Guidewire ClaimCenter™

Gosu Rules Guide

Release: 10.2.4

GUIDEWIRE

Product Name: Guidewire ClaimCenter
Product Release: 10.2.4
Document Name: Gosu Rules Guide
Document Revision: 10-December-2024

Contents

Support ... 9

Part 1
ClaimCenter rules ... 11

1 **Rules: A background** ... 13
Guidewire business rules, Guidewire Gosu rules ... 14
Gosu rule hierarchy ... 15
Gosu rule execution ... 15
Gosu rule management ... 16
2 **About ClaimCenter Gosu rules** ... 17
Rule structure ... 17
  Gosu rule syntax ... 18
  Rule members ... 18
  Understanding Gosu rule conditions ... 19
  Understanding Gosu rule actions ... 20
Exiting a Gosu rule ... 20
  How to use exitToNextRoot ... 20
Gosu annotations and ClaimCenter Gosu rules ... 21
  The effect of annotations on Gosu rules ... 21
  Determine the annotations on elements of Gosu code ... 21
Invoking a Gosu rule from gosu code ... 21
3 **Using the rules editor** ... 23
Working with rules ... 23
  View or edit a rule ... 23
  Changing rule order ... 23
  Create a new rule set category ... 24
  Create a new rule set ... 24
  Create a new rule ... 25
  Accessing a rule set from Gosu code ... 25
  Renaming or deleting a rule ... 25
  Making a rule active or inactive ... 26
About changing the root entity of a rule ... 26
  Change the root entity of a rule ... 27
4 **Writing rules: testing and debugging** ... 29
Generating logging information in Gosu rules ... 29
5 **Writing rules: examples** ... 31
Accessing fields on subtypes ... 31
Preventing repeated actions in Gosu rules ... 31
Taking action on more than one subitem ... 32
Checking entity permissions ... 32
Determining the original attributes of a changed entity ... 33
6 **ClaimCenter rule set categories** ... 35
Approval routing ... 36
  Approval rules and Transaction approval rules ... 36

- BulkInvoice Approval Assignment Rules........38
- Archive........38
  - Understanding archive events........39
  - About skipped and excluded claims in archiving........39
  - Default Group Claim Archiving rules........40
  - Claim Eligible for Archive rules........41
- Assignment........41
  - Default Group Activity Assignment rules........42
  - Default Group Claim Assignment rules........43
  - Default Group Exposure Assignment rules........43
  - Default Group ServiceRequest Assignment rules........44
  - Global Activity Assignment rules........44
  - Global Claim Assignment rules........46
  - Global Exposure Assignment rules........47
  - Global Matter Assignment rules........47
  - Global ServiceRequest Assignment rules........48
- BulkInvoice........48
  - Bulk Invoice Approval rules........48
- Closed........49
  - Activity Closed rules........49
  - Claim Closed rules........49
  - Exposure Closed rules........50
  - Matter Closed rules........51
- Event message........51
  - About messages and messaging events........52
  - About modifying data in Event Fired rules and messaging plugins........54
  - Event Fired rule example........54
  - ISO Event Message rules........54
  - MetroReport Event Message rules........55
  - System console logger Event Message Rule........56
  - Contact Auto Synchronization Failure Event Message rule........56
  - Policy System Notification Event Message rules........56
  - Asynchronous Document Storage Event Message rule........57
  - Contact Event Message rules........58
  - Instant check message rules........59
  - Solr Destination Event Message rules........60
- Exception........61
  - Activity Escalation rules........61
  - Claim Exception rules........62
  - Group Exception rules........65
  - User Exception rules........66
- Initial Reserve........66
  - Initial Reserve rules for auto vehicle damage........66
  - Initial Reserve rules for travel losses........67
- Loaded........68
  - Claim Loaded rules........68
  - Exposure Loaded rules........69
- Postsetup........69
  - Activity Postsetup rules........69
  - Claim Postsetup rules........70
  - Exposure Postsetup rules........70

- Matter Postsetup rules........70
- Transaction Postsetup rules........71

Presetup........71
- About claim and exposure setup........72
- Creating Transaction Set Presetup rules........73

Preupdate........76
- How ClaimCenter triggers Preupdate rules........76
- How ClaimCenter triggers Preupdate rules on custom entities........77
- Activity Preupdate rules........78
- Claim Preupdate rules........78
- Contact Preupdate rules........84
- Exposure Preupdate rules........85
- Group Preupdate rules........86
- Matter Preupdate rules........86
- Policy Preupdate rules........87
- RIAgreementPreupdate rules........87
- RITransactionSetPreupdate rules........87
- TransactionSet Preupdate rules........88
- User Preupdate rules........90

Reopened........90
- Claim Reopened rules........90
- Exposure Reopened rules........91
- Matter Reopened rules........91

Segmentation........91
- Claim Segmentation rules........92
- Exposure Segmentation rules........93
- Handling a strategy separately from a segment........96

Transaction Approval........96
- Transaction Approval rules........97

Validation........98
- About ClaimCenter validation levels........98
- About validation warnings and errors........99
- Adding new validation levels........100
- How ClaimCenter triggers validation rules........100
- How ClaimCenter triggers Validation rules on custom entities........101
- ClaimCenter Validation and Preupdate rules........101
- Determining the validity of an object........102
- Validation error reject methods........102
- Activity Validation rules........104
- Claim Closed Validation rules........104
- Claim Reopened Validation rules........105
- Claim Validation rules........106
- Contact Validation rules........109
- Exposure Closed Validation rules........109
- Exposure Reopened Validation rules........110
- Exposure Validation rules........110
- Group Validation rules........113
- Matter Closed Validation rules........113
- Matter Reopened Validation rules........114
- Matter Validation rules........114
- Person Validation rules........114

Policy Validation rules....... 114
RIAgreement Validation rules....... 115
RITransactionSet Validation rules....... 116
ServiceRequest Validation rules....... 116
TransactionSet Validation rules....... 117
User Validation rules....... 119
Workplan....... 119
Gosu methods to use with Workplan rules....... 120
Claim Workplan rules....... 120
Exposure Workplan rules....... 121
Matter Workplan rules....... 122
7 ClaimCenter rule reports 125
About the Rule Repository report....... 125
Generate a Rule Repository Report....... 125
About the Profiler Rule Execution report....... 125
Generate a Profiler Rule Execution Report....... 126
Understanding the Profiler Rule Execution Report....... 126
View rule information in the Profiler Chrono report....... 127

Part 2
Advanced topics....... 129

8 Assignment in ClaimCenter....... 131
Understanding assignment in Guidewire ClaimCenter....... 131
Assignment Entity Persistence....... 131
Assignment Queues....... 131
Primary and secondary assignment....... 132
Primary (user-based) assignment entities....... 132
Secondary (role-based) assignment entities....... 132
Assignment within the Assignment rule sets....... 133
Assignment outside the Assignment rule sets....... 133
Global and default Assignment rules....... 134
Global Assignment rules....... 134
Default Group Assignment rules....... 135
Assignment execution session....... 135
Assignment success or failure....... 136
Determining assignment success or failure....... 136
Assignment success or failure within the context of rules....... 136
Assignment success or failure outside the context of rules....... 136
Logging assignment activity....... 137
Assignment cascading....... 137
Assignment events....... 138
Assignment method reference....... 138
Assignment by assignment engine....... 138
Group assignment (within the Assignment rules)....... 139
Queue assignment....... 142
Immediate assignment....... 143
Claim-based assignment....... 144
Condition-based assignment....... 145
Proximity-based assignment....... 148
Round-robin assignment....... 152

Dynamic assignment.... 152
Manual assignment.... 155
Workload Count.... 156
9 Rules-based validation.... 159
Overview of rules-based validation.... 160
Entity validation order.... 160
About the validation graph.... 160
How ClaimCenter traverses the validation graph.... 161
How ClaimCenter works with owned arrays.... 161
Top-level entities that trigger validation.... 162
Overriding validation triggers.... 164
Validation performance issues.... 164
View the validation graph in the server log.... 165
10 Detecting claim fraud.... 167
Claim fraud.... 167
The Special Investigation Details screen.... 167
Special Investigation trigger rules.... 168
SI rule evaluation sequence.... 168
Using the Special Investigation rules.... 169
Set the initial life cycle stage.... 169
Advance the life cycle stage.... 170
Evaluating SI triggers.... 170
Escalating a claim to a supervisor.... 172
Assigning an activity to a SIU group.... 172
Adding an assigned user in a SIU role.... 172
11 Transaction Set validation.... 175
Transaction Set validation rules.... 175
TXV01000 – Total payments not greater than exposure limit.... 176
TXV02000 – Check exposure limit when increasing reserves.... 177
TXV03000 – Total payments not greater than incident limit.... 177
TXV04000 – Check incident limit when increasing reserves.... 178
TXV05000 – PIP.... 178
TXV06000 – Reserve threshold.... 179
TXV08000 – Check aggregate limits.... 179
TXV13000 – Warning if adverse party pays non-subrogation recovery.... 180
TXV17000 – Reserve lines compatible with service requests.... 180
TXV18000 – Currencies compatible with service requests.... 180
sumForPIPAgg enhancement methods.... 181
12 Working with queues.... 183
About activity queues.... 183
Example: Using a queue to handle FNOL claims.... 183
Assign a claim to an intake group.... 184
Assign a claim to a group supervisor.... 185
Create a FNOL review activity.... 185
Assign a FNOL review activity to an intake group.... 186
Assign a FNOL review activity to a queue.... 186
Example: Using a queue to void/stop checks.... 187
Create a stop/void queue.... 187
Assign an activity to the appropriate group.... 188
Add an activity to a queue.... 188

Support

For assistance, visit the Guidewire Community.

Guidewire customers

`https://community.guidewire.com`

Guidewire partners

`https://partner.guidewire.com`

part 1

ClaimCenter 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 ClaimCenter rule set categories.

Designing Gosu rules

In general, Guidewire strongly recommends that you develop and document the functional logic of rules before attempting to turn that logic into rules within ClaimCenter. 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 ClaimCenter rules work. It can also help you make decisions about changing your rules as your use of ClaimCenter evolves over time.

Gosu 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.

Gosu rule terminology

Guidewire uses the following terminology in discussing Gosu rules.

assignment engine

The assignment engine handles assignment of claims, exposures, and activities to users by repeatedly asking ClaimCenter to evaluate assignment rules for the item. Guidewire structures the assignment rule sets so that the assignment engine first asks to which claim handling office the assignment goes. It then asks to which group, and finally to which person (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 `Claim`. 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 ClaimCenter resources, such as PCF pages, Gosu 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 `Claim` or `Activity`. For an entity type, Guidewire also calls an object an *entity instance*.
- An instance of an Gosu class
- An instance of a 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}
```

The following example illustrates the logic of a validation rule.

If the claim has no loss date or the claim has a loss date in the future, mark the loss date field as invalid.

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. You create and develop workflow within Guidewire Studio.

Guidewire business rules, Guidewire Gosu rules

Guidewire provides two different types of rules in the base ClaimCenter configuration.

ClaimCenter business rules

You use business rules to create, edit, and manage activities. You work with business rules in the **Business Rules** screens, in ClaimCenter **Administration** > **Business Settings**.

ClaimCenter provides the following types of business rules:

- Activity rules
- Exposure rules
- Reserve rules

The target users of business rules are ClaimCenter administrators and business analysts. It is possible to create and deploy ClaimCenter business rules without starting the application server.

These rules perform different functions. For example, you use activity business rules for tasks such as the following:

- Adding an activity to the workplan
- Creating activities more selectively than through Gosu rules
- Changing an activity assignment
- Adding a new activity pattern to the workplan
- Disabling an activity

Gosu rules

You create and manage Gosu rules entirely in Guidewire Studio. Gosu rules require in-depth domain knowledge and technical expertise to create. After you make changes to Gosu rules, you typically need to restart the application server.

Gosu rule hierarchy

A rule set can be thought of as a logical grouping of rules that are specific to a business function within ClaimCenter. 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.

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 ClaimCenter 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 ClaimCenter. However, make each hierarchy specific to the rule set to which it belongs.

Gosu rule execution

The hierarchy of rules in ClaimCenter mirrors a decision tree that you might diagram on paper. ClaimCenter 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 following the line of the rule set.) ClaimCenter 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.

To illustrate how to use a rule hierarchy, take segmentation logic as an example. You would probably expect to describe a series of questions to figure out in which segment to put a claim. For convenience, you would want to describe these questions hierarchically. For example, if you answer question #1 as No, then skip directly to question #20 because questions #2-19 only pertain if you answered #1 as Yes.

You might go through logic similar to the following:

- **If this is an auto claim**, then go through a list of more detailed segmentation rules for auto claims:
  - If this has glass damage only, then segment as "Auto Glass" [done]
  - Otherwise, if this was a collision, then consider collision segmentation rules
    - If there are injuries, then segment as "injury accident" [done]
    - Else if there are more than two cars, segment as "multi-car accident" [done]
    - Else if there are two cars, segment as "2 car accident" [done]
    - Else segment as "single car accident" [done]
  - Otherwise, consider non-collision segmentation rules
    - If there are third-party exposures, segment as "non-accident liability" [done]
    - Else, if the cause of loss is theft, then segment as "theft" [done]
    - Else, segment as "non-accident vehicle damage" [done]
- **If this is a homeowner's claim**, then go through a list of more detailed segmentation rules for homeowner's claims…
- **If all else fails [Default]**, segment this as "unknown" [done]

You can see from this example that your decision tree follows a hierarchy that maps to the way ClaimCenter keeps rules in the hierarchy. If the first rule checks whether the claim is an auto claim, then ClaimCenter does not need to check any of the follow-up (child) rules. (That is, unless the claim is actually an auto claim.)

Gosu rule management

Guidewire strongly recommends that you tightly control access to changing rules within your organization. Editing rules is a complicated process. Because ClaimCenter 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.

Typically, the following kinds of people manage the ClaimCenter Gosu rules.

Business analysts

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 ClaimCenter.

Technical rule writers

One or more rule writers who are generally more technical than business analysts. 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 Gosu rules. These are rules that you can actually implement with the information available to ClaimCenter.

chapter 2

About ClaimCenter Gosu rules

In the base configuration, Guidewire provides a number of sample rules as examples of how to use rules to perform business logic. 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 ClaimCenter. Rules contain the Gosu code (condition and action) that perform the actual business logic.

Rule structure

Every Gosu rule has the following basic syntax:

`IF` {some conditions}

`THEN` {do some actions}

Each Gosu rule divides the rule functionality into different blocks of code, with a specific label. The following table summarizes each of these parts.

Rule block	Description

USES	A list of packages or classes that this rule uses. For example:
	`uses java.util.HashSet`
	`uses gw.lang.reflect.IType`

CONDITION	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 an auto claim) AND (the policy includes collision coverage)
	It is also possible to insert a statement list, instead of a simple expression. For example:
	`var o = new HashSet<IType>() {A, B, C, ...}`
	`return o.contains(typeof(...))`
	The `CONDITION` block must contain a `return` statement that returns a Boolean value.

ACTION	A list of actions to take. For example: • Mark the claim as flagged} • Add an activity for the adjuster to follow up

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, ClaimCenter gives you access to conditions and actions that are appropriate for the type of your current rule.

Gosu rule syntax

Guidewire Gosu rules use a programming language called Gosu, which is a high-level language tailored for expressing Gosu 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

All expressions use a dot notation to reference fields, subobjects, and methods. For example, to reference the license plate data from a vehicle associated with a claim, the field path is:

```
claim.Vehicles[0].LicensePlate
```

A statement can consist of one or many expressions.

IMPORTANT: Use only Gosu expressions and statements to create ClaimCenter Gosu rules. For example, do not attempt to define a Gosu function in a Gosu 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.

See also

- See the *Gosu Reference Guide* for more information about how to write Gosu statements.

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 (claims, for example).

Rule conditions

Rule conditions are 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 `ACTION` block. For example, you can use the following condition to test whether the contact information includes a home phone number:

```
contact.HomePhone == null
```

Rule actions

Rule actions are 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 in the `CONDITION` block evaluates to `true`. For example, if a contact information is missing a home phone number, you can use the following code to raise an error in ClaimCenter:

```
contact.rejectField("HomePhoneCountry", ValidationLevel.TC_LOADSAVE,
      displaykey.Validator.Phone.Home.CountryCode.Null, null, null)
```

Rule APIs

Rule APIs are a collection of Gosu methods accessed through the `gw.api.*` package. They include many standard mathematic, date, string, and financial methods. For example, you can use the following code to determine if an activity (`act`) is more than 15 days old and thus needs additional handling:

```
gw.api.util.DateUtil.differenceInDays(act.CreateTime,
        gw.api.util.DateUtil.currentDate()) > 15
```

Rule entities

Rule entities are the collection of business object entities that ClaimCenter supports. Guidewire Studio objects use the familiar "dot" notation to reference fields and objects.

For example, you can use the following object conditions to verify if the loss is an Auto claim involving a vehicle collision without the weather field entered:

```
Claim.LossType == "auto" and
Claim.LossCause == "vehcollision" and Claim.Weather == null
```

Understanding Gosu 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 reference 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 reference the activity in question as `Activity`. Other root objects, depending on your Guidewire application, are `Account`, `PolicyPeriod`, `Producer`, `Invoice`, `TroubleTicket`, and `Charge`.
2. ClaimCenter 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 using standard Boolean logic operators (`and`, `or`, `not`). For example:

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

IMPORTANT: The rule condition statement must evaluate to either Boolean `true` or `false`. If you create a condition statement that evaluates to `null`, ClaimCenter interprets this as `false`. This can happen inadvertently, especially if you create a condition statement with multiple conditions to evaluate. If your condition evaluates to `null` (`false`), ClaimCenter 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 `CONDITION` block. Every `CONDITION` block must contain a `return` statement that evaluates to a Boolean value of `true` or `false`. For example:

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

See also

- See the the *Gosu Reference Guide* for details on operator precedence and other syntactical information.

Understanding Gosu rule actions

Within the rule `ACTION` block, you create the outcome for the criteria identified in the rule `CONDITION` block. 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.

For example, suppose that you want an exposure to fail validation if a certain coverage exists but the claim does not contain that coverage. The following syntax creates this action:

```
for( Exposure in claim.Exposures ) {
  if( Exposure.OtherCoverage==true and (Exposure.OtherCoverageInfo==null or Exposure.OtherCoverageInfo=="") ) {
    // Mark each exposure in which this occurs as invalid
    Exposure.rejectSubField( Exposure, "OtherCoverageInfo", "loadsave", "You need to provide information
          about claimant's other coverage.", null, null )
  }
}
```

See also

- See "Validation error reject methods" on page 102 for a discussion of the various `reject` methods.

Exiting a Gosu rule

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

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

Exit – 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 ClaimCenter what to do next.

Flow control action	Description
`actions.exit`	This is the simplest version of an exit action. ClaimCenter stops processing the rule set as soon as it encounters this action.
`actions.exitAfter`	This exit action causes ClaimCenter to stop processing the rule set after processing any child rules.
`actions.exitToNext`	This exit action causes ClaimCenter 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. You use this exit action only rarely, within very complicated action sections.
`actions.exitToNextParent`	This exit action causes ClaimCenter 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 exit action causes ClaimCenter 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, a rule lists an exit action as the last action in the rule so that it occurs only after the rule executes all other defined actions.

How to use exitToNextRoot

It is useful to employ the `exitToNextRoot` method if you set up your rule set to make two separate decisions. For example, suppose that you set up the segmentation rules to do the following sequence of actions:

1. First, determine the severity rating of an injury.
2. Then, decide on the segmentation using the results of this determination.

To do this, you can structure the rule set as follows:

- Always process my child rules to evaluate severity…
  - If (conditions) Then (Set severity) [Done, but go to next top-level rule]
  - Otherwise…
- Always process my child rules to evaluate categorization…
  - If (conditions) Then (Set segment) [Done, no need to go further]
  - Otherwise…

After ClaimCenter makes the first decision (setting the severity), the decision logic is complete. However, ClaimCenter needs to move on to the next major block of decision-making, which is deciding on the categorization. After ClaimCenter sets the segment, the rule set is finally complete, so the rule can just use the simple exit method.

Gosu annotations and ClaimCenter Gosu rules

Guidewire ClaimCenter uses annotation syntax to add metadata to Gosu classes, constructors, methods, and properties. For example, you can add an annotation to a method to indicate its return type or what exceptions it might throw.

The variation that annotations cause in the visibility of classes, constructors, methods, and properties can make it seem that your Gosu code is incorrect. The location of your Gosu code might be the issue, not the code itself.

See also

- For information on working with and creating annotations, see the *Gosu Reference Guide*.

The effect of annotations on Gosu rules

Guidewire marks certain Gosu code in the base application with the `@scriptable-ui` annotation. That annotation restricts the usage, or *visibility*, of the code to non-rules Gosu code. The converse is the `@scriptable-all` annotation, which makes a class, constructor, method, or property visible in Gosu code everywhere.

Within the Gosu Rules editor, 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. The Rules editor does not recognize the property descriptor for that property. However, the Gosu compiler does recognize the property in other editors in Studio, such as the Gosu editor for classes and enhancements.

Determine the annotations on elements of Gosu code

About this task

To determine the annotations on an element of Gosu code, such as a class, constructor, method, or property, do the following:

Procedure

1. Place your cursor at the beginning of the line directly above the affected code.
2. Type an `@` sign.
   Studio displays a list of valid annotations.

Invoking a Gosu rule from gosu code

It is possible to invoke Gosu 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 the use of the `invoke` method on a rule set triggers all of the rules in that rule set. You cannot invoke individual rules within a rule set using the `invoke` method. ClaimCenter executes all of the rules in the invoked rule set in sequential order. If a given rule's `CONDITION` block expression evaluates to `true`, then ClaimCenter executes the Gosu code in the `ACTION` block for that rule.

chapter 3

Using the rules editor

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

Working with rules

ClaimCenter 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.

View or edit a rule

Procedure

1. In the Studio **Project** window, navigate to **configuration** > **config** > **Rule Sets**.

   Alternatively, you can use the 'breadcrumb' functionality (which bypasses the Studio **Project** window) located directly under the screen toolbar to navigate to a rule, for example:

   configuration > **config** > **rules** > **…**

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.

Changing rule order

If you want to change the order of your Gosu rules, you can drag and drop rules within the rule hierarchy in Guidewire Studio. If you move rules using drag and drop, ClaimCenter moves the chosen rule and all its children as a group. This behavior makes it easy to reposition entire branches of the hierarchy.

ClaimCenter also supports standard cut, copy, and paste commands, which you can use 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.

Move a rule

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.

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**.

Move a rule to a different level

Select the rule to move and 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.

Create a new rule set category

Procedure

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.

Results

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

Create a new rule set

Procedure

1. In the Studio **Project** window, expand **configuration** > **config** > **Rule Sets**.

   Although the label is **Rule Sets**, the primary children of this folder are actually rule set categories.
2. Do one of the following:
   - If an appropriate rule set category for your rule set does not exist, first perform the steps listed in "Create a new rule set category" on page 24. After creating a rule set category, move to the next step.
   - If an appropriate rule set category for your rule set does exist, move to the next step.
3. Select a rule set category, then select **New** > **Rule Set** from the context menu.
4. Enter the following information:

Field	Description
Name	Studio displays the rule name in the middle pane. 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 middle pane.

Field	Description
	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	ClaimCenter 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.

Create a new rule

Procedure

1. Select the rule set to 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.

Results

Studio creates the new rule as the last child rule of the currently selected rule (or rule set).

Accessing 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 triggers evaluation of every rule in that rule set, in sequential order. If the conditions for a rule evaluate to true, then ClaimCenter executes the actions for that rule.

Renaming or deleting a rule

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

Command	Description	Actions to take
Rule > Rename Rule	Renames the currently selected rule	1. Select the rule to rename in the center pane of Studio. 2. Select **Rename Rule** from the **Rule** menu, or, right-click and select **Rename Rule** 3. 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	1. Select the rule to delete in the center pane of Studio. 2. Select **Delete** from the **Edit** menu, or right-click and select **Delete**. ClaimCenter does not require you to confirm your decision before deleting the rule. You can use the **Delete** command to delete a rule only.

About renaming a rule

At a structural level, Guidewire ClaimCenter 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. ClaimCenter stores the rule definition classes in the following location in the installation directory:

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

If you rename a rule set, ClaimCenter 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, ClaimCenter ensures that the rule class names and the file names are always in synchronization. Always use Guidewire Studio to rename a rule. Never rename a file using your local file system.

Making a rule active or inactive

ClaimCenter 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.

About changing the root entity of a rule

ClaimCenter bases each Gosu rule on a specific business entity. In general, the rule set name reflects this entity. For example, in the Validation rule set category, you have Contact Validation rules and Person Validation rules. These rule set names indicate that the root entity for each rule set is—respectively—the `Contact` object and the `Person` object.

ClaimCenter 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. 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 any other circumstances.

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

1. You create a new entity in ClaimCenter, 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, **TestEntityRule**, for example. 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 cannot 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.

Change the root entity of a rule

About this task

The intent of the Rules editor **Change Root Entity** menu command is to enable you to edit a rule that you otherwise cannot open in Studio. This can happen, for example, if ClaimCenter is unable to parse the rule declarations. Do not use this command in any other circumstances.

Procedure

1. Select a rule within a rule set.
2. Right-click the rule name 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.

Results

After you complete this command:

- Studio performs a check to verify that the provided name 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 is 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.

See also

- “About changing the root entity of a rule” on page 26

chapter 4

Writing rules: testing and debugging

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

- Use class `gw.api.system.CCLoggerCategory` to print out helpful error messages to the application log files.
- Use comments embedded within rules to provide useful information while troubleshooting rules.

Generating logging information in Gosu rules

To trigger the flow of information from a Gosu rule to a log file, you need to configure the following:

- The appender definition (`<RollingFile>`) for the rule log file in `log4j2.xml`.
- The logger definition (`<Logger>`) associated with the rule file in `log4j2.xml`.
- The activation trigger for the logging category in a Gosu rule using `gw.api.system.CCLoggingCategory`.

Logging category definitions

A logging category defines how the Apache log4j logging utility handles different types of logging requests. In the base configuration, Guidewire provides a number of readily usable logging categories.

To define how ClaimCenter interacts with the ASSIGNMENT category, add something similar to the following code in file `log4j2.xml`.

```
<!-- Rolling file definition... -->
<RollingFile name= "RuleEngineLog" fileName="${guidewire.logDirectory}/ruleengine.log"
      filePattern="${guidewire.logDirectory}/ruleengine.log%d{.yyyy-MM-dd}">
 <PatternLayout pattern="${file.defaultPattern}" charset="UTF-8"/>
 <TimeBasedTriggeringPolicy/>
</RollingFile>

<!-- Logger definition... -->
<Logger name="Assignment" additivity="false" level= "info">
  <AppenderRef ref="RuleEngineLog">
</Logger>
```

Notice that this code does the following:

- It defines a rolling log file named `ruleengine.log` in the `<RollingFile>` element.
- It then links the `ruleengine.log` file to the `Assignment` logger category in the `<Logger>` element.
- It sets a default logging level of INFO.

Base configuration logging categories

There are several ways to view the base configuration logging categories:

- From the **Set Log Level** Server Tools screen.
- By running the `system_tools` command from a command prompt and adding the `-loggercats` option.

Triggering a logging category from a Gosu rule

For example, to use this API in Gosu code to perform assignment logging, do something similar to the following:

```
uses gw.api.system.CCLoggerCategory
...
var logger = CCLoggerCategory.ASSIGNMENT
...
logger.info("Print out this message.")
```

Logging example

The following code is an example of the use of a logging category in a Gosu rule. This code assumes that the necessary appender and logger definitions exist in file `log4j2.xml`.

```
var logger=gw.api.system.CCLoggerCategory.ASSIGNMENT

// Pass the user search criteria to the assignUserByProximityWithSearchCriteria function
if (activity.CurrentAssignment.assignUserByProximityWithSearchCriteria(usc, -1, true,
      activity.CurrentAssignment.AssignedGroup)) {
  logger.debug( DisplayKey.get("Rules.Assignment.DefaultGroup.Activity", actions.ShortRuleName) )
  logger.debug( DisplayKey.get("Rules.Assignment.AssignedUserIs", activity.AssignedUser) )
  actions.exit()
}
```

See also

- *Administration Guide*

chapter 5

Writing rules: examples

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

Accessing fields on subtypes

Various entities in ClaimCenter 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 to particular subtypes only, you cannot reference them in rules by using the primary root object. For example, the following illustrates an invalid way to reference the lienholder of a vehicle:

```
Exposure.lienholder[0].FirstName == "Joe" // 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:

```
(Exposure.lienholder[0] as Person).FirstName == "Joe" // valid
```

As another example, consider transaction validation rules. A transaction set has several subtypes, such as a `ReserveSet`, `CheckSet`, and others. To validate that primary checks being sent by mail have a mailing address that is not `null`, you can use the following in the rule condition:

```
TransactionSet.Subtype == "checkset" &&
  (transactionSet as CheckSet).PrimaryCheck.DeliveryMethod == "send" &&
  (transactionSet as CheckSet).PrimaryCheck.MailToAddress == null
```

It is important to cast the type of an object correctly, otherwise, a Runtime Error can occur.

Preventing repeated actions in Gosu rules

Many times, during exception rule execution, you want to take action on an exception the first time you discover the exception only. For example, suppose that you want to remind the adjuster to set the fault rating on an claim that is open for more than 30 days. If this action has not been completed within 30 days, then you might want to add an activity on day 31 to remind the adjuster. On day 32, if the adjuster still has not set the rating, you do not want to add another activity, even though ClaimCenter finds the same exception again.

One way to handle this is to use ClaimCenter methods for noting a custom event in the Claim History. The first time ClaimCenter finds an exception, you need to instruct ClaimCenter to note it in the claim’s history. Then, in the rule, include a check to verify if this event appeared before in the rule's exceptions. If the event already exists, then do not execute the actions again.

The syntax for this is similar to the following:

```
CONDITION (claim : entity.Claim):
return (gw.api.util.DateUtil.daysSince( claim.ReportedDate ) > 30)
      and (!exists (hist in claim.History where hist.CustomType == "a_n_f_r"))

ACTION (claim : entity.Claim, actions : gw.rules.Action) :
claim.createCustomHistoryEvent( "a_n_f_r", "description" )
claim.createActivity( parameterList)
```

In this example, ClaimCenter determines if the event occurred previously. You can also determine if the event occurred within a certain time period. In this way, you can repeat the rule's actions if enough time has past since ClaimCenter first noted the exception.

Taking action on more than one subitem

A common situation to handle is raising the priority on every activity on a claim that is overdue. Gosu provides a `for(... in ...)` syntax and an `if(...) {` *`then do something`* `}` syntax that you can use to construct fairly complicated actions:

```
for (act in claim.Activities) {
  if ( act.Status == "open" and act.TargetDate < gw.api.util.DateUtil.currentDate() ) {
    act.Priority = "urgent"
  }
}
```

Notice that the curly braces ({}) mark the beginning and end of a block of Gosu statements:

- The outer pair of braces contain statements to perform "for" each activity on the claim.
- The inner pair of braces contain statements to perform "if" the activity is overdue.

See also

- For information on the syntax to use to construct `for` and `if` statements, see the *Gosu Reference Guide*.

Checking entity permissions

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

- ClaimCenter exposes static permissions that are non-object-based (like the permission to create a user) as Boolean properties.
- ClaimCenter exposes permissions that take an object (like the permission to edit a claim) as methods that take an entity as their single parameter.
- ClaimCenter 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. ClaimCenter 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.

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 = u.Roles.hasMatch(\role -> role.Role.Privileges.hasMatch(\perm -> perm.Permission == p))
```

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

Determining the original attributes of a changed entity

The following example illustrates how to determine if the original fieldvalues on an entity have changed, and, for the changed values, what are the original values. For example, suppose that the Loss Location on the claim changes for some reason. You might want to retrieve the original values to compare with the current values to determine how the change in the Loss Location state.

It is not sufficient to determine the original values simply by knowing that there was a change to the Loss Location. The following expression does not provide the correct value, as it does not point back to the original Loss Location.

```
Claim.LossLocation.getOriginalValue("State")
```

Instead, to determine the original attributes of a changed object, you must first re-instantiate the original object.

In this example, the rule executes if the `isFieldChanged("LossLocation")` becomes `true`. Then, if the new Loss Location state is different from the original state, the rule creates a custom history event identifying the change.

