must implement the following code:
   ```
   <implementsEntity name="Validatable"/>
   ```
2. The preupdate rule sets that you want the custom entity to trigger must conform to the following naming convention:
   - Place your preupdate rules in the *Preupdate* rule set category and name the rule set `<entity name>Preupdate`.

   The preupdate rule must exist in a preupdate rule set that reflects the extension entities name and ends with `Preupdate`. In other words, 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 place your rules in this rule set.

   For example, if you create an extension entity named `NewEntityExt`, then you need to create a rule set to hold your preupdate rules and name it:
   ```
   NewEntityExtPreupdate
   ```
   For information creating new rules sets, see “Working with Rules” on page 21. See especially the section entitled “To create a new rule set”.

Transaction Preupdate Rule Example

IMPORTANT Avoid adding many rules to this rule set. This rule set can seriously affect performance because there are typically many transactions active in the system.

In the following example, the rule first determines if this transaction is of type `AccountAdjustment`. If so, it creates a history event object (`history`) to track the change.

Rule Conditions

```
transaction typeis AccountAdjustment
```

Rule Actions

```
var adjustment = transaction as AccountAdjustment
var history = new AccountHistory(adjustment)

history.Account = adjustment.Account
history.EventDate = adjustment.TransactionDate
history.EventType = HistoryEventType.TC_ACCOUNTADJUSTED
history.Transaction = adjustment
history.Description = adjustment.LongDisplayName

adjustment.Account.addToHistory(history)
```

Validation

BillingCenter provides the ability to write validation rules against the following entities:

- `Activity`
- `Contact`
- `Group`
- `Region`
- `User`

IMPORTANT You can only write validation rules against the listed entities. No other entity—even if it implements the `Validatable` delegate—triggers a validation rule.

The `EnableWorkQueueValidation` parameter determines whether validation is performed on BillingCenter-specific work queues. See "EnableWorkQueueValidation" on page 60 in the *Configuration Guide*.

Guidewire does not provide sample rules for validation in the base BillingCenter configuration.

chapter 7

BillingCenter Rule Reports

This topic provides information on how to generate reports that provide information on the BillingCenter business rules.

This topic includes:

- "Generating a Rule Repository Report" on page 41
- "Generating a Profiler Rule Execution Report" on page 42
- "Viewing Rule Information in the Profiler Chrono Report" on page 43

See also

- "Generating Rule Debugging Information" on page 27
- "Using Custom Logging Methods to Debug Rule Issues" on page 28

Generating a Rule Repository Report

To facilitate working with the Gosu business rules, BillingCenter provides a command line tool to generate a report describing all the existing business rules. This tool generates the following:

- An XML file that contains the report information
- An XSLT file that provides a style sheet for the generated XML file

To create a rule repository report

1. Navigate to the `BillingCenter/bin` directory.
2. At a command prompt, type:

```
gwbc regen-rulereport
```

This command generates the following files:

```
build/rules/RuleRepositoryReport.xml
build/rules/RuleRepositoryReport.xslt
```

After you generate these files, it is possible to import the XML file into Microsoft Excel, for example. You can also provide a new style sheet to format the report to meet your business needs.

Generating a Profiler Rule Execution Report

The Guidewire Profiler provides information about the runtime performance of specific application code. It can also generate a report listing the business rules that individual user actions trigger within Guidewire BillingCenter. The Profiler is part of the Guidewire-provided Server Tools. To access the Server Tools, Guidewire Profiler, and the rule execution reports, you must have administrative privileges.

To generate a rule execution report

1. Log into Guidewire BillingCenter using a user account with access to the Server Tools.
2. Access the Server Tools and click **Guidewire Profiler** on the menu at the left-hand side of the screen.
3. On the Profiler **Configuration** page, click **Enable Web Profiling for this Session**. This action enables profiling for the current session. Profiling provides information about the runtime performance of the application, including information on any rules that the application executes.

   Guidewire also provides a way to enable web profiling directly from within the BillingCenter interface. To use the alternative method, press ALT+SHIFT+p within BillingCenter to open a popup in which you can enable web profiling for the current session. If you use this shortcut, you do not need to access the Profiler directly to initiate web profiling. You still need to access the Profiler, however, to view the rule execution report.
4. Navigate back to the BillingCenter application screens.
5. Perform a task for which you want to view rule execution.
6. Upon completion of the task, return to Server Tools and reopen BillingCenter Profiler.
7. On the Profiler **Configuration** page, click **Profiler Analysis**. This action opens the default **Stack Queries** analysis page.
8. Under **View Type**, select **Rule Execution**.

See also

- "Guidewire Profiler" on page 175 in the *System Administration Guide*

Interpreting a Rule Execution Report

After you enable the profiler and perform activity within BillingCenter, the profiler **Profiler Analysis** screen displays zero, one, or multiple stack traces.

Stack trace	Profiler displays...
None	A message stating that the Profiler did not find any stack traces. This happens if the actions in the BillingCenter interface did not trigger any business rules.
One	A single expanded stack trace. This stack trace lists, among other information, each rule set and rule that BillingCenter executed as a result of user actions within the BillingCenter interface. The Profiler identifies each stack trace with the user action that created the stack trace. For example, if you create a new user within BillingCenter, you see `NewUser -> UserDetailPage` for its stack name.
Multiple	A single expanded stack trace and multiple stack trace expansion buttons. There are multiple stack trace buttons if you perform multiple tasks in the interface. Click a stack trace button to access that particular stack trace and expand its details.

Each stack trace lists the following information about the profiled code:

- **Time**
- **Name**
- **Frame (ms)**
- **Elapsed (ms)**
- **Properties and Counters**

Within the stack trace, it is possible to:

- Expand a stack trace completely by clicking **Expand All**.
- Collapse a stack trace completely by clicking **Collapse All**.
- Partially expand or collapse a stack trace by clicking the **+** (plus) or **–** (minus) next a stack node.

The profiler lists the rule sets and rules that BillingCenter executed in the **Properties and Counters** column in the order in which they occurred.

Viewing Rule Information in the Profiler Chrono Report

It is possible to use the Guidewire Profiler **Chrono** report to view frames filtered out by the **Rule Execution** report. You will need the following information from the **Rule Execution** report to access the correct information in the **Chrono** report:

- Name of the session
- Name of the stack
- Time offset

See also

- "Guidewire Profiler" on page 175 in the *System Administration Guide*

To view the Chrono report

1. Log into Guidewire BillingCenter using a user account with access to the Server Tools.
2. Access the Server Tools and click **Guidewire Profiler** on the menu at the left-hand side of the screen.
3. Generate a rule execution report. See "Generating a Profiler Rule Execution Report" on page 42 for details.
4. Select **Guidewire Profiler** → **Profiler Analysis** → **Web**.
5. Under **Profiler Result**, select **Chrono** from the **View Type** drop-down list.
6. Select the desired session, for example, **2015/05/05 09:23:25 web profiler session.**
7. Select the desired stack, for example, **DesktopActivities -> AssignActivity.**
8. As you select a stack, the **Profiler** page shows additional information about that stack at the bottom of the page.
9. Expand the **Time** node as needed to select the time offset that is of interest to you.

part II

Advanced Topics

chapter 8

Assignment in BillingCenter

This topic describes how Guidewire BillingCenter assigns a business entity or object.

This topic includes:

Understanding Assignment

Guidewire refers to certain business entities as *assignable* entities. This means, generally, that it is possible to designate a specific user as the party responsible for that entity. Guidewire defines the following entities as assignable in the BillingCenter base configuration:

- `Activity`
- `AgencyBillWorkflow`
- `BCWorkflow`
- `DelinquencyProcess`
- `DelProcessWorkflow`
- `Hold`
- `TroubleTicket`
- `UserRoleAssignment`

You can create additional assignable entities through the following methods:

- Modify the base configuration data model by extending an existing entity to make it assignable

- Modify the base configuration data model by creating a new entity and making it assignable

To be assignable, an entity must implement certain required delegates and interfaces, not the least of which is the `Assignable` delegate class.

You use Gosu assignment methods to set an assigned group and user for an assignable entity. You can use these assignment methods either within the Assignment rule sets or within any other Gosu code (a class or an enhancement, for example).

Assignment Persistence

BillingCenter persists the assignment any time that you persist the entity being assigned. Therefore, if you invoke the assignment methods on some entity from arbitrary Gosu code, you need to persist that entity. This can be, for example, either as part of a page commit in the BillingCenter interface or through some other mechanism.

Assignment Queues

BillingCenter does not currently support assignment queues.

BillingCenter Assignment

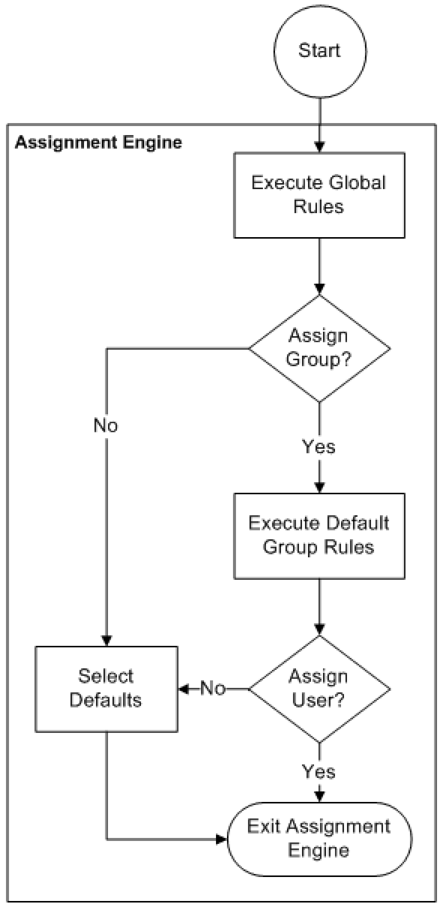
Assignment that uses the Assignment engine works within the context of the Assignment rules, which BillingCenter groups into the following categories:

- Global Assignment Rules
- Default Group Assignment Rules

In general, BillingCenter uses the global assignment rules to determine the group to which to assign the entity. It then the runs the default group assignment rules to assign the entity to a user in the chosen group. The Assignment engine runs the assignment rules until it either completes the assignment or it runs out of rules to run.

Guidewire designs the assignment rules to *trickle down* through the group structure. For example, if a user assigns an activity using automated assignment, the Assignment engine runs a global activity assignment rule first. Typically, the global rule sets a group. Then, the Assignment engine calls the assignment rule set for that group. This rule set can either perform the final assignment or assign the activity to another group. If assigned to another group, BillingCenter calls the rule set for that group. This process continues until the Assignment engine completes the final assignment. (For simplicity, these examples use activity throughout, although the same logic applies to all assignable entities.)

The following graphic illustrates this process.



To support this process, the assignment engine contains the sense of *execution session* and *current group*. You cannot explicitly set these items.

- The *execution session* is one complete cycle through the assignment rules. In the Assignment engine graphic, this means all the processes that occur between the *Start* point and the *Exit Assignment Engine* termination point.
- The *current group* is the group to which the Assignment engine assigns the assignable entity after it executes an assignment rule set.

IMPORTANT The Assignment engine sets the current group as it exits the rule set. To alleviate unintended assignments, Guidewire strongly recommends that you exit the rule set immediately after making a group assignment.

Several of the assignment methods (notably `assignByUserAttributes` and `assignUser`) require a current group to be set. If the current group is not set, these methods return `false` immediately. The combination of the current group and these assignment actions can give unpredictable results if you do not implement them carefully.