```
CONDTION (claim : entity.Claim):
return claim.isFieldChanged("LossLocation")

ACTION (claim : entity.Claim, actions : gw.rules.Action):
var add = Address( claim.getOriginalValue("LossLocation") )

if (add.State != Claim.LossLocation.State) {
  claim.createCustomHistoryEvent("DataChange",
        displaykey.Rules.Log.DataChange(add.State,claim.LossLocation.State ) )
}
```

Additional changed entity methods

Guidewire provides a number of methods that you can use to determine if an entity changes. (For information on each, place your cursor on the method signature and press `Ctrl+Q`.)

```
Entity.getOriginalValue(fieldname : String)
Entity.getAddedArrayElements(arrayFieldName : String)
Entity.getChangedArrayElements( arrayFieldName : String)
Entity.getRemovedArrayElements( arrayFieldName : String)
Entity.isArrayElementChanged(arrayFieldName : String)
Entity.isArrayElementAddedOrRemoved(arrayFieldName : String)
Entity.isFieldChanged(fieldname : String)
```

You can also use the read-only `Changed` property on the entity to determine if the entity contains changed fields. The `Changed` property becomes `true` only after a modification to the entity.

```
Entity.Changed
```

These methods are available in all Gosu code, not just Gosu rules.

See also

- For information on how to determine if an entity changed, see the *Gosu Reference Guide*.

chapter 6

ClaimCenter rule set categories

As part of the ClaimCenter base configuration, Guidewire provides Gosu sample rules in the following rule set categories.

Note: It is important to understand that you edit Gosu rules in Guidewire Studio. Certain Gosu rules relate to the business rules that you create and manage in the ClaimCenter Business Rules editor. For business rules that are in the same category as Gosu rules, such as Preupdate rules, ClaimCenter runs the Gosu rules first.

Rule set category	Rules
Approval Routing	Route approval activities for a transaction that requires approval to the appropriate user.
Archive	Manage the claim archive process.
Assignment	Determine the responsible party for an activity, claim, exposure, matter, or service request.
BulkInvoice Approval	Define the approval requirements for Bulk Invoices.
Closed	Perform an action whenever you close an activity, claim, exposure, or matter—for example, sending a notification.
Event Message	Handle communication with integrated external applications.
Exception	Specify an action to take under certain circumstances. For example, if no one touches a claim for a certain number of days, then create a review activity.
Initial Reserve	Determine the initial financial reserves that to create for a new exposure.
Loaded	Trigger automatic actions whenever you load a new claim using the FNOL import interface.
Postsetup	Trigger automatic actions whenever you add a new claim, activity, exposure, matter, or transaction within ClaimCenter. These rules run after the Assignment and Workplan rules complete their execution.
Presetup	Trigger automatic actions whenever you add a new claim, exposure, matter, or transaction within ClaimCenter. These rules run prior to the execution of rules in the Segmentation, Assignment, and Workplan rule set categories.
Preupdate	Trigger automatic actions whenever a claim or other high-level entity changes.
Reopened	Trigger automated actions on reopening a claim or exposure.
Segmentation	Categorize a new claim or an exposure based on complexity, severity of damage, or other attributes.
Transaction Approval	Verify that a user has the appropriate authorization to submit a financial transaction.

Rule set category	Rules
Validation	Check for missing information or invalid data on `Activity`, `Claim`, `Contact`, `Exposure`, `Group`, `Matter`, `Person`, `Policy`, `Region`, `RIAgreement`, `RITransaction`, `ServiceRequest`, `Transaction`, and `User` objects.
Workplan	Add initial activities to a claim, exposure, or matter as a checklist of work that various people need to do on the claim.

Approval routing

Approval routing Gosu rules route an approval activity for a transaction that requires approval. ClaimCenter uses approval routing rules to determine which person to assign the approval activity to. Typically, a supervisor is the approving authority. The transaction set and its transactions remain in a *pending approval* state until someone approves the activity.

See also

- "Approval rules and Transaction approval rules" on page 36
- "BulkInvoice Approval Assignment Rules" on page 38

Approval rules and Transaction approval rules

The Approval Rules rule set and the rule sets in the Transaction Approval rule set category form a pair. ClaimCenter calls these rule sets sequentially.

- ClaimCenter calls rule sets in the Transaction Approval rule set category whenever you submit any kind of financial transaction.
- ClaimCenter calls the Approval Rules rule set if the outcome of transaction approval is Yes or if the transaction exceeds the transaction authority limit, causing the transaction to need approval.

IMPORTANT: Avoid creating approval Gosu rules that generate an infinite loop. Suppose, for example, that a rule first assigns the transaction to a supervisor for approval, who subsequently approves it. Then, the rule assigns the transaction approval to someone who does not have the authority to approve the transaction. ClaimCenter runs the rule set again, and again reassigns the approval to the supervisor, generating a never-ending loop.

See also

- "Approval rules for TransactionSet" on page 36
- "Methods to set the approver of a transaction" on page 37
- "Transaction Approval" on page 96

Approval rules for TransactionSet

In the base configuration, Guidewire provides the following sample `TransactionSet` Approval Gosu rules. To work with these rules, navigate the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **ApprovalRouting** > **ApprovalRules**

ARR01000 - Western Regional Auto Supervisors

This rule sets the approving user to the supervisor of the Western Auto Group if the last approval was by a supervisor of one of the subordinate teams. The rule then exits the rule set. If all the conditions are false, the next rule executes.

This rule is an example of how to encapsulate complex logic in reusable, external class methods instead of implementing all logic inside the rule.

ARR02000 - Try to assign to group supervisor

This rule calls the `transactionSet.approveByGroupSupervisor` method. This method first verifies that ClaimCenter has not yet assigned the associated claim or exposure. It also verifies that the current user making the request is not the group supervisor. If so, the method then assigns approval to the group supervisor and the rule exits the rule set. If not, the method returns `false` and the next rule executes.

ARR03000 - Default Rule

This rule runs if neither of the previous rules succeeded in making an assignment to a group supervisor. The rule assigns approval to a higher level supervisor.

Methods to set the approver of a transaction

The following list describes the main methods used to set the person who can approve the transaction. These methods are available on the `TransactionSet` object.

Note: The `BulkInvoice` object has the same set of methods available to set the person who can approve the bulk invoice.

Method	Description
`approveByGroupSupervisor`	Use to assign an approval activity to a group supervisor based on the number of exposures associated with the transaction: • If the transaction set has transactions associated with only one exposure, the method assigns the approval activity to the supervisor of the group assigned to the exposure. • If the transaction set has transactions associated with multiple exposures, the method assigns the approval activity to the supervisor of the group assigned to the associated claim. In either case, if the supervisor of the group is the same as the user requesting approval, then the activity remains unassigned. **Note:** If you use this method, you must explicitly handle this last case.
`approveByUserSupervisor`	Use to assign an approval activity to a group supervisor based on the user who submitted the transaction: • If the submitting user is in the owning group and is not the supervisor, then approval goes to the group supervisor. • If the submitting user is the group supervisor, then ClaimCenter escalates the approval to the group supervisor of the next group up the group hierarchy. • If the submitting user is a group supervisor someplace higher in the hierarchy, then the approval escalates to the supervisor of the next group up the hierarchy. • If the submitting user is in *one and only one* group, then the approval goes to the supervisor of that group. Otherwise, approval goes to the supervisor of the owning group. Finally, if ClaimCenter worked its way through the previous conditions to the supervisor of the root group without making an assignment, then ClaimCenter assigns the approval to user *defaultowner*.
`setApprovingUser`	Use to assign an approval activity to a specified user. You must provide a user and group as parameters. The method checks whether the user is retired or inactive (`user.credential.Active != true`) and returns `false` if either of these conditions exist. It also returns `false` if the user or group is missing. It returns `true` otherwise. **IMPORTANT** Call this method only from within the Approval Routing rule set.

Example TransactionSet Approval rule

In the following example Transaction Set Approval rule, the rule assigns the approval to the supervisor, unless the supervisor is the one who entered the transaction.

```
USES:
uses gw.api.system.CCLoggerCategory
```

```
CONDITION (transactionSet : entity.TransactionSet):
return true

ACTION (transactionSet : entity.TransactionSet, actions : gw.rules.Action):
var exp = transactionSet.AllTransactions*.Exposure
// Verify that the supervisor has not already approved this TransactionSet

if (not exists( act in transactionSet.Claim.Activities where (act.TransactionSet == transactionSet
        and (act.ActivityPattern.Code == "approve_reserve_change"
        or act.ActivityPattern.Code == "approve_payment" )
        and act.Approved == true and act.AssignedUser == exp[0].AssignedGroup.Supervisor ))) {
  if ( transactionSet.approveByGroupSupervisor() ) {
    CCLoggerCategory.ASSIGNMENT.debug(" Assign to supervisor")
    CCLoggerCategory.ASSIGNMENT.debug(
      transactionSet.AllTransactions*.Exposure[0].AssignedGroup.Supervisor.Contact.DisplayName )
    actions.exit()
  }
}
```

Note: In actual practice, Guidewire recommends that you make `String` values into display keys or constant variables, as appropriate.

BulkInvoice Approval Assignment Rules

The `BulkInvoice` Approval Assignment Rules and the `BulkInvoice` Approval Rules rule sets form a pair. These Gosu rules work together similarly to the rule sets in the Approval Routing rule set category and the Transaction Approval rule set category. ClaimCenter calls these rule sets sequentially.

ClaimCenter runs the `BulkInvoice` Approval Assignment rule set for each bulk invoice to determine to whom to assign it for approval. See “Bulk Invoice Approval rules” on page 48:

In the base configuration, Guidewire provides the following sample Gosu rules in this rule set. To see them, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **ApprovalRouting** > **BulkInvoiceApprovalAssignmentRules**

You can customize these rules to meet your business needs.

ARB01000 - Western Regional Auto Supervisors

This rule sets the approving user to the supervisor of the Western Auto Group if the last approval was by a supervisor of one of the subordinate teams. The rule then exits the rule set. If all the conditions are false, the next rule executes.

This rule is an example of how to encapsulate complex logic in reusable, external class methods instead of implementing all logic inside the rule.

ARB02000 - Try to assign to group supervisor

This rule calls `bulkInvoice.approveByGroupSupervisor`. This method first verifies ClaimCenter has not yet assigned the associated claim or exposure. It then verifies that the current user making the request is not the group supervisor. If so, the method then assigns approval to the group supervisor and the rule exits the rule set. If not, the method returns `false` and the next rule executes.

ARB03000 - Default Rule

This rule runs if neither of the previous rules succeeded in making an assignment to a group supervisor. The rule assigns approval to a higher level supervisor.

Archive

Archiving is the process of moving a closed claim and associated data from the active ClaimCenter database to a document storage area. Archived claims occupy less space in the active database. You can still search for and retrieve archived claims.

See also

- *Application Guide*

Understanding archive events

Whenever ClaimCenter creates a claim, it also creates a `ClaimInfo` entity. If a `ClaimInfo` entity changes, including during claim archiving, retrieval, claim exclusion, and archive failure, ClaimCenter generates a `ClaimInfoChanged` event. To determine the archive state of a claim, use `ClaimInfo.ArchiveState`. This field, from the `ArchiveState` typelist, can have the following possible values in the base configuration:

Archive state	Meaning
archived	ClaimCenter has finished archiving the claim.
retrieving	ClaimCenter has marked the claim for retrieval.

The following actions set the archive state to one of these values:

- Archiving a claim generates a `ClaimInfoChanged` event, and sets `ClaimInfo.ArchiveState` to `archived`.
- Retrieving an archived claim generates a `ClaimInfoChanged` event, and sets `ClaimInfo.ArchiveState` to `retrieving`.

A null `ArchiveState` indicates that the claim is not in an archive state. This can mean one of the following:

- ClaimCenter has never archived the claim.
- ClaimCenter successfully retrieved the claim from the archive.

Besides the `ArchiveState` events, the `History` array of a claim receives a new entry each time that ClaimCenter archives or retrieves the claim. Archive Gosu rules that reference either these history entries or a `ClaimInfoChanged` event can distinguish archived and retrieved claims from other claims.

About skipped and excluded claims in archiving

Through the Archive Gosu rules, you can do the following:

Skip a claim	Skipping a claim during archiving makes that claim temporarily unavailable for archiving during this particular archiving pass. To skip a claim, call the following method on the claim and provide a reason: `claim.skipFromArchiving(String)` **IMPORTANT:** Calling this method on a claim terminates rule set execution.
Exclude a claim	Excluding a claim from archiving makes that claim unavailable for archiving during this and future archiving passes. The claim becomes unavailable for archiving on a semi-permanent basis. To exclude a claim from archiving, call the following method on the claim and provide a reason for the exclusion: `claim.reportArchiveProblem(String)` Calling this method on a claim does not terminate rule set execution.

Detecting Claims Excluded from Archiving

There are several cases in which ClaimCenter excludes claims from archiving.

Archiving Rule Conditions

ClaimCenter can exclude a claim from the set of claims to archive due to some condition related to the archiving Gosu rules. For example, a rule condition determines that the claim did not meet a certain business-related condition. In this case, ClaimCenter populates the following `ClaimInfo` properties, which you can use to view more details of the failure.

`ClaimInfo.ExcludeFromArchive`	If `true`, ClaimCenter excluded this claim from the archiving process.

`ClaimInfo.ExcludeReason`	Provides details on why ClaimCenter excluded this claim from the archiving process.

Archiving Process Failure

ClaimCenter can exclude a claim from the set of claims to archive due to the failure of the archive process itself. For example, there was a foreign key violation. In this case, ClaimCenter populates the following `ClaimInfo` properties, which you can use to view more details of the failure.

`ClaimInfo.ArchiveFailure`	A foreign key to an `ArchiveFailure` object that provides a one-line summary of why the claim failed the archive process
`ClaimInfo.ArchiveFailureDetails`	A foreign key to an `ArchiveFailureDetails` object that provides the full stack trace related to the failure.

Managing Claims Skipped or Excluded from Archiving

You can view information about skipped and excluded claims from within ClaimCenter by entering `ALT+SHIFT+T` to open Server Tools. then navigating to the **Info Pages** > **Archive Info** screen.

> **Note:** To see this screen, you must enable archiving. See the *Administration Guide* for details. To access the Server Tools, you must log into ClaimCenter using an administrative account.

The **Archive Info** screen contains three sections. The top section is an **Overview** section. Underneath that section is a section with archive source information showing availability of the store, of retrieval, and of delete functionality. The bottom section summarizes archives by data model version.

The **Overview** section lists:

- Total number of claims skipped by the archive process
- Number of claims excluded because of rules
- Number of claims excluded because of failure

For skipped and excluded claims, you can investigate each individual item. You can also reset any excluded claim so that the archive process attempts to archive that claim the next time the archive process runs.

See also

- *Administration Guide*

Default Group Claim Archiving rules

You use the Gosu rules in Default Group Claim Archiving rule set to prevent ClaimCenter from archiving an otherwise eligible claim. ClaimCenter invokes this rule set immediately before archiving the claim, after the claim meets certain internal criteria for archiving.

In the base configuration, Guidewire provides the following sample Gosu rules in this rule set. To see these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Archive** > **DefaultGroupClaimArchivingRules**

You must customize the archiving rules to meet your business needs.

ARC01000 - Claim State Rule

If the claim is open, ClaimCenter skips the claim.

ARC03000 - Bulk Invoice Item State Rule

If the claim has links to a bulk invoice item with a status of `draft`, `notvalid`, `approved`, `checkpendingapproval`, or `awaitingsubmission`, ClaimCenter skips this claim.

The intent is to not force a user to retrieve the claim if there is an item that needs escalation. An item with status In Review or Rejected does not prevent ClaimCenter from archiving the claim. The claim can retain one of these statuses long after the escalation and clearing of a bulk invoice item.

ARC04000 - Open Activities Rule

If the claim has open activities, ClaimCenter skips the claim.

Note: The Work Queue writer does not typically mark a claim with open activities for archiving. However, an activity can open on the claim between the time the worker queued the claim for archive and the time that the archive batch process actually processes the claim.

ARC05000 - Incomplete Review Rule

If a claim has a vendor review that is incomplete, ClaimCenter skips the claim.

ARC06000 - Unsynced Review Rule

If a claim has a vendor review that is not in synchronization with ContactManager, ClaimCenter skips the claim.

ARC07000 - Transaction State Rule

If a claim has un-escalated or un-acknowledged transactions, ClaimCenter skips the claim.

You can add your own archive-related rules to this rule set and modify the existing sample rules to create rules that reflect your unique business conditions. For example, you can write rules to do the following:

- If a claim is sensitive or had restricted access control, do not archive it.
- If a claim meets a certain condition, such as having medical payments over some amount, do not archive it.
- If a claim has a claimant who has other open claims pending, do not archive it.

Claim Eligible for Archive rules

Guidewire provides the following Gosu rules that affect the eligibility of a claim to for archiving. To see the rules, navigate to the following location in the Studio **Project** window:

Closed > **ClaimClosed** > **Reopened** > **ClaimReopened**

CCL04000 - Set archive eligibility date

If you close a claim, this rule sets the date on which a claim is eligible for archiving.

CRO04000 - Clear archive eligibility date

If you reopen a claim, this rule clears the archive eligibility date.

The two rules work as a pair. You must enable or disable the two rules in conjunction with each other.

See also

- “Claim Closed rules” on page 49
- “Claim Reopened rules” on page 90

Assignment

Certain business entities are *assignable*. For these entities, it is possible to determine—either through the use of assignment rules or through the use of Gosu assignment methods—the party responsible for that entity. In the base configuration, ClaimCenter defines the following entities as assignable:

- `Activity`
- `Claim`
- `Exposure`
- `Matter`
- `ServiceRequest`
- `UserRoleAssignment`

Note: `UserRoleAssignment` does not have a rule set defined for it in the base configuration. It is in this list because it is assignable.

If an entity is not assignable in the base configuration, you can extend it and make the new entity assignable. To make an entity assignable, it must implement either the `Assignable` or the `CCAssignable` delegate class.

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 rule set or simply as Gosu code, such as in a class or enhancement.

In general, ClaimCenter 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. ClaimCenter runs the assignment rules until it either completes the assignment or runs out of rules to run.

To work with the Assignment rule sets, navigate in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment**

See also

- For information on assignment and assignable entities in Guidewire ClaimCenter, see “Assignment in ClaimCenter” on page 131.

Default Group Activity Assignment rules

The Gosu Default Group Activity Assignment rules determine how ClaimCenter assigns activities within a group.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **DefaultGroupActivityAssignmentRules**

In the base configuration, Guidewire provides the following rules.

DGA01000 - SI Assign claim review activity to supervisor

If the activity is for a special investigation review (`SI_review`), the rule assigns the review activity to the current group’s supervisor. If the assignment is successful, the rule exits the rule set.

DGA02000 - SI Assign Escalation Activity to named SIU user

If the activity is for a special investigation escalation (`SIU_escalation`), the rule assigns the review activity to a user who has the SIU Investigator role. If the assignment is successful, the rule exits the rule set.

DGA03000 - SI Default SI Escalation activity routing

If the activity is for a special investigation escalation (`SIU_escalation`), assignment to a user was not successful in the previous rule. This rule assigns the activity to a user in the current group by round-robin. If the assignment is successful, the rule exits the rule set.

DGA04000 - Assign reinsurance review activity to reins user

If the activity is for a reinsurance review or synchronization, and if the claim specifies a reinsurance manager, attempt to assign it to the reinsurance manager. If there is no reinsurance manager on the claim, assign the activity by round-robin to the group assigned to the claim associated with the activity. If the assignment is successful, the rule exits the rule set.

DGA05000 - Assign to Subro group

If the activity is to determine the opportunity for a recovery (`subro_check`), the rule assigns the activity to the user who is the subrogation owner. If the assignment is successful, the rule exits the rule set.

DGA06000 - Test Assignment by Proximity

The rule first determines if it can use the geocoding feature locally and if there is a primary address for the claimant. If so, the rule searches for a nearby user that it can assign the activity to. In the base configuration, the search radius is five miles and the default user name is `"b"`. If the assignment is successful, the rule exits the rule set.

DGA07000 - Assign activity to named user

This rule determines if the activity for assignment has a user role associated with it and if there is already a user with that role on the claim. If so, the rule assigns the activity to a user with the proper role and then exits the rule set.

DGA08000 - Default - if users in group

This rule determines if there is at least one user in the group assigned to the activity. If so, the rule assigns the activity by round-robin and then exits the rule set.

DGA09000 - Default

If there is no user assigned to the activity, this rule assigns it to the group's supervisor for manual assignment to a user.

DGA10000 - Assignment by Proximity and Attribute Example

Guidewire disables this rule in the base configuration by default. It is a sample rule that shows how to use a custom attribute and proximity search to find a user and assign an activity to that user. The example also uses logger categories to log the results.

See also

- "Detecting claim fraud" on page 167
- *Application Guide*

Default Group Claim Assignment rules

The Gosu Default Group Claim Assignment rules determine how ClaimCenter assigns claims within a group.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **DefaultGroupClaimAssignmentRules**

In the base configuration, Guidewire provides the following rules.

DGC00500 - Balanced workload within group

If enabled, the rule first determines if weighted workload assignment is also enabled. If so, it looks for users in the current group that are members and who are available for work, those who are active and not on vacation. If the rule fines a user, it assigns the claim by weighted workload. Whether assignment is successful or not, the rule exits the rule set.

DGC01000 - Default - if users in group

If there are users in the group assigned to the activity, assign the claim by round-robin. If the assignment is successful, the rule exits the rule set.

DGC02000 - Default - if no users in group

If the claim is still not assigned to a user, this rule assigns it to the group's supervisor for manual assignment to a user.

Default Group Exposure Assignment rules

The Gosu Default Group Exposure Assignment rules determine how ClaimCenter assigns exposures within a group. These rules apply only if the claim owner does not assign the exposure.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **DefaultGroupExposureAssignmentRules**

In the base configuration, Guidewire provides the following rules.

DGE00500 - Balance workload within group

If enabled, the rule first determines if weighted workload assignment is enabled. If so, it looks for users in the current group that are members and who are available for work, those who are active and not on vacation. If the rule finds a user, it assigns the exposure by weighted workload. Whether assignment is successful or not, the rule exits the rule set.

DGE01000 - Group exposures with same adjuster

This rule determines if anyone in the assigned group has already been assigned an exposure on this claim. If so, the rule assigns that user to the exposure, too. If the assignment is successful, the rule exits the rule set.

DGE02000 - Default - users in team

If there is no assigned user for the exposure, this rule determines if the assigned group for the exposure contains any users. If so, the rule assigns the exposure by round-robin.

DGE03000 - Default - no users in team

If there is still no assigned user for the exposure, this rule assigns the exposure to the group's supervisor for manual assignment to a user.

Default Group ServiceRequest Assignment rules

The Gosu Default Group Service Request Assignment rules determine how ClaimCenter assigns service requests to users within a group.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **DefaultGroupServiceRequestAssignmentRules**

In the base configuration, Guidewire provides the following rule.

DGS01000 - Default

This rule assigns the service request to the claim's assigned user and to the default group for that user. The rule exits regardless of whether the assignment is successful.

Global Activity Assignment rules

The Gosu Global Activity Assignment rules determine how ClaimCenter first assigns activities to a group. After these rules run, the Default Group Activity Assignment rules run to assign the activity to a user in the group.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **GlobalActivityAssignmentRules**

In the base configuration, Guidewire provides the following rules.

GAA01000 - SI - Assign claim review to claim owner's group

If the activity is for a special investigation review (`SI_review`), the rule assigns the review activity to the claim's assigned group. If the assignment is successful, the rule exits the rule set.

GAA02000 - SI - Assign SIU escalation activity to SIU group

If the activity is for a special investigation escalation (`SIU_escalation`), in the base configuration, the rule assigns the review activity to an SIU group:

1. The group of the SIU Investigator on the claim, if there is such a user.
2. The Western SIU group if the first assignment was not possible.

If the assignment is successful, the rule exits the rule set.

GAA03000 - Subro - Assign to Subro group

If the activity is to determine the opportunity for a recovery (`subro_check`), in the base configuration, the rule assigns the review activity to a subrogation group:

1. The group of the Subrogation Owner on the claim, if there is such a user.
2. The HQ Subro Unit group if the first assignment was not possible.

If the assignment is successful, the rule exits the rule set.

GAA04000 - Assign reinsurance review to Reinsurance group

If the activity is for a reinsurance review or synchronization, attempt to assign it to the reinsurance manager and group. If a reinsurance manager does not exist on the claim, attempt to assign the activity by round-robin to a user in the Reinsurance group. If the assignment is successful, the rule exits the rule set.

GAA05000 - Homeowners PolicyType

If the policy for the claim that is related to the activity is a Homeowners policy, execute the following child rules.

GAA05100 - Damaged items list

If the activity pattern is for damaged items, attempt to assign it to the owner of the claim or exposure, whichever is relevant for the activity. If the assignment is successful, the rule exits the rule set.

GAA05200 - Contact insured about living expenses

If the activity pattern is to contact the insured about living expenses, attempt to assign it to the owner of the claim or exposure, whichever is relevant for the activity. If the assignment is successful, the rule exits the rule set.

GAA05300 - Get property inspected

If the activity pattern is to inspect the damage property, attempt to assign it to the owner of the claim or exposure, whichever is relevant for the activity. If the assignment is successful, the rule exits the rule set.

GAA05400 - Get claimant medical reports

If the activity pattern is to get medical report from the claimant, attempt to assign it to the owner of the claim or exposure, whichever is relevant for the activity. If the assignment is successful, the rule exits the rule set.

GAA05500 - Verify coverage

If the activity pattern is to verify the coverage, attempt to assign it to the owner of the claim or exposure, whichever is relevant for the activity. If the assignment is successful, the rule exits the rule set.

GAA06000 - Activity Exceptions

This rule serves as a folder to group child rules that process assignment of some special activities.

GAA06100 - Legal

If the activity pattern is to initiate a legal review, attempt to assign the activity in the following order:

1. Attempt to assign the activity to an assigned claim user role matching that of the activity, and to the group associated with that user. If successful, exit the rule set.
2. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the insured party's primary address. If successful, exit the rule set.
3. If the previous assignment was not successful, attempt to assign the group by round-robin, starting with the group type associated with the activity pattern and including subgroups if necessary. If successful, exit the rule set.

GAA06200 - Independent appraisal

If the activity pattern is to initiate an independent appraisal, attempt to assign the activity in the following order:

1. Attempt to assign the activity to an assigned claim user role matching that of the activity, and to the group associated with that user. If successful, exit the rule set.
2. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the loss location. If successful, exit the rule set.
3. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the primary address of the claimant on the exposure. If successful, exit the rule set.
4. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the insured party's primary address. If successful, exit the rule set.

GAA06300 - Nurse Visit

If the activity pattern is to schedule a visit from a nurse, attempt to assign the activity in the following order:

1. Attempt to assign the activity to an assigned claim user role matching that of the activity, and to the group associated with that user. If successful, exit the rule set.
2. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the insured party's primary address. If successful, exit the rule set.

GAA06400 - Property inspection

If the activity pattern is to schedule a property inspection, attempt to assign the activity in the following order:

1. Attempt to assign the activity to an assigned claim user role matching that of the activity, and to the group associated with that user. If successful, exit the rule set.
2. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the insured party's primary address. If successful, exit the rule set.

GAA06500 - Police

If the activity pattern is to obtain a police report, attempt to assign the activity in the following order:

1. Attempt to assign the activity to an assigned claim user role matching that of the activity, and to the group associated with that user. If successful, exit the rule set.
2. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the loss location. If successful, exit the rule set.
3. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the primary address of the claimant on the exposure. If successful, exit the rule set.
4. If the previous assignment was not successful, attempt to assign the activity to a group in a location near the insured party's primary address. If successful, exit the rule set.

GAA06600 - Assignment Review

If the activity pattern is to review the assignment of the activity, if there is an assigned group for the claim, assign the activity to the group's supervisor.

GAA07000 - Default

If the activity is still unassigned to a group, this rule assigns it the owner of the claim or exposure, as appropriate.

Global Claim Assignment rules

The Gosu Global Claim Assignment rules determine how ClaimCenter first assigns claims to a group.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **GlobalClaimAssignmentRules**

In the base configuration, Guidewire provides the following rules.

GCA00010 - Claim

This rule serves as a folder to group its child rules.

GCAA00010 - Auto

If the claim loss type is Auto, this rule uses the loss type and claim segment to get the top two matching group type choices for assigning this claim. The rule then tries the following assignment options, in order, and exits the rule set if one of them succeeds:

1. Primary group type, near the claim's loss location
2. Secondary group type, near the claim's loss location
3. Primary group type, near the insured party's primary address
4. Secondary group type, near the insured party's primary address

GCAP00010 - Property

If the claim loss type is property (PR), this rule performs the same steps to assign the claim as the previous rule.

GCAL00010 - Liability

If the claim loss type is liability (GL), this rule performs the same steps to assign the claim as the previous rule.

GCAWC0010 - WC

If the claim loss type is Workers' Compensation (WC), this rule performs the same steps to assign the claim as the previous rule.

The rule does further processing if the attempts to assign the claim all fail. For example, there is neither a primary address for the insured party nor a loss location. The rule then tries to assign the claim to the first group to which the claim creator belongs.

GCATR0040 - Travel

If the claim loss type is travel (TRAV), this rule performs the same steps to assign the claim as the previous rule.

For this rule, the first four assignment attempts often fail. For example, the Travel group's location rarely matches the loss location, and it often does not match the insured party's primary address. As with the previous rule, the rule then tries to assign the claim to the first group to which the claim creator belongs.

Global Exposure Assignment rules

The Gosu Global Exposure Assignment rules determine how ClaimCenter first assigns exposures to a group.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **GlobalExposureAssignmentRules**

In the base configuration, Guidewire provides the following rules.

GEA010000 - Auto Claims

This rule determines if the exposure has a loss type of Auto and that there is an exposure on the claim that has either of the following properties:

- The loss party is the insured party on the claim.
- The exposure type is not bodily injury damage.

If the exposure matches these criteria, it is processed by the following child rule. Otherwise, if the claim has only third-party injury exposures, this claim was already assigned to a liability adjuster, and all exposures go to that adjuster.

GEA01100 - 3rd-party injury exposures

This rule determines if the exposure has both the following properties:

- The loss party is third party
- The exposure type is bodily injury damage

If so, the rule uses the claim's loss type and the exposure's segment to get the top two matching group type choices for assigning this claim. The rule then tries the following assignment options, in order, and exits the rule set if one of them succeeds:

1. Primary group type, near the insured party's primary address
2. Secondary group type, near the insured party's primary address

GEA02000 - Non-auto claims

This rule determines if the exposure has a loss type other than Auto. If so, the child rule executes.

GEA02100 - Default

This rule assigns the exposure to the assigned owner and group for the claim. It then exists the rule set.

GEA03000 - Default

If the exposure is still not assigned, this rule assigns the exposure to the assigned owner and group for the claim. It then exits the rule set.

Global Matter Assignment rules

The Gosu Global Matter Assignment rules determine how ClaimCenter first assigns matters to a group.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **GlobalMatterAssignmentRules**

In the base configuration, Guidewire provides the following rule.

GMA01000 - Default

This rule assigns the matter to the assigned owner and group for the claim. It then exits the rule set.

Global ServiceRequest Assignment rules

The Gosu Global Service Request Assignment rules determine how ClaimCenter first assigns service requests to a group.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Assignment** > **GlobalServiceRequestAssignmentRules**

In the base configuration, this rule set has the following rule.

GSA01000 - Default

This rule assigns the service request to the assigned owner and group for the claim. It then exits the rule set.

BulkInvoice

In ClaimCenter, the approval process for a bulk invoice is much the same as the approval process for transaction sets. However, there is one critical difference: There are no authority limits for bulk invoices, although authority limits are verified on the individual checks in the bulk invoice. The Bulk Invoice Approval rules alone control whether a bulk invoice itself requires approval.

The BulkInvoice Approval Rules and the BulkInvoice Approval Assignment Rules rule sets form a pair. These Gosu rules work together similarly to the rule sets in the Approval Routing and Transaction Approval rule set categories. ClaimCenter calls these rule sets sequentially.

See also

- “BulkInvoice Approval Assignment Rules” on page 38
- “Approval routing” on page 36
- “Transaction Approval” on page 96
- For information on bulk invoice web service APIs, validating bulk invoices, and bulk invoice processing performance, see the *Integration Guide*.

Bulk Invoice Approval rules

ClaimCenter runs the BulkInvoice Approval rules for each bulk invoice to determine whether the bulk invoice requires approval.

In the base configuration, ClaimCenter provides a sample approval rule for bulk invoices that always requires approval. Navigate in the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **BulkInvoiceApproval** > **BulkInvoiceApprovalRules**

ClaimCenter provides a single rule in this rule set in the base configuration, BIA010000 - Always Require Approval. Guidewire disables this rule in the base configuration. This rule contains code showing how to require approval each time a bulk invoice is submitted.

Writing an approval rule for a bulk invoice is similar to writing an approval rule for a transaction set. Some important differences to keep in mind, however, are the following:

- For a typical `TransactionSet` approval rule, the rule condition requires verification of the subtype of the transaction set, such as `TransactionSet.subtype == "checkset"`. This verification is required because ClaimCenter passes all created transaction sets, regardless of subtype, to the same approval rules. In contrast, the bulk invoice approval rules have no need for subtype verification in their rule conditions. The only implicit entity that ClaimCenter ever passes to the bulk invoice approval rules is a bulk invoice.

- Bulk invoice approval rules have an implicit reference to the bulk invoice submitted for approval. You can access this reference in a rule condition or action block through the `BulkInvoice` entity. This behavior is different from the rules in the Transaction Approval rule set category, which have an implicit reference to the transaction set being approved. See “Transaction Approval” on page 96.

Automatic approval of bulk invoices

If none of the approval rules determine that the bulk invoice requires approval, ClaimCenter automatically and immediately approves the bulk invoice. ClaimCenter then does the following:

- Marks the bulk invoice as Approved and sets the approval date to the current day.
- Transitions the bulk invoice status to Pending upon bulk invoice item validation.
- Transitions all the bulk invoice items that have a status of Draft to Approved status.

Closed

The Closed rule sets fire whenever you close an activity, claim, exposure, or matter. They fire immediately on closing one of these entities, not before. You can use these Gosu rules to send a notification or to generate additional activities.

To work with this rule set, navigate to the following location in the Studio **Project** window:sl

configuration > **config** > **Rule Sets** > **Closed**

The following examples, which work in conjunction with one another, show some aspects of how to use these rule sets:

- “Example – closing a claim from exposure closed rule” on page 51
- “Example – closing claim activities while closing a claim” on page 50

Activity Closed rules

ClaimCenter executes the Activity Closed rules immediately after an activity closes. Use these Gosu rules to create follow-up actions and activities.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Closed** > **ActivityClosed**

In the base configuration, this rule set contains the following sample rule:

CAC01000 - Salvage
First verifies that an activity to salvage a vehicle is complete. If so, the rule sets the vehicle recovery date to the current date.

Claim Closed rules

These Gosu rules execute immediately after a claim closes. Use these rules to create follow-up actions and activities.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Closed** > **ClaimClosed**

In the base configuration, this rule set has the following rules.

CCL01000 - Notify external systems
Notifies external systems that the claim is closed.

CCL02000 - Suspend AutoSync for Related Contacts
Any `Contact` on a closed claim must have its AutoSync status set to Suspended to prevent attempts by ContactManager and ClaimCenter to update the closed claim. This action helps facilitate the synchronization of contact information between ClaimCenter and ContactManager. See also the *Contact Management Guide*.

Guidewire provides the following Gosu rule (CCL03000) that affects the purging of archived claims. To see this rule, navigate to the following location in the Studio **Project** window:

Closed > **ClaimClosed** > **Reopened** > **ClaimReopened**

CCL03000 - Sample rule to set purge date

Guidewire disables this rule in the base configuration. The rule sets a purge date on the claim to seven years from the closing of the claim file. If you enable this rule, you must also enable the following rule:

- Claim Reopened Rules: CRO03000 - Sample rule to remove purge date.

These Gosu rules are samples only and are not active in the base configuration. The two rules work as a pair. You must enable or disable the two rules in conjunction with each other.

CCL04000 - Set archive eligibility date

Sets `claim.DateEligibleForArchive` to a date created by adding the value of configuration parameter `DaysClosedBeforeArchive` to the current system date.

This rule works in conjunction with the claim reopened rule CRO04000 - Clear archive eligibility date. See “Claim Eligible for Archive rules” on page 41.

Example – closing claim activities while closing a claim

The following example Closed rule, which you would add to the Claim Closed rule set, runs whenever ClaimCenter closes a claim. The rule works in conjunction with the “Example – Closing a Claim from Exposure Closed Rule” example, which closes a claim whenever the last exposure closes. This example rule closes all open activities associated with the claim that are not available to closed claims.