Consider the following rule:

```
activity.CurrentAssignment.assignGroup(someGroup)

if ( activity.CurrentAssignment.assignByUserAttributes(someAttributes, false,
        activity.CurrentAssignment.AssignedGroup) ) {
  actions.exit()
}
```

At first, it would seem that this rule assigns the activity to someone in `someGroup` who has the appropriate attributes. However, it does the following:

- If the rule is in a global rule set, it does nothing at all.
- If the rule is in the rule set of another group—say group X—it assigns the activity to a user in group X.

The reason for this is that the logic for `assignByUserAttributes` only cares about the current group in the assignment context. (The current group is `null` in the global rule set and group X in group X's rule set). The `assignByUserAttributes` method does not care about what group the activity is currently assigned.

In actuality, as the rules trickle down through the organization, the Assignment engine sets the current group as it exits the rule set. To restate, Guidewire strongly recommends that you exit the rule set immediately after making a group assignment.

Therefore, for the previous example, you need to immediately exit the rule upon assigning the activity to a group (`someGroup`). Then, you would use a separate rule to assign the activity to a user who is a member of `someGroup` as determined by that user's attributes.

Rule 1

```
if ( activity.CurrentAssignment.assignGroup(someGroup) ) {  actions.exit() }
```

Rule 2

```
if ( activity.CurrentAssignment.assignByUserAttributes(someAttributes, false,
        activity.CurrentAssignment.AssignedGroup) ) {
  actions.exit()
}
```

See also

- "Assignment Execution Session" on page 51

Global Assignment Rules

Initially, the Assignment engine invokes the Global assignment rules for the entity type in question. There is a rule set for each entity type, meaning there is a rule set each to handle activities, disbursements, and trouble tickets.

The Assignment engine runs the global rules one time only. There are three valid outcomes for a Global Assignment rule:

- One of the rules assigns both a group and a user. In this case, the assignment process completes and the Assignment engine exits.
- One of the rules assigns a group. In this case the assignment process continues with the Default assignment rules.
- None of the rules perform any assignment. In this case, the Assignment engine assigns the entity to a default user and group and the Assignment engine exits.

BillingCenter triggers the global assignment rules any time that you use auto-assignment to assign an entity.

There is one global rule set for each type of assignable entity. These rules make the initial assignment decision. This, in turn, triggers further assignment rules for a particular group to narrow the selection criteria. (These rules can define geographical or business criteria to use in the selection process, for example.) Then, the assignment rules distribute the work to a specific user or queue within the chosen group.

Assignment logic is often *trickle-down* or hierarchical. For example, BillingCenter might use the Assignment rules to determine the ultimate assignment owner by first determining each of the following:

- The regional group
- The state group
- The local group
- The loss type, and finally,

- The individual

Guidewire recommends that you do not make user assignments directly using the Global assignment rules. Instead, use these rules to trigger further group selection rules. For example, suppose that you make an assignment directly to a user who does not have the requisite job role, or requisite authority. In this case, there is no other rule that governs the situation, and the Assignment engine makes no assignment.

It is possible for you to implement your own assignment class and call it directly from within BillingCenter, if you choose. Guidewire does not restrict you to using the base configuration Assignment rules only.

Default Group Assignment Rules

The Default Group assignment rules again contain one rule set for each assignable entity type. These rule sets contain rules that execute anytime that you manually assign an item from within BillingCenter. (These rules govern assignment selection also if you do not specify any other rule set for a group.)

After the Assignment engine invokes the rules, there are again three possibilities:

- One of the rules assigns a User. In this case, the assignment process is done and the Assignment engine exits.
- None of the rules assigns a User. However, one of the rules assigns a a different Group. In this case, the Assignment engine runs the Default Assignment rules again.

None of the rules perform any assignment. In this case, the assignment fails and the Assignment engine exits.

Assignment Execution Session

Guidewire provides a `gw.api.assignment.AssignmentEngineUtil` class that provides several useful helper assignment methods. Do not use these helper methods as you create new assignment logic. Instead, use this class to provide a bridge between any existing deprecated assignment methods without a group parameter and the current assignment methods that do take a group parameter.

The following helper method, in particular is useful in that it returns the default group stored in the `ExecutionSession`:

```
getDefaultGroupIDFromExecutionSession
```

Formerly, a number of (now deprecated) assignment methods did not use the `GroupBase` parameter, and instead relied on the implicit Assignment engine state. As you can invoke these methods outside of the Assignment engine, you can not rely on this implicit state during assignment. Therefore, Guidewire requires that all assignment methods take a `GroupBase` parameter. You can use the `getDefaultGroupIDFromExecutionSession` method to retrieve the current `GroupBase` value in `ExecutionSession`.

The returned `GroupBase` value from the `getDefaultGroupIDFromExecutionSession` method is the result of the last run of the rule set (global or default), *instead of the current value*. If a rule changes the assigned group but does not exit (by calling `actions.exit`), the assigned group is different from all the following rules in the current rule set.

If, instead, you want the currently assigned group on an object, an `Activity` object for example, then use the following:

```
activity.CurrentAssignment.AssignedGroup
```

The `CurrentAssignment.AssignedGroup` method returns the currently assigned group on the current assignment. If a rule sets the `AssignedGroup` value and does not exit (does not call `actions.exit`), all the following rules use this value for the current group.

As the return value of `getDefaultGroupIDFromExecutionSession` can be `null`, you need to check for this condition.

Deprecated Methods

Do not use the following deprecated methods on `AssignmentEngineUtil`:

- `getCurrentGroupFromES`
- `getCurrentGroupIDFromES`

Primary and Secondary Assignment

At its core, the concept of assignment in Guidewire BillingCenter is basically equivalent to ownership. The user to whom you assign an activity, for example, is the user who owns that activity, and who, therefore, has primary responsibility for it. Ownership of a `TroubleTicket` entity, for example, works in a similar fashion. Guidewire defines an entity that someone can own in this way as *assignable*.

Assignment in Guidewire BillingCenter is always made to a user and a group, as a pair (group, user). However, the assignment of the user is not dependent on the group. In other words, you can assign a user that is not a member of the specified group.

IMPORTANT In the base configuration, BillingCenter provides assignment methods that take only a user and that assign the entity to the default group for that user. Guidewire has now deprecated these types of assignment methods. *Do not use them.* Guidewire recommends that you rework any existing assignment methods that only take a user into assignment methods that take both a user and a group.

Guidewire BillingCenter distinguishes between two different types of assignment:

Primary (User-based) Assignment	Also known as user-based assignment, this type of assignment assigns an object that implements the `Assignable` delegate to a particular user.
Secondary (Role-based) Assignment	Also known as role-based assignment, this type of assignment assigns an object that implements the `RoleAssignments` array to a particular user role. (Each role is held by a single user at a time, even though the user who holds that role can change over time.)

Primary (User-based) Assignment

BillingCenter uses *primary* assignment in the context of ownership. For example, only a single user (and group) can own an activity. Therefore, an `Activity` object is primarily assignable. *Primary* assignment takes place any time that BillingCenter assigns an item to a *single* user.

Primary assignment objects *must* implement the `Assignable` and `BCAssignable` delegates. In the BillingCenter base configuration, the following objects implement the required delegates:

- `Activity`
- `Hold`
- `TroubleTicket`

It is common for BillingCenter to implement the `Assignable` delegate in the main entity definition file and the `BCAssignable` delegate in an entity extension file.

Secondary (Role-based) Assignment

BillingCenter uses *secondary* assignment in the context of user roles assigned to an entity that does not have a single owner. For example, an entity can have multiple roles associated with it as it moves through BillingCenter, with each role assigned to a specific person. Since each of the roles can be held by only a single user, BillingCenter represents the relationship by an array of `UserRoleAssignment` entities. These `UserRoleAssignment` entities are primarily assignable and implement the `Assignable` delegate.

Thus, for an entity to be secondarily assignable, it must have an associated array of role assignments, the `RoleAssignments` array. (This array contains the set of `UserRoleAssignment` objects.) In the BillingCenter base configuration, the following objects all contain a `RoleAssignmentss` array:

- `DelProcessWorkflow`
- `Disbursement`
- `TroubleTicket`

It is possible for an entity to be both primarily and secondarily assignable.

Secondary Assignment and Round-robin Assignment

Secondary assignment uses the assignment owner to retrieve the round-robin state. What this means is that different secondary assignments on the same assignment owner can end up using the same round-robin state and they can affect each other.

In general, if you use different search criteria for different secondary assignments, you do not encounter this problem as the search criteria is most likely different. However, if you want to want to make absolutely sure that different secondary assignments follow different round-robin states, then you need to extend the search criteria. In this case, add a flag column and set it to a different value for each different kind of secondary assignment. (See "Round-Robin Assignment" on page 57 for more information on this type of assignment.)

Assignment within the Assignment Rules

Guidewire provides sample assignment rules in the **Assignment** folder in the Studio **Resources** tree. These rule include both primary and secondary assignment. For assignments that use the Assignment engine (which is all rules in the **Assignment** folder), use the following assignment properties:

Assignment type	Use...
Primary	`entity.currentAssignment` This property returns the current assignment, regardless of whether you perform either primary or secondary assignment. If you attempt to use it for secondary assignment, then the property returns the same object as the `CurrentRoleAssignment` property.
Secondary	`entity.currentRoleAssignment` This property returns `null` if you attempt to use it for primary assignment.

If you assign an entity outside of the Assignment engine (meaning outside of the Assignment rules), then you do not need to use `currentAssignment`. Instead, you can use any of the methods that are available to entities that implement the `Assignable` delegate directly, for example:

```
activity.assign(group, user)
```

BillingCenter does not require that `user` be a member of `group`, although this is the most usual practice. In other words, you can assign a user that is not a member of the specified group.

Assignment Success or Failure

All assignment methods—*except for those that invoke the Assignment engine*—return a `Boolean` value that is `true` if BillingCenter successfully found an assignee and `false` otherwise. Thus, you have the following cases:

- Assignment Methods that Use the Assignment Engine
- Assignment Methods that Do Not Use the Assignment Engine

Assignment Methods that Use the Assignment Engine

You can invoke the Assignment engine by calling the `autoAssign` method from outside the Assignment rules, which invokes the Assignment engine automatically.

If you invoke the Assignment engine and it does not successfully assign a user, it then assigns the assignable item to the group supervisor. If it is not possible to assign the item to the group supervisor, then the Assignment engine assigns the assignable item to the BillingCenter default owner (`defaultowner`). (In some cases, it might not be possible to make an assignment to a group supervisor. For example, possibly the assignment rules failed to assign the item to a Group even, or, perhaps, the assigned Group has no supervisor.)

BillingCenter provides user `defaultowner` as the assignee of last resort. (This user's first name is *Default* and the last name is *Owner*, with a user name of `defaultowner`.) Guidewire recommends, as a business practice, that you have someone in the organization periodically search for outstanding work assigned to this user. If found, then you need to reassign these items to a proper owner. Guidewire also recommends that the Rule Administrator investigate why BillingCenter did not assign an item of that type, so that you can correct any errors in the rules.

IMPORTANT Do not attempt to make direct assignments to user `defaultowner`.

Assignment Methods that Do Not Use the Assignment Engine

If you call one of the assignment methods directly, from outside the Assignment rules, then the method simply returns `true` (if it has found a valid assignment) or `false` otherwise. In this case, it is your responsibility to determine what to do if no assignment is found. This can be assigning the item to the group supervisor, or invoking another assignment method, for example.

Determining Success or Failure

Guidewire recommends that you always determine if the assignment actually succeeded.