```
CONDITION (claim : entity.Claim):
return true

ACTION (claim : entity.Claim, actions : gw.rules.Action):
for( act in claim.Activities) {
  if (act.Status == "open" and act.ActivityPattern != null
      and act.ActivityPattern.ClosedClaimAvlble == false) {
    act.complete()
    if (act.Description == null) { act.Description = "Closed by ClaimCenter." }
    else { act.Description = act.Description + "Closed by ClaimCenter." }
  }
}
```

Note: In actual practice, Guidewire recommends that you make `String` values into display keys or constant variables, as appropriate.

Exposure Closed rules

The Exposure Closed rules execute immediately after the rules in the Exposure Closed Validation Rules rule set run. Use these Gosu rules to take automated actions on the closure of the exposure.

To work with this rule set, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Closed** > **ExposureClosed**

In the base configuration, this rule set has the following rule.

CEX01000 - Notify external systems

Notifies external systems that the exposure is closed.

See also

- “Exposure Closed Validation rules” on page 109

Example – closing a claim from exposure closed rule

The following example Closed rule, which you would add to the Exposure Closed rule set, runs whenever ClaimCenter closes an exposure. This rule closes the claim file from the Exposure Closed Rules rule set if all other exposures on the claim are closed.

```
CONDITION (exposure : entity.Exposure):
return !exists (exp in exposure.Claim.Exposures where exp.State !="closed")

ACTION (exposure : entity.Exposure, actions : gw.rules.Action):
exposure.Claim.close( "completed", "Claim number " + exposure.Claim.ClaimNumber
      + " closed by ClaimCenter because last open exposure was closed." )
```

Note: In actual practice, Guidewire recommends that you make `String` values into display keys or constant variables, as appropriate.

Matter Closed rules

A matter is the set of data organized around a single lawsuit or potential lawsuit. This data includes information on the attorneys involved, the trial details, and the lawsuit details. These rules execute immediately after a matter closes. Use these Gosu rules to take automated actions on the closure of a matter.

To work with this rule set, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Closed** > **MatterClosed**

In the base configuration, this rule set has the following rule.

CMC01000 - Set Litigation Status

This rule first ensures that all matters on the associated claim have a status of Closed. If so, this rule sets the litigation status on the associated claim to `"complete"`.

Event message

IMPORTANT: ClaimCenter runs the Event Message rules as part of the database bundle commit process. Use these Gosu rules only to create integration messages.

In the base configuration, there is a single rule set, Event Fired, in the Event Message rule category. To see these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired**

The rules in this rule set:

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

ClaimCenter calls the Event Fired rules if an entity involved in a bundle commit triggers an event of interest to a message destination. A message destination registers an event to indicate its interest in that event.

As part of the event processing:

- ClaimCenter runs the rules in the Event Fired rule set a single time for every event of interest to a message destination.
- ClaimCenter runs the Event Fired rule set a single time for each destination that is listening for that particular event. Thus, it is possible for ClaimCenter to run the Event Fired rules sets multiple times for each event, one time for each destination interested in that event.

See also

- *Integration Guide*

About messages and messaging events

The topics that follow describe how messaging and message events work.

Messaging events

ClaimCenter automatically generates standard events for most top-level objects. In general, event generation occurs for any addition, modification, removal, or retirement of a top-level entity. For example, ClaimCenter 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)
```

IMPORTANT: Do not create custom code that generates update events for any entity type that can possibly be part of the archive bundle.

See also

- *Configuration Guide*
- *Integration Guide*

Message destinations and message events

You use the Studio Messaging editor to link one or more events to a message destination:

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

Using the Studio Messaging editor, it is possible to associate (register) one or more events with a particular message destination. For example, in the base configuration, ClaimCenter associates the following events with the `metro` message destination (`DestID` 67):

- `MetroReportAdded`
- `MetroReportChanged`

If you modify or add a `MetroReport` object, ClaimCenter generates the relevant event. Then, during a bundle commit of the `MetroReport` object, ClaimCenter notifies any Event Message rule interested in the event of the occurrence of the event. You can 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, which encapsulates all the necessary behavior for delivering a notification to an external system, uses three different plugin interfaces to implement the destination. Each plugin handles different parts of what a destination does, as follows:

- 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 plugin implementation classes in Studio first in the Plugins Registry editor. Whenever you first create a new plugin registry, Studio prompts you for a name for the registry and a plugin interface name. Use that plugin registry name in the Messaging editor in Studio to register each destination.

Note: You must register your plugin in two different editors in Studio, first in the Plugin Registry, and then in the Messaging editor.

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

Generating messages

Use the `messageContext.createMessage` method 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 a claim changes.

```
var session = messageContext.SessionMarker
var claim = messageContext.Root as Claim

// Create a message for this claim.
var msg = messageContext.createMessage("Message for ClaimChanged")
msg.putEntityByName( "assigneduser", claim.AssignedUser )

// Add assigned user to session map, so it's available in the session.
session.addToTempMap( "assigneduser", claim.AssignedUser )

// Create a message for each exposure.
for (exposure in claim.Exposures) {
  msg.messageRoot = exposure
  var user : User = session.getFromTempMap( "assigneduser" ) as User
  msg.putEntityByName( "assigneduser", user )
  msg.putEntityByName( "claim", claim )
}
```

Detecting object changes in an Event Fired rule

You might want to listen for a change to 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, such as an update of the address. The `User` object itself does not contain an address. Instead, ClaimCenter 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 touch the `User` object.

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

```
USES:
uses gw.api.util.ArrayUtil

CONDITION (messageContext : entity.MessageContext):
//DestID 68 is the Console Message Logger
return messageContext.DestID == 68 and messageContext.EventName == "ContactChanged"

ACTION (messageContext : entity.MessageContext, actions : gw.Rules.Action):
var message = messageContext.createMessage( "Event: " + messageContext.EventName )
var contact = messageContext.Root as Contact
var fields = contact.PrimaryAddress.ChangedFields
print( ArrayUtil.toString( fields ) )
```

About modifying data in Event Fired rules and messaging plugins

Be very careful about modifying entity data in Event Fired rules and messaging plugin implementations. Use these Gosu rules to perform only the minimal data changes necessary for integration code. Entity changes in these code locations do not cause the application to run or re-run validation or preupdate rules. Therefore, do not change fields that might require those rules to run. Instead, change only the fields that are not modifiable from the user interface. For example, set custom data model extension flags used only by messaging code.

ClaimCenter does not support the following:

- Adding or deleting business entities from Event Fired rules or messaging plugins, even indirectly through other APIs.
- Calling, even in the Event Message rule set, any message acknowledgment methods or any skipping methods, such as `reportAck`, `reportError`, or `skip`. Use those methods only in messaging plugins.
- Creating messages outside the Event Message rule set.

See Also

- *Integration Guide*
- *Configuration Guide*
- *Contact Management Guide*

Event Fired rule example

In the following example, the rule creates a `CashReceiptDocument` whenever a user creates a new Recovery transaction in ClaimCenter.

```
USES:
uses java.util.HashMap

CONDITION (messageContext : entity.MessageContext):
return messageContext.Root typeis Recovery and (messageContext.Root).Status == "submitting"

ACTION (messageContext : entity.MessageContext, actions : gw.Rules.Action):
//This code uses custom fields that must be added to the Recovery and Document entities
var rec = messageContext.Root as Recovery
var values = new HashMap()
var template : gw.plugin.document.IDocumentTemplateDescriptor

values["InsuredName"]  = rec.Claim.Insured
values["ReceivedFrom"] = rec.Payer
values["DateReceived"] = rec.CreateTime
values["AmtReceived"]  = rec.Amount
values["PmntNo"]       = null
values["CheckDate"]    = rec.TransactionDate
values["CompletedBy"]  = rec.UpdateUser
values["ClaimNo"]      = rec.Claim.ClaimNumber
values["PolicyNo"]     = rec.Claim.Policy.PolicyNumber
values["CostCat"]      = rec.CostCategory
values["RecoveryCat"]  = rec.RecoveryCategory
values["ReceiptNo"]    = rec.CashReceiptNumberExt   //Custom field

var doc              = new Document(rec.Claim)
doc.MimeType         = "application/pdf"
doc.Claim            = rec.Claim
doc.Name             = "CashReceivedTicket"
doc.Exposure         = rec.Exposure
doc.TypeExt          = "Recovery"                   //Custom field
doc.SubTypeExt       = "Other"                      //Custom field
doc.PrivilegedExt    = "No"                         //Custom field
doc.ProcessMethodExt = "Sent_to_File"               //Custom field
doc.Claimant         = rec.Claim.claimant
doc.Status           = "approved"
doc.DocUID           = "ID-" + java.util.Calendar.getInstance().getTimeInMillis()

gw.document.DocumentProduction.createDocumentSynchronously(template, values, doc)
```

ISO Event Message rules

ClaimCenter uses event rules to determine if changes to an object require that information be sent to ISO.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **EFR01000 - ISO**

In the base configuration, ClaimCenter provides the following event message rules for ISO integration.

EFR01000 - ISO
This rule determines if the message context destination ID is 66. If so, the destination is ISO and the child rules can run.

EFR01100 - Exposure
Determines if the object that fired the message event is `Exposure`. If so, the child rule can run.

EFR01110 - Exposure Changed
If the message event indicates that the exposure is new, changed, or has been marked valid, call a method that sends a message to ISO. For details, see the method `checkForExposureChanges` in the file `ISO.gs`.

EFR01200 - Claim
Determines if the object that fired the message event is `Claim`. If so, the child rule can run.

EFR01110 - Claim Changed
If the message event indicates that the claim is changed or has been marked valid, call a method that sends a message to ISO. For details, see the method `checkForClaimChanges` in the file `ISO.gs`.

EFR01300 - Claim Contact Role
Determines if the object that fired the message event is `ClaimContactRole`. If so, the child rule can run.

EFR01310 - Claim Contact Role Changed
If the message event indicates that the claim contact role is new, changed, or removed, call a method that sends a message to ISO. For details, see the method `checkForClaimChanges` in the file `ISO.gs`.

EFR01400 - Policy
Determines if the object that fired the message event is `Policy`. If so, the child rule can run.

EFR01410 - Policy Changed
If the message event indicates that the policy has changed, call a method that sends a message to ISO. For details, see the method `checkForClaimChanges` in the file `ISO.gs`.

See also

- *Application Guide*

MetroReport Event Message rules

ClaimCenter uses event rules to check the status of a metropolitan police report request.

- If the status is `InsufficientData`, ClaimCenter does not send the report request to the Metropolitan Reporting Bureau. Instead, it creates an activity as described in "Claim Preupdate rule for metropolitan police report requests" on page 79.
- If the status is `Sending Order`, ClaimCenter fires an event and sends a message to the Metropolitan Reporting Bureau requesting a report.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **EFR02000 - Metro**

In the base configuration, ClaimCenter provides the following event message rules for metropolitan police report integration.

EFR02000 - Metro
Basically a folder that contains the Metro Report rules.

EFR02100 - Metro Report
Filter for the following two child messages. Checks for changes to the status of the `MetroReport`.

EFR02110 - Send Order Message

ClaimCenter sends out the initial report request to Metropolitan Reporting Bureau with all the claim information and the type of the report it requests.

EFR02120 - Send Inquiry Message

ClaimCenter sends an inquiry on a scheduled basis to Metropolitan Reporting Bureau for each outstanding report requested. The interval for sending the inquiry is set to hourly in the base configuration.

See also

- For information on Metropolitan Police Reports and the MetroReport object, see the *Integration Guide*.
- For information on related Gosu rules, see "Claim Preupdate rule for metropolitan police report requests" on page 79.

System console logger Event Message Rule

ClaimCenter uses the event rules to determine whether to send a message to the system console message logger.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **EFR04000 - Console**

In the base configuration, ClaimCenter provides the following event message rule for the system console message logger.

EFR04000 - Console

This rule determines if the message context destination ID is 68, indicating that the system console message logger is the destination. If so, the rule creates a message that says that this event, `messageContext.EventName`, was fired.

See also

- "Detecting object changes in an Event Fired rule" on page 53

Contact Auto Synchronization Failure Event Message rule

ClaimCenter uses an event rule to send a message indicating that ClaimCenter failed to synchronize a contact with the contact management system.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **EFR05000 - Contact Auto Sync Failure**

In the base configuration, ClaimCenter provides the following event message rule for a contact auto sync failure.

EFR05000 - Contact Auto Sync Failure

This rule determines if the message context destination ID is 80, the Contact Auto Sync Failure destination. If so, the rule creates a message indicating the `AddressBookUID` of the contact that failed to synchronize. The logic to detect events is defined in the messaging destination, `messaging-config.xml`. The event this destination listens for is `ContactAutoSyncFailed`.

See also

- *Application Guide*

Policy System Notification Event Message rules

ClaimCenter uses an event rule to send a message indicating that ClaimCenter must synchronize with the policy system. In the base configuration, these Gosu rules are disabled.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **EFR06000 - Policy System Notification**

In the base configuration, ClaimCenter provides the following event message rules for a sending notifications to the policy system.

EFR06000 - Policy System Notification

This rule and its child rules are disabled in the base configuration. If enabled, this rule determines if the message context destination ID is 69, indicating that ClaimCenter needs to send a message to the policy system. If so, the child rules can execute.

EFR06100 - Claim

This rule and its child rules are disabled in the base configuration. If enabled, this rule determines if the object that fired the message is a `Claim`. If so, the child rules can run.

EFR06110 - Claim Resync

Guidewire disables this rule in the base configuration. If enabled and the message event name is `ClaimResync`, this rule sends a message that resynchronizes the claim with the policy.

EFR06120 - Notification

Guidewire disables this rule in the base configuration. If enabled, this rule determines if there is a policy system notification event registered. If so, the rule retrieves the message event and creates a new message that uses the message context. In the base configuration, there is one such event defined, an event for Claim Exceeds Large Loss.

Asynchronous Document Storage Event Message rule

ClaimCenter uses an event rule to send a message indicating that ClaimCenter must store a document asynchronously, at some point in the future.

To work with this rule, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **EFR07000 - Async Document Storage**

In the base configuration, ClaimCenter provides the following event message rule for storing a document asynchronously.

EFR07000 - Async Document Storage

This rule determines if the message context destination ID is 324, a document storage message, and that the event involves storing a document (`DocumentStore`). If so, the rule sends a message that identifies the document that ClaimCenter is to process asynchronously.

Example asynchronous document storage failed Event Fired rule

Guidewire recommends that, in working with asynchronous document storage, that you create an Event Fired rule to handle the failure of the database to store a document properly. If a document fails to store properly, ClaimCenter creates a `FailedDocumentStore` event on `Document`.

To be useful, you need to add the `FailedDocumentStore` event to a messaging destination in `messaging-config.xml`. For example, you can add this event to message destination 324, which already tracks the `DocumentStore` event.

You then need to create an event rule to process the event as you see fit. You can, for example:

- Email a notification message to the creator of the `Document` entity.
- Create an activity on the primary or root object.
- Attempt to correct the document issues and create a new `DocumentStore` message.

The following example rule illustrates a possible Event Fired rule to handle the failure of the database to store a document.

```
CONDITION (messageContext : entity.MessageContext):
return messageContext.DestID == 324 and messageContext.EventName == "FailedDocumentStore"

ACTION (messageContext : entity.MessageContext, actions : gw.Rules.Action):
var document = messageContext.Root as Document
var fixedIssue = false
var sendEmail = false

if (fixedIssue) {
  messageContext.createMessage( "DocumentStore" )
}
else if (document.CreateUser.Contact.EmailAddress1 != null && sendEmail) {
  gw.api.email.EmailUtil.sendEmailWithBody(document,
  document.CreateUser.Contact.EmailAddress1,
  document.CreateUser.Contact.getDisplayName(),
  null, null, "Failed Document Store",
  //CREATE MESSAGE BODY HERE )
}
```

See also

- *Configuration Guide*

Contact Event Message rules

ClaimCenter uses an event rule to send a message indicating that ClaimCenter must synchronize with the contact management system. In the base configuration, these Gosu rules are disabled.

To work with this rule, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **EFR08000 - Contact**

In the base configuration, ClaimCenter provides the following event message rules for handling contact messages with the contact management system.

EFR08000 - Contact

This rule determines if the message context destination ID is 99, indicating that ClaimCenter needs to send a message to the contact management system. If so, the child rules can execute. These messages are used if ClaimCenter is integrated with ContactManager.

EFR08100 - Linked

This rule determines if a contact fired the message, and, if so, if all these additional conditions are true:

- The contact is stored in the contact management system—has an `AddressBookUID`.
- Automatic synchronizing of contacts is enabled.
- The contact is not an internal user, represented by the `UserContact` entity.
- The claim is not in draft save mode, indicated by the state of the New Claim Wizard.

If so, the child rules can execute.

EFR08110 - Update or add

Guidewire disables this rule in the base configuration. It is deprecated because it is for use only with ContactManager 7.0, and that version of ContactManager cannot be integrated with ClaimCenter 8.0.

EFR08115 - Tags changed

Guidewire disables this rule in the base configuration. It is deprecated because it is for use only with ContactManager 7.0, and that version of ContactManager cannot be integrated with ClaimCenter 8.0.

EFR08120 - UpdateInAB

This rule determines if ClaimCenter has updated a contact synchronously in ContactManager. If so, the rule sends a message that causes ClaimCenter to update all copies of the contact.

Whenever ClaimCenter updates a contact in ContactManager, ClaimCenter does not get a notification back from ContactManager that the contact was updated. This behavior prevents infinite looping of update messages between the applications.

However, ClaimCenter can have multiple local instances of a given linked contact. Whenever ClaimCenter updates one contact instance synchronously in ContactManager, ClaimCenter triggers the `ContactUpdatedInAB` event for that contact update. That event enables ClaimCenter to create an `AutoSyncWorkItem` to update all the other local instances of the contact.

This rule is not used in the base configuration because contact updates now go through an asynchronous API. However, the rule remains enabled in ClaimCenter to support any custom code that might continue to use synchronous update.

EFR08160 - Update

This rule determines if the contact has been updated or changed, if ContactManager is integrated, and if the contact has changes that need to be synchronized with ContactManager. If so, and the change did not come from ContactManager, the rule sends the changes to ContactManager.

EFR08200 - Not Linked

This rule determines if a contact fired the message, and, if so, if all these additional conditions are true:

- The contact is not stored in the contact management system—does not have an `AddressBookUID`—and therefore is not linked.
- The contact is not an internal user, represented by the `UserContact` entity.
- The claim is not in draft save mode, indicated by the state of the New Claim Wizard.

If so, the child rule can execute.

EFR08210 - Update Or Add

This rule determines if the contact has been updated or changed and if ContactManager is integrated. Because the contact is not linked, there is no need to determine if automatic contact synchronization is on.

If these conditions are true and the change did not come from ContactManager, the rule sends the changes to ContactManager.

EFR08300 - ClaimContact

This rule determines if a `ClaimContact` object fired the message, and, if so, if the event indicates that the claim contact was removed from a claim.

If so, the child rule can execute.

EFR08310 - Removed

This rule first determines the following:

- That the claim contact is stored in the contact management system—has an `AddressBookUID`.
- That automatic synchronization is enabled for this contact.
- That the contact is not an internal user, represented by the `UserContact` entity.

If these conditions are true, and the change did not come from ContactManager, and the contact is on only one claim, the rule sends the change to ContactManager.

See also

- *Contact Management Guide*

Instant check message rules

The ClaimCenter instant disbursements feature integrates with a payment gateway. This integration sends checks from ClaimCenter to a payment gateway, where they are processed as payouts. The `Check` entity in ClaimCenter is used to represent the payout when a check has a payment method of *instant*. ClaimCenter uses event rules to send a message indicating that ClaimCenter is requesting an instant payout from the payment gateway. The process of moving an instant check from `Requesting` to `Requested` is handled by the Instant Check Transport message queue.

To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **EFR09000 - Instant Check Transport**

In the base configuration, ClaimCenter provides the following event message rules for the instant payment gateway integration.

EFR09000 - Instant Check Transport

This rule determines if the instant payment integration feature is enabled and the message context destination ID is 71. If so, the destination is the Instant Check Transport message queue and the child rule can run.

EFR09100 - Check Added Or Changed

Child rule that determines if the object that fired the message event is a `Check` being added or updated. If so, the `shouldMakeOutboundPayment()` method in the `InstantCheckUtil.gs` class is called to determine if a Message entity must be created and the rule checks that the Status field on Check has changed. In the `InstantCheckTransportPlugin.gs` (Destination ID: 71) plugin class, the Message entity is processed, and an API call to the payment gateway provider initiates the payout.

See also

- *ClaimCenter Integration Guide*

Solr Destination Event Message rules

ClaimCenter uses Event Fired rules to update the Solr index that supports free text search for contacts on claims. These rules run whenever a change to contacts can be caused by the following events:

- A claim is added, changed, removed, or purged.
- A claim contact is added to, changed in, or removed from a claim.
- A claim role is added to, changed in, or removed from a claim.
- Any claim information is removed or purged.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **EventMessage** > **EventFired** > **SolrDestFilter**

In the base configuration, ClaimCenter provides the following event message rules for communicating contact changes to the Solr free text search system.

SolrDestFilter

This rule determines if both the following Solr message requirements are `true`:

- The message context destination ID is the same as the one being used by the Solr system.
- Free text search is enabled.

If both conditions are `true`, the child rules can execute.

Claim

This rule determines if a `Claim` object fired the message. If so, its child rules can run.

ClaimAdded

This rule determines if the event that fired the message was that a claim was added. If so, the rule sends a message to Solr to handle the claim added event.

ClaimChanged

This rule determines if the event that fired the message was that a claim was changed. If so, the rule sends a message to Solr to handle the claim changed event.

ClaimRemoved

This rule determines if the event that fired the message was that a claim was removed. If so, the rule sends a message to Solr to handle the claim removed event.

ClaimPurged

This rule determines if the event that fired the message was that a claim was purged. If so, the rule sends a message to Solr to handle the claim purged event.

ClaimContact

This rule determines if a `ClaimContact` object fired the message. If so, its child rules can run.

ClaimContactAdded

This rule determines if the event that fired the message was that a claim contact was added. If so, the rule sends a message to Solr to handle the claim contact added event.

ClaimContactContactChanged

This rule determines if the event that fired the message was that a claim contact was changed. If so, the rule sends a message to Solr to handle the claim contact changed event.

ClaimContactRemoved

This rule determines if the event that fired the message was that a claim contact was removed. If so, the rule sends a message to Solr to handle the claim contact removed event.

ClaimContactRole

This rule determines if a `ClaimContactRole` object fired the message. If so, its child rules can run.

ClaimContactRoleAdded

This rule determines if the event that fired the message was that a claim contact role was added. If so, the rule sends a message to Solr to handle the claim contact added event.

ClaimContactRoleChanged

This rule determines if the event that fired the message was that a claim contact role was changed. If so, the rule sends a message to Solr to handle the claim contact role changed event.

ClaimContactRoleRemoved

This rule determines if the event that fired the message was that a claim contact role was removed. If so, the rule sends a message to Solr to handle the claim contact role removed event.

ClaimInfo

This rule determines if a `ClaimInfo` object fired the message. If so, its child rules can run.

ClaimInfoChanged

This rule determines if the event that fired the message was that claim information was changed. If so, the rule sends a message to Solr to handle the claim information changed event.

The `ClaimInfo` entity changes only if ClaimCenter archives the claim or restores the claim from archive. In either case, it is important to update the Solr search index to either remove or restore the claim.

ClaimInfoPurged

This rule determines if the event that fired the message was that claim information was purged. If so, the rule sends a message to Solr to handle the claim information purged event.

See also

- *Configuration Guide*

Exception

Use the Exception rule sets to specify an action to take if an activity or claim is overdue and is in the escalation state. Additionally, there are rule sets to handle user exceptions and group exceptions. Neither of these rule sets contain sample rules in the base configuration. You can use Gosu rule sets in the Exception category to perform action such as sending an email, creating an activity, or setting a flag.

ClaimCenter runs the exception rule sets at regularly scheduled intervals.

See also

- For information on exception batch processes, see the *Administration Guide*.

Activity Escalation rules

ClaimCenter runs scheduled process `activityesc` every 30 minutes (by default). This batch process runs against all open activities in ClaimCenter to find activities to escalate, using the following criteria:

- The activity has an escalation date that is non-null.
- The escalation date is in the past.
- ClaimCenter has not yet escalated the activity.

ClaimCenter marks each activity that meets the criteria as escalated.

After the batch process runs, ClaimCenter runs the escalation rules for all the activities that have hit their escalation date. Use this rule set to specify what actions to take if the activity becomes escalated. For example, you can use this rule set to reassign escalated activities.

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.

Note: ClaimCenter does not run Activity Escalation rules on activities or users who are external to ClaimCenter. For example, ClaimCenter ignores any activity assigned to a third-party claims adjuster who is external to ClaimCenter and flagged as such in ClaimCenter.

The simplest approach to escalated activities is to add an activity on the claim for the appropriate supervisor to determine why the work is not yet complete. You can mark the claim as a flagged claim as well. You can also decide that there is no follow-up action needed for certain kinds of activities.

In the base configuration, ClaimCenter provides two sample Activity Escalation rules. To see the rules, navigate to the following location in the Studio Project window:

configuration > **config** > **Rule Sets** > **Exception** > **ActivityEscalationRules**

In the base configuration, this rule set contains the following rules.

AER01000 - High priority activities

If the activity has a high priority, set a flag on the claim for an overdue high priority activity and indicate the subject of this activity.

AER02000 - Urgent priority activities

If the activity has an urgent priority, set a flag on the claim for an overdue urgent priority activity and indicate the subject of this activity.

See also

For information on setting a claim flag, see “Flagging a claim as an exception” on page 65.

Claim Exception rules

ClaimCenter monitors claims for unusual conditions or conditions of potential concern to make it possible to alert adjusters or supervisors. This process has multiple parts:

- Determining the list of claims to evaluate
- Running claim exception rules for each claim on the list to find exceptions and take appropriate actions

For ClaimCenter to evaluate a claim, the claim must meet one of the following criteria.

Significant change

The claim must have changed in some significant way since the last time ClaimCenter ran the exception checking process. Significant changes include editing the claim basics, adding or editing an exposure, and adding or changing a financial transaction. ClaimCenter does not consider adding notes or documents, for example, a significant change because the exception rules do not typically evaluate the content of a new note or document.

Idle for a length of time

The claim is idle, aging without any significant change, for a certain length of time. Several configuration parameters, defined in file `config.xml`, impact how ClaimCenter determines a claim is idle:

- Configuration parameter `IdleClaimThresholdDays` defines the number of days after which ClaimCenter identifies a non-modified claim as idle. In the base configuration, this parameter is set to seven days.
- Configuration parameter `IdleClosedClaimThresholdDays` defines the number of days after which ClaimCenter identifies a closed claim as idle. In the base configuration, this parameter is set to seven days.

To view the Claim Exception rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Exception** > **ClaimExceptionRules**

In the base configuration, ClaimCenter provides the following Claim Exception rules.

CER01000 - Setting SIU Life Cycle State

Advances the claim life cycle stage based on the number of days that have passed since claim inception. See "Advance the life cycle stage" on page 170.

CER02000 - At least one activity for claim owner

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CER03000 - At least one activity for exposure owner

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CER04000 - Recalculate Claim Metrics

Guidewire disables this rule in the base configuration. This rule verifies that all claim metrics, exposure metrics, and claim indicators are created on every claim. Any missing metrics or indicators are added to the claim.

Enable this rule only if a new metric or indicator that has been added to the system is to be added to all claims.

CER05000 - BizRules

This rule executes the business rules that are set to trigger on a claim exception. The rule runs after the other Claim Exception Gosu rules have completed.

See also

- For a discussion of Claim Exception rules that relate to fraud and special investigations, see "Detecting claim fraud" on page 167.
- *Application Guide*

Checking for recently modified claims

Guidewire defines *recently modified* as any claim for which the update time is newer than the last time the exception rules ran on the claim. The check for recently modified claims examines the following:

- The update time on the claim
- The update time on any exposure on the claim
- The update time on any transaction on the claim
- The update time on any claim contact or claim contact role on the claim
- The update time on any matter on the claim

About exception batch processing

The following list describes the batch processes that work with changed and idle claims.