- In general, if you call an assignment method directly and it was successful, then you need do nothing further. However, you need to take care if you call an assignment method that simply assigns the item to a group (one of the `assignGroup` methods, for example). In this case, it might be necessary to call another assignment method to assign the item to an actual user.
- If you use the Assignment engine within the context of the Assignment rules, you can exit the rule (not consider any further rules), or perform other actions before exiting.

Logging Assignment Activity

Guidewire recommends also that you log the action anytime that you use one of the assignment methods. The following example—called from within the context of the Assignment rules—illustrates how to log assignment activity.

```
if (Activity.CurrentAssignment.assignUserByRoundRobin( false,
        Activity.CurrentAssignment.AssignedGroup) ) {
  Activity.CurrentAssignment.confirmManually( User( 2 /* Super User */ ) )
  gw.api.util.Logger.logDebug( "Assigned to: " + Activity.AssignedUser.DisplayName )
  actions.exit()
}
```

Notice that if you put the exit action inside the `if{...}` block, the Assignment engine continues to the next rule. (That is, if the current one is not successful in choosing a group or person to which to assign the item.) It is extremely important that you plan an effective exit from within a rule. Otherwise, it is possible for the Assignment engine to make further unanticipated and unwanted assignments. *You need only use the* `actions.exit` *method if performing assignment from within the Assignment Rules.*

Assignment Cascading

For certain top-level assignable entities, BillingCenter needs to re-assign some subobjects any time that it assigns a top-level entity. For example, if you reassigns a troubleticket, BillingCenter reassigns all open activities connected to that troubleticket. This includes all activities currently assigned to the same group and user as the claim.

For certain top-level assignable entities, it is possible that some subobjects also need to be re-assigned any time that BillingCenter assigns the top-level entity. For example, if you reassign a trouble ticket, BillingCenter reassigns all open activities connected to that trouble ticket currently assigned to the same group and user as the trouble ticket.

This cascading behavior is Guidewire application-specific. You cannot configure it.

- If you call the assignment methods through the Assignment engine, the Assignment engine cascades the assignments at the end of the process, as the engine exits with a completed assignment.
- If you call the assignment methods directly, then the assignment methods perform cascading of the assignment immediately as each method exits. For this reason, if you perform assignment outside of the Assignment rules, Guidewire strongly recommends that your Gosu code first determine the appropriate assignee (or set of assignees). It can then call a single assignment method on the assignee (or set of assignees). Do not rely on the recursive structure assumed by the assignment rules to perform assignment outside of the assignment rules.

Assignment Events

Any time that the assignment status changes on an assignment, BillingCenter fires an assignment event. There are three possible events that can trigger an assignment change event:

- `AssignmentAdded`
- `AssignmentChanged`
- `AssignmentRemoved`

The following list describes these events.

Old status	New status	Event fired	Code
Unassigned	Unassigned	None	None
Unassigned	Assigned	`AssignmentAdded`	`Assignable.ASSIGNMENTADDED_EVENT`
Assigned	Assigned	`AssignmentChanged`—if a field changes, for example, the assigned user, group, or date	`Assignable.ASSIGNMENTCHANGED_EVENT`
Assigned	Unassigned	`AssignmentRemoved`	`Assignable.ASSIGNMENTREMOVED_EVENT`

Assignment Method Reference

IMPORTANT Guidewire now deprecates assignment method signatures that do not have a `Group` parameter. Studio indicates this status by marking through the method signature in Gosu code and Studio code completion does not display deprecated methods. You can, however, see them in listed in the full list of assignment methods (with Studio again indicating their deprecated status). Do not use a deprecated assignment method—one without the `Group` parameter—outside of the Assignment rules. In general, Guidewire recommends that you do not use deprecated methods at all. If your existing Gosu code contains deprecated methods, Guidewire recommends that you rework your code to use non-deprecated methods.

Guidewire divides the assignment methods into the following general categories:

- Assignment by Assignment Engine
- Group Assignment (within the Assignment Rules)
- Queue Assignment

- Immediate AssignmentRound-Robin Assignment
- Dynamic Assignment

For the latest information and description on assignment methods, consult the *Gosu API Reference*.

Assignment by Assignment Engine

This type of assignment invokes the Assignment engine. Do not call the following from within the Assignment rules themselves.

autoAssign

```
public boolean autoAssign()
```

IMPORTANT Do not call this assignment method in the context of the Assignment rules. The method invokes the Assignment engine. Calling it within the context of the Assignment rules can potentially create an infinite loop. If you attempt to do so, BillingCenter ignores the method call and outputs an error message.

This method invokes the Assignment engine to assign the entity that called it. Call this method outside of the Assignment rules if you want to invoke the Assignment engine to carry out assignment. For example, you can call this method as part of entity creation to perform the initial assignment.

```
var trb : TroubleTicket
trb.autoAssign()
```

Group Assignment (within the Assignment Rules)

Methods that only assign a group are *most useful* from within the Global Assignment rule set (which is only responsible for assigning a group). In all other contexts, you need to assign both a user and a group. Therefore, it makes more sense to use one of the other types of assignment methods which perform both assignments. These methods include the following:

- `assignGroupByRoundRobin`

It is important to understand that these methods do not assign the assignable item to a user. They simply pick a group to use during the rest of the assignment process. To complete the assignment, you must use one of the other assignment methods to assign the item to a user.

assignGroupByRoundRobin

```
boolean assignGroupByRoundRobin(groupType, includeSubGroups, group)
```

This method assigns an entity to a group (or one of its subgroups) using round-robin assignment. The method restricts the set of available groups to those matching the specified group type. Use this method to distribute work (assignable items) among the different groups equitably.

For example, suppose you have several groups of people who specialize in handling complex issues. For some high-complexity items, you might want to rotate among the groups on an equitable basis. The passed-in `groupType` (if non-null) ensures that the method only considers for round-robin assignment those groups that have the matching value for their `groupType` fields. Thus, if there are three groups, all with the same group type, this method performs round-robin assignment between the three groups.

The `assignGroupByRoundRobin` method ignores any group for which the `Group.LoadFactor` value is zero (0).

The following example assigns a troubleticket to one of the “special handling” groups within the currently assigned group.

```
TroubleTicket.assignGroupByRoundRobin(  "special handling", true, Activity.AssignedGroup )
```

Queue Assignment

BillingCenter does not currently support the use of assignment queues.

Immediate Assignment

The following methods perform “immediate” or direct assignment to the specified user or group.

- `assign`
- `assignUserAndDefaultGroupassignToCreator`

assign

```
boolean assign(group, user)
```

This method assigns the assignable entity to the specific group and user. This is perhaps the simplest of possible assignment operations. Use this method if you want a specific known user and group to own the entity in question.

In the following example, the assignment method assigns a trouble ticket to the owner of an already assigned trouble ticket.

```
var TroubleTickets= find (trbtkt in TroubleTicket where trbtkt.TroubleTicketNumber contains "100")

for (trb in TroubleTickets) {
  if (trb != TroubleTicket
          and trb.AssignedGroup == Group( "default_data:1" /* Default Root Group */ )
          and trb.AssignedUser != null) {
    if ( TroubleTicket.CurrentAssignment.assign( trb.AssignedGroup, trb.AssignedUser ) ) {
      gw.api.util.Logger.logDebug( "##### This is the Default Group TroubleTickt Assignment rule "
              + gw.api.util.StringUtil.substring(actions.getRule().DisplayName,0, 8))
      gw.api.util.Logger.logDebug( "Assigned User is " + trb.AssignedUser)
      actions.exit()
    }
  }
}
```

assignUserAndDefaultGroup

```
boolean assignUserAndDefaultGroup(user)
```

This method assigns the assignable entity to the specified user, selecting a default group. The default group is generally the first group in the set of groups to which the user belongs. In general, use this method only if a user belongs to a single group, or if the assigned group really does not matter.

It is possible that the assigned group can affect visibility and permissions. Therefore, Guidewire recommends that use this method advisedly. For example, you might want to use this method only under the following circumstances:

- The users belong to only a single group.
- The assigned group has no security implications.

The following example, assigns an Activity to the current user and does not need to specify a group.

```
Activity.CurrentAssignment.assignUserAndDefaultGroup(User.util.CurrentUser)
```

assignToCreator

```
boolean assignToCreator(sourceEntity)
```

This method assigns the assignable entity to the user who created the supplied `sourceEntity` parameter. The following example assigns an activity to the creator of a trouble ticket.

```
Activity.CurrentAssignment.assignToCreator( TroubleTicket )
```

Round-Robin Assignment

The round-robin algorithm rotates through a set of users, assigning work to each in sequence. See “Secondary (Role-based) Assignment” on page 52 for a discussion on secondary assignment and round-robin states.

assignUserByRoundRobin

```
boolean assignUserByRoundRobin(includeSubGroups, currentGroup)
```

This method uses the round-robin user selector to choose the next user from the current group or group tree to receive the assignable. If the `includeSubGroups` parameter is `true`, the selector performs round-robin assignment not only on the current group, but also through all its subgroups. This is useful, for example, if BillingCenter currently assigns set of accounts to a particular group and you want all of the users within that group to share the workload.

The following example assigns an activity to the next user in a set of users in a group.

```
Activity.CurrentAssignment.assignUserByRoundRobin( false, Activity.AssignedGroup )
```

Dynamic Assignment

Dynamic assignment provides a generic hook for you to implement your own assignment logic, which you can use to perform automated assignment under more complex conditions. For example, you can use dynamic assignment to implement your own version of load balancing assignment.

There are two dynamic methods available, one for users and the other for groups. Both the user- and group-assignment methods are exactly parallel, with the only difference being in the names of the various methods and interfaces.

```
public boolean assignGroupDynamically(dynamicGroupAssignmentStrategy)
public boolean assignUserDynamically(dynamicUserAssignmentStrategy)
```

These methods take a single argument. Make this argument a class that implements one of the following interfaces:

```
DynamicUserAssignmentStrategy
DynamicGroupAssignmentStrategy
```

Interface Methods and Assignment Flow

The `DynamicUserAssignmentStrategy` interface defines the following methods. (The Group version is equivalent.)

```
public Set getCandidateUsers(assignable, group, includeSubGroups)
public Set getLocksForAssignable( assignable, candidateUsers)
public GroupUser findUserToAssign( assignable, candidateGroups, locks)
boolean rollbackAssignment(assignable, assignedEntity)
Object getAssignmentToken(assignable)
```

The first three methods are the major methods on the interface. Your implementation of these interface methods must have the following assignment flow:

1. Call `DynamicUserAssignmentStrategy.getCandidateUsers`, which returns a set of assignable candidates.
2. Call `DynamicUserAssignmentStrategy.getLocksForAssignable`, passing in the set of candidates. It returns a set of entities for which you must lock the rows in the database.
3. Open a new database transaction.
4. For each entity in the set of locks, lock that row in the transaction.
5. Call `DynamicUserAssignmentStrategy.findUserToAssign`, passing in the two sets generated in step 1 and step 2 previously. It returns a `GroupUser` entity representing the user and group to which you need to assign the entity.
6. Commit the transaction, which instructs BillingCenter to update and unlock the locked entities.

   Dynamic assignment is not complete after these steps. The interface methods allow for the failure of the commit operation by adding one last final step.
7. If the commit fails, roll back all changes made to the user information, if possible. If this is not possible, save the user name and reassign that user to the assignable item as you save the item at a later time.

Implementing the Interface Methods

Any class that implements the `DynamicUserAssignmentStrategy` interface (or the Group version) must provide implementations of the following methods:

- `getCandidateUsers`
- `getLocksForAssignable`
- `findUserToAssign`
- `rollbackAssignment`
- `getAssignmentToken`

getCandidateUsers

Your implementation of the `getCandidateUsers` method must return the set of users to consider for assignment. (As elsewhere, the `Group` parameter establishes the root group to use to find the users under consideration. The Boolean `includeSubGroups` parameter indicates whether to include users belonging to descendant groups, or only those that are members of the parent group.)

getLocksForAssignable

The `getLocksForAssignable` method takes the set of users returned by `getCandidateUsers` and returns a set of entities to lock. By locked, Guidewire means that the current machine grabs the database rows corresponding to those entities (which must be persistent entities). Any other machine that needs to access these rows must wait until the assignment process finishes. Round-robin and dynamic assignment require this sort of locking. This ensures that multiple machines do not perform simultaneous assignments and assign multiple activities (for example) to the same person, instead of progressing through the set of candidates.

findUserToAssign

Your implementation of the `findUserToAssign` method must perform the actual assignment work, using the two sets of entities returned by the previous two methods. (That is, it takes a set of users and the set of entities for which you need to lock the database rows and performs that actual assignment.) The method must do the following:

- It makes any necessary state modifications (such as updating counters, and similar operations).
- It returns the `GroupUser` entity representing the selected User and Group.

Make any modifications to items such as load count, for example, to entities in the bundle of the assignable. This ensures that BillingCenter commits the modifications at the same time as it commits the assignment change.

rollbackAssignment

Guidewire provides the final two API methods to deal with situations in which, after the assignment flow, some problem in the bundle commit blocks the assignment. This can happen, for example, if a validation rule causes a database rollback. However, at this point, BillingCenter has already updated the locked objects and committed them to the database (as in step 6 in the assignment flow).

If the bundle commit does not succeed, BillingCenter calls the `rollbackAssignment` method automatically. Construct your implementation of this method to return `true` if it succeeds in rolling back the state numbers, and `false` otherwise. In the event that the assignment does not get saved, you have the opportunity in your implementation of this method to re-adjust the load numbers.

getAssignmentToken

If the `rollbackAssignment` method returns `false`, then BillingCenter calls the `getAssignmentToken` method. Your implementation of this method must return some object that you can use to preserve the results of the assignment operation. The basic idea is that in the event that BillingCenter is not able to commit an assignment, your logic does one of the following:

- PolicyCenter rolls back any database changes that you have already made.

• PolicyCenter preserves the assignment in the event that you invoke the assignment logic again.

Sample DynamicUserAssignmentStrategy Implementation

The following code illustrates an implementation of the `LeastRecentlyModifiedAssignmentStrategy` class. This is a very simple application of the necessary concepts needed to create a working implementation. The class performs a very simple user selection, simply looking for the user that has gone the longest without modification.

Since the selection algorithm needs to inspect the user data to perform the assignment, BillingCenter returns the candidate users themselves as the set of entities to lock. This ensures that the assignment code can work without interference from other machines.

```
package gw.api.assignment.examples

uses gw.api.assignment.DynamicUserAssignmentStrategy
uses java.util.Set
uses java.util.HashSet

@Export
class LeastRecentlyModifiedAssignmentStrategy implements DynamicUserAssignmentStrategy {

  construct() { }

  override function getCandidateUsers(assignable:Assignable, group:Group, includeSubGroups:boolean ) :
          Set {
     var users = (group.Users as Set<GroupUser>).map( \ groupUser -> groupUser.User )
     var result = new HashSet()
     result.addAll( users )
     return result
  }
  override function findUserToAssign(assignable:Assignable, candidates:Set, locks:Set) : GroupUser {
    var users : Set<User> = candidates as Set<User>
    var oldestModifiedUser : User = users.iterator().next()
    for (nextUser in users) {
      if (nextUser.UpdateTime < oldestModifiedUser.UpdateTime) {
        oldestModifiedUser = nextUser
      }
    }

    return oldestModifiedUser.GroupUsers[0]
  }

  override function getLocksForAssignable(assignable:Assignable, candidates:Set) : Set {
    return candidates
  }

  //Must return a unique token
  override function getAssignmentToken(assignable:Assignable) : Object {
    return “LeastRecentlyModifiedAssignmentStrategy_” + assignable
  }

  override function rollbackAssignment(assignable:Assignable, assignedEntity:Object) : boolean {
    return false
  }
}
```

Using Assignment Methods in Assignment Pop-ups

In Guidewire BillingCenter, you typically reassign an existing entity through an **Assignment** popup screen. This screen is usually two-part:

• You use the upper part to select from a pre-populated list of likely assignees, including (for example) the activity owner. It also includes an option to perform rule-based assignment.

• You use the lower part of the popup to search for a specific assignee.

You can run assignment methods directly, from outside of the Assignment engine. Therefore, it is possible to modify the upper part of this popup to call a specific assignment method without invoking the Assignment engine. This requires both new Gosu code and some PCF configuration.

One possibility is the following:

1. Create a new Gosu class, implementing the `gw.api.assignment.Assignee` interface. Use this class to perform your business logic to assign the passed-in `Assignable` object. The following is an example of this.

```
package gw.api

class ExtensionAssignee implements gw.api.assignment.Assignee {

  construct()

  override function assignToThis(assignableBean : com.guidewire.pl.domain.assignment.Assignable) {
    var users = gw.api.database.Query.make(User).compare("PublicID", Equals, userID)
    var usr = users.select().AtMostOneRow
    assignableBean.autoAssign( usr.GroupUsers[0].Group, usr )
  }

  function toString() : String {
    return "Bakeriffic Assignment"
  }
}
```

2. Modify the PCF files to include this new `Assignee` object, removing others as necessary. One option, for example, would be to modify the assignment popup to add a new method in the Code section, such as the following. You can then reference it in the `valueRange` section of the assignment widget in place of the call to `SuggestedAssignees`.

```
function getSuggestedAssignees() : gw.api.assignment.Assignee[] {
  var assignees = TroubleTicket.SuggestedAssignees
  assignees[0] = new gw.api.ExtensionAssignee()
  return assignees
}
```

chapter 9

Document Creation

This topic describes synchronous and asynchronous document creation in Guidewire BillingCenter. It briefly describes the integration points between a document management system and Guidewire BillingCenter. (For detailed integration information, see “Document Management” on page 217 in the *Integration Guide*.) It also covers some of the more important document management APIs and document production classes.

This topic includes:

Synchronous and Asynchronous Document Production

Guidewire BillingCenter supports the following types of document creation:

- *Synchronous* document creation completes immediately after it you initiate it.
- *Asynchronous* document creation completes at a future time after you initiate it.

BillingCenter uses an `IDocumentProduction` plugin interface to manage document creation. Guidewire also provides a Gosu helper class with a number of public `createDocumentXX` methods to facilitate working with document creation.

In the context of `IDocumentProduction` plugin, *synchronous* versus *asynchronous* refers to the perspective of BillingCenter. In other words, after a `createDocumentXX` method call returns, did the integrated document production application create the document already (synchronous creation)? Or, is the `IDocumentProduction` implementation responsible for future creation and storage of the document (asynchronous creation)?

The following table restates the differences between synchronous and asynchronous document creation:

Type	Document contents
Synchronous	Generated immediately and returned to the calling method for further processing. In this scenario, the caller assumes responsibility for persisting the document to the Document Management system (if desired).
Asynchronous	Possibly not generated for some time, or possibly require extra workflow processing or manual intervention. The document creation system does not return the contents of the document, although it can return a URL or other information allowing for the checking of state. The `IDocumentProduction` implementation is responsible for adding the document to the Document Management system and notifying BillingCenter upon successful creation of the document.

See also

- "Summary of All BillingCenter Plugins" on page 153 in the *Integration Guide*
- "Document Management" on page 217 in the *Integration Guide*

Integrating Document Functionality with BillingCenter

Note: For information on how to integrate document-related functionality with Guidewire BillingCenter, see the information on plugins in "Document Management" on page 217 in the *Integration Guide*.

Implementing document-related functionality in a Guidewire application often requires integration with two types of external applications:

- Document Management Systems (DMS), which store document contents and metadata
- Document Production Systems (DPS), which create new documents

This integration involves the following main plugin interfaces, along with several minor ones. The following table summarizes information about the main plugin interfaces.

Interface	Used for...
`IDocumentContentSource`	Storage and retrieval of document contents only, with no metadata.
`IDocumentMetadataSource`	Storage and retrieval of document metadata only, with no contents. This plugin is disabled in the base configuration, and document metadata is stored with the `Document` object. Although many (if not most) DMS applications store both document contents and metadata about the documents, Guidewire provides two separate plugin interfaces. This separation is due to the different performance characteristics of the two kinds of data: • Document metadata is generally a collection of relatively short strings. Thus, you can usually implement remote transmission using SOAP interfaces. • Document contents are generally a large (up to many megabytes) chunk of data, often binary data, which cannot be easily or cheaply moved around between applications.
`IDocumentProduction`	Document creation. See "The DocumentProduction Class" on page 69.

Interface	Used for...
`IDocumentTemplateDescriptor`	Describe the templates used to create documents. See "The IDocumentTemplateDescriptor Interface" on page 65 for details of this interface.
`IDocumentTemplateSource`	Retrieval of document templates, which are a part of document creation. In the base configuration, BillingCenter provides a default implementation of the `IDocumentTemplateSource` plugin that retrieves the document templates from XML files stored on the server file system. This default implementation reads files from the application `configuration` module: `*BillingCenter*/modules/configuration/config/resources/doctemplates` If you desire to use a different path, then you need to supply the following parameters to specify an alternate location: • `templates.path` • `descriptors.path` If you choose to do this, Guidewire strongly recommends that you use absolute path names. Do not use relative path names as using a path relative to the document plugin directory can cause checksum problems at server start. The default implementation also checks the files system for template files if a user performs a search or retrieval operation. In general practice, however, Guidewire expects you to implement a storage solution that meets your business needs, which can include integration with a Document Management System.

The following table summarizes information about the minor plugin interfaces (those used less frequently).

Interface	Used for...
`IDocumentTemplateSerializer`	Customizing the reading and writing of `IDocumentTemplateDescriptor` objects. Use this sparingly. The default implementation of `IDocumentTemplateSerializer` uses an XML format that closely matches the fields in the `DocumentTemplateDescriptor` interface. This is intentional. The purpose of `IDocumentTemplateSerializer` is to serialize template descriptors and provide the ability to define the templates within simple XML files. This XML format is suitable for typical implementations.
`IPDFMergeHandler`	Creation of PDF documents. You use this mainly to set parameters on the default implementation.

See also

- "Summary of All BillingCenter Plugins" on page 153 in the *Integration Guide*
- "Document Management" on page 217 in the *Integration Guide*

The IDocumentTemplateDescriptor Interface

A *document template descriptor* works in conjunction with an a document template, meaning a Microsoft Word MailMerge template, a PDF form, or a similar item. The descriptor file tells BillingCenter how to populate the fields on the template. Within BillingCenter, the `IDocumentTemplateDescriptor` interface defines the API that any object that represents a document template descriptor must implement.

A document template descriptor contains the following different kinds of information:

Category	Contains
Template Metadata	Metadata about the template itself (for example, the template ID, name, and similar items)
Document Metadata	Metadata defaults to apply to any documents created from the template (for example, the document status)

Category	Contains
Context Objects	Set of values that are referenced by the values to be inserted into the document template, known as *Context Objects*. This includes both default values and a set of legal alternative values for use in the document creation user interface.
Form Fields	Set of field names and values to insert into the document template, including some formatting information.
Document Locale	Locale in which to generate the document.