Batch process	Description	Default time
`claimexception`	The Claim Exception batch process runs the claim exception rules on claims that have been modified significantly since the last time that ClaimCenter ran this process. The claims processed are:	Every day at 2:00 a.m.

Batch process	Description	Default time
	• Claims that have been updated since the last time the batch process processed it • Claims with any of the following updates: ◦ exposure ◦ transaction ◦ claim contact ◦ claim contact role ◦ matter • Claims that have never been processed by the batch process For all these cases, the Claim Exception batch process creates a `ClaimException` row in the database for the claim if none currently exists. If the row exists, the batch process updates the `ExCheckTime` field of the `ClaimException` to the current time.	
`idleclaimexception`	The Idle Claim Exception batch process runs the claim exception rules on claims that are idle. A claim is idle if the exception rules have not been run on it in the number of days specified by the `IdleClaimThresholdDays` configuration parameter. In the base configuration, the value of this configuration parameter is 7 days. More specifically, the Idle Claim Exception batch process evaluates all claims that have a foreign key to an existing `ClaimException` row in the database. If the `ExCheckTime` field of the `ClaimException` row is older than Today minus `IdleClaimThresholdDays`, the batch process runs the claim exception rules on that claim.	Every Sunday at 12:00 noon.

To run separate schedules for changed claims and idle aging claims, set configuration parameter `SeparateIdleClaimExceptionMonitor` to `true` in file `config.xml`.

Configuration parameters that affect changed and idle claims

The following list describes the configuration parameters that specifically affect changed and idle claims.

Configuration parameter	Description	Default value
`IdleClaimThresholdDays`	Number of days without significant change after which ClaimCenter considers a claim idle.	7
`IdleClosedClaimThresholdDays`	Number of days without significant change after which ClaimCenter considers a closed claim idle.	7
`SeparateIdleClaimExceptionMonitor`	If `true`, ClaimCenter runs separately scheduled processes for determining changed claims (`claimexception`) and idle aging claims (`idleclaimexception`). If `false`, ClaimCenter runs the `idleclaimexception` process also as it runs the `claimexception` process.	`true`

See also

- For information on ClaimCenter batch processes, see the *Administration Guide*.
- For information on ClaimCenter configuration parameters, see the *Configuration Guide*.

Claim exceptions and Gosu rules

There are many kinds of exceptions for which you can write rules:

- Work that has not been completed within a certain number of days, but for which there is no activity-based escalation

For example, you can look for auto claims that are more than 30 days since being reported but still do not have a fault rating set.

- Unusual reserve changes or values

  For example, you can look for claims with three or more reserve increases.

- Items of interest

  You can decide to flag claims that have important new information. For example, if a claim notes the existence of a plaintiff attorney, you can flag the claim for potential litigation automatically.

Flagging a claim as an exception

One action to take in regard to escalated claims is to flag a claim as an exception. The following list describes `Claim` fields and one method that you can use with flagged claims. ClaimCenter displays the three fields in the **New Loss** screen for the claim.

Property or method	Description
`Flagged`	Sets the flagged status of a claim to a typecode defined in the `FlaggedType` typelist: • `isflagged` • `neverflagged` • `wasflagged` Used to determine a status of a claim.
`FlaggedDate`	Indicates the date on which a flag was set.
`FlaggedReason`	A text field that provides a reason the flag was set. Helps the adjuster understand the nature of the exception noticed.
`clearFlag`	A method that toggles the `Flagged` field and clears the `FlaggedReason` field. Use this method after the claim is no longer considered to have any exceptions.

You can use the `setFlag` method in a rule to set the value of the flagged reason. For example:

```
claim.setFlag("Claim reported within 30 days of opening policy."
      + " Claim referred to SIU for investigation.")
```

Note: In actual practice, Guidewire recommends that you make `String` values into display keys or constant variables, as appropriate.

IMPORTANT: Avoid writing exception rules that cause ClaimCenter to report the same exceptions repeatedly. See “Taking action on more than one subitem” on page 32.

Group Exception rules

ClaimCenter runs the Group Exception rules on a scheduled basis to look for certain conditions on groups that might require further attention. You can use these Gosu rules to define the follow-up actions for each exception found. This rule set is empty in the base configuration. To work with this rule set, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Exception** > **GroupExceptionRules**

In the base configuration, Guidewire schedules the Group Exception work queue to run the group exception rules on all groups in ClaimCenter every day at 4:00 a.m. However, Guidewire disables the Group Exception work queue by default. You must enable the work queue by setting the number of workers for this work queue to a number greater than zero in file `work-queue.xml`.

User Exception rules

ClaimCenter runs the User Exception rules on a scheduled basis to look for certain conditions on users that might require further attention. You can use these Gosu rules to define the follow-up actions for each exception found. This rule set is empty in the base configuration. To work with this rule set, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Exception** > **UserExceptionRules**

In the base configuration, Guidewire schedules the User Exception work queue to run the user exception rule sets on all users in ClaimCenter every day at 3:00 a.m. However, Guidewire disables the User Exception work queue by default. You must enable the work queue by setting the number of workers for this work queue to a number greater than zero in file `work-queue.xml`.

See also

- *Integration Guide*

Initial Reserve

ClaimCenter uses Initial Reserve rules to create initial financial reserves on new exposures. The Initial Reserve rule category, which executes immediately after the Postsetup rules, automates creation of reserves for new exposures.

Note: If your enterprise does not automate creation of reserves, you probably do not use need to use this rule category.

ClaimCenter uses these Gosu rules to determine the initial financial reserves to create for a new exposure. Each segment type has specific reserve requirements. For example, the reserve amount you establish for a moderate vehicle damage exposure is likely to be less than the reserve you create for a serious bodily injury exposure.

The rules use the `createInitialReserve` method to automate the creation of the reserve.

Note: Reserves created by using these rules are not submitted for approval. Authority limit validation is skipped, and they go straight into **Submitting** status.

ClaimCenter does not associate a `TransactionSet` with any reserves added through these rules until it processes the new reserves. The most common approach for creating initial reserves is to write a rule or rule set for each exposure segment type.

In the base configuration, ClaimCenter provides Initial Reserve rules for Auto Vehicle Damage, Travel Baggage Delay or Loss, Trip Cancellation, and Travel Medical Payments. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **InitialReserve**

Initial Reserve rules for auto vehicle damage

The Initial Reserve Auto rules create an initial reserve for an Auto loss type. To these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **InitialReserve** > **InitialReserve** > **Auto**

Guidewire provides the following Initial Reserve Auto rules in the ClaimCenter base configuration.

IRR01000 - Auto

This rule determines if the loss type is Auto. If so, the child rules execute.

IRR01100 - Vehicle damage

This rule determines if the exposure is Vehicle Damage. If so, the child rules execute.

IRR01110 - Minor

This rule determines if the exposure segment is Auto Low, if the claim is not open, and if there is not exactly one exposure and one transaction. If so, and if there are no financial holds, the rule creates an initial reserve for an exposure. The call to `createInitialReserve` specifies a cost type of `claim_cost`, a cost category of `body`, and the value of the script parameter `InitialReserve_AutoMinorVehicleDamageBody`.

IRR01120 - Medium

This rule determines if the exposure segment is Auto Medium, if the claim is not open, and if there is not exactly one exposure and one transaction. If so, and if there are no financial holds, the rule creates an initial reserve for an exposure. The call to `createInitialReserve` specifies a cost type of Claim Cost, a cost category of Body, and the value of the script parameter `InitialReserve_AutoMediumVehicleDamageBody`.

Additionally, regardless of the status of financial holds, the rule creates an initial reserve with a cost type of Adjusting and Other Expenses to cover inspection of the auto damage. This initial reserve value is specified in the script parameter `InitialReserve_AutoMediumVehicleDamageInspection`.

IRR01130 - High complexity

This rule determines if the exposure segment is Auto High, if the claim is not open, and if there is not exactly one exposure and one transaction. If so, if and there are no financial holds, the rule creates an initial reserve for an exposure. The call to `createInitialReserve` specifies a cost type of Claim Cost, a cost category of Body, and the value of the script parameter `InitialReserve_AutoMajorVehicleDamageBody`.

Additionally, regardless of the status of financial holds, the rule creates an initial reserve with a cost type of Adjusting and Other Expenses to cover inspection of the auto damage. This initial reserve value is specified in the script parameter `InitialReserve_AutoMediumVehicleDamageInspection`.

See also

- “Segmentation” on page 91
- *Application Guide*

Initial Reserve rules for travel losses

The Initial Reserve Travel rules create an initial reserve for a Travel loss type. To view these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **InitialReserve** > **InitialReserve** > **Travel**

Guidewire provides the following Initial Reserve Travel rules in the ClaimCenter base configuration.

IRR02000 - Travel

This rule determines if the loss type is Travel. If so, the child rules execute.

IRR02100 - Baggage

This rule determines if the exposure type is Baggage. If so, the child rules execute.

IRR02110 - Baggage Delay

This rule determines if the incident for this Baggage exposure is Delay Only. If so, and if there are no financial holds:

- If baggage is delayed for more than 4 hours and less than a day, the rule:
  - Sets the initial reserve to the value of the `InitialReserve_TravelBaggageLoss` script parameter if this value is lower than the exposure limit.
  - Otherwise, sets the initial reserve to the exposure limit.
- If baggage is delayed for more than a day, the rule multiplies the value of the `InitialReserve_TravelBaggageLoss` script parameter by the number of days lost.
  - If this product is lower than the exposure limit. the rule sets the initial reserve to this product.
  - Otherwise, the rules set the initial reserve to the exposure limit.

Note: The rule hardcodes the values of 4 and 24 for hours delayed.

IRR02120 - Baggage Loss

This rule determines if the incident for this Baggage exposure is not Delay Only, which means that the baggage was lost. If so, and if there are no financial holds:

- If there is an exposure limit, the rule creates an initial reserve for that amount.
- If there is no exposure limit, create an initial reserve for the approved content total if the value is greater than zero.

IRR02200 - Trip Cancellation

This rule determines if the exposure type is Trip Cancellation or Delay. If so, and if there are no financial holds:

- The rule sets the initial reserve to the total financial impact if that value is available and is lower than the exposure limit.
- Otherwise, the rule sets the initial reserve to the exposure limit.

IRR02300 - Medical payments

This rule determines if the exposure type for this Travel loss type is Medical Payments. If so, and if there are no financial holds, the rule sets the initial reserve to the exposure limit.

See also

- *Application Guide*

Loaded

Rules in the Loaded rule set category execute only in concert with the FNOL (First Notice of Loss) import interface and relate only to imported claims. To add Gosu rules to this category, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Loaded**

Important

Place rules that affect the following types of claims in the Presetup rule set category rather than the Loaded rule set category:

- Claims that ClaimCenter itself creates
- Claims that must execute before Segmentation

Claim Loaded rules

Claim Loaded rules execute whenever an FNOL import loads a new claim. ClaimCenter executes Loaded rules after it finishes importing, or *loading*, the claim from an external FNOL source. Some uses of this rule category are:

- To make any changes necessary to convert initial claim data from an external format into a suitable structure for ClaimCenter, such as:
  - Sanity checks
  - Data aggregations and transformations
  - Data and naming reconciliations
- To record the import of claims
- To assign review of the new FNOL to a user prior to running the full automated setup process

A typical application of the Loaded rules is to ensure that ClaimCenter makes a record of claims that it imports from an external FNOL application. Making a record typically consists of logging the transaction. The following rule illustrates this application.

```
USES:
uses gw.api.system.CCLoggerCategory
```

```
CONDITION (claim : entity.Claim):
return Claim.State != null and Claim.State != "draft"

ACTION (claim : entity.Claim, actions : gw.rules.Action):
CCLoggerCategory.IMPORT.debug("In Claim No. : "+ claim.ClaimNumber + " - In " + actions.getRuleSet()
      + " - In Rule " + actions.getRule() )
actions.exit()
```

Note: In actual practice, Guidewire recommends that you make `String` values into display keys or constant variables, as appropriate.

Exposure Loaded rules

ClaimCenter does not execute the Exposure Loaded rules. Guidewire intends these rule to trigger after an FNOL import into ClaimCenter for the `Claim` entity only. If you add Gosu rules to this rule set, you must trigger the rules manually.

In the following example, the rule logs a message to the system console if the import loads an exposure linked to an existing claim. As with any Exposure Loaded rules, you must manually trigger this rule.

```
USES:
uses gw.api.system.CCLoggerCategory

CONDITION (exposure : entity.Exposure):
return (exposure.State == null or exposure.State == "draft") and exposure.Claim.State == "open"

ACTION (exposure : entity.Exposure, actions : gw.rules.Action);
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()
      + " Rule set.")
CCLoggerCategory.IMPORT.debug( " " )
CCLoggerCategory.IMPORT.debug( "Loaded Exposure Loaded default rules " + actions.getRule() )
```

Note: In actual practice, Guidewire recommends that you make `String` values into display keys or constant variables, as appropriate.

Postsetup

Postsetup rules run automatically whenever ClaimCenter adds a new claim, exposure, activity, matter, or transaction. ClaimCenter runs these rules after the Assignment and Workplan rules complete their execution. The Postsetup rules run immediately before the rule sets in the Initial Reserve rule set category and the Preupdate rule set category.

Postsetup rules can perform a wide variety of actions, depending on your business requirements. Postsetup rules often depend on the results of the rules executed during setup or other tasks. You can design these rules to trigger based on specific conditions that the setup rules create.

Note: ClaimCenter runs the Postsetup rules as part of the “save and setup” operation on claims and exposures. See “About claim and exposure setup” on page 72 for more information.

See also

- “Initial Reserve” on page 66
- “Preupdate” on page 76

Activity Postsetup rules

The Activity Postsetup rules fire just prior to completing the setup process and saving any changes to activities. They run after the assignment rules have run. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Postsetup** > **ActivityPostsetup**

In the base configuration, ClaimCenter provides the following sample Activity Postsetup rules.

APS01000 - Add user in SIU Role to claim if necessary

If the current activity is an SIU escalation, this rule determines whether the claim already has an SIU investigator assigned. If there is no assigned SIU investigator, and the owner of the activity is an SIU investigator, the rule assigns the owner of the activity to the claim.

APS02000 - Subro - Add user to Role on claim if necessary

This rule applies if the current activity is a subrogation check and the claim does not already have a subrogation owner. If so, and there is an assigned user for this activity, assign that user to the claim as the subrogation owner.

APS03000 - Reins - Add user to Role on claim

This rule applies if the current activity is a claim reinsurance review and there is an assigned user for this activity. In that case, assign that user to the claim as the reinsurance manager.

Claim Postsetup rules

The Claim Postsetup rules fire just prior to completing the setup process and saving any changes to claims. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Postsetup** > **ClaimPostsetup**

In the base configuration, Guidewire provides a single, disabled, sample Claim Postsetup rule.

CPS01000 - Add Auto-body shop

Guidewire disables this rule in the base configuration. If the current claim is an Auto claim, attempt to find and add an auto body shop as a vendor service to the claim.

Example Claim Postsetup rule

In the following example, the Claim Postsetup rule creates a new activity for FNOL (first notice of loss).

```
CONDITION (claim : entity.Claim):
return claim.isValid( "newloss", true )

ACTION (claim : entity.Claim, actions : gw.rules.Actions):
var strClaimNumber = claim.DuplicateClaimNumbers

if ( strClaimNumber != null ) {
  /* Create new claim activity if there are possibly duplicate claims listed. */
  var description = "Check that claim is not a duplicate of " + strClaimNumber
  claim.createActivity( null, ActivityPattern.finder.getActivityPatternByCode( "fnol" ),
        "Check that duplicate claim does not exist", description, null, null, null, null )
} else {
  /* Create new claim activity */
  claim.createActivityFromPattern( null, ActivityPattern.finder.getActivityPatternByCode( "fnol" ) )
}
```

Note: In actual practice, Guidewire recommends that you make `String` values into display keys or constant variables, as appropriate.

Exposure Postsetup rules

The Exposure Postsetup rules fire just prior to completing the setup process and saving any changes to exposures, after assignment and workplan rules have run. Guidewire does not provide any sample rules in the base ClaimCenter configuration.

To add Gosu rules to this rule set, navigate in the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Postsetup** > **ExposurePostsetup**

Matter Postsetup rules

The Matter Postsetup rules fire just prior to completing the setup process and saving any changes to matters, after assignment and workplan rules have run. To work on this rule set, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Postsetup** > **MatterPostsetup**

In the base configuration, ClaimCenter provides one Matter Postsetup rule.

MPS01000 - Set Litigation Status

If a matter is created, set the litigation status of the claim to Litigated.

Example Matter Postsetup rule

In the following example, the Matter Postsetup rule sets up an activity for the claim owner if the matter has a bodily injury exposure in its array of exposures. The activity is to initiate a legal review of a bodily injury claim 10 days before a scheduled mediation meeting with the insured.

```
CONDITION (matter : entity.Matter):
return true

ACTION (matter : entity.Matter, actions : gw.rules.Actions):
var exp = matter.Exposures*.Exposure.firstWhere( \ elt -> elt.ExposureType ==
      ExposureType.TC_BODILYINJURYDAMAGE)

if (exp !=null) {
  var act = matter.Claim.newActivity( ActivityPattern.finder.getActivityPatternByCode( "legal_review" ), exp)
  act.TargetDate = gw.api.util.DateUtil.addDays( act.Matter.MediationDate, -10 )
  act.assignToClaimOwner()
}
```

Transaction Postsetup rules

The Transaction Postsetup rules fire under the following conditions:

- Immediately after the Exposure Closed rule set, after closing an exposure that has a transaction associated with it.
- Immediately after the Transaction Approval Rules rule set except if you are in the process of creating a reserve.

These rules also fire under the following conditions:

- ClaimCenter approves a TransactionSet
- ClaimCenter voids, stops, or escalates a check
- ClaimCenter records a payment
- ClaimCenter voids a recovery
- ClaimCenter closes an exposure

To work on this rule set, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Postsetup** > **TransactionPostsetup**

In the base configuration, ClaimCenter provides the following Transaction Postsetup rules.

TPS01000 - Check Aggregate Limits

If the transaction set is within 20 percent of its limit, check to see if there is already an activity assigned to check for approaching aggregate limits. If not, create a general reminder activity to check on the transaction within 5 days.

TPS02000 - Check Aggregate Deductible

If the aggregate deductibles are within 20 percent of the limit, check to see if there is already an activity assigned to check for approaching aggregate deductible limits. If not, create a general reminder activity to check on the deductibles within five days.

Presetup

ClaimCenter runs the Presetup rules automatically whenever ClaimCenter adds a new activity, claim, exposure, matter, or transaction. Presetup rules do not apply to modifications made to existing claims. You use Presetup rules to perform actions on new claims before claim or exposure setup. *Setup* typically encompasses running rules in the Segmentation, Assignment, and Workplan rule set categories. If you need to modify a new claim before beginning setup, the Presetup rule category provides a place for the rules that perform the actions.

To work on Presetup rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Presetup**

In the base configuration, ClaimCenter provides one Presetup rule, the Claim Presetup rule CPR01000 - WCDefaultExposures.

See also

- "Workers' Compensation Claim Presetup example" on page 75
- "Segmentation" on page 91
- "Assignment" on page 41
- "Workplan" on page 119

About claim and exposure setup

Guidewire uses the term *setup* as a generic term for invoking multiple rules (Presetup, Segmentation, Assignment, Workplan, and Postsetup, for example). The term *setup* also applies to initialization work that ClaimCenter performs whenever it first creates a Claim or Exposure as an open item. ClaimCenter typically performs this kind of setup under the following circumstances:

- At the end of the **New Claim** wizard, ClaimCenter sets up the new claim and all its exposures.
- After creating a new exposure through ClaimCenter, ClaimCenter sets up the new exposure after you click **Update**.
- Whenever you add a new claim through the `addFNOL` SOAP call, ClaimCenter performs a setup before it commits data to the database.

About adding a new claim using the ClaimAPI.addFNOL method

ClaimCenter performs the following setup tasks whenever you add a new claim through the `ClaimAPI` web service method `addFNOL`. ClaimCenter:

1. Adds the FNOL claim.
2. Invokes the Loaded rule set category rules on the FNOL claim. See "Loaded" on page 68.
3. Creates an `import` history event.
4. Performs the sequence of actions classified as *save-and-setup*, setting the claim status to `open` at the end of the process. See "About save and setup processes for an FNOL claim" on page 72.
5. Commits the claim. This commit includes all exposures, activities, and other linked objects created during the save-and-setup process.
6. At commit time, runs the rules in the Preupdate, Validation, and Event Message rule set categories, validating the claim at the `loadsave` level.

See also

- "Preupdate" on page 76
- "Validation" on page 98
- "Event message" on page 51
- *Integration Guide*

About save and setup processes for an FNOL claim

Guidewire ClaimCenter performs the following steps for a save-and-setup operation on an FNOL claim. ClaimCenter:

1. Creates a new claim number (if one does not exist).
2. Creates a claim snapshot.

3. Runs the rules in the Presetup rule set category on the claim and each exposure.
4. Runs the rules in the Segmentation rule set category on each exposure, then the claim.
5. Runs the rules in the Assignment rule set category on the claim, then each exposure.
6. Runs the rules in the Workplan rule set category on the claim, then each exposure. This process generates the workplan.
7. Runs the rules in the Postsetup rule set category on the claim and each exposure.
8. Updates history timestamps so that client-added history events can be correctly stamped.
9. Sets the status of the claim and each exposure to `open`.
10. Ensures that each exposure has a valid claim order field set.
11. Sets initial reserves.

At the end of the setup process for an FNOL claim, ClaimCenter returns the public ID of the newly created claim.

See also

- "Assignment" on page 41
- "Postsetup" on page 69
- "Presetup" on page 71
- "Segmentation" on page 91
- "Workplan" on page 119

About save and setup processes for exposures

ClaimCenter performs the following save-and-setup pre-processing operations on an exposure after the `Exposure.saveAndSetup` method creates it. ClaimCenter:

1. Sets the claim order on the exposure.
2. Runs rules in the Presetup rule set category on the exposure.
3. Runs the segmentation engine on the exposure.
4. Runs the strategy engine on the exposure.
5. Runs the assignment engine on the exposure.
6. Runs the workplan engine on the exposure.
7. Runs rules in the Postsetup rule set category on the exposure.
8. Sets the exposure's status to `open`.
9. Creates initial reserves.
10. Commits the exposures bundle, which contains the exposure, activities, and other objects creating during the call.

See also

- "Postsetup" on page 69
- "Presetup" on page 71

Creating Transaction Set Presetup rules

The Transaction Set Presetup rules execute prior to transaction setup. *Setup* for transactions encompasses submitting a `TransactionSet` for approval and updating the T-accounts. In the base configuration, ClaimCenter does not provide any Gosu rules for this rule set.

The Transaction Set Presetup rule set is useful primarily for a limited purpose, making the same modifications to the following transactions:

- Transactions created in the ClaimCenter user interface
- Transactions created in rules or outside rules in Gosu

Whenever you use the rules for this purpose, you can modify transaction values that affect financial calculations. These values include the transaction `Amount`, the `Eroding` state, the `Exposure`, `CostType`, or `CostCategory`, or the `ScheduledSendDate` on a payment's check.

If you create or modify transactions as described in the following cases, you do not need to use Presetup rules on them:

- If you need to modify transactions created only in the ClaimCenter user interface, rather than using rules, you can modify them in the `beforeCommit` attribute of the PCF page. Use the financial APIs for transactions in your Gosu code.
- If you need to modify transactions created only in rules or Gosu, use the financial APIs to create the transactions correctly in the first place instead of modifying them later.

About financial APIs for transactions

There are classes and methods that provide extensive control over payments and transaction line items as you set up checks, and these APIs also update the T-accounts if used correctly.

If you want to create, modify, or remove transactions programmatically, Guidewire recommends that you use the Gosu methods described in the following topics.

- *Configuration Guide*

IMPORTANT: Guidewire does not provide a mechanism for editing previously saved checks or recoveries exclusively with rules or Gosu. The editing of a check must be initiated in the user interface.

About ClaimCenter execution of Transaction Set Presetup rules

ClaimCenter executes the Transaction Set Presetup rule set under the following circumstances:

- If you click the **Finish** at the end of the **Check** wizard
- If you click **Save** or **Update** on the **New Reserves**, **New Recovery Reserves**, or **New Recoveries** screen
- If you create a claim in the **New Claim** wizard in **Auto - First and Final** mode, which creates a check

ClaimCenter also runs the Transaction Set Presetup rules:

- During `CheckCreator.prepareForCommit`.
- During `BulkInvoiceItem` processing of its placeholder check.

ClaimCenter does not execute these Presetup rules if you use the following methods on `Claim`, `Exposure`, or `ReserveLine` to create a transaction:

- `createRecovery`
- `setAvailableReserves`
- `setOpenRecoveryReserves`

Using the Transaction Set Presetup rules

You use the rules in the Transaction Set Presetup rule set to modify transactions and checks before they go through the setup process. Through these rules, you can:

- Add to or remove a payment from a check.
- Add to or remove a line item from a transaction.
- Modify the amount of a line item.
- Add or remove a `Recovery`, a `Reserve`, or `RecoveryReserve` from a `TransactionSet` of the appropriate subtype.

Note: Reserves and recovery reserves only have one line item.

The Transaction Set Presetup rules provide support for modifying the amounts of single and recurring checks from rules just before the checks are set up. They also provide support for adding new payment line items through the user interface just before setting up the transaction. For example, after a payment is submitted, but before it is set up and then saved, you might want to add a new payment to a check. You might want to add the payment both from the user interface and programmatically to the check while it is in Awaiting Submission status. You might want to add the payment, for example:

- To adjust a future payment
- To add a line item for a certain type of tax computed in rules

Performing your business logic in the Transaction Set Presetup rule set does cause ClaimCenter to update the T-accounts properly.

Guidewire provides the following methods that you can use in this rule set:

- `Transaction.addNewLineItem(Amount, Comments, LineCategory)`
- `RecoveryReserveSet.newRecoveryReserve(Exposure, CostType, CostCategory)`
- `RecoverySet.newRecovery()`
- `ReserveSet.newReserve(Exposure, CostType, CostCategory)`
- `Check.addNewPayment(Exposure, CostType, CostCategory)`

If you want to change a line item, use the standard `getLineItems` method to retrieve the line item. Then, in this rule set, call any of the standard setters on the line item—for example, `lineitem.setAmount`—so that ClaimCenter correctly computes the T-accounts.

IMPORTANT: If you use financial API methods in a Presetup rule, do not call `prepareForCommit` in the rule as ClaimCenter prepares the transactions for commit as part of the setup process. In addition, do not call the constructors of financial transaction entities, such as `new Payment` or `new RecoveryReserve`, in Presetup rules. Instead, use the financial APIs to create these entities outside rules.

See also

- "About financial APIs for transactions" on page 74

About modifying the check scheduled send date

It is possible to modify the check property for the schedule send date, `ScheduledSendDate`, in the Transaction Presetup rules or from Gosu called from the application user interface (PCF) code. The date determines the place to include the check amount, such as:

- In Future Payments (tomorrow or later)
- In Awaiting Submission (today), which is included in Total Payments and similar financial calculations.

If you update the date inappropriately, ClaimCenter throws an exception with the following message:

```
Check contains a payment whose LifeCycleState is inconsistent with the Check's Scheduled Send
Date.
```

Workers' Compensation Claim Presetup example

One common application of Presetup rules is to create exposures for a new Workers' Compensation claim. Workers' Compensation claims typically have only three possible exposure assignments:

- An employee is injured on the job and loses some work time. The employee is entitled to reimbursement for both medical costs and lost wages, and the employer's workers' compensation insurance covers the loss.

  This type of claim has two exposures, one for Medical Details and another for Time Loss.

- An employee is injured on the job, but misses no work time. The employee is entitled to reimbursement only for medical costs, and the employer's workers' compensation insurance covers the loss.

  This type of claim has only a single Medical Details exposure.
- An employee is injured and the employer's insurance does not cover the loss. The claimant is therefore obliged to seek compensation in court.

  This type of claim has only a single Employer Liability exposure.

There is an existing Claim Presetup rule that creates the standard workers' compensation exposures that you see whenever setting up a Workers' Compensation claim in ClaimCenter. To see this rule, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Presetup** > **ClaimPresetup** > **CPR01000 - WCDefaultExposures**

In the base configuration, this rule has the following form:

```
CONDITION (claim : entity.Claim):
return claim.LossType == "WC"

ACTION (claim : entity.Claim, actions : gw.rules.Action):
claim.createRelevantWorkCompExposures()
```

The condition determines that the rule applies to any Workers' Compensation claim. The action calls the method `createRelevantWorkCompExposures`, defined in an enhancement to the claim entity in `FNOL.gsx`. The method contains the instructions for evaluating and assigning exposures for workers' compensation claims. To see this method, navigate to the following location in the Studio **Project** window:

configuration > **gsrc** > **libraries** > **FNOL**

See also

- *Application Guide*

Preupdate

Use the Preupdate rule sets to perform domain logic or validation that must be performed before the entity in question is committed. Only a change to an entity marked as `Validatable` in its definition file can trigger a preupdate rule.

Typically, the Preupdate rules execute before the rules in the Validation rule set category.

The Preupdate rules can perform a wide variety of actions. One application of the preupdate rules is to create an activity based on a condition. For example, if a vehicle covered in a claim has been declared a total loss, a rule can create an activity to inspect the car for its salvage potential.

To access this rule set category, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate**

See also

- "Rules-based validation" on page 159
- "Validation" on page 98
- "Postsetup" on page 69

How ClaimCenter triggers Preupdate rules

The following ClaimCenter root entities can all trigger preupdate rules in the ClaimCenter base configuration:

```
Activity
Claim
Contact
Exposure
Group
```

```
Matter
Policy
RIAgreement
RITransactionSet
Subrogation
TransactionSet
User
```

Note: Not all these entities have Preupdate rules in the base configuration.

For an entity to trigger a preupdate rule, that entity must implement the `Validatable` delegate. This requirement applies both to entities in the base ClaimCenter configuration and to custom entities that you create.

Entities that implement the `Validatable` delegate trigger Validation rules as well.

ClaimCenter runs the preupdate and validation rules in response to the following events:

- An instance of a validatable entity is created or modified.
- For an entity connected by an array or foreign key to a validatable entity:
  - The validatable entity's array key or foreign key is defined with `triggersValidation="true"`.
  - The entity to which the array or foreign key connects is created or modified.

IMPORTANT: In running the Preupdate rule set, ClaimCenter 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.

The same behavior also applies to entities newly created in a preupdate rule. For example, if you create a new activity in a preupdate rule, ClaimCenter does not then run the Activity preupdate rules on that activity.

See Also

- For information on custom entities and validation rules, see "How ClaimCenter triggers validation rules" on page 100.
- *Configuration Guide*
- *Integration Guide*

How ClaimCenter triggers Preupdate rules on custom entities

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

For an extension entity to trigger a preupdate rule, all of the following must be true:

1. The entity must be a root entity, not a subtype of an entity.
2. The entity must implement the `Validatable` delegate.

   For any extension entity that you create, if you want it to trigger a preupdate rule, the entity must implement the following code:

   ```
   <implementsEntity name="Validatable"/>
   ```
3. A rule set must exist in the Preupdate rule set category that conforms to the following naming convention:

   ```
   <custom_entity_name>Preupdate
   ```

   For example, if you create an extension entity named `NewEntityExt`, create a rule set in the Preupdate rule set category to hold your preupdate rules and name it `NewEntityExtPreupdate`

See also

- For information on creating new rules sets, see "Working with rules" on page 23.
- For information on how to create an entity that implements the `Validatable` delegate, see the *Configuration Guide*.

Activity Preupdate rules

Use the Activity Preupdate rules to modify activities and related entities. In the base configuration, whenever you reassign a claim, ClaimCenter reassigns activities that are associated with the claim and belong to the previous owner to the new claim owner. You can write Activity Preupdate rules that provide different behavior.

In the base configuration, there are no Gosu rules in this rule set. To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ActivityPreupdate**

Claim Preupdate rules

Use the Claim Preupdate rules to modify claims and related entities. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ClaimPreupdate**

In the base configuration, the Claim Preupdate rule set includes rules that are useful in working with the following types of items:

Item type	See…
Document creation	"Claim Preupdate rules for document creation" on page 78
Claim fraud	"Claim Preupdate rules for Special Investigation Unit (SIU)" on page 79
Metropolitan reports	"Claim Preupdate rule for metropolitan police report requests" on page 79
Catastrophes	"Claim Preupdate rules for catastrophes" on page 79
Change of medical diagnosis	"Claim Preupdate rules for changed primary diagnosis code" on page 80
Subrogation	"Claim Preupdate rules for subrogation" on page 80
Auto first and final	"Claim Preupdate rule for auto first and final" on page 81
Travel trip incidents	"Claim Preupdate rules for travel" on page 81
Reinsurance notifications	"Claim Preupdate rules for reinsurance notifications" on page 82
Claim health metrics	"Claim Preupdate rule for claim health metric updates" on page 82
Coverage in question	"Claim Preupdate rules for coverage in question" on page 82
Change of loss date and removing policy verification	"Claim Preupdate rule to unverify policy if loss date changes" on page 83
Workload assignment balancing	"Claim Preupdate rules to balance workload assignment" on page 83
Workers' compensation	"Claim Preupdate rule to set workers' comp class code" on page 84
Executing Claim Preupdate business rules	"Claim Preupdate rule to execute business rules" on page 84

Claim Preupdate rules for document creation

There are two Claim Preupdate rules that create documents if the claim description field has changed. Guidewire does not enable these Gosu rules in the base configuration.

In the base configuration, this rule set contains the following rules.

CPU01000 - Create Document Synchronously

Sample rule that shows how to create and store a document synchronously. Guidewire disables this rule in the base configuration.

CPU02000 - Create Document Asynchronously

Sample rule that shows how to create and store a document asynchronously. Guidewire disables this rule in the base configuration.

Claim Preupdate rules for Special Investigation Unit (SIU)

The Claim Preupdate SIU rules and the Claim Exception SIU rules work together to assist in determining claim fraud. These rules identify certain characteristics of a claim that increase the suspicion that fraud has occurred and assign points to each of these characteristics.

See also

- For descriptions of the SIU rules, see "Detecting claim fraud" on page 167.
- See also "Claim Exception rules" on page 62.

Claim Preupdate rule for metropolitan police report requests

The Metropolitan Reporting Bureau at `www.metroreporting.com` is a nationwide service in the United States for obtaining accident and incident reports. The Claim Preupdate rule set has one rule that works with this service, CPU08000 - Metro Report Request. This rule starts an internal process to determine the type of each report requested by a claim, one at a time. The rule verifies that the claim is in draft state and then calls `libraries.Metro.validateAndStartReportFlows(claim)`. This method verifies that the claim contains the data required to request each of the requested reports.

- If there are any missing fields, ClaimCenter creates an activity called Metropolitan Report Request Failed and assigns it to the user that created the MetroReport request. The **Description** text area of the activity contains information on the type of report that was requested. It also contains any data that must be supplied to successfully make the request. For each `MetroReport` request for which the required data is not defined on the claim, ClaimCenter marks the status as `insufficientData`.
- If the required data is specified and the claim update completes successfully, ClaimCenter changes the `MetroReport` status to `validated`. After the report status becomes valid, ClaimCenter changes the Metropolitan report status to `Sending Order`.

IMPORTANT: ClaimCenter integrates with Metropolitan Reporting Bureau in the base configuration. However, you must have an account with Metropolitan Reporting Bureau for report requests to work.

See also

- For information on Metropolitan Police Reports and the MetroReport object, see the *Integration Guide*.
- For information on event rules for Metropolitan Police Reports, see"MetroReport Event Message rules" on page 55.

Claim Preupdate rules for catastrophes

In the base configuration, Guidewire provides several Claim Preupdate rules related to catastrophe categorization in the Claim Preupdate rule set.

CPU09000 - Related to Catastrophe

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CPU13000 - Catastrophe History

Determines if there is a change to the claim **Catastrophe** field. If there is, the rule updates History and marks any open *Review for Catastrophe* activity complete.

CPU19000 - Geocode Catastrophe Claims

Determines if the claim **Catastrophe** field is set (not `null`) and if the claim's loss location has never been geocoded. If so, the rule schedules the claim's loss location for geocoding. This rule takes effect only if geocoding is enabled.

ClaimCenter uses these rules in conjunction with the catastrophe profile that an administrative user creates through the **Administration** > **Catastrophes** screen.

Claim Preupdate rules for changed primary diagnosis code

In the base configuration, Guidewire provides the following Claim Preupdate rules to record if a primary diagnosis for an injury incident has changed.

CPU10000 - Primary Diagnosis Code Changed

This rule screens for injury incidents that have injury diagnoses, for the following rules.

CPU10100 - WC

If the loss type is Workers' Comp, this rule determines:

- If the diagnosis change was to set the diagnosis to primary for the first time. In this case, the rule saves the primary diagnosis ICD code only to the history.
- If an existing primary diagnosis has changed. In this case, the rule saves both the original code and the new code to history.

CPU10200 - Default

This rule has the same logic as the WC rule, except that it applies to all other injury incidents if the WC rule does not apply.

Claim Preupdate rules for subrogation

In the base configuration, Guidewire provides the following Claim Preupdate Gosu rules that apply to subrogation.

CPU10800 - Create Subro Summary

This rule is conditional on a claim subrogation that both exists, as determined by the method call `claim.ActivateSubroModule`, and has an empty subrogation summary. If there is such a subrogation on the claim, this rule creates a summary for it.

The method `ActivateSubroModule` is a `Claim` enhancement defined in `GWSubroNonFinancialClaimEnhancement.gsx` that applies various tests to see if a subrogation exists and returns a `Boolean` value based on these tests.

CPU12000 - Subro Status - Close

This rule is conditional on a change of the subrogation status to `closed`. If it has been, this rule sets the close date on the summary to today's date and creates a history event for the closure.

CPU14000 - Subro Adverse Party role

This rule is conditional on a claim subrogation that has a summary. If so, it Iterates over the subrogation adverse parties. If a party has a changed `AdverseParty` field, the rule iterates over the `claim.Contact` objects to find the one with the same contact as that of the adverse party. If a claim contact is found that does not have the role `"adverseparty"`, the rule adds this role as a new `ClaimContactRole` to this claim contact.

See also

- *Application Guide*

Subrogation rules and scenarios

You can find some financial rules governing subrogation in Guidewire Studio. In the Studio **Project** window, navigate to **configuration** > **config** > **Rule Sets** > **Preupdate** > **ClaimPreupdate**.

The following descriptions are scenarios for subrogation rules you could add to your application. For more information on configuring rules, see *Gosu Rules Guide*.

Flag a claim as a possible subrogation opportunity

- **Condition** – The claim's subrogation status has not been set and the loss cause is a rear-end collision.
- **Action** – Set the claim's subrogation status to `Review` and the reason to `Rear-end collision`.

You can add more conditions that flag a claim for possible subrogation, or you can add an action to create an activity to review the claim.

Create an activity to send the first subrogation letter

- **Condition** – The **Strategy** is **Pursue**, and this activity does not exist.
- **Action** – Create an activity that creates and sends the first dunning letter with a particular template.

Create an activity to send the second subrogation letter

This rule can be part of the **Activity Closed** rule set. An activity to send a third demand letter would be similar.

- **Condition** – The previous activity is complete and a certain time has passed.
- **Action** – Create an activity that completes and sends the second dunning letter by using its particular template.

Claim Preupdate rule for auto first and final

In the base configuration, Guidewire provides a single Claim Preupdate rule that applies to an auto first and final claim. This kind of claim is a simple auto claim, such as a cracked windshield, that requires only a first and final payment.

CPU15000 - AutoFirstAndFinal

This rule is conditional on a claim of type `FirstAndFinal` that is open, has a loss type of Auto, and has one exposure and one transaction.

The rule auto-completes all non-approval activities and then sets the `FaultRating` to `nofault`.

Claim Preupdate rules for travel

In the base configuration, Guidewire provides the following Claim Preupdate rules that apply to Travel claims with at least one incident.

CPU16000 - Travel

This rule filters for claims with a loss type of `TRAV` and passes them to its child rules for processing.

CPU16100 - Trip Incident

This rule filters for travel related claims that have at least one trip incident and passes them to the next rule for processing. A *trip incident* concerns cancellation or delay of the trip. For example, if a trip is cancelled, it might be necessary to reimburse transportation and accommodation costs that were prepaid.

CPU16110 - Trip Segment

This rule looks for trip segments in the claim's trip incidents and calls the method `onPreUpdate` for each segment found, passing it a value indicating transportation as the expense. The method, declared in `GWTripExpenseDelegateEnhancement.gsx`, evaluates the `Assessment` field of the claim's trip expenses. Based on the value it finds in this field, the method can create history events and activities for further action.

CPU16120 - Trip Accommodation

This rule looks for trip accommodations in the claim's trip incidents and calls the method `onPreUpdate` for each accommodation found, passing it a value indicating accommodation as the expense. See the previous rule for a description of what `onPreUpdate` does.

See also

- *Application Guide*

Claim Preupdate rules for reinsurance notifications

In the base configuration, Guidewire provides the following Claim Preupdate rules that apply to reinsurance notifications.

CPU17000 - RI Notifications

This rule serves as a folder grouping two reinsurance notification rules.

CPU17100 - Exceeds Notification Threshold

This rule calls the method `setReinsuranceNotificationIfGrossTotalIncurredOverThreshold`, a `Claim` enhancement defined in `GWClaimReinsuranceNotificationEnhancement.gsx`. The method determines if the notification threshold has been exceeded for each reinsurance agreement. If so, the method can create an activity to review the claim for reinsurance, as appropriate.

CPU172000 - Set Reinsurance Fields from User

This rule can add notes to the claim based on user changes:

- A change in `ReinsuranceReportable` status that is not accompanied by a change in `ReinsuranceFlaggedStatus`
- A change in the `Text` field that occurs with either setting (flagging) or clearing (unflagging) the `ReinsuranceFlaggedStatus` field.

See also

- *Application Guide*

Claim Preupdate rule for claim health metric updates

In the base configuration, Guidewire provides a single Claim Preupdate rule that applies to claim health updates.

CPU18000 - Update Claim Health

This rule calls `claim.scheduleHealthUpdate` to update the claim health metrics.

See also

- *Application Guide*

Claim Preupdate rules for coverage in question

In the base configuration, Guidewire provides the following Claim Preupdate rules that apply to coverage being in question.

CPU20000 - Coverage in question

This rule filters for claims that have coverage in question or claims for which the coverage in question has changed and passes them to its child rules.

CPU20100 - Set Coverage in Question

This rule applies tests to determine if it will set `claim.CoverageInQuestion` to `true`. All these tests must be `true` for the rule to apply:

- The original `Boolean` value of `CoverageInQuestion` was not `true`.
- There is no history entry indicating that coverage is in question.
- A call to `claim.isCoverageInQuestion` returns `true`.

CPU 20200 - Activities

This rule applies tests to determine if it will create an activity related to coverage being in question. The rule applies if `claim.CoverageInQuestion` is `true` and there is no activity already for checking coverage in question or for checking on a recovery opportunity:

- If no cost payments have been made, the rule creates an activity for checking coverage in question.
- If cost payments have been made, the rule creates an activity for checking into a recovery opportunity.

CPU 20300 - Create history

This rule makes a history entry if there has been a change to the `claim.CoverageInQuestion` field:

- If `claim.isCoverageInQuestion` returns `true`, the new history entry says that coverage is in question for the claim.
- If coverage is not in question and the previous value of `claim.CoverageInQuestion` was `true`, the new history entry says that the user turned off coverage in question.

See also

- *Application Guide*

Claim Preupdate rule to unverify policy if loss date changes

In the base configuration, Guidewire provides a single Claim Preupdate rule that removes policy verification if the claim's loss date changes.

CPU 21000 - Unverify Policy if Loss Date Changed

This rule changes the policy from verified to unverified if the loss date, `claim.LossDate`, changes.

See also

- *Integration Guide*

Claim Preupdate rules to balance workload assignment

In the base configuration, Guidewire provides the following Claim Preupdate rules that balance workload assignments.

CPU30000 - Workload Assignment Balancing

This rule tests to see if the weighted workload assignment feature is enabled in configuration. If `gw.api.system.CCConfigParameters.WeightedAssignmentEnabled.Value` is `true`, its child rules run.

CPU30100 - Claim Closed

If the claim's state has changed to closed, this rule makes an appropriate log entry. The rule then updates the current user's workload to account for the closed claim.

CPU30200 - Claim Reassignment

If the claim has been reassigned to the current user, this rule makes an appropriate log entry. The rule then updates the current user's workload to account for the reassignment.

CPU30300 - Claim Reopened

If the claim has been reopened after previously being closed, this rule makes an appropriate log entry. The rule then updates the current user's workload to account for the reopened claim.

CPU30400 - New Claim

If the claim is new or has just been assigned for the first time, this rule makes an appropriate log entry. The rule then updates the current user's workload to account for the new claim.

CPU30500 - Claim Workload Affected

This rule applies if the workload has changed but the assigned user has not. In particular:

- The claim is open.
- A field on the claim that affects workload has changed.
- The assigned user has not changed.

- The claim's state is not affected.

If all these conditions are true, the rule makes a log entry indicating that the workload has changed for the user for this claim. The rule then updates the current user's workload to account for the change.

See also

- *Application Guide*
- *Configuration Guide*

Claim Preupdate rule to set workers' comp class code

In the base configuration, Guidewire provides the following Claim Preupdate rules that set the Workers' Compensation class code.

CPU31000 - Workers Comp
This rule filters for a claim loss type of workers' compensation. In the base configuration, it has one enabled child rule.

CPU31100 - Class Code Selection
This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CPU31200 - Compensability Decision
This rule adds a history entry whenever there is a change in Workers' Compensation compensability for the claim. It also updates the claim's `DateCompDcsnMade` field to the current date.

See also

- *Application Guide*
- *Integration Guide*

Claim Preupdate rule to execute business rules

In the base configuration, Guidewire provides the following Claim Preupdate rule that triggers execution of business rules that trigger on an update to a claim.

CPU40000 - BizRules
This rule executes the business rules that are set to trigger on update of a claim. Guidewire recommends that you always run this rule after the other Claim Preupdate Gosu rules have completed.

See also

- *Application Guide*

Contact Preupdate rules

Use the Contact Preupdate rules to ensure correct modification of contacts that are added to or already parties to claim. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ContactPreupdate**

Guidewire provides the following rules in the base configuration:

COP01000 - Update Check Address
This rule applies whenever ClaimCenter receives a change of address from the contact management system for a contact that is in sync with that system. This rule ensures that all future dated checks with this contact as a payee have the new contact address, and that the old address is removed from the check.

COP02500 - Set all Vendors to auto sync
All contacts added to the claim are set to automatically synchronize with the contact management system.

COP03000 - Add default tags

This rule ensures that the required, minimum contact tags are applied to each contact added to a claim. In the base configuration, every claim contact of any type must have the Claim Party tag, and every vendor contact must have the Vendor tag.

You can change the minimal tags applied to contacts by configuring the class `CCContactMinimalTagsImpl`.

See also

- *Contact Management Guide*

Exposure Preupdate rules

Use the Exposure Preupdate rules to modify exposures and related entities. If a rule encounters an exception, the bounding database transaction is rolled back, which prevents the update of the exposure.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ExposurePreupdate**

In the base configuration, ClaimCenter provides the following Exposure Preupdate rules:

EPU01000 - Assign default coverage

This rule ensures that coverage, if not already set, is set properly for three workers' compensation coverage subtypes:

- `WCWorkersCompWAGES`
- `WCEmpLiabCov`
- `WCWorkersCompMED`

EPU02000 - Salvage

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EPU03000 - SI Triggers

This rule is the parent for two Exposure Preupdate SIU rules, which are disabled in the base configuration. See "Exposure Preupdate SIU rules" on page 171.

EPU04000 - Update Deductible on Updated Exposure Coverage

This rule checks for a deductible on coverage. If there is one, then the rule creates a deductible tracking object on the claim.

EPU05000 - Update Deductible on Updated Coverage Deductible

This rule calls methods that detect if a deductible has changed and ensures that the deductible is updated if there is a need to do so.

EPU10000 - Workload Assignment Balancing

This rule tests to see if the weighted workload assignment feature is enabled in configuration. If `gw.api.system.CCConfigParameters.WeightedAssignmentEnabled.Value` is `true`, the following rules run.

EPU10100 - Exposure Closed

If the exposure's state has changed to closed, this rule makes an appropriate log entry. The rule then updates the current user's workload to account for the closed exposure.

EPU10200 - Exposure Reassignment

If the exposure has been reassigned to the current user, this rule makes an appropriate log entry. The rule then updates the current user's workload to account for the reassignment.

EPU10300 - Exposure Reopened

If the exposure has been reopened after previously being closed, this rule makes an appropriate log entry. The rule then updates the current user's workload to account for the reopened exposure.

EPU10400 - New Exposure

If the exposure is new or has just been assigned for the first time, this rule makes an appropriate log entry. The rule then updates the current user's workload to account for the new exposure.

EPU10500 - Exposure Workload Affected

This rule applies if the workload has changed but the assigned user has not. In particular:

- The exposure is open.
- A field on the exposure that affects workload has changed.
- The assigned user has not changed.
- The exposure's state is not affected.

If all these conditions are true, the rule makes a log entry indicating that the workload has changed for the user for this exposure. The rule also:

- Updates the current user's workload to account for the change.
- Creates a new `SITrigger` if none exists.
- Updates the claim's `SIScore` by the number of points specified in the rule.

EPU20000 - BizRules

This rule executes the business rules that are set to trigger on update of an exposure. The rule runs after the other Exposure Preupdate Gosu rules have completed.

See also

- *Application Guide*

Group Preupdate rules

Use Group Preupdate rules to modify groups and related entities. In the base configuration, there are no rules in this rule set. To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **GroupPreupdate**

ClaimCenter runs the Group Preupdate rules upon a save of an instance of a `Group` entity. Any exceptions found cause the bounding database transaction to roll back, thus effectively vetoing the update.

After running these rules, Claim Center runs the Group Validation rules.

See also

- "Group Validation rules" on page 113

Matter Preupdate rules

Use the Matter Preupdate rules to modify matters and related entities. This rule set is empty in the base configuration. To work with these rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **MatterPreupdate**

In the following example, the rule creates a new activity to review the claim file 30 days before the date set to litigate the claim in court.

```
CONDITION (matter : entity.Matter):
where (Activity.ActivityPattern != ActivityPattern.finder.getActivityPatternByCode( "trial_review" ) ) )

ACTION (matter : entity.Matter, actions : gw.rules.Action):
var act = matter.Claim.newActivity( ActivityPattern.finder.getActivityPatternByCode( "trial_review" ), null )
act.TargetDate = gw.api.util.DateUtil.addDays( act.Matter.TrialDate, -30 )
act.assignToClaimOwner()
```

Policy Preupdate rules

Use the Policy Preupdate rules to modify policies and related entities. To work with these Gosu rules, navigate in the Studio **Project** window to **configuration** > **config** > **Rule Sets** > **Preupdate** > **PolicyPreupdate**.

In the base configuration, this rule set has one rule that is disabled by default, PPU01000 - Assign Default Coverage. The rule is disabled because its function, linking workers' compensation exposures to policy coverages, is now handled in the **New Claim** wizard. See the comments for this rule for more information.

In the following example, the rule standardizes the policy number by using standard Gosu library functions. The nested functions trim any spaces in the number and turn all alpha characters to lower case.

```
CONDITION (policy : entity.Policy):
return true

ACTION (policy : entity.Policy, actions : gw.rules.Action):
policy.PolicyNumber =
        gw.api.util.StringUtil.toLowerCase( gw.api.util.StringUtil.trim( policy.PolicyNumber ) )
```

RIAgreementPreupdate rules

ClaimCenter runs the `RIAgreement` Preupdate rule set whenever a user adds or edits a reinsurance agreement. Use this rule set to modify `RIAgreement` entities and related entities. Exceptions cause the bounding database transaction to roll back, effectively vetoing the update. This rule set is empty by default in the base configuration.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **RIAgreementPreupdate**

IMPORTANT: Modifying financial values on an `RIAgreement` object that is used by the `IReinsuranceTransactionPlugin` to set up `RITransaction` objects does not affect the calculations in the current commit. The reason is that the `IReinsuranceTransactionPlugin` has already run and created the `RITransaction` objects. Instead, modify `RIAgreement` objects between the invocation of the `IReinsurancePlugin` and the invocation of the `IReinsuranceTransactionPlugin`. `IReinsurancePlugin` retrieves the agreements from an external system. `IReinsuranceTransactionPlugin` creates `RITransaction` objects based on the regular financials values and the `RIAgreement` objects.

See also

- "RIAgreement Validation rules" on page 115
- *Application Guide*

RITransactionSetPreupdate rules

ClaimCenter runs the `RITransactionSet` Preupdate rule set whenever a user adds or edits a reinsurance transaction. This rule set is empty by default in the base configuration.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **RITransactionSetPreupdate**

Do not modify the amount of an `RITransaction` in this rule set. Exceptions cause the bounding database transaction to roll back, effectively vetoing the update. You can use rules in this rule set to update extension fields. However, there are limitations on using this rule set.

IMPORTANT: The `ClaimAmount`, `ReportingAmount`, `ReserveLine`, and `RIAgreement` fields of an `RITransaction` entity must not be modified in the `RITransactionSet` Preupdate rules. Changes made in this rule set to `RITransaction` entities do not affect automatic reinsurance financials calculations for `RITransaction` entities because the calculations are updated whenever the `RITransaction` object is created. You can modify the `IReinsuranceTransactionPlugin` implementation to set the `ClaimAmount`, `ReportingAmount`, `ReserveLine`, and `RIAgreement` fields correctly whenever an `RITransaction` object is created.

See also

- "RITransactionSet Validation rules" on page 116
- *Application Guide*
- *Integration Guide*

TransactionSet Preupdate rules

ClaimCenter runs the Transaction Set preupdate rules whenever a user adds or edits a transaction. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **TransactionSetPreupdate**

In working with the Transaction Set preupdate rules, keep the following in mind:

- Do not add a large number of rules to this rule set. This rule set can seriously affect performance because there are typically many transactions active in the system.
- Do not modify fields that affect financial calculations, such as the transaction `Amount`, the `Eroding` state, the `Exposure`, `CostType`, or `CostCategory`, or the `ScheduledSendDate` on a payment's check. You can modify other fields that do not affect financial calculations, including extension fields
- Do not create new transactions in preupdate rules. Preupdate rules do not run on transactions created through preupdate rules. Instead, create the new transactions in a class that implements the `IPreUpdateHandler` plugin interface, which runs immediately before preupdate rules. In the base implementation, ClaimCenter provides a class that you can use for this purpose, `CCPreupdateHandlerImpl`, which is registered in the plugin registry `IPreupdateHandler.gwp`. See the *Integration Guide*.

In the base configuration, ClaimCenter provides the following Transaction Preupdate rules.

TPU01000 - Create Activity After Check Denial

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

TPU02000 - Create Activity After Recovery Denial

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

TPU03000 - Add Budget Lines to Matter

This rule runs whenever you write a check to make a payment on a matter. It checks to see if you have set the **Budget Lines** screen to display for each matter. If so, and the line item category for the check matches a line category used by this screen, the rule adds a budget line item record for the matter.

More specifically, the rule runs on new `CheckSet` transactions if the script parameter `UtlizeMatterBudget` has been set to `true`. The rule looks for transactions that include a payment with a cost category of `"legal"` and have an associated `Matter`. For each of these transactions, if there is no budget line record for the line item category, the rule adds one.

See the *Application Guide*.

TPU04000 - Reinsurance

Guidewire disables this rule in the base configuration. It is designed to find transaction sets that have not exceeded the claim's reinsurance reporting threshold so the two reinsurance rules that follow can process these transaction sets.

TPU04100 - Large Loss Identification

Guidewire disables this rule in the base configuration. It calls `transactionSet.Claim.setReinsuranceIfTotalIncurredOverThreshold`, which is defined in `GWClaimReinsuranceEnhancement`.

TPU04200 - Continuous WC Injury Payments Exceeds 12 Months

Guidewire disables this rule in the base configuration. It runs if the loss type is workers' comp and the transaction set is a `CheckSet`. The rule calls `transactionSet.Claim.setReinsuranceIfContinuousWCInjuryPaymentsExceed12Months`, which is defined in `GWClaimReinsuranceEnhancement`.

TPU05000 - Large Loss Notification

Guidewire disables this rule in the base configuration. It determines if the claim's large loss notification status is either null or set to `"None"` and the claim exceeds the large loss threshold. If so, the rule calls `transactionSet.Claim.addLargeLossEvent` to notify the policy system. The method is defined in `GWClaimReinsuranceEnhancement`.

TPU06000 - Unlink Deductible After Check Denial

This rule runs on checks that have been denied. For each check in the check set, if the status changed to `"Denied"`, the rule calls `check.unlinkDeductibles`. This method, defined in `GWCheckDeductibleEnhancement`, finds all the deductible line items for payments on the check and unlinks them from their deductibles.

TPU07000 - Update Claim Health

This rule calls `claim.scheduleHealthUpdate` to update the claim health metrics, which might have changed as a result of running the previous rules in this rule set.

TPU08000 - Clean Up Payment Method Related Check Fields

This rule is useful for cleaning up data if the payment method changes. It nulls out unwanted fields if the user changes the payment method between check and EFT (Electronic Check Transfer).

This rule calls the method `gw.entity.GWCheckEnhancement.removeUnusedPaymentMethodRelatedFields` on the `Check` object. In the base configuration, this method nulls out fields related to a specific payment method.

For payment by `Check`, the method nulls the following fields:

- `BankAccountNumber`
- `BankName`
- `BankAccountType`
- `BankRoutingNumber`

For payment by EFT, the method nulls the following fields:

- `MailingAddress`
- `MailTo`
- `DeliveryMethod`

TPU04000 - RI Notification Threshold Exceeded

This rule calls `transactionSet.Claim.setReinsuranceIfTotalIncurredOverThreshold`, which is defined in `GWClaimReinsuranceNotificationEnhancement`. The method determines if the notification threshold has been exceeded for each reinsurance agreement. If so, the method can create an activity to review the claim for reinsurance, as appropriate.

TPU05000 - BizRules

This rule executes the business rules that are set to trigger on update of a transaction set. The rule runs after the other Transaction Set Preupdate Gosu rules have completed.

TPU06000 - Escalate Instant Payment Check Status

This rule escalates an instant check in `Awaiting submission` status to `Requesting` if the check's scheduled send date has been reached and the instant disbursements (instant payment integration) feature is enabled in `config.xml`. An instant check is a check with a payment method of `Instant` and represents a payment processed by an external payment gateway to an individual.

See also

- "TransactionSet Validation rules" on page 117
- *Application Guide*

User Preupdate rules

User Preupdate rules modify `User` and related entities prior to user validation. ClaimCenter runs the User Preupdate rules on a save of the `User` entity. Any exceptions found in a rule in this rule set cause the bounding database transaction to roll back, thus effectively vetoing the update.

To work with these Gosu rules, navigate in the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **UserPreupdate**

In the base configuration, this rule set contains a single rule only.

UPU00005 - User Workload Resync On Status Change

The rule first verifies that weighted workload is enabled and that the user's vacation status or active status has changed. It then logs the changes that were made and resynchronizes the workload for the group to which the user belongs. See the rule in Guidewire Studio for more information.

ClaimCenter runs the User Validation rule set after this rule set. See "User Validation rules" on page 119.

See also

- *Application Guide*
- *Configuration Guide*

Reopened

The Reopened rule sets provide a means of taking automatic action whenever a `Claim`, `Exposure`, or `Matter` is reopened.

Claim Reopened rules

The Claim Reopened rule set executes after a claim has been reopened, after Claim Reopened Validation rules run but before preupdate and validation rules run. These rules take automated actions on reopening a claim.

To work with these Gosu rules, navigate in the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Reopened** > **ClaimReopened**

In the base configuration, ClaimCenter provides the following Claim Reopened rules.

CRO01000 - Notify external systems

If a claim is reopened, this rule adds a Claim Reopened event to notify external systems.

CRO02000 - Allow AutoSync for Related Contacts

Helps facilitate the synchronization of contact information between ClaimCenter and ContactManager.

CRO03000 - Sample rule to remove purge date

Guidewire disables this rule in the base configuration. If enabled, this rule removes a purge date on the claim that you reopen. If you enable it, you must also enable the following Claim Closed rule:

- CCL03000 - Sample rule to set purge date

These Gosu rules are samples only and are not active in the base configuration. The two rules work as a pair. You must enable or disable the two rules in conjunction with each other.

CRO04000 - Clear archive eligibility date

If claim archiving is enabled, this rule removes the archive eligibility date on the reopened claim.

See also

- "Claim Closed rules" on page 49
- *Contact Management Guide*

- *Application Guide*

Exposure Reopened rules

The Exposure Reopened rule set executes after an exposure has been reopened, after Exposure Reopened Validation rules run, but before preupdate and validation rules run. These rules take automated actions on reopening an exposure.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Reopened** > **ExposureReopened**

In the base configuration, ClaimCenter provides the following Exposure Reopened rule.

ERO01000 - Notify external systems
If an exposure is reopened, this rule adds a Exposure Reopened event to notify external systems.

Example Exposure Reopened rule

In the following example, the rule uses a script parameter to set the reserve amount for the logged-in user if a Workers' Compensation exposure is reopened. It also uses `try... catch...` exception handling to deal with potential exceptions.

```
CONDITION (exposure : entity.Exposure):
return exposure.Type == "WCInjuryDamage"

ACTION (exposure : entity.Exposure, actions : gw.rules.Action):
try {
  exposure.setAvailableReserves( "claimcost", "medical", ScriptParameters.InitialReserve_ReopenClaim,
        User.util.CurrentUser )
} catch (e) {
  CCLoggerCategory.Rules.error("Caught exception: " + e)
}
```

See also

- *Application Guide*
- *Configuration Guide*

Matter Reopened rules

The Matter Reopened rule set executes after a matter has been reopened, after Matter Reopened Validation rules run, but before preupdate and validation rules run. These rules take automated actions on reopening the matter.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Reopened** > **MatterReopened**

In the base configuration, ClaimCenter provides the following Matter Reopened rule.

MRO01000 - Set Litigation Status
If a matter is reopened, this rule sets `matter.Claim.LitigationStatus = "litigated"`.

Segmentation

The purpose of the segmentation rules is to set the `Segment` property on new claims and exposures based on the complexity, severity of damage, and other attributes. Optionally, you can also use these rules to set the `Strategy` property. Segment values are defined in the **ClaimSegment** typelist.

ClaimCenter runs segmentation rules just prior to running assignment rules if automated assignment is selected for a new claim or exposure. Typically, values set for the segmentation for a new claim or exposure are used in determining the assignment of the claim or exposure.

To arrive at a decision on the segment of an exposure, ClaimCenter examines the following:

- The fields on the exposure, such as, for an injury, severity, body part injured, nature of injury, first as opposed to third-party claimant
- Possibly, the fields on the claim, such as the cause of loss and the time between the loss date and the time that the claim was reported

It is easier to make decisions about the segmentation of the claim as a whole after each exposure has been categorized. For example, an auto claim could be categorized as complex if there are any third-party injury exposures. For this reason, the default ClaimCenter workflow first segments each exposure and then segments the claim.

See also

- *Application Guide*

Claim Segmentation rules

You can use the Claim Segmentation rules to categorize each claim as a whole based on complexity, severity of damage, and other attributes. The results set the `Segment` field of a `Claim`.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Segmentation** > **ClaimSegmentationRules**

In the base configuration, ClaimCenter provides the following Claim Segmentation rules.

CSG01000 - Auto

Filters for auto claims, which have a loss type of `"AUTO"`, and passes these claims to its child rules, the next two rules. If the loss type is not `"AUTO"`, the rule CSG02000 - Property, executes.

CSG01100 - Auto

Finds an exposure with the most important auto segmentation. If not found, the code repeats for the rest of the segmentations in reverse order of importance.

For the first exposure found by these searches, the rule sets the claim segment to that of the found exposure and then exits the claim segmentation rules.

If no exposure is found, this rule drops into the default rule.

CSG01200 - Auto default

This rule sets the claim segment to the lowest auto segment value, `"auto_low",` and then exits the claim segmentation rules.

CSG02000 - Property

Filters for property claims, which have a loss type of `"PR"`, and passes these claims to its child rules, the next two rules. If the loss type is not `"PR"`, the rule CSG03000 - WC, executes.

CSG02100 - Property

Declares an array of property segment values to be used to find an exposure with the most important segmentation. This array is used to order the segments from lowest to highest. If you add new segment types, you must update the array declared in this rule.

Finds the exposure with the highest segment value in the array and assigns the claim segment to that exposure segment value.