BillingCenter stores all classes used by document plugins in the **Classes** → **gw** → **document** package. In the base configuration, this consists of the `DocumentProduction` class. This class relies heavily on `IDocumentTemplateDescriptor` objects in the creation of document objects.

The IDocumentTemplateDescriptor API

The `IDocumentTemplateDescriptor` API consists entirely of getter methods, with the addition of a single setter method (for `DateModified`). As a result, the following sections list the getter names and the return type information, broken into the following categories:

- Template Metadata
- Document Metadata
- Context Objects
- Form Fields
- Document Locale

Note: Many—but not all—of the `IDocumentTemplateDescriptor` getter methods exist as properties on `IDocumentTemplateDescriptor` object as well.

Template Metadata

The following list describes the getter methods associated with *template* metadata that the `IDocumentTemplateDescriptor` API manages.

getXXX method	Return type	Returns
`getDateEffective` `getDateExpiration`	`Date`	Effective and expiration dates for the template. If a user searches for a template, BillingCenter displays only those templates for which the specified date falls between the effective and expiration dates. However, it is possible to use Gosu rules to create an *expired* template as Gosu-based document creation uses templates only. Do not attempt to use this as a mechanism for establishing different versions of templates with the same ID. All template IDs must be unique.
`getDateModified` `setDateModified`	`Date`	Date on which the template was last modified. In the base configuration, BillingCenter sets this date from the information on the XML descriptor file itself. However, as both getter and setter methods exist for this property, it is possible to set this date through the `IDocumentTemplateSource` implementation.
`getDescription`	`String`	Human-readable description of the template or the document it creates.

getXXX method	Return type	Returns
`getDocumentProductionType`	`String`	If present, you can use this property to control which `IDocumentProduction` implementation to use to create a new document from the template. See “Document Management” on page 217 in the *Integration Guide* for information on implementing and configuring the `IDocumentProduction` plugin.
`getIdentifier`	`String`	Additional human-readable identifier for the template. This often corresponds to a well-known domain-specific document code. It can indicate, for example, to which state-mandated form this template corresponds.
`getKeywords`	`String`	Set of keywords that you can use in a search for the template.
`getMetadatePropertNames` `getMetadataPropertyValues`	`String[]`	Method `getMetadataPropertyNames` returns the set of extra metadata properties that exist in the document template definition. You can use these properties in conjunction with the `getMetadataPropertyValues` method as a flexible extension mechanism. For example, you can add arbitrary new fields to document template descriptors. BillingCenter then passes these fields onto the internal entities that it uses to display document templates in the interface. If the extra property names correspond to properties on the `Document` entity, BillingCenter passes the values along to documents that it creates from the template.
`getMimeType`	`String`	Type of document that this document template creates.In the base configuration. You can also use this to determine which `IDocumentProduction` implementation to use to create documents from this template. For information on implementing and configuring the IDocumentProduction plugin, see “Document Management” on page 217 in the *Integration Guide*.
`getName`	`String`	A human-readable name for the template.
`getPassword`	`String`	If present, this property holds the password that BillingCenter requires for the user to be able to create a document from this template. Not all document formats support this requirement. For example, Gosu templates do not support this functionality.
`getRequiredPermission`	`String`	Value corresponding to a type code from the `SystemPermissionType` typelist that BillingCenter requires for a user to use this template. BillingCenter does not display templates for which the user does not have the appropriate permission.
`getScope`	`String`	Contexts in which you can use this template. Possible values are: • **gosu** – Indicates that you can only create this document template from Gosu code. These templates do not appear in the BillingCenter interface • **ui** – Indicates you must only use this template from the BillingCenter interface as it usually requires some kind of human interaction • **all** – Indicates that you can use this document template from any context
`getTemplateId`	`String`	Unique ID of the template. For most template types, this must be the same as the file name of the document template file itself (for example, `ReservationRights.doc`). One exception is InetSoft-based report templates, for which the `templateId` is the same as the name of the report to use to generate the document.

Document Metadata

The following list describes the getter methods associated with *document* metadata that the `IDocumentTemplateDescriptor` API manages.

getXXX method	Return type	Returns
`getTemplateType`	`String`	Value corresponding to a type code from the `DocumentType` typelist. If you create a document using this template, BillingCenter sets the `type` field to this value. The XML descriptor file lists this property (and only this property) by a different name. The XML file lists this as `type` rather than as `templateType`.
`getDefaultSecurityType`	`String`	Value corresponding to a type code from the `DocumentSecurityType` typelist. If you create a document using this template, BillingCenter sets the `securityType` field to this value.

Context Objects

The following list describes the getter methods associated with the *context objects* that the `IDocumentTemplateDescriptor` API manages.

getXXX method	Return type	Returns
`getContextObjectNames`	`String[]`	Set of context object names that the document template defines. See "The DocumentProduction Class" on page 69 for an example of how template descriptor files display context objects.
`getContextObjectType`	`String`	Type of the specified context object. Possible values include the following: • `String` • `Text` • `Bean` – this is means any entity type • Name of any entity type, for example `Account`.
`getContextObjectAllowsNull`	`Boolean`	`true` if `null` is a legal value, `false` otherwise.
`getContextObjectDisplayName`	`String`	Human-readable name for the given context object. BillingCenter displays this name in the document creation interface.
`getContextObjectDefaultValueExpression`	`String`	Gosu expression that evaluates to the desired default value for the context object. BillingCenter uses this expression to set the default context object for the document creation interface, or as the context object value if it creates a document automatically.
`getContextObjectPossibleValuesExpression`	`String`	Gosu expression that evaluates to the desired set of legal values for the given context object. BillingCenter uses these values to display a list of options for the user in the document creation interface.

Form Fields

The following list describes the getter methods associated with the *form fields* that the `IDocumentTemplateDescriptor` API manages.

getXXX method	Return type	Returns
`getFormFieldDisplayValue`	`String`	Value to insert into the completed document, given the field `name` and `value`. (The `value`, for example, can be the result of evaluating the expression returned from `getFormFieldValueExpression`). If desired, you can then implement some processing on the returned string, for example, substituting NA (not applicable) for `null` or formatting the dates correctly.
`FormFieldNames`	`String[]`	Set of form fields that the document template defines. See "The DocumentProduction Class" on page 69 for an example of how template descriptor files show form fields.
`getFormFieldValueExpression`	`String`	Gosu expression that evaluates to the desired value for the form field. The Gosu expression is usually written in terms of one or more context objects. However, you can use any legal Gosu expression.

Document Locale

The following list describes the getter methods associated with the document *locale* that the `IDocumentTemplateDescriptor` API manages.

getXXX method	Return type	Returns
`getLocale`	`String`	Locale in which to create the document from this template descriptor. A return value of `null` indicates an unknown language. In most cases, the method returns the default language for the application.

The DocumentProduction Class

Guidewire Studio provides a helper `DocumentProduction` Gosu class (**configuration** → **gsrc** → **gw** → **document** → **DocumentProduction**) with the following public methods to facilitate working with document creation:

Asynchronous methods	Synchronous methods
• `asynchronousDocumentCreationSupported`	• `synchronousDocumentCreationSupported`
• `createDocumentAsynchronously`	• `createAndStoreDocumentSynchronously`
	• `createDocumentSynchronously`

For the most part, these methods require the same passed-in parameters.

Parameter	Type	Description
`template`	`IDocumentTemplateDescriptor`	Template descriptor to use in creating the document. The file name must be the same as the file name of the document template file itself. **IMPORTANT** If you do not supply a locale in the `IDocumentTemplateDescriptor`, then BillingCenter uses the default locale.
`parameters`	`Map`	Set of objects—keyed by name—to supply to the template generation process to create the document. The `parameters` Map must not be `null` and must contain, at a minimum, a name and value pair for each context object required by the given document template. It can also contain other information, which the document production system can use to perform additional operations.
`document`	`Document`	`Document` entity corresponding to the newly generated content. The passed-in `Document` entity can contain some fields that have already been set, and the document production system can set other fields in the course of performing the document creation.
`fieldValues`	`Map`	Set of values—keyed by field name—to set on the `Document` entity created at the end of the asynchronous creation process.

How to Determine the Supported Document Creation Type

BillingCenter provides the following methods that you can use to determine if the `IDocumentTemplate` plugin supports the required document creation mode:

- `asynchronousDocumentCreationSupported`
- `synchronousDocumentCreationSupported`

The asynchronousDocumentCreationSupported Method

Call the `asynchronousDocumentCreationSupported` method to determine whether the `IDocumentProduction` implementation supports asynchronous creation for the specified template. This method returns `true` if it is possible, and `false` otherwise. It also returns `false` if a template cannot be found with the specified name or if you pass in `null` for the template name.

This method has the following signature:

```
public static function asynchronousDocumentCreationSupported(template :
        IDocumentTemplateDescriptor) : boolean
```

The synchronousDocumentCreationSupported Method

Call the `synchronousDocumentCreationSupported` method to determine whether the `IDocumentProduction` implementation supports synchronous creation for the specified template. This method returns `true` if it is possible, and `false` otherwise. It also returns `false` if BillingCenter cannot find a template with the specified name or if you pass in `null` for the template name.

This method has the following signature:

```
public static function synchronousDocumentCreationSupported(template : IDocumentTemplateDescriptor) :
        boolean
```

Asynchronous Document Creation Methods

Guidewire provides the following methods for asynchronous document creation:

- `createDocumentAsynchronously`

createDocumentAsynchronously Method

Call the `createDocumentAsynchronously` method to create a new document asynchronously, based on the named template and the supplied parameters. The method returns immediately, but the actual document creation takes place over an extended period of time. If required, it is the responsibility of the external document production system to create a `Document` entity after document creation is complete.

The last method parameter (`fieldValues`) provides a set of mappings of field name to value to set in the metadata of the new `Document` after you create it.

This method has the following signature:

```
public static function createDocumentAsynchronously(template : IDocumentTemplateDescriptor,
        parameters : Map, fieldValues : Map) : String
```

Note: The `IDocumentProduction` implementation is responsible for persisting both the document contents and creating a new `Document` entity if necessary.

Synchronous Document Creation Methods

Guidewire provides the following methods for synchronous document creation:

- `createAndStoreDocumentSynchronously`
- `createDocumentSynchronously`

createAndStoreDocumentSynchronously Method

Call the `createAndStoreDocumentSynchronously` method to create and store a document without any further user interaction. This method creates a document synchronously and passes it to the `IDocumentContentSource` plugin for persistence.

This method has the following signature:

```
public static function createAndStoreDocumentSynchronously(template : IDocumentTemplateDescriptor,
        parameters : Map, document : Document)
```

The createDocumentSynchronously Method

Call the `createDocumentSynchronously` method to create a new document synchronously, based on the named template and the supplied parameters. BillingCenter adds the generated document to the configured `IDocumentContentSource` implementation for storage in the Document Management System (DMS).

Use this method if you want the generated content to be visible in the BillingCenter interface and you do not necessarily want to persists the newly generated content. For example, you can use this method to generate content simply for viewing in BillingCenter or for printing.

This method returns a `DocumentContentsInfo` object related to the template type, which contains the results of the document creation. The following are some examples of possible return objects:

- For Gosu-based templates, the returned object consists of the actual contents of the generated document.
- For Microsoft Mail Merge documents (chiefly MS Word and Excel documents), the returned object consists of a JavaScript file that the client machines runs to produce the document content.
- For a server-generated PDF document, the returned object consists of the actual contents of the generated document.

This method has the following signature:

```
public static function createDocumentSynchronously(
        templateDescriptor: IDocumentTemplateDescriptor,
        parameters : Map) : DocumentContentsInfo,
        document : Document) : DocumentContentsInfo
```

Document Templates

See the following topics for additional information on document templates.