If no exposure is found, this rule drops into the default rule.

CSG02200 - Property default

This rule sets the claim segment to the lowest property segment value, `"prop_low",` and then exits the claim segmentation rules.

CSG03000 - WC

Filters for workers' comp claims, which have a loss type of `"WC"`, and passes these claims to its child rules, the next two rules. If the loss type is not `"WC"`, the rule CSG04000 - GL, executes.

CSG03100 - WC

Finds an exposure with the most important workers' comp segmentation. If not found, the code repeats for the rest of the segmentations in reverse order of importance. For the first exposure found by these searches, the rule sets the claim segment to that of the found exposure and then exits the claim segmentation rules. If no exposure is found, this rule drops into the default rule.

CSG03200 - WC default

This rule sets the claim segment to the lowest workers' comp segment value, `"wc_med_only",` and then exits the claim segmentation rules.

CSG04000 - GL

Filters for liability claims, which have a loss type of `"GL"`, and passes these claims to its child rules, the next two rules. If the loss type is not `"GL"`, the next rule, CSG05000 - Travel, executes.

CSG04100 - GL

Finds an exposure with the most important liability segmentation. If not found, the code repeats for the rest of the segmentations in reverse order of importance. For the first exposure found by these searches, the rule sets the claim segment to that of the found exposure and then exits the claim segmentation rules. If no exposure is found, this rule drops into the default rule.

CSG04200 - GL

This rule sets the claim segment to the lowest liability segment value, `"liab_low",` and then exits the claim segmentation rules.

CSG05000 - Travel

Filters for travel claims, which have a loss type of `"TRAV"`, and passes these claims to its child rules, the next two rules. If the loss type is not `"TRAV"`, the rule set exits the claim segmentation rules.

CSG05100 - Travel

Finds an exposure with the most important travel segmentation. If not found, the code repeats for the rest of the segmentations in reverse order of importance. For the first exposure found by these searches, the rule sets the claim segment to that of the found exposure and then exits the claim segmentation rules. If no exposure is found, this rule drops into the default rule.

CSG05200 - Travel default

This rule sets the claim segment to the lowest travel segment value, `"travel_low",` and then exits the claim segmentation rules.

Exposure Segmentation rules

You can use the Exposure Segmentation rules to categorize each exposure based on complexity, severity of damage, and other attributes. The results set the `Segment` field of an `Exposure`. These rules run before the Claim Segmentation rules so that claim segmentation rules can use the results of exposure segmentation.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Segmentation** > **ExposureSegmentationRules**

In the base configuration, ClaimCenter provides the following Exposure Segmentation rules.

ESG01000 - Auto

Filters for a claim associated with this exposure that has a loss type of `"AUTO"`, and passes this claim to its child rules, the next four rules. If the loss type is not `"AUTO"`, the rule ESG02000 - Property, executes.

ESG01100 - Auto low

The rule tests for the following conditions:

- Exposure type is `"vehicledamage"`, `"BodilyInjuryDamage"`, `"PIPDamages"`, `"PropertyDamage"`, or `"TowOnly"`.
- Severity of the incident is `"minor"` or `"moderate-auto"`.

If the tests evaluate to `true`, the rule sets `exposure.Segment = "auto_low"` and exits the rule set. If none of these conditions are `true`, the next rule executes.

ESG01200 - Auto medium

The rule tests for the following conditions:

- Exposure type is personal injury damages or loss party is third party.

If the tests evaluate to `true`, the rule sets `exposure.Segment = "auto_mid"` and exits the rule set. If none of these conditions are `true`, the next rule executes.

ESG01300 - Auto high

Tests to see if any one of the following is true:

- Severity of the incident is a severe injury, a major injury, or a fatality.
- The vehicle was a total loss.

If any of the tests evaluate to `true`, the rule sets `exposure.Segment = "auto_high"` and exits the rule set.

If none of these conditions are `true`, the default rule executes.

ESG01400 - Auto default

If the exposure segment has not been set yet, this rule sets it to `"auto_mid"` and exits the rule set.

ESG02000 - Property

Filters for property claims, which have a loss type of `"PR"`, and passes these claims to its child rules, the next four rules. If the loss type is not `"PR"`, the rule ESG03000 - WC, executes.

ESG02100 - Property Low

The rule tests to see if both of the following conditions are true:

- Exposure type is either `"PersonalPropertyDamage"` or `"PropertyDamage"`.
- Severity of the incident is `"minor"`.

If the tests evaluate to `true`, the rule sets `exposure.Segment = "prop_low"` and exits the rule set. If none of these conditions are `true`, the next rule executes.

ESG02200 - Property Medium

The rule tests to see if any one of the following is true:

- Severity of the incident is moderate property,
- Exposure type is loss of use damage.
- The loss party was a third party.

If any of the tests evaluate to `true`, the rule sets `exposure.Segment = "prop_mid"` and exits the rule set.

If none of these conditions are `true`, the next rule executes.

ESG02300 - Property High

The rule tests to see if both of the following conditions are true:

- Severity of the incident is either `"major-prop"` or `"fatal"`.
- The loss party is the insured party.

If both tests evaluate to `true`, the rule sets `exposure.Segment = "prop_high"` and exits the rule set. If none of these conditions are `true`, the default rule executes.

ESG02400 - Property default

If the exposure segment has not been set yet, this rule sets it to `"prop_mid"` and exits the rule set.

ESG03000 - WC

Filters for workers' comp claims, which have a loss type of `"WC"`, and passes these claims to its child rules, the next three rules. If the loss type is not `"WC"`, the rule ESG04000 - GL, executes.

ESG03100 - Medical

If the exposure type is workers' comp injury damage, this rule assigns the segment `"wc_med_only"` and exits the rule set. If not, the next rule executes.

ESG03200 - Lost Wages

If the exposure type is lost wages, this rule assigns the segment `"wc_lost_time"`. If not, the default rule executes.

ESG03300 - Default

Sets the segment to `"wc_liability"` and then exits the rule set.

ESG04000 - GL

The rule filters for liability claims, which have a loss type of `"GL"`, and passes these claims to its child rules, the next four rules. If the loss type is not `"GL"`, the rule ESG05000 - Travel, executes.

ESG04100 - GL Low

The rule tests to see if any of the following is true:

- Exposure type is general damage or loss of use damage.
- Severity of the incident is minor.

If any of the tests evaluate to `true`, the rule sets `exposure.Segment = "liab_low"` and exits the rule set. If none of these conditions are `true`, the next rule executes.

ESG04200 - GL Medium

The rule tests to see if any of the following is true:

- Exposure loss party is a third party.
- Severity of the incident is `"moderate-gen"`.

If any of the tests evaluate to `true`, the rule sets `exposure.Segment = "liab_mid"` and exits the rule set. If none of these conditions are `true`, the next rule executes.

ESG04300 - GL High

If the exposure severity is `"severe-gen"`, this rule assigns the segment `"liab_high"` and exits the rule set. If not, the default rule executes.

ESG04400 - GL Default

The rule sets the segment to `"liab_mid"` and exits the rule set.

ESG05000 - Travel

The rule filters for travel claims, which have a loss type of `"TRAV"`, and passes these claims to its child rules, the next four rules. If the loss type is not `"TRAV"`, the rule set exits the Claim Segmentation rules.

ESG05100 - Travel Low

The rule tests to see if any of the following is true:

- Exposure type is baggage or trip cancellation or delay.
- Severity of the incident is minor.

If any of the tests evaluate to `true`, the rule sets `exposure.Segment = "travel_low"` and exits the rule set. If none of these conditions are `true`, the next rule executes.

ESG05200 - Travel Medium

The rule tests to see if any of the following is true:

- Exposure type is `"MedPay"` or `"PersonalPropertyDamage"`.
- Severity of the incident is `"moderate-gen"`.

If any of the tests evaluate to `true`, the rule sets `exposure.Segment = "travel_mid"` and exits the rule set. If none of these conditions are `true`, the next rule executes.

ESG05300 - Travel High

The rule tests to see if any of the following is true:

- Exposure type is `"VehicleDamage"`, `"PropertyDamage"`, or `"PIPDamage"`.
- Severity of the incident is `"severe-gen"` or `"major-gen"`

If any of the tests evaluate to `true`, the rule sets `exposure.Segment = "travel_high"` and exits the rule set. If none of these conditions are `true`, the default rule executes.

ESG05400 - Travel Default

The rule sets the segment to `"travel_low"` and exits the rule set.

Handling a strategy separately from a segment

There are a number of reasons for defining the handling strategy for a claim or exposure separately from the segment:

- There can be many different categories of claims for which it is interesting to report statistics separately. However, there is usually a much smaller number of strategies (also known as handling approaches), such as fast track as opposed to investigate. The segment can capture the more detailed understanding of categorization and govern how ClaimCenter assigns the work, while the strategy can provide an assessment of how much effort is put into adjusting the claim.
- You can experiment with different handling strategies for the same segment of claim. For example, you decide to employ a new approach for a certain segment of claims in one office. You want to be able to analyze the results later to determine if average outcomes were better following the new strategy.

See also

- *Application Guide*

Setting the strategy property

Note: The strategy property can be set independently of any claim or exposure segment. In this example, the segment is used to set the strategy, which is a common approach, but, there are other ways that a strategy can be set.

You can set the strategy value after setting the segment by using the following property:

```
claim.Strategy = strategy_value
```

The `ClaimStrategy` typelist defines the valid strategy values. In the simplest form, the segmentation rules simply use the claim's segment and map a segment to a strategy. For example, the following rule could run after the segmentation is set for an Auto claim.

Note: For this rule to run after the two existing auto claim segmentation rules, you would have to change the two rules to not exit if each rule set completes.

The following code sample illustrates claim segmentation rule CSG01300 - Set Auto Claim Strategy.

```
CONDITION (claim : entity.Claim):
return true

ACTION (claim : entity.Claim, actions : gw.rules.Action):
if (claim.Segment == "auto_high" or claim.Segment == "auto_mid") {
  claim.Strategy = "auto_normal"
}
else {
  claim.Strategy = "auto_fast"
}
actions.exit()
```

Transaction Approval

Transaction approval rules ensure that a user has authorization to submit certain financial transactions. A transaction set contains one or more transactions that are submitted as a group for approval. If a user attempts to save a transaction set, ClaimCenter can ensure that the transaction be marked as requiring approval.

ClaimCenter calls this rule set to handle transaction approvals of all kinds, not just those that involve authority limits, such as reserves or payments. Guidewire recommends that you create rules in this rule set that are used only to determine whether a transaction requires approval or not.

The Approval Routing and Transaction Approval rule set categories form a pair. ClaimCenter calls these rule sets sequentially.

- ClaimCenter calls rules in the Transaction Approval rule set category whenever you submit any kind of financial transaction.
- ClaimCenter calls rules in the Approval Routing rule set category if the transaction approval rules mark the transaction as requiring approval. It also calls this rule set if the transaction authority limit fails, thus requiring that the transaction needs approval. See "Approval routing" on page 36.

You must ensure that if you approve the activity, you approve the transaction also. If a transaction requires approval, ClaimCenter uses the Approval Routing rules to determine who must give the approval.

In working with the Approval Routing rules:

- Use the `TransactionSet.requireApproval` method to require approval.
- Use the `TransactionSet.getRequestingUser` method to determine the person on whose behalf ClaimCenter runs the rules. For example, suppose that Andy Applegate initiates the reserve, and ClaimCenter routes the transaction to Sue Smith for approval. After Sue approves the transaction, she becomes the `requestingUser` for the next person in the approval chain.

IMPORTANT: Often, the Transaction Approval rules iterate through all the transactions on the `TransactionSet` entity. Do not use these rules to set a claim to a new value in the course of working with a transaction.

Transaction Approval rules

The Transaction Approval rules determine the approval requirements for financial transactions. ClaimCenter runs these rules for each transaction to determine if the transaction requires approval. In a typical `TransactionSet` approval rule, the rule condition verifies the subtype of the transaction set, such as `TransactionSet.subtype == "checkset"`. This subtype verification is required because all created transaction sets, regardless of subtype, are passed to the same approval rules.

To work with the Transaction Approval rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **TransactionApproval** > **TransactionApprovalRules**

In the base configuration, ClaimCenter provides the following Transaction Approval rules:

TAP01000 - PIP

This rule determines if approval is needed for a transaction with an exposure that has reached its limit for incident, exposure, or several measures of personal injury coverage (PIP).

The rule first verifies that the transaction set subtype is `checkset` and that there is at least one transaction in the transaction set with a cost type of `claimcost`. The rule then tests each transaction in the transaction set to see if the related exposure's authority limit has been reached for any of the following:

- Incident limit
- Exposure limit
- PIP non-medical aggregate limit
- PIP replacement services aggregate limit
- PIP per-person aggregate limit
- PIP claim aggregate limit

If one of these limits has been reached and the supervisor has not already approved payment for the transaction, an activity is created for that purpose.

TAP02000 - Sample - Manual Auth Limit Checking

This sample rule is disabled by default in the base configuration.

If enabled, the rule checks authority limits on `ReserveSet` and `CheckSet` objects, excluding bulk invoice checks. It provides an alternative to automatic authority limit checking, which does check bulk invoice checks.

> **Note:** This rule is appropriate only if the system configuration parameter `CheckAuthorityLimits` is `false`.

TAP03000 - Sample - Custom Calculation

This sample rule is disabled by default in the base configuration. The rule uses a customized total incurred value. For more information on Total Incurred Value, see the class `util.financials.CustomCalcs`.

TAP04000 - Claim and Exposures Should Be At Ability to Pay

This rule detects claims and exposures associated with the transaction set that are not at a specified validation level. The transaction set must be a check set. If the associated claim or exposure is not at Ability to Pay, the check set is sent for approval.

TAP05000 - Optionally Check for Duplicate Checks

This rule can look for duplicate checks that might exist if a check is created programmatically, through the web service `ClaimFinancialsAPI.creatCheckSet`.

> **Note:** If a check is created in the ClaimCenter user interface, the check for duplicates has already happened and has been handled there. In this case, `CheckSetRequiresApprovalForDuplicateChecks` is set to `false`.

The rule first determines if the transaction set is a check set. If it is and `CheckSetRequiresApprovalForDuplicateChecks` set to `true`, the rule looks for duplicate checks in the transaction set. If the rule finds a duplicate check, it sends the check set for approval.

Validation

ClaimCenter uses validation levels to ensure that certain conditions are met before it is possible to move to a critical stage in the processing workflow of claim objects. These validation levels apply, for example, as you create a new claim or make a payment. You can view these stages as maturity levels—as a claim gains data, it achieves new levels of maturity. A ClaimCenter object can always achieve a new and higher level of maturity, but it can never go backwards to a lower validation level.

A validation rule sets the maturity level to which the rule applies. ClaimCenter then enforces rule errors only if set at or below the level the object has achieved.

To work with validation Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation**

Notes:

- ClaimCenter runs the validation rules at many different points in the ClaimCenter workflow. ClaimCenter also runs these rules whenever it commits a claim or other validatable business entity to the database.
- ClaimCenter does not run validation rules on claims that are in the in-process Draft state in the New Claim wizard.

See also

- “Rules-based validation” on page 159
- “ClaimCenter Validation and Preupdate rules” on page 101

About ClaimCenter validation levels

In the base configuration, ClaimCenter has five levels of validation. Some of these validation levels are internal to ClaimCenter, and you cannot change them. Others are editable, and you can access them through Guidewire Studio by using the `ValidationLevel` typelist.

Validation Priority Levels

ClaimCenter uses the `priority` attribute of the typekeys to order the validation levels. The lower the priority number, the higher the validation level. Thus, in the base configuration, `newloss` with a priority of 300 is more restrictive than `loadsave` with a priority of 400. The following list describes the base configuration validation levels, listed in increasing order of maturity.

Level	Name	Priority
loadsave	Load and save	400
newloss	New loss completion	300
ISO	Valid for ISO	250
external	Send to external system	200
payment	Ability to pay	100

System Validation Levels

In the base configuration, ClaimCenter defines three validation levels that you cannot delete, although you can override their properties. ClaimCenter requires that these validation levels exist for internal code use. In the `ValidationLevel` Typelist editor, you see the following three validation levels in gray text, indicating that they are not editable.

Level	Priority	Description
loadsave	400	The loosest possible validation level against which ClaimCenter validates claims. All data must pass `loadsave` to be saved to the database.
newloss	300	The lowest level at which a claim can exit the New Claim wizard.
payment	100	The lowest level at which it is possible to pay a claim or an exposure. For example, in the base configuration, ClaimCenter does not permit you to enter the New Check wizard if the claim is not in Ability to Pay.

Non-system Validation Levels

ClaimCenter provides other validation levels in the base configuration, listed in the table that follows. Because Guidewire defines these levels in the `ValidationLevel.ttx` typelist, it is possible to modify, remove, or even add to them by using the Studio Typelist editor.

Level	Priority	Description
iso	250	The level at which the claim passes ISO validation.
external	200	The level at which a claim can be submitted to an external system. Validation occurs whenever an external submission or decline is triggered by a workflow button.

Note: It is possible to add additional validation levels. See “Adding new validation levels” on page 100 for details.

About validation warnings and errors

It is possible for ClaimCenter to generate validation warnings that provide information only. To generate this type of warning, the issue must not affect the claim or exposure, but it can affect other business practices. For example, if an automobile claim is missing weather information, a warning can remind the user to enter the information. The following code sample, added to the Claim Validation Rules rule set, illustrates this concept.

```
CONDITION (claim : entity.Claim):
return claim.LossType=="auto" and claim.Weather==null

ACTION (claim : entity.Claim, actions : gw.rules.Action):
```

```
claim.rejectField("Weather", null, null, "newloss",
      "Remember to enter weather information for an auto claim!")
```

It is also possible for ClaimCenter to generate validation errors that prevent the claim or exposure from being saved in ClaimCenter. For example, it is not possible for a user to save a claim in which the loss date is in the future. The following code sample is from the Claim Validation Rule CLV03000 - Future loss date.

```
CONDITION (claim : entity.Claim):
return Claim.LossDate > gw.api.util.DateUtil.currentDate()

ACTION (claim : entity.Claim, actions : gw.rules.Action):
Claim.rejectField("LossDate", "newloss",  displaykey.Rules.Validation.Claim.ProvideLossDateNotInTheFuture,
      null, null)
```

In this example, ClaimCenter marks the `Claim.LossDate` field as invalid and prompts the user to fix it.

See also

- For a discussion of the `rejectField` method, see “Validation error reject methods” on page 102.

Adding new validation levels

It is possible to add additional validation levels to meet your business needs.

System Validation Levels

You can override the name and the priority of the three system validations levels `loadsave`, `newloss`, and `payment`. For example, you can change the name of the `payment` validation level from *Ability to pay* to *Ability to pay expenses*. You can also modify the priority so this level is easier to reach. You cannot, however, delete the three system validation levels.

Non-system Validation Levels

You can modify or delete any non-system validation level.

Custom Validation Levels

You can add new validation levels. If you do so, then you must provide a linkage to the actions you would like to control through the new validation level and to the validation level itself. You do this by calling the following method on a `Claim` or `Exposure` object or any other validatable object:

```
isValid(validationLevel, includeExposures)
```

For example:

```
Claim.isValid("newLoss", false)
```

The `isValid` method determines if the claim or exposure is valid for the action you are attempting to perform. Calling this method is required for checking actions triggered in preupdate rules or from within ClaimCenter. It is important to understand that ClaimCenter runs the Validation rules after the preupdate rules, so the `Claim.ValidationLevel` property reflects the previous validation level. However, this is not true for rules in the Event Message rule set category, which ClaimCenter runs after validation. See “Event message” on page 51.

How ClaimCenter triggers validation rules

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

ClaimCenter runs the validation (and preupdate) 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.

See Also

- For information on preupdate rules and custom entities, see “How ClaimCenter triggers Preupdate rules on custom entities” on page 77.
- For information on how to create an entity that implements the `Validatable` delegate, see the *Configuration Guide*.

How ClaimCenter triggers Validation rules on custom entities

It is possible to create validation rules for custom entities, entities that you create yourself and that are not part of the base Guidewire ClaimCenter configuration.

For an extension entity to trigger a validation rule, all of the following must be true:

1. The entity must be a root entity, not a subtype of an entity.
2. The entity must implement the `Validatable` delegate.

   For any extension entity that you create, if you want it to trigger a validation rule, the extension entity must implement the following code:

   ```
   <implementsEntity name="Validatable"/>
   ```
3. A rule set must exist in the Validation rule category that conforms to the following naming convention:

   `custom_entity_name``ValidationRules`

   For example, if you create an extension entity named `NewEntityExt`, create a rule set in the Validation rule category to hold your validation rules and name it `NewEntityExtValidationRules`.

See also

- For information on creating new Gosu rules sets, see “Working with rules” on page 23.
- For information on how to create an entity that implements the `Validatable` delegate, see the *Configuration Guide*.

ClaimCenter Validation and Preupdate rules

With the exception of in-process claims in the New Claim wizard, ClaimCenter runs preupdate and validation rules every time that it commits data to the database. Guidewire calls the process of committing data to the database a *database bundle commit*.

Validation rules execute after ClaimCenter runs all preupdate rules and internal preupdate callbacks. Execution of the validation rules is followed by event fired rules before ClaimCenter writes data to the database.

> **Note:** Claims being processed in the New Claim wizard are saved in Draft state until the user completes the wizard. Whenever ClaimCenter saves the claim in Draft state, it runs only the preupdate rules. Whenever the user finishes the wizard, ClaimCenter runs both the preupdate and validation rules on the claim as described in this topic. In fact, at the end of the New Claim wizard, it explicitly validates the claim and any existing exposures. This results in claim validation rules always running twice at the end of the New Claim wizard. It may also result in exposure validation rules running once or twice, if exposures were added or modified before the wizard is complete.

It is possible for the preupdate rules to modify objects during execution, requiring changes to the database. However, ClaimCenter does not permit you to modify objects during validation rule execution that require changes to the database. To allow this would make it impossible to ever completely validate data—data validation would be an ever-shifting target.

See Also

- For more information on how the Gosu preupdate and validation rules work together, see “Rules-based validation” on page 159.
- For information on how to create custom entities that trigger validation (and preupdate) rules, see “How ClaimCenter triggers Preupdate rules on custom entities” on page 77.

Determining the validity of an object

ClaimCenter provides a `validate` method that you can call on a number of different business objects to determine if that object is valid. For example, the `validate` method on `Claim` executes the Claim Validation Rules on the supplied claim. The method then returns a `ValidationResult` object containing any errors. If the validation was successful, the method returns an empty validation object. Also, if `validateExposures` is `true`, then the method validates the claim and all the exposures on the claim. Otherwise, the method validates the claim only.

```
claim.validate(validateExposures)
```

This method works on a single object at a time and returns a validation object that you must then handle. For this reason, Guidewire does not recommend that you use this method for bulk operations. For example, if you want to process a large number of converted claims by escalating them to whatever level they can achieve, use the `ClaimAPI` method, `startClaimValidation`, as follows:

```
ClaimAPI.startClaimValidation(loadCommandPublicID)
```

The startClaimValidation method does not perform any validation. Instead, it launches a batch process that creates a separate work item to validate each claim. The batch process runs unattended and does not return a result. The `loadCommandPublicID` parameter is a `String` value that identifies the conversion batch process that imported the claims. The `loadCommandPublicID` value is available through the `TableImportResult` object returned from a table import operation. For example:

```
TableImportAPI.integrityCheckStagingTableContentsAndLoadSourceTables()
```

To determine if a claim and its exposures are valid, call the following method on a `Claim` or `Exposure` object:

```
claim.isValid(validationLevel, includeExposures)
exposure.isValid(validationLevel)
```

See also

- "Claim Validation rules" on page 106

Validation error reject methods

You can use a form of the `reject` method to prevent continued processing of the object and to inform the user of the problem and how to correct it. The `reject` methods are available on all validatable entities. In the base configuration, these entities are:

```
Activity
Claim
Contact
Exposure
Matter
Person
Policy
RIAgreement
RITransactionSet
ServiceRequest
TransactionSet
User
```

The following table describes the various forms of the `reject` method.

Method form	Description
`reject`	Indicates a problem and provides an error message, but does not point the user to a specific field.

Method form	Description
`rejectField`	Indicates a problem with a particular field or field path, provides an error message, and directs the user to the correct field to fix.
`rejectSubField`	Similar to `rejectField`, but also indicates that the problem is with a field on a subobject.

The method signature can take one of the following forms:

```
reject( errorLevel, strErrorReason, warningLevel, strWarningReason )
rejectField( strRelativeFieldPath, errorLevel, strErrorReason, warningLevel, strWarningReason )
rejectSubField( relatedBean, strRelativeFieldPath, errorLevel, strErrorReason, warningLevel,
      strWarningReason )
```

IMPORTANT: Guidewire ClaimCenter currently supports the listed `reject` methods only, even though additional methods can show in Studio. Specifically, ClaimCenter does not support `reject` methods that take a `flowtStepId` argument. ClaimCenter ignores this argument if you attempt to set it.

The following table lists the `reject` method parameters and describes their use. Identified issues or problems are either errors (blocking) or warnings (non-blocking). Note that you can indicate failure as both an error and a warning simultaneously. However, if the failure is both an error and a warning, use different error and warning levels for the failure.

Method parameter	Description
`errorLevel`	Corresponding level to be set as a result of the validation error.
`relatedBean`	Object related to the root entity that is causing the error.
`strErrorReason`	Message to show to the user indicating the reason for the error.
`strRelativeFieldPath`	Relative path from the root entity to the field that failed validation. Using a relative path (as opposed to a full path) sets a link to the problem field from within the message shown to the user. It also identifies the invalid field on-screen.
`strWarningReason`	Message to show to the user indicating the reason for the warning.
`warningLevel`	Corresponding level effected by the validation warning.

Validation Rules and Entity Types

You define a validation rule in a rule set that is declared for a specific entity type. For example, the rule set would be for `Claim`, `User`, or one of the other business entities. A rule declares its primary object in the rule `CONDITION` statement.

IMPORTANT: Although it is possible to access business entities other than the declared type within a rule, call the `reject` method only on the declared entity in the rule set. Studio does not detect violations of this guideline. If you do call `reject` on an entity that is not the declared type, ClaimCenter can lose this reject call. This loss can result from ClaimCenter traversing the set of validatable objects that resets the previous result for the entity.

About Error Text Strings

Guidewire recommends that you use one of the following ways to insert an error text string into a `reject`:

- Method `getMessage`
- Display keys

Method getMessage

The `getMessage` method returns an instance of `UserMessage`, which represents all the information needed to render a user message. Guidewire recommends that you also call the `localize` method to handle any localization issues. The following `ClaimContact` Role Configuration example for exposure validation checks roles at the exposure level and generates an error message for each invalid role.

```
/* If anything relevant has changed, check for invalid role
   configurations at the exposure level and report each individually
   on the Contacts array if they exist
*/
var b = \ bean : KeyableBean -> typeof bean == entity.ClaimContact ||
    typeof bean == entity.ClaimContactRole
if (!exposure.isValid("newloss")
    || exposure.Bundle.InsertedBeans.toList().hasMatch(b)
    || exposure.Bundle.UpdatedBeans.toList().hasMatch(b)
    || exposure.Bundle.RemovedBeans.toList().hasMatch(b)) {
  var failures = exposure.validateRoles()
  for (failure in failures) {
    exposure.reject("newloss", failure.getMessage().localize(), null, null)
  }
}
```

Display Keys

You can also use display keys to return a value as the error text string. A display key provides globalization support. For example, in the `display.properties` file for `U.S English`, the following display key is defined as User has duplicate roles:

`displaykey.Java.Admin.User.DuplicateRoleError`

You can also use display keys that require a parameter or parameters. In the following example, the `display.properties` file defines the following display key with placeholder `{0}`:

```
Java.UserDetail.Delete.IsSupervisorError = Cannot delete user because they are supervisor
      of the following groups\: {0}
```

Suppose then, that you use the following display key code for a rule validating Groups. In this case, ClaimCenter has already retrieved the value of `GroupName`.

`displaykey.Java.UserDetail.Delete.IsSupervisorError( GroupName )`

This display key returns the following from the `display.properties` file:

`Cannot delete user because they are supervisor of the following groups: AutoClaim1`

Activity Validation rules

Use the Gosu Activity Validation rules to provide validation for `Activity` objects. To work with `Activity` Validation rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **ActivityValidationRules**

In the base configuration, ClaimCenter provides a single rule in this rule set.

ACV01000 - WC Compensability

This rule tests for an activity of type Claim Acceptance with a `Status` of Complete and an associated workers' comp claim with a `Compensable` status of either `null` or Pending. For any matching activity, the rule rejects the activity, but does not close it, and displays the following message:

```
Prior to completing the activity, {0}, please change the Compensability status to either
Accepted or Denied
```

In the base configuration, ClaimCenter defines this message in file `display.properties` as `displaykey.Rules.Validation.Activity.Compensable.Status(activity.Subject)`

Claim Closed Validation rules

The Claim Closed Validation rules execute whenever ClaimCenter closes a claim. These rules execute:

- Before ClaimCenter commits the data to the database.
- Before ClaimCenter executes the Claim Closed rule set.
- Before ClaimCenter executes the Claim Preupdate rule set.
- Before ClaimCenter executes the Claim Validation rule set.

You can use these rules to display validation errors or warnings if ClaimCenter must not close a claim for some reason. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **ClaimClosedValidationRules**

In the base configuration, ClaimCenter provides the following rules in this rule set.

CCV01000 - Open exposures

If the claim being closed has any open exposures, this rule prevents the claim from being closed. Open exposures must be closed before closing the claim.

CCV02000 - Open activities

If the claim being closed has open activities that are not allowed on a closed claim, this rule prevents the claim from being closed. All the open activities must first be completed or skipped before the claim can be closed. The rule does not prevent the claim from being closed for activities that are allowed on closed claim.

CCV03000 - Property checks

This rule checks if the claim being closed has a loss type of PR, but the rule has no actions. Its purpose is to filter for property claims in case you want to add child rules that perform claim closed validation on property claims.

CCV04000 - At Fault must be determined for non-WC

In the base configuration, the claims that can require a determination of which party is at fault are claims with a loss type of AUTO, PR, and GL. For example, these claims can be on policies for personal auto, commercial auto, general liability, other liability, businessowners, personal umbrella, commercial property, Homeowners, and inland marine.

This rule first checks if the claim being closed has one of these three loss types—AUTO, PR, or GL—and `FaultRating` is not set but payments have been made. If the claim matches these criteria, this rule prevents the claim from being closed. The fault rating must be determined before closing the claim.

CCV05000 - Subrogation Status must not be Review

This rule first checks if the claim being closed is being reviewed for subrogation—`claim.SubrogationStatus == "review"`. If so, the rule prevents the claim from being closed. The subrogation must be set to Open or Closed before the claim can be closed.

CCV06000 - if Third party Liability indicated, then

This rule first checks if the claim being closed has third party liability and has not been reviewed for subrogation. If so, the rule prevents the claim from being closed. The claim's third party liability requires that subrogation be closed before the claim can be closed.

CCV07000 - Open matters

If the claim being closed has any open matters, this rule prevents the claim from being closed. All open matters must first be closed before the claim can be closed.

CCV08000 - Open Service Requests

If the claim being closed has any active service requests that have not been completed, this rule prevents the claim from being closed. All open service requests must be closed before the claim can be closed.

Claim Reopened Validation rules

The Claim Reopened Validation rules execute whenever ClaimCenter reopens a claim. In particular, they execute:

- After the Claim Validation rules execute.
- Before ClaimCenter commits the data to the database.
- Before ClaimCenter inserts a new entity or updated entity into the database.

- Before ClaimCenter executes the Claim Reopened rules.

The Claim Reopened Validation rules perform their actions only if the entity is valid.

You can use these rules to raise validation errors or warnings if ClaimCenter cannot reopen a claim for some reason. To work with these Gosu rules, navigate to the following location in the Studio **Project** window

configuration > **config** > **Rule Sets** > **Validation** > **ClaimReopenedValidationRules**

This rule set is empty in the base ClaimCenter configuration.

Claim Validation rules

Use the Claim Validation rules to ensure that claim data stored in ClaimCenter is sufficient and valid for downstream processing. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **ClaimValidationRules**

In the base configuration, ClaimCenter provides the following rules in this rule set.

CLV01000 - Not set - Coverage in Question

Guidewire disables this rule in the base configuration. If the claim has a null `CoverageInQuestion` property, the rule prevents the claim from being used until the property is set.