Document Templates

For general information on document templates, how to create them, and how to use them, see:

- "Gosu Templates" on page 359 in the *Gosu Reference Guide*
- "Document Management" on page 217 in the *Integration Guide*
- "Data Extraction Integration" on page 441 in the *Integration Guide*

Document Template Localization

For information on localizing document templates, see:

- "Localizing Templates" on page 75 in the *Globalization Guide*
- "Document Localization Support" on page 80 in the *Globalization Guide*

IMPORTANT The base configuration *Sample Acrobat* document (`SampleAcrobat.pdf`) uses Helvetica font. If you intend to create a document that uses Unicode characters (for example, one that uses an East Asian language), then the document template must support a Unicode font. Otherwise, the document does not display Unicode characters correctly.

Document Creation Examples

Note: See also "Document Management" on page 217 in the *Integration Guide* for more information on working with documents in BillingCenter.

IMPORTANT The following examples refer to Guidewire ClaimCenter objects and entities. However, these examples illustrate concepts and principals that are applicable to document creation in all Guidewire applications.

The following examples use an HTML template to construct a document that is in reference to an accident that took place in Great Britain. (FSA is the regulator of all providers of financial services in the United Kingdom, thus, the references to FSA regulations in the document.) After creation, the document looks similar to the following:

To:
Peter Smith
535 Main Street
Sutter, CA 12345

Dear Peter Smith,

This letter is being sent in accordance with FSA requirements ICOB 7.5.1, 7.5.4, and 7.5.5.
Your claim, number 54321, was filed on November 25, 2005. We are currently investigating this claim and will contact you and other involved parties shortly.

You can expect an update from your claims representative, Mary Putnam, no later than 30 days after the listed claim filing date.

If you need more information, please contact me at +44 555 123 1234.

Sincerely,
John Sandler
XYZ Insurance

To construct this document, you need the following files:

- `FSAClaimAcknowledgement.gosu.htm`
- `FSAClaimAcknowledgement.gosu.htm.descriptor`

FSAClaimAcknowledgement.gosu.htm

File `FSAClaimAcknowledgement.gosu.htm` sets the text to use in the document. It contains template variables (`%...%`) that ClaimCenter replaces with values supplied by `FSAClaimAcknowledgement.gosu.htm.descriptor` during document production.

```
<html xmlns="http://www.w3.org/TR/REC-html40">

  <head>
    <meta http-equiv=Content-Type content="text/html; charset=windows-1252">
    <title>FSA Claim Acknowledgement Letter</title>
    <style></style>
  </head>

  <body>
    <b>To:</b><br>
    <%=InsuredName%><br>
    <%=InsuredAddress1%><br>
    <%=InsuredCity%>, <%=InsuredState%> <%=InsuredPostalCode%><br>
    <br>Dear <%=InsuredName%>,<br>
    <br>This letter is being sent in accordance with FSA requirements ICOB 7.5.1, 7.5.4, and 7.5.5.<br>
    <br>Your claim, number <%=ClaimNumber%>, was filed on <%=ClaimReportedDate%>.<br>
    <br>We are currently investigating this claim and will contact you and other involved parties
      shortly.  You can expect an update from your claims representative, <%=AdjusterName%>,
      no later than 30 days after the listed claim filing date.<br>
    <br>If you need more information, please contact me at +44 555 123 1234.<br>
    <br>Sincerely,<br>
    <%=AdjusterName%><br>
    XYZ Insurance<br>
  </body>

</html>
```

FSAClaimAcknowledgement.gosu.htm.descriptor

The descriptor document serves a number of purposes:

- First, it serves to classify the created document.
- Secondly, it specifies the `ContextObject` objects that it needs to fill in the document template. These are manually set by the user or BillingCenter automatically sets these through business rules during document creation.
- It also provides a mapping between the names of templatized fields in the destination document and the context objects (`FormField` objects).

```
<?xml version="1.0" encoding="UTF-8"?>
<DocumentTemplateDescriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.guidewire.com/schema/claimcenter/document-template.xsd"

   id="FSAClaimAcknowledgement.gosu.htm"
   name="Claim Acknowledgement Letter"
   description="ICOB 7.5.1 acknowledgement letter."
   type="letter_sent"
   lob="GL"
   state="UK"
   mime-type="text/html"
   date-effective="Mar 15, 2004"
   date-expires="Mar 15, 2007"
   keywords="UK, acknowledgement">

  <ContextObject name="To" type="Contact">
    <DefaultObjectValue>Claim.Insured</DefaultObjectValue>
    <PossibleObjectValues>Claim.getRelatedContacts()</PossibleObjectValues>
  </ContextObject>

  <ContextObject name="From" type="Contact">
    <DefaultObjectValue>Claim.AssignedUser.Contact</DefaultObjectValue>
    <PossibleObjectValues>Claim.getRelatedUserContacts()</PossibleObjectValues>
  </ContextObject>

  <FormFieldGroup name="main">
    <DisplayValues>
      <DateFormat>MMM dd, yyyy</DateFormat>
    </DisplayValues>
    <FormField name="ClaimNumber">Claim.ClaimNumber</FormField>
    <FormField name="ClaimReportedDate">Claim.ReportedDate</FormField>
    <FormField name="InsuredName">To.DisplayName</FormField>
    <FormField name="InsuredPrefix">(To as Person).Prefix</FormField>
    <FormField name="InsuredLastName">(To as Person).LastName</FormField>
    <FormField name="InsuredAddress1">To.PrimaryAddress.AddressLine1</FormField>
    <FormField name="InsuredCity">To.PrimaryAddress.City</FormField>
    <FormField name="InsuredState">To.PrimaryAddress.State</FormField>
    <FormField name="InsuredPostalCode">To.PrimaryAddress.PostalCode</FormField>
    <FormField name="CurrentDate">gw.api.util.DateUtil.currentDate()</FormField>
    <FormField name="ClaimNoticeDate">Claim.LossDate</FormField>
    <FormField name="AdjusterName">From.DisplayName</FormField>
    <FormField name="AdjusterPhoneNumber">From.WorkPhone</FormField>
    <FormField name="InsuranceCompanyName">Claim.Policy.UnderwritingCo</FormField>
    <FormField name="InsuranceCompanyAddress">From.PrimaryAddress.AddressLine1</FormField>
    <FormField name="InsuranceCompanyCity">From.PrimaryAddress.City</FormField>
    <FormField name="InsuranceCompanyState">From.PrimaryAddress.State</FormField>
    <FormField name="InsuranceCompanyZip">From.PrimaryAddress.PostalCode</FormField>
  </FormFieldGroup>

</DocumentTemplateDescriptor>
```

Method createAndStoreDocumentSynchronously Example 1

In the following example, the `createAndStoreDocumentSynchronously` method takes following parameters to construct the previously shown document:

Parameter	Value	Description
`descriptor`	`FSAClaimAcknowledgement.gosu.htm.descriptor`	Template descriptor to use in constructing the document.
`parameters`	`contextObjects`	Relevant information related to the document contained in a `HashMap` object.
`document`	`Document`	`Document` entity to contain the newly constructed document

The following (ClaimCenter) Gosu code creates the document.

```
//Creates a map object and sets document information (ContextObjects) needed for content creation
var parameters = new java.util.HashMap()
parameters.put("Claim", claim)
parameters.put("To", claim.maincontact)
parameters.put("From", claim.AssignedUser.Contact)
parameters.put("CC", null)

//Creates a new Document entity and sets metadata directly on the document object
var document : Document = new Document(claim)
document.Claim = claim
document.Name = "Claim Acknowledgement"
document.Type = "letter_sent"
document.Status = "draft"

//Creates the document template descriptor
var plugin = gw.plugin.Plugins.get(gw.plugin.document.IDocumentTemplateSource)
var descriptor = plugin.getDocumentTemplate("FSAClaimAcknowledgement.gosu.htm",
       \ code -> gw.i18n.ILocale.EN_US)

gw.document.DocumentProduction.createAndStoreDocumentSynchronously(descriptor, parameters, document)
```

Method createAndStoreDocumentSynchronously Example 2

Gosu also provides the ability to test whether document creation is possible before attempting the operation. You can check for the ability to create a document synchronously or asynchronously by using one of the following methods.

- `synchronousDocumentCreationSupported`
- `asynchronousDocumentCreationSupported`

These methods take the name of a template descriptor as the lone argument and determine whether the `IDocumentProduction` plugin supports the specified template descriptor. Each method returns `true` if the plugin does support that template descriptor file and `false` otherwise. Each method also return `false` if it cannot find a template descriptor with the specified name.

The following sample (ClaimCenter) Gosu code illustrates the use of these methods:

```
//Creates a map object and sets document information (ContextObjects) needed for content creation
var parameters = new java.util.HashMap()
parameters.put("Claim", claim)
parameters.put("To", claim.maincontact)
parameters.put("From", claim.AssignedUser.Contact)
parameters.put("CC", null)

//Creates a new Document entity and sets metadata directly on the document object
var document : Document = new Document(claim)
document.Claim = claim
document.Name = "Claim Acknowledgement"
document.Type = "letter_sent"
document.Status = "draft"

//Creates the document template descriptor (of type IDocumentTemplateDescriptor)
var plugin = gw.plugin.Plugins.get(gw.plugin.document.IDocumentTemplateSource)
var descriptor = plugin.getDocumentTemplate("SampleAcrobat.pdf",
        \ code -> gw.i18n.ILocale.EN_US)

if (gw.document.DocumentProduction.synchronousDocumentCreationSupported(descriptor)) {
  gw.document.DocumentProduction.createAndStoreDocumentSynchronously(descriptor, parameters, document)
}
```

Document Creation in Guidewire BillingCenter

The BillingCenter `gw.billingcenter.IBillingCenterAPI` API provides a number of useful methods that you can use to manage documents:

- `IBillingCenterAPI.addDocumentToAccount`
- `IBillingCenterAPI.addDocumentToPolicyPeriod`
- `IBillingCenterAPI.addDocumentToProducer`

These methods have the following signatures:

```
addDocumentToAccount(document, accountID)
addDocumentToPolicyPeriod(document, policyPeriodID)
addDocumentToProducer(document, producerID)
```

They take the following method parameters:

Parameter	Type	Description
`document`	`Document`	The `Document` entity to add.
`accountID`	`String`	The public ID of an existing account to which the document is to be added.
`policyPeriodID`	`String`	The public ID of an existing policy period to which the document is to be added.
`producerID`	`String`	The public ID of an existing producer to which the document is to be added.

All of these methods:

- Add the passed-in Document object to an existing Account, PolicyPeriod, or Producer. The methods actually add the passed-in Document object to the bundle specified by the passed-in public ID.
- Return the public ID (PublicID) of the Document object.

Method createAndStoreDocumentSynchronously Example

```
var document: Document = new Document(account)
document.Name = "Created by a Rule" + java.lang.System.currentTimeMillis()
document.Type = DocumentType.TC_DISPUTE
document.Status = DocumentStatusType.TC_DRAFT

var parameters = new java.util.HashMap()
parameters.put("Account", account)
parameters.put("To", account.AccountName)
parameters.put("From", account.UpdateUser)
parameters.put("CC", null)

//Creates the document template descriptor
var plugin = gw.plugin.Plugins.get(gw.plugin.document.IDocumentTemplateSource)
var descriptor = plugin.getDocumentTemplate("CreateEmailSent.gosu.htm",
        \ code -> gw.i18n.ILocale.EN_US)

gw.document.DocumentProduction.createAndStoreDocumentSynchronously(descriptor, parameters, document)
```

Troubleshooting

The following issues with document creation can occur on occasion:

- IDocumentContentSource.addDocument Called with Null InputStream
- UnsupportedOperationException Exception
- Document Template Descriptor Upgrade Errors
- “Automation server cannot create object” Error
- “IDocumentProduction implementation must return document...” Error

IDocumentContentSource.addDocument Called with Null InputStream

In certain cases, BillingCenter calls IDocumentContentSource.addDocument with a null InputStream. This can occur as a result of the following scenario:

1. The user creates a Document entity with some content and clicks **Update** in the application interface.
2. The document fails validation for some reason. However, BillingCenter uploads the document contents to the configured IDocumentContentSource before validation is run. Therefore, BillingCenter has already called addDocument and has already read the InputStream.
3. The user fixes the problem and clicks **Update** again. At this point, any changes made to the document by the previous call to IDocumentContentSource.addDocument have been lost. So, BillingCenter calls addDocument again. This gives the IDocumentContentSource implementation a chance to set any fields (such as DocUID) that need to be set on the Document entity before BillingCenter stores it.

This scenario assumes that you can match up the document content passed in the first call to addDocument with the Document entity from the second call to addDocument. This is not true of every Document Management System. Therefore, you might need to add some caching code to your IDocumentContentSource implementation so that the connection can be preserved.

UnsupportedOperationException Exception

Occasionally, the following exception occurs:

```
java.lang.UnsupportedOperationException: Asynchronous client-side MailMerge-based generation
        is not supported!
```

This exception occurs because the sample `IDocumentProduction` implementation does not support server-side creation of Microsoft Word or Excel documents. Therefore, you cannot create documents asynchronously based on templates of these types.

Document Template Descriptor Upgrade Errors

The following error can occur during document template descriptor upgrade:

```
IOException encountered: Content is not allowed in prolog.
```

The cause of this error, generally, is that one or more descriptor files has some characters before the "`<?xml`" at the beginning of the file. That XML tag must be the very first thing in the file.

“Automation server cannot create object” Error

Occasionally, the following (or similar) error occurs during attempts to create a document from a template.

```
"Automation server cannot create object"
```

Creation of Microsoft Word and Excel documents from templates is done on the user's machine. Therefore, if the user's machine does not have Word or Excel, this kind of error occurs. You can, however, still create a document from a PDF or Gosu template instead.

“IDocumentProduction implementation must return document...” Error

You may see the following error while calling a `DocumentProduction.create*` method:

```
"The IDocumentProduction implementation must return document contents to be called from a rule"
```

This error message indicates a problem with the `DocumentContentsInfo.ResponseType` property on the return value of the document production plugin document creation code. The exact plugin method is the `IDocumentProduction` plugin method `createDocumentSynchronously`.

In the default configuration, the built-in implementation of this plugin expects the `responseType` field has the response type value `DocumentContentsInfo.DOCUMENT_CONTENTS`.

If you modify the document production plugin or any related code such that it returns another response type such as `URL`, BillingCenter displays this error. The built-in `IDocumentProduction` implementation expected an actual document contents as an input stream of bytes, not other response types such as `DOCUMENT_URL`.

However, you might want to support a document production implementation that supports the response type `DOCUMENT_URL`. If so, you may need to modify the built-in document production plugin implementation. In Studio, edit the built-in Gosu class `gw.document.DocumentProduction`. There are multiple method signatures for the method `createAndStoreDocumentSynchronously`, all of which check the response type. You must modify the `DocumentProduction` class to appropriately handle the `DOCUMENT_URL` response type.

Your modifications depend on the meaning of the URL in your system:

- If the URL refers directly to the external Document Management System (DMS), the document is already stored. The document production plugin does not need to do anything directly with returned data. Remove the error checking line that requires the document type `DOCUMENT_CONTENTS`. Check if the value is `DOCUMENT_URL`, and if so skip the following lines that assume the result is an input stream.
- If the URL refers to a separate external resource other than the DMS, the document production plugin must immediately retrieve that external data. Next, submit the resulting input stream to the document content source. Base your code to submit the document on the existing `createAndStoreDocumentSynchronously` code that submits an input stream to the document content source.

If your URL has some other meaning than the choices listed above, contact Guidewire Customer Support.

Large Size Microsoft Word Documents

Occasionally, you see the same document template create Microsoft Word documents with different file sizes. The issue, in this case, is that some users have their individual Microsoft Word applications configured to save documents as `Word97-2003 6.0/95 - RTF.doc`. This causes Word to generate the file as Rich Text Format (RTF), but save the generated file with the `.doc` extension. In general, RTF documents have a much larger file size than the basic `.doc` Word files.

To set a default file format for saving new documents, do the following:

1. On the Word **Tools** menu, click **Options**, and then click the **Save** tab.
2. In the **Save Word files as** box, click the file format that you want.

This procedure can be different for each version of Microsoft Word.

Note: Guidewire recommends that you check the Microsoft Word documentation specific to your version for exact instructions.

chapter 10

Sending Emails

This topic describes how to send email messages from Guidewire BillingCenter.

This topic includes:

Guidewire BillingCenter and Email

The Guidewire platform includes support for sending emails from BillingCenter. You can access this capability from any Gosu code. For example, you can access email functionality from Gosu rules or in Gosu embedded in the BillingCenter PCF screens.

BillingCenter provides the following email functionality:

- Support for various types of email recipients (To, CC, and BCC)
- Support for templates that can be used to populate the subject and body of the email
- Support for attaching documents stored in the configured DMS (Document Management System) to the email message

Because email messages are sent using the same integration infrastructure as event-based messages, you can use the same administrative tools for monitoring the status of messages. You can view unsent messages in the BillingCenter **Administration** interface.

The Email Object Model

Guidewire BillingCenter uses the following two classes to define email messages:

- `gw.api.email.Email`
- `gw.api.email.EmailContact`

Both these classes are simple data containers with almost no logic.

gw.api.email.Email

The `Email` class contains the following fields, most of which are self-explanatory:

Field	Description
`Subject`	Subject of the email
`Body`	Body of the email
`Sender`	EmailContact
`ReplyTo`	EmailContact (It is possible for this to be different from the Sender.)
`ToRecipients`	List of EmailContacts
`CcRecipients`	List of EmailContacts
`BccRecipients`	List of EmailContacts
`Documents`	List of DocumentBase entities to be attached to the email

gw.api.email.EmailContact

The `EmailContact` class contains three fields:

Field	Description
`Name`	Name of contact
`EmailAddress`	Email address of contact
`Contact`	Contact entity, which can be `null`. If this is set, it sets the `Name` and `EmailAddress` fields to the appropriate values from the specific `Contact` entity.

Email Utility Methods

Besides the `Email` and `EmailContact` classes, Guidewire also provides a set of static utility methods in the `gw.api.email.EmailUtil` class for generating and sending emails from Gosu:

```
gw.api.email.EmailUtil.sendEmailWithBody( KeyableBean entity, Email email )
gw.api.email.EmailUtil.sendEmailWithBody( KeyableBean entity,
                                          Contact to,
                                          Contact from,
                                          String subject,
                                          String body )
gw.api.email.EmailUtil.sendEmailWithBody( KeyableBean entity,
                                          String toEmailAddress,
                                          String toName,
                                          String fromEmailAddress,
                                          String fromName,
                                          String subject,
                                          String body )
```

All three methods take an entity as the first parameter. This parameter can be `null`. However, if specified, use the application entity to which this email is related, such as a specific account or activity. BillingCenter uses this parameter only while processing the email for transmission. See "Email Transmission" on page 81.

Emails that Use an Email Object

This variation of the `sendEmailWithBody` method requires that you create a `gw.api.email.Email` entity, and then define its properties to build the `Email` entity. For example:

```
...
var testEmail : gw.api.email.Email
testEmail.Body = "This is a test."
testEmail.Subject = "Test"
...
gw.api.email.EmailUtil.sendEmailWithBody( thisClaim, testEmail)
```

Emails that Use Contact Objects

The second variation of the `sendEmailWithBody` method uses `Contact` objects for the `to` and `from` parameters. In Gosu, you can obtain `Contact` objects from various places. For example, in a account rule, to send an email from an insurance company employee to the insured, do the following:

- Set the `to` parameter to `Account.insured`.
- Set `from` to `Account.AssignedUser.Contact`.

The following Gosu example generates an email from the current assigned user to that user's supervisor:

```
gw.api.email.EmailUtil.sendEmailWithBody(thisAccount, thisAccount.AssignedGroup.Supervisor.Contact,
        thisAccount.AssignedUser.Contact, "A account got a AccountValid event", "This is the text." )
```

Emails that Use an Email Address

Use the following variation of the `sendMailWithBody` method if you do not have a full `Contact` object for a recipient or sender. The contact might have been generated dynamically through some other application. The `sendMailWithBody` method uses a name and email address instead of entire `Contact` records and does not require that you have access to a `Contact` record. In the following example, all arguments are `String` objects:

```
gw.api.email.EmailUtil.sendEmailWithBody( Entity, toName, toEmail, fromName, fromEmail, subject, body)
```

Email Transmission

Guidewire BillingCenter, from the user's perspective, sends emails asynchronously by using the BillingCenter Messaging subsystem. If there is a method call for one of the `EmailUtil.sendEmail` methods, BillingCenter creates a `Message` entity with the contents and other information from the `Email` object.

- If the `entity` parameter is non-null, BillingCenter adds the `Message` entity to the `entity` bundle. BillingCenter persists the `Message` entity any time that it creates the bundle.
- If the `entity` parameter is `null`, BillingCenter persists the `Message` entity immediately.

You must configure a `MessageTransport` class to consume the email Messages and do the actual sending. BillingCenter processes messages one at a time, and sends out the emails associated with that message. For more information, see "Configuring BillingCenter to Send Emails" on page 84.

Understanding Email Templates

You use email templates to create the body of an email message. Unlike Document templates (but like Note templates), BillingCenter performs no Gosu interpretation on the Email templates themselves. Thus, email templates are generally only suitable for boilerplate text that does not require modification, or for presenting in the application interface as a starting point for further modification. You cannot use Email templates for mail-merge-style operations.

WARNING Do not add, modify, or delete files from any directory other than the `modules/configuration` application directory. Otherwise, you can cause damage to BillingCenter.

Email Template Files

An email template consists of two separate files:

- A descriptor file, whose name must end in `.gosu.descriptor`, which contains some metadata about the template.
- A template file, whose name must end in `.gosu`, which contains the desired contents of the email body.

IMPORTANT The names of the descriptor and template files must match.

An email descriptor file contains the following fields:

Field	Description
`Name`	The name of the template
`Topic`	The topic of the template (a String value)
`Keywords`	A list of keywords (which can be used to search the templates)
`Subject`	The subject of the emails created using this template
`Body`	The body of the emails created using this template

For example, `EmailReceived.gosu.descriptor` defines an *Email Received* descriptor file:

```
<?xml version="1.0" encoding="UTF-8"?>
<serialization>
  <emailtemplate-descriptor
    name="Email Received"
    keywords="email"
    topic="reply"
    subject="Email Received"
    body="EmailReceived.gosu"
  />
</serialization>
```

The `EmailReceived.gosu.descriptor` file pairs with the actual template file (`EmailReceived.gosu`):

Thank you for your correspondence. It has been received and someone will contact you shortly to follow up on your feedback.