CLV02000 - Coverage indicated but no details provided

This rule is disabled in the base configuration because exposures are validated in a dedicated exposure ruleset. The rule serves only as an example.

The rule tests each exposure of the claim to determine if the `OtherCoverage` property is set but the detail field for other coverage is empty. For every exposure that matches these criteria, the rule calls `claim.rejectSubField`. This method call sets the claim to New Loss and displays a message saying to indicate the details of the other coverage.

CLV03000 - Future loss date

If the claim has a loss date that is later than today’s date and time, the rule calls `claim.rejectSubField`. This method call sets the claim to New Loss and displays a message saying to change the date to one that is not in the future.

CLV04000 - `ClaimContact` Role Configuration

This rule determines if anything that is relevant to contact roles has changed for the claim. If so, the rule checks for invalid role configurations for the claim and reports each one it finds individually on the `Contacts` array.

CLV05000 - Policy expiration date after effective date

If the policy for the claim expired before the policy was in effect, this rule prevents the claim from being saved. The policy must be in effect before it expires for a claim to be filed on it.

CLV06000 - Policy original effective date

If the policy for the claim has an original effective date that is later than the current effective date, `claim.Policy.EffectiveDate`, this rule prevents the claim from being saved. The original effective date of the policy must not be later than the policy effective date.

CLV07000 - Liability can not exceed 100

This rule first determines if the claim has gone to subrogation and that there are adverse parties involved. It prevents the claim from proceeding if either of the following is the case:

- Total liability for adverse parties exceeds 100%.
- Total liability for the insured and the adverse parties together exceeds 100%.

CLV08000 - Expected Recovery exceeds 100

This rule first determines if the claim has gone to subrogation and that there are adverse parties involved. It prevents the claim from proceeding if both the following are the case:

- There was a change to the expected recovery amount.
- The expected recovery from the adverse parties exceeds 100%.

CLV09000 - ISO Validation

This rule tests if the current claim has ISO enabled. If so, it passes the claim to its child rules.

CLV09100 - Not draft

This rule checks the claim to see if it is still in Draft state. If so, the rule prevents the claim from going to ISO. The state must be at least ISO for ISO to be enabled.

CLV09200 - Policy

If there is no policy related to the claim, this rule prevents the claim from going to ISO. There must be an associated policy for ISO to be enabled.

CLV09300 - LossLocation

If there is no loss location for the claim, this rule prevents the claim from going to ISO. There must be a loss location for ISO to be enabled.

CLV09400 - PolicyNumber

If there is no policy number for the policy that is related to the claim, this rule prevents the claim from going to ISO. There must be a policy number for the associated policy for ISO to be enabled.

CLV09500 - PolicyType

If there is no policy type for the policy that is related to the claim, this rule prevents the claim from going to ISO. There must be a policy type for the associated policy for ISO to be enabled.

CLV09600 - ClaimNumber

If there is no claim number for the claim, this rule prevents the claim from going to ISO. There must be a claim number on the claim for ISO to be enabled.

CLV09700 - LossDate

If there is no loss date for the claim, this rule prevents the claim from going to ISO. There must be a loss date for the claim for ISO to be enabled.

CLV09800 - State

This rule tests if the current claim is missing a `State` entry for the loss location. If the state is missing and the country US or CA (Canada) or there is no country, the rule prevents the claim from going to ISO.

CLV09900 - Insured

This rule groups three child rules that perform validations on values for the insured party.

CLV09910 - Insured exists

If there is no insured party for the claim, this rule prevents the claim from going to ISO. There must be an insured party for the claim for ISO to be enabled.

CLV09920 - Name

This rule performs a series of tests on the name of the insured party. If the insured party is a person, the person must have a first and last name. If the insured party is a company, the company must have a name. If any of these tests fails, the rule prevents the claim from going to ISO. The appropriate name fields must be filled in for ISO to be enabled.

CLV09930 - Address

This rule performs a series of tests on the address of the insured party. If the address is missing the first line of the address or the city, the rule prevents the claim from going to ISO. Additionally, if the country is US, CA (Canada), or null (no country), the `State` field must be filled in. All these fields must be filled in for ISO to be enabled.

CLV091000 - Workers Comp

This rule filters for a workers' comp claim. Its child rules apply various tests for this type of claim.

CLV091010 - Claimant Exists

This rule checks to see if the claimant is missing. If there is no claimant, the rule prevents the claim from going to ISO. There must be a claimant for a workers' comp claim for ISO to be enabled.

CLV091020 - Name

If there is a claimant, the rule checks if the person has a first and last name. If either is missing, the rule prevents the claim from going to ISO. The claimant for a workers' comp claim must have a first and last name for ISO to be enabled.

CLV091030 - Address

If there is a claimant, this rule performs a series of tests on the address of the claimant. If the address is missing the first line of the address or the city, the rule prevents the claim from going to ISO. Additionally, if the country is US, CA (Canada), or null (no country), the `State` field must be filled in. All these fields must be filled in for ISO to be enabled.

CLV091030 - Injury Description

If the injury description is missing, the rule rejects the claim. A workers' comp claim must have an injury description for ISO to be enabled.

CLV091030 - Body Part

If either there are no body parts described or if the first body part's details are missing, the rule prevents the claim from going to ISO. A workers' comp claim must have at least one body part indicated for ISO to be enabled.

CLV091100 - Description

This rule evaluates if the claim is eligible to send search messages to the ISO system. If so and there is no content for the `Description` property, the rule prevents the claim from going to ISO. If it is to be searchable, a workers' comp claim must have a description for ISO to be enabled.

CLV10000 - Catastrophe

This rule determines if the claim is for a catastrophe. If so, the following child rules execute.

CLV10100 - Check Perils

This rule determines if the claim has a catastrophe peril. If it does, the claim's loss cause and loss type must match the peril's loss cause and loss type for the claim to be categorized as a catastrophe.

CLV10200 - Check Loss Date and Zone

This rule determines if:

- The claim's loss date is within the catastrophe's date range.
- The claim's location matches the zone type for which the catastrophe is valid.

Both conditions must be true for the claim to be categorized as a catastrophe.

CLV11000 - Subrogation Status And Liability

For a claim that is in subrogation, the total liability percentage for insured and adverse parties must equal 100%. This rule determines the actual percentage for this type of claim and rejects it if the percentage is not 100%.

CLV12000 - Injury diagnosis validity dates

This rule determines if the claim has an injury diagnosis. If so the child rules can run.

CLV120100 - WC

This rule determines if the start date and end date of each injury diagnosis for a workers' comp claim are within the valid dates for the injury code. If not, the claim is rejected.

CLV120200 - Default

This rule determines if the start date and end date of each injury diagnosis for a workers' comp claim are within the valid dates for the injury code. If not, the claim is rejected.

CLV13000 - Full Denial Reasons

This rule enforces a limit of five denial reasons for a workers' comp claim.

CLV14000 - Subrogation role changes

If this claim has a subrogation, the following child rules execute.

CLV14100 - Subrogation owner role removed

This rule prevents the user from directly removing the subrogation owner role from a user on the claim. Instead, the user must reassign the subrogation.

CLV14200 - Responsible party role removed

This rule prevents the user from directly removing the responsible party role from a claim contact. Instead, the user must remove the contact from the responsible parties list on the subrogation.

CLV15000 - Applicable Deductible CovTerm Changed

This rule checks if the claim loss cause has changed, which results in a change to the deductible coverage term. It prevents the claim from validating at load and save if a deductible has not been paid with the message:

```
The applicable deductible was updated as the loss cause has changed. You may need to
inform insured about the changed deductible amount.
```

If a deductible has been paid, it prevents the claim from validating at new loss completion with the message:

```
Changing the loss cause may change the applicable deductible. Review the deductible
amount on the exposure and any checks where deductible has been applied.
```

Contact Validation rules

Use the Contact Validation rules to ensure that contacts data stored is sufficient and valid for downstream processing. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **ContactValidationRules**

This rule set contains the following rules. In the base configuration, Guidewire disables all the rules in this rule set.

Home phone country code must be valid

Guidewire disables this rule in the base configuration. This rule checks to see if a home phone number is defined for the contact. If there is one, the rule verifies that the country code is valid. If it is not, the contact is rejected.

Work phone country code must be valid

Guidewire disables this rule in the base configuration. This rule checks if a work phone number is defined for the contact. If there is one, the rule verifies that the country code is valid. If it is not, the contact is rejected.

Fax phone country code must be valid

Guidewire disables this rule in the base configuration. This rule checks if a fax phone number is defined for the contact. If there is one, the rule verifies that the country code is valid. If it is not, the contact is rejected.

Exposure Closed Validation rules

The Exposure Closed Validation rules execute whenever ClaimCenter closes an exposure. These rules execute:

- Before ClaimCenter commits the data to the database.
- Before ClaimCenter executes the Exposure Closed rule set.
- Before ClaimCenter executes the Exposure Preupdate rule set.
- Before ClaimCenter executes the Exposure Validation rule set.

You can use these rules to display validation errors or warnings if ClaimCenter must not close an exposure for some reason. To work with these Gosu rules, navigate to the following location in the Studio **Project** window

configuration > **config** > **Rule Sets** > **Validation** > **ExposureClosedValidationRules**

In the base configuration, ClaimCenter provides the following rules in this rule set.

ECV01000 - Open activities

If the exposure being closed has any open activities that are not permitted in closed claims, this rule prevents the exposure from being closed. Open activities not permitted in closed claims must be closed before closing the exposure.

ECV02000 - Injury-specific checks

This rule determines if the exposure being closed is for bodily injury. It is an example of a parent rule that could filter exposures for a set of child rules.

ECV03000 - Vehicle damage-specific checks

This rule determines if the exposure being closed is for vehicle damage. It is an example of a parent rule that could filter exposures for a set of child rules.

ECV04000 - Property-specific checks

This rule determines if the exposure being closed is for a property loss. It is an example of a parent rule that could filter exposures for a set of child rules.

ECV05000 - Content-specific checks

This rule determines if the exposure being closed is related to contents. It is an example of a parent rule that could filter exposures for a set of child rules.

ECV06000 - Loss of Use-specific checks

This rule determines if the exposure being closed is for loss of use. It is an example of a parent rule that could filter exposures for a set of child rules.

ECV07000 - Stop Closing Of Exposure With Unpaid Deductible

This rule determines if If this exposure's coverage has an unpaid deductible. If the exposure has at least one payment, it cannot be closed until the deductible is waived.

Exposure Reopened Validation rules

The Exposure Reopened Validation rules execute whenever a user reopens an exposure. The rules execute:

- Before ClaimCenter commits data to the database
- Before the Exposure Reopened rule set executes
- During the standard operation of the Preupdate and Validation rule sets.

Use these rules to raise validation errors or warnings if there is some reason to not reopen the exposure.

To work with these Gosu rules, navigate in the following location in the Studio **Project** window

configuration > **config** > **Rule Sets** > **Validation** > **ExposureReopenedValidationRules**

In the base configuration, ClaimCenter provides the following rule in this rule set.

ERV01000 - Claim closed

If the exposure being reopened is on a claim that is not open, this rule rejects the reopen action. An exposure cannot be reopened unless the claim is open.

Exposure Validation rules

Use the Exposure Validation rules to ensure that exposure data stored in ClaimCenter is sufficient and valid for downstream processing.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **ExposureValidationRules**

In the base configuration, ClaimCenter provides the following rules in this rule set.

EXV01000 - Incident not null

This rule checks to see if an incident is not associated with the exposure and rejects the exposure if that is the case. An incident must be associated with an exposure for the exposure to be valid.

EXV02000 - Other coverage but no info

Guidewire disables this rule in the base configuration. The rule causes ClaimCenter to enforce that exposures fill in Other Coverage data during validation.

EXV03000 - ClaimContact Role Configuration

This rule determines if anything that is relevant to contact roles has changed for the exposure. If so, the rule checks for invalid role configurations for the exposure and reports each one it finds individually on the `Contacts` array.

EXV04000 - Coverage not null

Guidewire disables this rule in the base configuration. The restrict prevents the writing of checks for Workers' Comp exposures that have no appropriate coverage in the policy.

EXV05000 - ISO Validation

This rule tests if the current exposure has ISO enabled. If so, it passes the exposure to its child rules.

EXV05100 - Not draft

This rule checks the exposure to see if it is still in Draft state. If so, the rule prevents the exposure from going to ISO. The state must be at least ISO for ISO to be enabled.

EXV05200 - Primary Coverage

If there is no primary coverage for the exposure, this rule prevents the exposure from going to ISO. There must be exposure coverage for ISO to be enabled.

EXV05300 - Coverage Subtype

If there is no coverage subtype for the exposure, this rule prevents the exposure from going to ISO. There must be a coverage subtype for ISO to be enabled.

EXV05400 - Description

This rule determines first if the claim for the exposure has a loss type that is not workers' comp. If so, the rule then tests to see if there is a description of the incident. If there is no incident description, the rule prevents the exposure from going to ISO.

The rule tests for the following specific incident descriptions:

- Vehicle incident
- Fixed property incident
- Incident

EXV05500 - Non Workers Comp

This rule determines if the claim for the exposure has a loss type that is not workers' comp. If so, the following child rules execute.

EXV05510 - Claimant Exists

This rule tests to see if there is a claimant for the exposure. If there is no claimant, the rule prevents the exposure from going to ISO.

EXV05520 - Name

If there is a claimant, the rule checks if the person has a first and last name. If either is missing, the rule prevents the exposure from going to ISO. The claimant must have a first and last name for ISO to be enabled.

EXV05530 - Address

If there is a claimant, this rule performs a series of tests on the address of the claimant. If the address is missing the first line of the address or the city, the rule prevents the exposure from going to ISO. Additionally, if the country is US, CA (Canada), or null (no country), the `State` field must be filled in. All these fields must be filled in for ISO to be enabled.

EXV05600 - Bodily Injury Exposure

This rule determines if the exposure is of type bodily injury damage. If so, the following child rule executes.

EXV05610 - Injury

If there are no body parts described or if the first body part's details are missing, the rule prevents the exposure from going to ISO. A bodily injury exposure must have at least one body part indicated for ISO to be enabled.

EXV05700 - Vehicle Exposure

This rule determines if the exposure is of type vehicle damage. If so, the following child rules execute.

EXV05710 - Vehicle

This rule checks to see if any of the following vehicle incident fields are missing:

- VehicleIncident.Vehicle
- VehicleIncident.Vehicle.Year
- VehicleIncident.Vehicle.Make and VehicleIncident.Vehicle.Manufacturer

If so, the rule prevents the exposure from going to ISO. A vehicle damage exposure must have at least the year and make specified for ISO to be enabled.

EXV05720 - Normal Vehicle

This rule tests the vehicle incident to see if the vehicle is *normal*: not an ATV, snowmobile, or boat. If so, the child rules execute.

EXV05721 - Vin Present

This rule tests the normal vehicle indicated for the incident to see if the vehicle identification number (VIN) exists. The loss party must not be a third party and the vehicle year must be after 1981 for this rule to apply. If there is no VIN number, the rule prevents the exposure from going to ISO.

EXV05722 - Vehicle Identification Number

This rule tests the normal vehicle indicated for the incident to verify that all the following conditions are true:

- The year is defined.
- The year is one that requires a VIN.
- The vehicle identification number is defined.
- The VIN has a valid format.

The rule uses constants defined in `gw.api.iso.ISOValidationConstants` to do these validations. If any of these tests fail, the rule prevents the exposure from going to ISO.

EXV05730 - Off-road Vehicle

This rule tests the vehicle incident to see if the vehicle is an ATV or snowmobile. If so, the child rules execute.

EXV05731 - Serial Number Present

This rule tests the off-road vehicle indicated for the incident to see if the vehicle serial number exists. If there is no serial number, the rule prevents the exposure from going to ISO.

EXV05732 - Off Road Type Present

This rule checks to see if the `OffRoadStyle` has been set for the vehicle associated with the incident. This value must be set for an ATV or snowmobile before the incident can go to ISO.

EXV05740 - Boat

This rule tests the vehicle incident to see if the vehicle is a boat. If so, the child rules execute.

EXV05741 - HIN Present

This rule tests the boat object indicated for the incident to verify that all the following conditions are true:

- The loss party is not a third party.
- The year is defined.
- The year is one that requires a hull identification number (HIN).
- The hull identification number is defined.

The rule uses constants defined in `gw.api.iso.ISOValidationConstants` to do these validations. If any of these tests fail, the rule prevents the exposure from going to ISO.

EXV05742 - HIN Format Correct

This rule tests the boat object indicated for the incident to verify that all the following conditions are true:

- The year is defined.
- The year is one that requires a hull identification number (HIN).
- The hull identification number is defined.
- The HIN has a valid format.

The rule uses constants defined in `gw.api.iso.ISOValidationConstants` to do these validations. If any of these tests fail, the rule prevents the exposure from going to ISO.

EXV05743 - Boat Type Present

This rule tests the vehicle incident to see if the type of boat is defined. If not, the rule prevents the exposure from going to ISO.

EXV05800 - LossOfUseDamage Exposure

This rule checks the exposure to see if the loss party or claimant is the same as the insured party. If not, the rule prevents the exposure from going to ISO.

EXV06000 - Settlement Date is Valid

This rule checks a workers' comp exposure to see if the settlement date is later than the current date—is in the future. If so, the rule prevents the exposure from going to ISO.

EXV07000 - Salvage

This rule checks the exposure to see if there was an amount recovered for salvage. If this amount is negative or zero, the rule rejects the exposure. A salvage recovery cannot be negative or zero, but it can be created as an ordinary payment instead.

EXV08000 - Incident changing

This rule prevents changing an exposure's incident if the exposure is related to one or more ClaimCenter services. This rule prevents the exposure from going to new loss completion.

Group Validation rules

ClaimCenter runs the Group Validation and Group Preupdate rules whenever it saves the `Group` entity. Any exceptions found during validation cause the bounding database transaction to roll back, thus effectively vetoing the update. In the base configuration, this rule set is empty.

Guidewire recommends that you use the LoadSave validation level for rules pertaining to validation for Users and Groups.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **GroupValidationRules**

Matter Closed Validation rules

The Matter Closed Validation rules execute whenever ClaimCenter closes a matter. These rules execute:

- Before ClaimCenter commits the data to the database.
- Before ClaimCenter executes the Matter Closed rule set.
- Before ClaimCenter executes the Matter Preupdate rule set.
- Before ClaimCenter executes the Matter Validation rule set.

Use these rules to raise validation errors or warnings if ClaimCenter must not close a matter for some reason.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **MatterClosedValidationRules**

In the base configuration, ClaimCenter provides the following Matter Closed Validation rule:

MCV01000 - Close Before Trial

This rule is provided as a test of validation warnings in Closed validation rules. The rule determines if the trial date is in the future. If so, the matter cannot be closed because the trial has not yet occurred.

Matter Reopened Validation rules

The Matter Reopened Validation rules execute whenever a matter is reopened and before ClaimCenter commits the data to the database. The Matter Reopened Validation rules execute:

- Before ClaimCenter inserts a new or updated entity into the database.
- Before ClaimCenter executes the Matter Reopened rule set.
- Before ClaimCenter executes the Matter Preupdate rule set.
- Before ClaimCenter executes the Matter Validation rule set.

You can use these rules to raise validation errors or warnings if ClaimCenter must not reopen a matter for some reason. In the base configuration, this rule set is empty.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **MatterReopenedValidationRules**

Matter Validation rules

The rules in this rule set ensure that `Matter` data stored in ClaimCenter is sufficient and valid for downstream processing. ClaimCenter does not enforce validation levels on `Matter` objects, so the Matter Validation rule set always executes, regardless of the validation level.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **MatterValidationRules**

In the base configuration, this rule set is empty.

Person Validation rules

The single rule in the Person rule set ensures that the phone number on the `Person` entity has valid countries defined for it. Guidewire disables this rule in the base configuration.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **PersonValidationRules**

In the base configuration, ClaimCenter provides the following Person Validation rule.

Cell phone has valid country code

Guidewire disables this rule in the base configuration. This rule checks to see if a cell phone number is defined for the contact. If there is one, the rule verifies that the country code is valid. If it is not, the contact is rejected.

See also

- For more phone validation rules, see “Contact Validation rules” on page 109.

Policy Validation rules

The rules in Policy Validation rule set ensure that policy data stored in ClaimCenter is sufficient and valid for downstream processing. ClaimCenter does not enforce validation levels on Policy objects, so the Policy Validation Rule Set always fires, regardless of the validation level.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **PolicyValidationRules**

In the base configuration, ClaimCenter provides the following Policy Validation rules.

POV01000 - Effective and Expiration Dates

This rule checks to see if the effective date of the policy is later than the expiration date. If so, the rule prevents the policy from being saved. The policy effective date cannot be later than the policy expiration date.

POV02000 - Original Effective Date

This rule checks to see if an original effective date exists, which means that the policy effective date has changed. If so, the rule then checks to see if this original date is later than the policy effective date. If so, the rule prevents the policy from being saved. The original effective date cannot be later than the policy effective date.

POV03000 - Endorsement Dates

This rule checks each policy endorsement to see if the effective date of the endorsement is after its expiration date. If so, the rule prevents the policy from being saved. The effective date of an endorsement cannot be later than the expiration date of the endorsement.

RIAgreement Validation rules

Use the rules in the RIAgreement Validation rule set to ensure that reinsurance agreement data is sufficient and valid.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **RIAgreementValidationRules**

In the base configuration, ClaimCenter provides the following RIAgreement Validation rules.

RIAV01000 - Required Fields

This rule serves as a parent to group the following child rules.

RIAV01005 - General

This rule determines if the reinsurance agreement is not a facultative proportional agreement. If not—for example, the agreement is a treaty that is non-proportional—the `TopOfLayer` field, represented in ClaimCenter as the coverage limit, must have a value. If this field is null, the agreement cannot be saved.

RIAV01010 - Non-Proportional

This rule first determines if the agreement is a non-proportional agreement. If so, there must be values for the attachment point and the ceded share. If either or both of these values are `null`, the agreement cannot be saved.

RIAV01020 - Proportional

This rule determines if the agreement is a proportional treaty. If so, its child rules execute.

RIAV01021 - Quota Share

This rule determines if the agreement is a quota share treaty. If so, the value of the ceded share must not be `null`. If it is, the agreement cannot be saved.

RIAV01022 - Surplus

This rule determines if the agreement is a surplus treaty. If so, the value of the attachment point must not be `null`. If it is, the agreement cannot be saved.

RIAV01030 - Agreement Group

This rule checks to see if the reinsurance group for the agreement is null, and, if so, if reinsurance codings are also not specified. If there are no reinsurance codings, the agreement is assumed to be new. A new reinsurance agreement must have a reinsurance group specified before the agreement can be saved.

RIAV02000 - Unique Agreement Name

This rule determines if the agreement is part of an agreement group. If so, the agreement must have a unique name in the group before it can be saved.

RIAV03000 - Proportional Agreement Percentage Total

This rule determines if the agreement is in an agreement group and if the agreement is proportional. If so, the total proportional share percentage of all the agreements in the group cannot be greater than 100%. If the total proportional share exceeds 100%, the agreement cannot be saved.

See also

- *Application Guide*

RITransactionSet Validation rules

You can use the RITransactionSet Validation rule set to ensure that reinsurance transaction data is sufficient and valid.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **RITransactionSetValidationRules**

In the base configuration, ClaimCenter provides the following RITransactionSet Validation rules.

RITSV02000 - Ceded Reserves Are Positive

This rule retrieves all the agreements for the reinsurance ceded reserve transactions in the transaction set. It then determines for each agreement if either or both of the following conditions that will cause the rule to prevent the transaction from proceeding are true:

- Calculated (total non-adjusted) ceded reserves are less than zero.
- Total ceded reserves are less than zero.

RITSV03000 - Ceded Reserves Less Than Open Reserve

This rule retrieves all the agreements for the reinsurance ceded reserve transactions in the transaction set. It then determines for each agreement if either or both of the following conditions that will cause the rule to prevent the transaction from proceeding are true:

- There are calculated (total non-adjusted) ceded reserves (they are greater than zero) and they are greater than the open reserves.
- There are total ceded reserves (they are greater than zero) and they are greater than the open reserves.

RITSV04000 - RI Recoverables Are Positive

This rule retrieves all the agreements for the reinsurance recoverable transactions in the transaction set. It then determines for each agreement if either or both of the following conditions that will cause the rule to prevent the transaction from proceeding are true:

- Calculated (total non-adjusted) reinsurance recoverables are less than zero.
- Total reinsurance recoverables are less than zero.

RITSV05000 - RI Recoverables Less Than Payments

This rule retrieves all the agreements for the reinsurance recoverable transactions in the transaction set. It then determines for each agreement if either or both of the following conditions that will cause the rule to prevent the transaction from proceeding are true:

- There are calculated (total non-adjusted) reinsurance recoverables (they are greater than zero) and they are greater than the net amount paid.
- There are total reinsurance recoverables (they are greater than zero) and they are greater than the net amount paid.

ServiceRequest Validation rules

Use the rules in the Service Request rule set to ensure that service request data is valid for downstream processing. ClaimCenter does not enforce validation levels on `ServiceRequest` objects. Errors will cause validation to fail, regardless of the validation level.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **ServiceRequestValidationRules**

In the base configuration, ClaimCenter provides the following `ServiceRequest` Validation rules.

SRVR01000 - Service Request History

This rule stops a service request and notifies the user if the data in the service request history is not consistent.

SRVR02000 - Service Request Invoice Currencies

This rule stops a service request and notifies the user if the currency of the service request invoice does not match the currency of the associated check.

In the base configuration, this currency compatibility check is also done for checks in the transaction validation rule set. The constraint is enforced on both sides, by service requests and checks, in case only one of the other validation rules is running during a commit.

See also “TransactionSet Validation rules” on page 117.

TransactionSet Validation rules

ClaimCenter runs the Transaction Set Validation rules whenever ClaimCenter saves a `TransactionSet` object. ClaimCenter does not enforce validation levels on `Transaction` objects. Thus, the Transaction Set Validation rule set always fires, regardless of the validation level.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **TransactionSetValidationRules**

In the base configuration, ClaimCenter provides the following Transaction Set Validation rules.

TXV01000 - Total payment not > Exposure limit

This rule either blocks completely or permits, with warnings, a payment that would cause one of the following conditions:

- Making a new payment and adding it to the payments already made would cause the Total Payments to exceed the Exposure limit.
- Making a new payment and adding it to the payments already made plus the remaining reserves would cause the Total Payments to exceed the exposure limit.

The action of the rule depends on the settings of its `warning1` and `warning2` variables. See “TXV01000 – Total payments not greater than exposure limit” on page 176.

TXV02000 - Check exp limit when increasing reserves

This rule executes whenever you attempt to create a new reserve. In the base configuration, the rule either blocks or permits, with warnings, a reserve increase that causes the total reserves to exceed the exposure limit. For example the sum of the combined current payments, the new reserve amount, and remaining reserve exceeds the exposure limit.

The action of the rule depends on the settings of its `warning` variable. See “TXV02000 – Check exposure limit when increasing reserves” on page 177.

TXV03000 - Total Payments not > Incident limit

This rule executes whenever you attempt to create a new check. In the base configuration, the rule either blocks or permits with warnings a payment that causes either total payments or total payments and remaining reserves to exceed the incident limit.

The action of the rule depends on the settings of its `warning1` and `warning2` variables. See “TXV03000 – Total payments not greater than incident limit” on page 177.

TXV04000 - Check incid limit when increasing reserves

This rule executes whenever you attempt to create a new reserve. In the base configuration, the rule either blocks or permits, with warnings, a reserve increase that causes the total reserves to exceed the incident limit. For example the sum of the combined current payments, the new reserve amount, and remaining reserve exceeds the incident limit.

The action of the rule depends on the settings of its `warning` variable. See “TXV04000 – Check incident limit when increasing reserves” on page 178.

TXV05000 - PIP

This rule executes whenever you attempt to save a check. In the base configuration, the rule determines whether adding this new payment would cause the total payments to exceed one of the following limits:

- `PIPNonmedAgg` – Non-medical aggregate limit
- `PIPReplaceAgg` – Lost wages and replacement services aggregate limit
- `PIPPersonAgg` – PIP per person aggregate limit
- `PIPClaimAgg` – PIP per claim aggregate limit

See also

- “TXV05000 – PIP” on page 178
- “sumForPIPAgg enhancement methods” on page 181

TXV06000 - Reserve threshold

This rule provides a hard-coded limit of 1,000,000 in claim currency for creating a single reserve. See “TXV06000 – Reserve threshold” on page 179.

TXV07000 - Default `CheckSet` Rules

This rule executes whenever you attempt to create a check. It requires that a payment that is to be sent to the payee and is not electronic have an address to which the check can be mailed.

TXV08000 - Check Aggregate Limits

This rule notifies the user if a transaction would result in exceeding an aggregate limit. In the base configuration, ClaimCenter displays a warning in these cases, but allows the transaction to go through. See “TXV08000 – Check aggregate limits” on page 179.

TXV10000 - Check

Guidewire disables this rule in the base configuration. If enabled, this rule prevents the user from writing checks for workers’ comp exposures that have no coverage or for which the exposure has not reached the payment validation level.

TXV11000 - Block Payments that Exceed Reserves

Guidewire disables this rule in the base configuration. If enabled, this rule executes whenever you attempt to create a new check. If the check pays claim costs, the rule blocks the check if would cause the claim amount to exceed the available reserves.

TXV12000 - Pending Reserves & Matching Pending Payments

This rule looks for new reserves on reserve lines that have pending payments. If the newly created reserves would make remaining reserves less than pending payments, the rule sends the user a warning message.

TXV13000 - Warning if Adverse Party pays non-Subro Recovery

This rule prevents a recovery paid by a responsible party from being created if the recovery transaction has not been assigned the subrogation category.

See “TXV13000 – Warning if adverse party pays non-subrogation recovery” on page 180.

TXV14000 - Check Send Date Validation

Guidewire disables this rule in the base configuration. If enabled, this rule executes whenever you attempt to create a check. It verifies that all checks in the check set are scheduled to be sent on business calendar days, and not on weekends or holidays.

TXV15000 - Financial Holds

This rule filters for a transaction set that has financial holds and either is a reserve set or is a check set that does not have deleted or canceled checks. If so, the following child rules execute for this transaction set.

TXV15100 - Coverage In Question

This rule determines if the reserve set or check set is for a claim that has coverage in question. If so, the rule prevents the transaction from going through and sends an appropriate message to the user.

TXV15200 - Incident Only

This rule determines if the reserve set or check set is for a claim that is incident only. If so, the rule prevents the transaction from going through and sends an appropriate message to the user.

TXV15300 - Unverified Policy

Guidewire disables this rule in the base configuration. If enabled, the rule determines if the reserve set or check set is for a claim that does not have a verified policy. If so, the rule prevents the transaction from going through and sends an appropriate message to the user.

TXV16000 - New Vendor Management Rules

Guidewire disables this rule in the base configuration. If enabled, the rule determines if the transaction is for a check set that has changed. If so, the child rule executes if enabled.

TXV16100 - Vendor payees should be linked and synched in CM

Guidewire disables this rule in the base configuration. If enabled, the rule determines if the check is a new check, a check that has a change in payee, or a check that is being approved. If so and there is a vendor type payee on the check, the rule lets the check go through only if the vendor is linked and synchronized with ContactManager.

TXV17000 - Reserve Lines compatible with Service Requests

This rule validates all checks linked to a service request invoice. The rule determines if each check's set of reserve lines is compatible with at least one of the service request invoices.

See "TXV17000 – Reserve lines compatible with service requests" on page 180.

TXV18000 - Check currencies compatible with Service Request Invoices

This rule validates the currency for a check that is associated with a service request invoice. It prevents the check from being saved if the check and invoice currencies do not match.

See "TXV18000 – Currencies compatible with service requests" on page 180.

See also

- "Transaction Set validation" on page 175

User Validation rules

ClaimCenter runs the User Validation and User Preupdate rules on a save of the `User` entity. Any exceptions found during validation cause the bounding database transaction to roll back, thus effectively vetoing the update.

Guidewire recommends that you use the LoadSave validation level for rules pertaining to validation for users and groups.

In the base configuration, this rule set is empty. To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Validation** > **UserValidationRules**

Workplan

A *workplan* adds initial activities to a Claim, Exposure or Matter object as a checklist of work that various people need to perform. Guidewire broadly divides these activities into the following categories:

- Activities that are performed a single time for the entire claim

  These activities include contacting the insured, investigating the accident scene, and getting a police report. You can add these types of activities as part of the claim's workplan rules and associate them with the claim as a whole.