Sincerely,

By default, email templates live in the following location in Studio:

Other Resources → **emailtemplates**

See also

For general information on templates, how to create them, and how to use them, see:

- "Gosu Templates" on page 359 in the *Gosu Reference Guide*
- "Data Extraction Integration" on page 441 in the *Integration Guide*

Creating an Email Template

BillingCenter stores all email template files (both descriptor and template files) in the following location:

Other Resources → **emailtemplates**

To create a new email template

1. Navigate to **Other Resources** → **emailtemplates**, right-click, and select **Other file** from the **New** menu.
2. For the template file, do the following:
   a. Enter the name of the email template and add the `.gosu` extension.
   b. Enter the body of the email template in the view tab that opens. This file defines the text of the email message to send. For example:

```
Greetings:

Please contact <%= activity.AssignedUser %> at <%= activity.AssignedUser.Contact.WorkPhone %>
in regards to <%= activity.Subject %>

Thank you for your patience while we resolve this issue.

Sincerely,
```

3. For the descriptor file, do the following:
   a. Enter the name of the template descriptor file and add the `.gosu.descriptor` extension.
   b. Enter the body of the descriptor file in the view tab that opens. This file defines metadata about the template. For example, the following partial template descriptor file defines the email subject line and specifies the file to use for the email body text. For example:

```
<emailtemplate-descriptor
  name="Request for XYZ"
  keywords="activity, email"
  topic="request"
  subject="We require XYZ for &lt;%= activity.Subject %&gt;"
  body="NeedXYZ.gosu"
  requiredsymbols="Activity"
/>
```

See also

- "Understanding Email Templates" on page 82
- "Creating an Email Template" on page 83
- "Localizing an Email Template" on page 83
- "Gosu Templates" on page 359 in the *Gosu Reference Guide*
- "Data Extraction Integration" on page 441 in the *Integration Guide*

Localizing an Email Template

Localizing an email template is generally a straight-forward process. To localize an email template:

- Create a locale folder for the template files in following location:

  Other Resources → **emailtemplates** → *locale-folder*
- Within the locale folder, place localized versions of the email template file and its associated template descriptor file.

To use a localized email template in BillingCenter, you must perform a search for the localized template as you create a new email.

See also

- “Localizing Templates” on page 75 in the *Globalization Guide*
- “Creating Localized Documents, Emails, and Notes” on page 76 in the *Globalization Guide*

The IEmailTemplateSource Plugin

In the base configuration, Guidewire defines an `IEmailTemplateSource` plugin implementation that provides a mechanism for Guidewire BillingCenter to retrieve one or more Email templates. You can then use these templates to pre-populate email content. (The method of the `EmailTemplateSource` plugin is similar to that of `INoteTemplateSource` plugin.) You configure the `IEmailTemplateSource` plugin as you would any other plugin.

Class LocalEmailTemplateSource

In the base configuration, the `IEmailTemplateSource` plugin implements the following class:

```
gw.plugin.email.impl.LocalEmailTemplateSource
```

This default plugin implementation constructs an email template from files on the local file system. Therefore, it is not suitable for use in a clustered environment.

Class `LocalEmailTemplateSource` provides the following locale-aware method for searching for an email template. It returns an array of zero, one, or many `IEmailTemplateDescriptor` objects that match the locale and supplied values.

```
getEmailTemplates(locale, valuesToMatch)
```

These parameters have the following meanings:

`locale`	Locale on which to search.
`valuesToMatch`	Values to test. You can include multiple values to match against, including: • topic • name • key words • available symbols

Configuring BillingCenter to Send Emails

To configure Guidewire BillingCenter to send emails, you must first define an email server *destination* to process the emails. In the base configuration, BillingCenter defines an *email* destination with ID 65. In BillingCenter, a destination ID is a unique ID for each *external system* defined in the **Messaging** editor in Studio. Exactly one messaging destination uses the built-in email transport plugin, that destination has one ID associated with it, and that ID is 65.

To view the definition of the message destination in Studio, navigate to the following location in Studio:

Messaging → **email**

You cannot change the ID for this message destination. You can, however, set other messaging parameters as desired. For information on the **Messaging** screen, see “Using the Messaging Editor” on page 131 in the *Configuration Guide*.

You also need to configure the base configuration messaging transport that actually sends the email messages. The BillingCenter Gosu email APIs send emails with the built-in email transport. In the Studio **Plugins** editor, this plugin has the name `emailMessageTransport`.

To view or modify the `emailMessageTransport` plugin, navigate to the following location:

Plugins → **gw** → **plugin** → **messaging** → **MessageTransport** → **emailMessageTransport**

This plugin can implement one of the following classes or you can create your own message transport class:

- `EmailMessageTransport`
- `JavaxEmailMessageTransport`

This plugin has several default parameters that you need to modify for your particular configuration.

Configuration Parameter	Description
`SMTPHost`	Name of the SMTP email application. BillingCenter attempts to send mail messages to this server. For example, this might be `EmailHost1.acmeinsurance.com`.
`SMTPPort`	SMTP email port. Email servers normally listen for SMTP traffic on port 25 (the default), but you can change this if necessary.
`defaultSenderName`	Provides a default name for outbound email, such as `XYZ Company`.
`defaultSenderAddress`	Provides a default *From* address for out-bound email. This indicates the email address to which replies are sent.

BillingCenter uses the default name and address listed as the sender if you do not provide a *From Contact* in the Gosu that generates the email message. Otherwise, Studio uses the name and primary email address of the From Contact.

See also

- For information on events, message acknowledgements, the types of messaging plugins, and similar topics. see “Messaging and Events” on page 303 in the *Integration Guide*.

Class EmailMessageTransport

In the base configuration, BillingCenter defines a `gw.plugin.email.impl.EmailMessageTransport` Gosu class that provides the following useful methods (among others):

```
createHtmlEmailAndSend(wkSmtpHost, wkSmtpPort, email)
createEmail(wkSmtpHost, wkSmtpPort, email) : HtmlEmail
```

These parameters have the following meanings:

`wkSmtpHost`	SMTP host name to use for sending mail
`wkSmtpPort`	SMTP host port number
`email`	`Email` object

Method `createHtmlEmailAndSend` simply calls method `createEmail` with the required parameters. Method `createEmail` returns then an HTML email object that the calling method can then send. Method `createEmail` constructs the actual email documents for the email.

Error Handling

In the base configuration, the `emailMessageTransport` implementation has the following behavior in the face of transmission problems:

- If the mail server configuration is incorrectly set within BillingCenter itself, then BillingCenter does not send any messages until you resolve the problem. This ensures that BillingCenter does not lose any emails.
- If some of the email addresses are bad, then BillingCenter skips the emails with the bad addresses.
- If all of the email address are bad, then BillingCenter marks the message with an error message and skips that particular message. BillingCenter reflects this skip in the message history table.

Class JavaxEmailMessageTransport

In the base configuration, BillingCenter defines a `gw.plugin.email.impl.JavaxEmailMessageTransport` Gosu class that provides the following useful methods (among others):

```
createHtmlEmailAndSend(wkSmtpHost, wkSmtpPort, email)
populateEmail(out, email)
```

These parameters have the following meanings:

`wkSmtpHost`	SMTP host name to use for sending mail
`wkSmtpPort`	SMTP host port number
`email`	`Email` object
`out`	`MimeMessage` object

In the base configuration, the `createHtmlEmailAndSend` method on `JavaxEmailMessageTransport` does the following:

- It sets the SMTP host name and port.
- It retrieves the default `Session` object. (If one does not exist, it creates one.) In the base configuration, this method does not restrict access to the Session object. In other words, there is no authentication set on this session.
- It creates a new `MimeMessage` object (using the default session) and passes it to the `populateEmail` method.
- It throws a messaging exception if any exception occurs during the operation.

The `populateEmail` method takes the `MimeMessage` object and creates a container for it that is capable of holding multiple email bodies. Use standard `javax.mail.Multipart` methods to retrieve and set the subparts.

> **Note:** There are many reasons why there can be different versions of an email from the same information. (Locale is not one of those reasons as BillingCenter localizes email information before writing it to the message queue.) For example, you can split an email that exceeds some maximum size into multiple emails. Or, you can generate one email body for internal users and another, different, email body for external users.

The method then adds the following to the mime object:

- Sender
- Headers
- Recipients
- Subject
- Documents (if any)
- Email body

The `populateEmail` method returns an email object that you can then send.

Authentication Handling

In the base configuration, the `JavaxEmailMessageTransport.createHtmlEmailAndSend` method does not provide authentication on the session object. If you want to provide user name and password authentication for email messages, then you must do one of the following:

- Modify the `JavaxEmailMessageTransport.createHtmlEmailAndSend` method to provide authentication. In the base configuration, BillingCenter sets the `Authenticator` parameter in the following session creation method to `null` (meaning none).

  ```
  Session.getDefaultInstance(props, null)
  ```

  If you want to add authentication, then you must create and add your own `Authenticator` object to this method call.

- Modify the `JavaxEmailMessageTransport.createHtmlEmailAndSend` method to use the standard `javax.mail.internet.MimeMessage` encryption and signature methods.

Working with Email Attachments

By default, the `emailMessageTransport` plugin interacts with an external document management system as the *system user* during retrieval of a document attached to an email. It is possible, however, to retrieve the document on behalf of the user who generated the email message instead. To do this, you need to set the `UseMessageCreatorAsUser` property in the `emailMessageTransport` plugin.

To set the UseMessageCreatorAsUser property

1. In Studio, navigate to the following location:

   Configuration → **Plugins** → **gw** → **plugin** → **messaging** → **MessageTransport** → **emailMessageTransport**

2. In the **Parameters** area, click **Add** to create a new parameter entry.
3. Enter the following:

Name	`UseMessageCreatorAsUser`
Value	`true`

Sending Emails from Gosu

Note: Creating an email message and storing it in the Send Queue occurs as part of the same database transaction in which the rules run. This is the same as regular message creations triggered through business rules.

To send an email from Gosu, you need to first create the email, typically through the use of email templates (described in “Understanding Email Templates” on page 82). You then need to send the email using one of the `sendEmailWithBody` methods described in “Email Utility Methods” on page 80.

Saving an Email Message as a Document

BillingCenter does not store email objects created through the `Email` class in the database. However, it is possible to save the contents, recipient information, and other information of an email as a document in the configured DMS application. Because the `sendEmailXXX` methods all take all the sending information explicitly, you can save the email information by using the following process in Gosu code:

1. Create the email subject and body, and determine recipients.
2. Send the email by calling the appropriate `EmailUtil` method.
3. If BillingCenter does not encounter an exception, create a document recording the details of the sent email.

Create Document from Email Example

See “Document Creation” on page 63 for details of the `DocumentProduction` class methods.

```
// First, construct and send the email
var toEmailAddress : String = "Recipient email address"
var toName = "Recipient’s Name"
var fromEmailAddress : String = "Sender email address"
var fromName : String = "Sender's name"
var subject : String = "Email Subject"
var body : String = "Email Body"
gw.api.email.EmailUtil.sendEmailWithBody( null, toEmailAddress, toName, fromEmailAddress, fromName,
        subject, body )

// Next, create the document recording the email
var contextObjects = new java.util.HashMap()
```

```
contextObjects.put("To", toEmailAddress)
contextObjects.put("From", fromEmailAddress)
contextObjects.put("Subject", subject)
contextObjects.put("Body", body)
...
var document : Document = new Document()
document.Name = "EmailSent"

// Set other properties as needed
var template : gw.plugin.document.IDocumentTemplateDescriptor
var templateName =  "EmailSent.gosu"
...

// Call this method to create and store the document for later retrieval
gw.document.DocumentProduction.createDocumentSynchronously(template, contextObjects, document)
```