- Activities that are performed for each exposure or matter

For example, these activities can include contacting the individual claimants and getting a damage estimate for each vehicle for an auto claim. You can add these types of activities as part of the exposure or matter workplan rules and associate them with the exposure or matter.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Workplan**

Gosu methods to use with Workplan rules

The purpose of Workplan rule set is to add the most important or the most commonly overlooked activities to the workplan for the claim. The following list describes the main methods to use in workplan rules. These rules are available for `Claim`, `Exposure`, and `Matter` business entities.

Method	Description
`createActivity`	Adds an activity based on an activity pattern, but allows you to override some of the defaults.
`createActivityFromPattern`	Adds an activity based on an activity pattern, using all of the pattern's defaults.
`hasDuplicateClaimNumbers`	Returns `true` if there are other claims that are possible duplicates of the claim being checked. The `Claim.DuplicateClaimNumbers` property contains a comma-delimited list of claim numbers that might be duplicates of the current claim, or `null` if no duplicates were found.

See also

- *Application Guide*
- *Configuration Guide*
- *Best Practices Guide*

Claim Workplan rules

The Claim Workplan rules add initial activities to the claim as a checklist of work that various people need to perform on the claim.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Workplan** > **ClaimWorkplan**

In the base configuration, ClaimCenter provides the following Claim Workplan rules.

CLW0100 - Not First and Final

This rule is the parent rule for the child rules that follow. It filters out claims that are first and final because that type of claim is not likely to need further work.

CLW01000 - Contact insured

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW02000 - Thirty day review

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW03000 - New users

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW03100 - Verify coverage

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW04000 - Auto claims

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW04100 - Scene inspection

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW04200 - New users

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW04210 - Police report

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW05000 - Property claims

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW05100 - Property inspection

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW05200 - New users

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW05210 - Police report

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW05300 - Verify coverage

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW06000 - Liability claims

This rule determines if the claim loss type is `GL` for its child rules. For example, in the base configuration, the line of business for this loss type can be Businessowners, General Liability, Other Liability, or Personal Umbrella.

CLW06100 - New users

This rule determines if the user experience level is low. If so, any child rules can run.

CLW07000 - Workers comp claims

This rule determines if the claim is a Workers' Compensation claim for its child rules.

CLW07100 - Initial claim acceptance

This rule creates an activity to determine compensability based on the claim loss date, the report date, and the Workers' Comp compensability parameters.

CLW07200 - Contact claimant

This rule creates an activity to contact the claimant for the Workers' Comp claim.

CLW07300 - New users

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW07310 - Get Employee Injury Notice

This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

CLW08000 - BizRules

This rule executes the business rules that are set to trigger on creation of a claim. The rule runs after the other Claim Workplan Gosu rules have completed.

See also

- *Application Guide*

Exposure Workplan rules

The Exposure Workplan rules add initial activities to the exposure as a checklist of work that various people need to perform on the exposure.

IMPORTANT: If you are adding activities for each exposure, avoid adding redundant activities. For example, if the same person is a claimant on multiple exposures, do not add a separate activity to contact the claimant for each exposure.

To work with these Gosu rules, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Workplan** > **ExposureWorkplan**

In the base configuration, ClaimCenter provides the following Exposure Workplan rules.

EXW01000 - Contact claimant
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW02000 - Auto exposure
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW02100 - Vehicle inspection
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW03000 - Property exposure
: This rule determines if the exposure type is property damage. If so, any child rules execute. There are no child rules in the base configuration.

EXW04000 - Injury exposure
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW04100 - Medical report
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW04200 - IME
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW05000 - WC Injury Exposure
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW05100 - Medical report
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW06000 - WC Lost Wages Exposure
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW06100 - Wage Statement
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW07000 - Homeowners Policy
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW07100 - Get list of damaged items
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW07200 - Contact insured about living expenses
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW07300 - Get property inspected
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW07400 - Get claimant medical reports
: This Gosu rule is disabled in the base configuration because it is handled in the Business Rules editor.

EXW08000 - BizRules
: This rule executes the business rules that are set to trigger on creation of an exposure. The rule runs after the other Exposure Workplan Gosu rules have completed.

See also

- *Application Guide*

Matter Workplan rules

The Matter Workplan rules add initial activities to the matter as a checklist of work that various people need to perform on the matter. In the base configuration, there are no rules in this rule set.

To work with this rule set, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Workplan** > **MatterWorkplan**

Example Matter Workplan rule

In the following example Matter Workplan rule, the rule creates an activity requesting that the user handling the claim refer the matter to legal counsel for review.

```
CONDITION (matter : entity.Matter):
return true

ACTION (matter : entity.Matter, actions : gw.rules.Action):
matter.Claim.createActivityFromPattern( null, ActivityPattern.finder
      .getActivityPatternByCode( "review_legal" ) )
```

chapter 7

ClaimCenter rule reports

This topic provides information on how to generate reports that provide information on the ClaimCenter Gosu rules.

About the Rule Repository report

To facilitate working with the Gosu rules, ClaimCenter provides a command prompt tool to generate a report describing all the existing 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

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.

Generate a Rule Repository Report

Procedure

1. Open a command prompt and navigate to the ClaimCenter installation directory.
2. Execute the following command:

```
gwb genRuleReport
```

Results

This command generates the following files:

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

About the 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 rules that individual user actions trigger within Guidewire ClaimCenter. 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.

See also

- *Administration Guide*

Generate a Profiler Rule Execution Report

Procedure

1. Log into Guidewire ClaimCenter using an administrative account.
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 only.
4. Navigate back to the ClaimCenter 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 ClaimCenter 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**.

Understanding the Profiler Rule Execution Report

After you enable the Web profiler and perform activity within ClaimCenter, the profiler **Profiler Analysis** screen displays zero, one, or multiple stack traces. The following list describes the meaning these 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 ClaimCenter interface did not trigger any rules.
One	A single expanded stack trace. This stack trace lists, among other information, each rule set and rule that ClaimCenter executed as a result of user actions within the ClaimCenter 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 ClaimCenter, 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 ClaimCenter executed in the **Properties and Counters** column in the order in which they occurred.

View rule information in the Profiler Chrono report

About this task

It is possible to use the Guidewire Profiler **Chrono** report to view frames filtered out by the **Rule Execution** report.

Procedure

1. Log into Guidewire ClaimCenter using an administrative account.
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.
4. Make a note of the following information from the rule exception report:
   - Name of the session
   - Name of the stack
   - Time offset
5. Select **Guidewire Profiler** > **Profiler Analysis** > **Web**.
6. Under **Profiler Result**, select **Chrono** from the **View Type** drop-down list.
7. Select the desired session.

   For example:

   2016/05/05 09:23:25 web profiler session
8. Select the desired stack.

   For example:

   DesktopActivities > **AssignActivity**
9. As you select a stack, the **Profiler** page shows additional information about that stack at the bottom of the page.
10. Expand the **Time** node as needed to select the time offset that is of interest to you.

See also

- "Generate a Profiler Rule Execution Report" on page 126
- *Administration Guide*

part 2

Advanced topics

chapter 8

Assignment in ClaimCenter

This topic describes how Guidewire ClaimCenter assigns a business entity or object to a user or group.

Understanding assignment in Guidewire ClaimCenter

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

Assignment in Guidewire ClaimCenter is usually made to a user and a group, as a pair (group, user). However, the assignment of a user is independent of the group. In other words, you can assign an entity to a user who is not a member of the specified group.

IMPORTANT: In the base configuration, ClaimCenter provides assignment methods that take only a user and that assign the entity to the default group for that user. Guidewire deprecates 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.

See also

- “Primary and secondary assignment” on page 132

Assignment Entity Persistence

ClaimCenter persists an assignment anytime that you persist the entity being assigned. Therefore, if you invoke the assignment methods on an entity from Gosu code, you must persist that entity. For example, you can persist an entity as part of a page commit in the ClaimCenter interface.

Assignment Queues

Each group in Guidewire ClaimCenter has a set of `AssignableQueue` entities associated with it. For example, in the base configuration, ClaimCenter assigns each group a queue named `FNOL`. It is possible to modify the set of associated queues.

Guidewire supports assigning `Activity` entities only to queues. Guidewire deprecates the `assignToQueue` methods on `Assignable` in favor of the `assignActivityToQueue` method. This makes the restriction more clear.

If you assign an activity to a queue, you cannot simultaneously assign it to a user as well.

See also

- "Queue assignment" on page 142
- "Working with queues" on page 183

Primary and secondary assignment

Guidewire ClaimCenter distinguishes between the following different types of assignment.

Primary assignment

Also known as user-based assignment, primary assignment assigns an entity to a single owner only. The entity must implement either the `Assignable` delegate or the `CCAssignable` delegate, or both.

Secondary assignment

Also known as role-based assignment, secondary assignment assigns an object 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. The entity must implement the `RoleAssignments` array.

It is possible for an assignable entity to be both primarily and secondarily assignable. For example, in the ClaimCenter base configuration, the `Claim` entity implements the `Assignable` delegate, the `CCAssignable` delegate, and the `RoleAssignments` array.

See also

- "Primary (user-based) assignment entities" on page 132
- "Secondary (role-based) assignment entities" on page 132

Primary (user-based) assignment entities

ClaimCenter 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 anytime that ClaimCenter assigns an item to a single user.

To be available for user-based or primary assignment, an entity must implement one or both of the following delegates:

- `Assignable` delegate
- `CCAssignable` delegate

In the ClaimCenter base configuration, the following objects implement the required delegates:

- `Activity`
- `Claim`
- `Exposure`
- `Matter`
- `ServiceRequest`
- `Subrogation`
- `UserRoleAssignment`

`CCAssignable` implements the `Assignable` delegate.

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

Secondary (role-based) assignment entities

ClaimCenter 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 ClaimCenter, with each role assigned to a specific person. As each of the roles can be held by only a single user, ClaimCenter 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 ClaimCenter base configuration, the following objects contain a `RoleAssignments` array:

- `Claim`
- `Exposure`

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 152 for more information on this type of assignment.

Assignment within the Assignment rule sets

All assignment rules in the **Assignment** folder use the Assignment engine. In working within the context of the assignment rules, use the following assignment properties.

Primary assignment

`entity.CurrentAssignment`

This property returns the current assignment, regardless of whether you attempt to perform 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 assignment

`entity.CurrentRoleAssignment`

This property returns `null` if you attempt to use it for primary assignment.

Secondary assignment and the Assignment rules

`UserRoleAssignment` entities share the rule set of their primary entities. Thus, ClaimCenter assigns the `UserRoleAssignment` entities for an activity using the Activity Assignment rule sets, and so on. It is for this reason that assignment statements in the rules always use the `CurrentAssignment` formulation, such as the following:

```
Activity.CurrentAssignment.assignUserAndDefaultGroup( user )
```

It is important for rules to use this syntax, because it allows ClaimCenter to use the rules (in this case) for both activities and activity-related `UserRoleAssignment` entities.

Assignment outside the Assignment rule sets

It is possible to assign an entity outside of the assignment rules, which use the ClaimCenter Assignment engine. For example, you can assign an entity in arbitrary Gosu code, outside of the assignment rules.

If you assign an entity outside the Assignment engine, then you do not need to use `entity.currentAssignment` to retrieve the current assignment. 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)
```

Global and default Assignment rules

Guidewire provides a number of sample assignment rules in the base configuration. You can access these sample rules in Guidewire Studio, in the **Assignment** rule set category folder. All assignment that takes place in the context of the Assignment rules uses the ClaimCenter Assignment engine to execute the assignment.

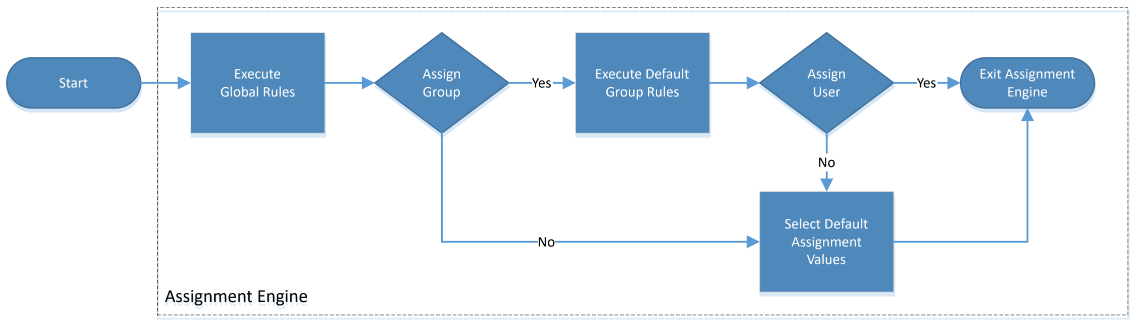
ClaimCenter groups the Assignment rules into the categories described in the following topics:

- "Global Assignment rules" on page 134
- "Default Group Assignment rules" on page 135

In general, ClaimCenter 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, ClaimCenter 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 flowchart, the execution session 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 avoid unintended assignments, Guidewire recommends that you exit the rule set immediately after making a group assignment.

See also

- "Assignment execution session" on page 135

Global Assignment rules

ClaimCenter triggers the Global assignment rules whenever you use auto-assignment to assign an entity. The main purpose of the Global assignment rules is to assign an entity (for example, an activity) to a specific group.

In the base configuration, ClaimCenter provides a number of sample rule sets in the Global assignment rules. Thus, there are is a rule set each to handle activities, claims, exposures, matters, service requests, and subrogation. The Assignment engine runs the rules in the Global rule set a single time, in a sequence set by the rule set hierarchy. If all of the initial Global rules fail to assign a group, the Assignment engine runs the Default rule set.

ClaimCenter runs rule in the Segmentation rule set category just prior to running assignment rules whenever you select automated assignment for a new claim or exposure. See "Segmentation" on page 91 for details.

While it is possible, 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 ClaimCenter, if you choose. Guidewire does not restrict you to using the base configuration Assignment rules only.

Global assignment success or failure

After the Global assignment rules complete, the Assignment engine checks to see if the rules succeeded in assigning the entity to a group. The following list describes the possible assignment outcomes.

Rules assigned the entity to a group	The Assignment engine calls the Default Group assignment rule set for that object type and uses rules in that rule set to assign an owner for the object.
Rules did not assign the entity to a group	The Assignment engine assigns the object to the Default User and to the root group of the ClaimCenter group hierarchy. The Assignment engine then exits.

Default Group Assignment rules

The Assignment engine calls the Default Group assignment rules to assign an entity to a user, after the Global assignment rules assigned the entity to a group.

In the base configuration, ClaimCenter provides a number of sample rule sets in the Default Group assignment rules. Thus, there are is a rule set each to handle activities, claims, exposures, matters, service requests, and subrogation.

Default group assignment success or failure

After the Default Group assignment rules complete, the Assignment engine checks to see if the rules succeeded in assigning the entity to a user. The following list describes the possible assignment outcomes.

Rules assigned the entity to a user	The Assignment engine exits as the assignment of the entity is complete.
Rules did not assign the entity to a user, but did assign the entity to a new group	The Assignment engine re-executes the Default Group rules. You can use the Default Group rules to reassign the entity to different groups at various levels in the ClaimCenter group hierarchy.
Rules did not assign the entity to either a user or a new group	The Assignment engine assigns the entity to the supervisor of the current group.

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`

Assignment success or failure

There are two general types of assignment methods:

- Assignment methods that execute in assignment rules in the context of the Assignment engine
- Assignment methods that execute in Gosu code outside of the Assignment engine

Determining assignment success or failure

Guidewire recommends that you always determine if 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, such as one of the `assignGroup` methods. In this case, it is almost always necessary to call another assignment method to assign the item to an actual user.
- If you call an assignment method in an Assignment rule, you can exit the rule (not consider any further rules), or perform other actions before exiting. For information on the different ways to exit a rule, see “Exiting a Gosu rule” on page 20.

Assignment success or failure within the context of rules

Rule-based assignment methods that execute directly within an assignment rule return `true` if the method makes a valid assignment. If the assignment does not make an assignment, the method returns `false`. If the method returns `false`, it is the responsibility of the calling code to determine what to do. The code can, for example, assign the item to the group supervisor or to invoke another assignment method.

Assignment success or failure outside the context of rules

It is possible to invoke the Assignment engine from outside of the assignment rules by calling the `autoAssign` method. Calling this method from outside the assignment rules invokes the Assignment engine automatically.

If you invoke assignment and it is not possible to assign the item to a user, the default assignment is to the group supervisor. In some cases, it is not possible to make an assignment to a group supervisor. For example, this can be if the assigned group has no supervisor. If it is not possible to assign the item to the group supervisor, then ClaimCenter assigns the item to the ClaimCenter default owner (`defaultowner`).

ClaimCenter 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 user `defaultowner`. If the search finds one of these assignments, the searcher must reassign these items to a proper owner. Guidewire also recommends that the Rule Administrator investigate why ClaimCenter 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`.

Logging assignment activity

Guidewire recommends that you log the action anytime that you use one of the assignment methods. The following example, called in an Assignment rule, shows how to print assignment activity.

```
uses gw.api.system.BCLoggerCategory

ACTION (activiity : entity.Activitym, actions : gw.rules.Action):

  var logger = BCLoggerCategory.ASSIGNMENT

  if (activity.CurrentAssignment.assignUserByRoundRobin(false, activity.CurrentAssignment.AssignedGroup)) {
    logger.info("Assigned to" + activity.AssignedUser.DisplayName)
    actions.exit()
  }
```

Placing the exit action inside the `if` block forces ClaimCenter to continue to the next assignment rule if the current assignment is not successful. A rule is not successful if it cannot choose a group or user to which to assign the item. It is important that you plan an effective exit from a rule. Otherwise, ClaimCenter can make further unanticipated and unwanted assignments. If you perform assignment from an Assignment rule, you need only to use the `actions.exit` method to exit the rule.

See also

- "Exiting a Gosu rule" on page 20

Assignment cascading

For certain top-level assignable entities, ClaimCenter needs to re-assign the related subobjects anytime that it assigns the top-level entity.

For example, if you reassign a claim, ClaimCenter does the following:

- ClaimCenter reassigns all open activities connected to that claim, including all activities currently assigned to the same group and user as the claim. ClaimCenter assigns the activities connected to the claim only, and not the activities connected to any linked exposure.
- ClaimCenter reassigns all non-closed exposures for the claim currently assigned to the same group and user as the claim.

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

Anytime that the assignment status changes on an assignment, ClaimCenter creates an assignment event. The following events can trigger an assignment change event:

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

The following list describes these events.

Old status	New status	Event	Code
Unassigned	Unassigned	None	None
Unassigned	Assigned	`AssignmentAdded`	`Assignable.ASSIGNMENTADDED_EVENT`
Assigned	Assigned	`AssignmentChanged`	`Assignable.ASSIGNMENTCHANGED_EVENT`
Assigned	Unassigned	`AssignmentRemoved`	`Assignable.ASSIGNMENTREMOVED_EVENT`

Note: A change can trigger the `AssignmentChanged` event for any number of reasons. It can happen, for example, if a field such as the assigned user, the assigned group, or the date changes.

Assignment method reference

Guidewire divides the assignment methods into the general categories described in the following topics.

Assignment type	Description	More information
Assignment by assignment engine	Assigns an object outside of the Assignment rules.	"Assignment by assignment engine" on page 138
Group assignment	Assigns an object to a user group only.	"Group assignment (within the Assignment rules)" on page 139
Queue assignment	Assigns an activity to a queue for later assignment to an actual person.	"Queue assignment" on page 142
Immediate assignment	Assigns an object directly to the specified user and group.	"Immediate assignment" on page 143
Claim-based assignment	Assigns an activity based on the claim owner.	"Claim-based assignment" on page 144
Condition-based assignment	Determines a set of qualified assignees and then uses round-robin assignment to assign the object among the members of the set.	"Condition-based assignment" on page 145
Proximity-based assignment	Assigns an object based on its proximity to a specific geographical location.	"Proximity-based assignment" on page 148
Round-robin assignment	Rotates through a set of users, assigning work to each in sequence.	"Round-robin assignment" on page 152
Dynamic assignment	Uses custom logic to perform automated assignment under more complex conditions.	"Dynamic assignment" on page 152
Manual assignment	Creates a new activity directing the specified user to manually perform (or review) the assignment.	"Manual assignment" on page 155

Note: For the latest information and description on assignment methods, consult the Gosudoc. You can also place the Studio cursor within a method signature and press `Ctrl+Q`.

Assignment by assignment engine

It is possible to invoke the Assignment engine, from outside the Assignment rules, to assign an object. To do so, use the `autoAssign` method. However, do not call this assignment method from within the Assignment rules themselves.

autoAssign

```
public boolean autoAssign()
```

This method invokes the rules-based 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 actv = Claim.newActivity( null, null )
actv.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, ClaimCenter ignores the method call and outputs an error message.

Group assignment (within the Assignment rules)

Methods that only assign a group are most useful if called in rules in the Global Assignment rule set, which are responsible for assigning a group only. In all other contexts, you need to assign both a user and a group. Therefore, outside the Group Assignment rules, it makes more sense to use one of the other types of assignment methods that perform both assignments.

Note: It is important to understand that the group 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.

assignGroup

```
boolean assignGroup(GroupBase group)
```

Use this method to assign the indicated group. You use this method in situations in which you know the group to which you want to assign the object. This group assignment method ignores load factors.

The following example assigns an activity associated with a SIU (Special Investigation Unit) escalation to the “Western SIU” group.

```
var siuGroup = find (g in Group where g.Name == "Western SIU").AtMostOneRow

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

assignGroupByLocation

```
boolean assignGroupByLocation(groupType, address, directChildrenOnly, group)
```

This method uses a location-based assigner to assign an assignable entity based on a given address. This group assignment method ignores load factors. Use this assignment method to assign a group based on geographic closeness to a particular address. For example, use this method to assign the assignable entity to a group based in a processing office in a specific time zone.

The parameters for this assignment method have the following meaning:

`groupType`	• If the value of `grouptype` is non-null, the method ignores (filters out) those groups that do not match the provided group type value. • If the value of `groupType` is null, the method ignores group type in the candidate pool of groups.
`address`	Using the supplied address, the method searches for the smallest zone that contains this address, which, in the United States, is the ZIP code (the postal code). If the method does not find a match, it expands the search to the next larger zone, which, again in the United States, is the county. If still not finding zone

	matches with the supplied address, the method expands the size of the zone again and searches for matches at the (U.S.) state level. The method continues this search zone expansion until it finds one or more zone matches with the supplied address. If the method finds multiple groups that match the criteria, it performs simple round-robin assignment among those groups. (Simple round-robin assignment does not use load factors.) The method bases this round-robin assignment on the zone type that the rule engine used to find the initial group match. For example, suppose that the method finds no groups that match by postal code, but does find a few groups that match by county. In that case, the assignment method performs round-robin assignment through the groups that match by county and ignores any groups that match by state.
`directChildrenOnly`	• If the value of `directChildrenOnly` is `true`, the method searches for groups directly underneath the provided group only. It does not search the children of those groups or any further descendants • If the value of `directChildrenOnly` is `false`, the method expands the candidate pool of groups to all descendants of the provided group.
`group`	• If the value of `group` is non-null, the method begins the search with the direct children of the specified group. ClaimCenter does not include the specified group itself in the search. • If the value of `group` is null, the method begins the search with the direct children of the root group. Depending on the size of the group hierarchy, this can potentially cause performance issues if `directChildrenOnly` is set to `false`

Interactions between method parameters

The `group`, `groupType`, and `directChildrenOnly` parameters work in conjunction with each other. For the sake of illustration, assume that `group` and `groupType` hold meaningful values and that `directChildrenOnly` is set to `true`.

1. The method looks at the children of the specified group and filters out any child groups that do not match the provided group type. If no child groups remain after filtering for group type, the method stops and does not assign the work item.
2. Of the remaining child groups (if any), the method looks for group locations that match the smallest zone of the provided address. Finding multiple groups that match as this zone level, the method uses simple round-robin to assign the work item to a group. If no child group matches any zone of the provided address, at any level, the method stops and does not continue searching.

Now, suppose that `group` and `groupType` hold meaningful values, but `directChildrenOnly` is set to `false`.

1. In this case, the method includes in the candidate pool of groups all the descendant groups of the provided group. It removes from consideration any group that does not match the provided group type.
2. The method then uses the zone information of this set of groups to determine which of these groups match with the zone data of the provided address. Assuming multiple matches, the method uses simple round-robin assignment to assign the work item to a group.

Method assignGroupByLocation example

The following code example uses the `getGroupTypeBasedOnClaimSegment` method to find the top two group choices for assigning this claim.

```
var result = libraries.Claimassignment.getGroupTypeBasedOnClaimSegment(claim.LossType, claim.Segment)
var primarygrouptype = result[0]
var secondarygrouptype = result[1]

if (claim.LossLocation != null) {

  if (claim.CurrentAssignment.assignGroupByLocation(primarygrouptype, claim.LossLocation, false,
      claim.CurrentAssignment.AssignedGroup)) {
    actions.exit()
  }

  if (claim.CurrentAssignment.assignGroupByLocation(secondarygrouptype, claim.LossLocation, false,
      claim.CurrentAssignment.AssignedGroup)) {
    actions.exit()
  }

}
```

```
if (claim.Insured.PrimaryAddress != null) {

  if (claim.CurrentAssignment.assignGroupByLocation(primarygrouptype, claim.Insured.PrimaryAddress,
      false, claim.CurrentAssignment.AssignedGroup)) {
    actions.exit()
  }

  if (claim.CurrentAssignment.assignGroupByLocation(secondarygrouptype, claim.Insured.PrimaryAddress,
      false, claim.CurrentAssignment.AssignedGroup)) {
    actions.exit()
  }
}
```

In this example, the assignment logic tries the following assignment options in the listed order until one succeeds:

- Attempt to assign the claim to a group of the primary group type that is near the claim's loss location.
- Attempt to assign the claim to a group of the secondary group type that is near the claim's loss location.
- Attempt to assign the claim to a group of the primary group type that is near the insured's primary address.
- Attempt to assign the claim to a group of the secondary group type that is near the insured's primary address.

assignGroupByRoundRobin

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

This assignment method assigns an item to a group (or one of its subgroups) using round-robin assignment. Use this method to distribute work (assignable items) among multiple groups. The `assignGroupByRoundRobin` method ignores any group for which the `Group.LoadFactor` value is zero (0).

The parameters for this assignment method have the following meaning:

`groupType`	• If the value of `grouptype` is non-null, the method ignores (filters out) those groups that do not match the provided group type value. • If the value of `groupType` is null, the method ignores group type in the candidate pool of groups.
`includeSubGroups`	• If the value of `includeSubGroups` is `true`: ◦ The method considers for assignment all descendants (subgroups) of the provided group. ◦ The method ignores the group load factor in making a round-robin assignment among the various groups. • If the value of `includeSubGroups` is `false`: ◦ The method considers for assignment the direct children of the provided group only. ◦ The method considers the load factor of each child group in making the round-robin assignment.
`group`	• If the value of `group` is non-null, the method begins the search with the direct children of the specified group. ClaimCenter does not include the specified group itself in the search. • If the value of `group` is null, the method begins the search with the direct children of the root group. Depending on the size of the group hierarchy, this can potentially cause performance issues if `includeSubGroups` is set to `false`

Round-robin assignment using load factors

In general, load factors have two purposes or uses:

- The first purpose is to provide a binary on/off switch. If the load factor for a user or group is set to zero (0), it disables assignment to that user or group. Conversely, if the load factor for a group or user is a non-zero value, it enables assignment to that group or user.
- The second purpose is to provide a means to compare the availability of groups or users for assignment. The use of load factors with round-robin assignment is only relevant if assigning within a single group. During round-robin assignment within a group, load factors determine how frequently a given group receives an assignment as compared to its sibling groups. For example, suppose that one group has a factor of 100 and its sibling group has a factor of 50. During round-robin assignment using load factors, the first group receives twice as many assignments

as the second. The same logic holds true for round-robin assignment of users within a group as well. Again, load factors determine the how frequently a given user receives an assignment as compared with the other members of the group.

Whether the `assignGroupByRoundRobin` method uses the group load factor in making a round-robin assignment depends on the value of `includeSubGroups`.

- If `includeSubGroups` is `true`, the method ignores the load factor of each group in making the assignment.
- If `includeSubGroups` is `false`, the method considers the load factor of the group in making an assignment. Load factors are only relevant as a comparative within the group.

Interactions between method parameters

The `group`, `groupType`, and `includeSubGroups` parameters work in conjunction with each other. For the sake of illustration, assume that `group` and `groupType` hold meaningful values and that `includeSubGroups` is set to `false`.

1. The method looks at the direct children of the specified group and filters out any child groups that do not match the provided group type. If no child groups remain after filtering for group type, the method stops and does not assign the work item.
2. Of the remaining child groups (if any), the method uses round-robin assignment to assign the work item to a group. The method considers the load factor of each child group in making the round-robin assignment.

Now, suppose that `group` and `groupType` hold meaningful values, and `includeSubGroups` is set to `true`.

1. In this case, the method includes in the candidate pool of groups all the descendant groups of the provided group. It then filters out any groups from the candidate pool that do not match the provided group type.
2. The method uses round-robin assignment on the remaining groups to assign the work item to a group. The method does not consider the load factor of any group in making the round-robin assignment.

Method assignGroupByRoundRobin example

The following example assigns a complicated automobile claim to one of the child groups that handle complex claims within the currently assigned group.

```
claim.assignGroupByRoundRobin( "autocomplex", false, claim.AssignedGroup)
```

Queue assignment

Each group in Guidewire ClaimCenter has an associated queue to which you can assign items. This is a way of putting assignable entities in a placeholder location without having to assign them to a specific person. Currently, Guidewire only supports assigning Activity entities to a Queue.

Within ClaimCenter, an administrator can define and manage queues through the ClaimCenter **Administration** screen.

See also

- "Assignment Queues" on page 131
- "Working with queues" on page 183

assignActivityToQueue

```
boolean assignActivityToQueue(queue, currentGroup)
```

Use this method to assign this activity to the specified queue. The activity entity then remains in the queue until someone or something reassigns it to a specific user. It is possible to assign an activity in the queue manually (by the group supervisor, for example) or for an individual to choose the activity from the queue.

To use this method, you need to define an `AssignableQueue` object.

Defining an AssignableQueue object by using the group name

You can use the name of the group to retrieve a queue attached to that group.

```
Activity.AssignedGroup.getQueue(queueName)     //Returns an AssignableQueue object
Activity.AssignedGroup.AssignableQueues        //Returns an array of AssignableQueue objects
```

In the first case, you need to know the name of the queue. You cannot do this directly, as there is no real unique identifier for a Queue outside of its group.

Note: In the ClaimCenter base configuration, each Group has an FNOL queue defined for it by default.

If you have multiple queues attached to a group, you can do something similar to the following to retrieve one of the queues. For example, use the first method if you do not know the name of the queue. Use the second method if you know the name of the queue.

```
var queue = Activity.AssignedGroup.AssignableQueues[0]
var queue = Activity.AssignedGroup.getQueue( "QueueName" )
```

You can then use the returned `AssignableQueue` object to assign an activity to that queue.

```
Activity.CurrentAssignment.assignActivityToQueue( queue, group )
```

Immediate assignment

Immediate assignment methods assign an object directly to the specified user or group.

assign

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

This method assigns the assignable entity to the specified group and user, the simplest of 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 another exposure to the owner of an already assigned exposure.

```
for (exp in Exposure.Claim.Exposures) {
  if (exp != Exposure  and exp.AssignedGroup == Exposure.CurrentAssignment.AssignedGroup
        and exp.AssignedUser != null) {
    if (Exposure.CurrentAssignment.assign( exp.AssignedGroup, exp.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 if a user only 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)
```

assignToIssueOwner

```
boolean assignToIssueOwner()
```

This method assigns the assignable entity to the “Issue Owner”. This concept only really applies to Guidewire ClaimCenter, in which the issue owner is the owner of the associated Claim. Use this method to assign a subsidiary entity (for example, an Exposure or a Matter) to the owner of the primary entity. such as the Claim.

The following example assigns an exposure to the issue owner (the owner of the associated claim).

```
Exposure.CurrentAssignment.assignToIssueOwner()
```

assignToCreator

```
boolean assignToCreator(sourceEntity)
```

This method assigns the assignable entity to the user who created the supplied `sourceEntity` parameter.

The following example assigns an exposure to creator of the claim associated with the exposure.

```
exposure.CurrentAssignment.assignToCreator( exposure.Claim )
```

assignToPreviousOwner

```
boolean assignToPreviousOwner()
```

Assigns the entity to the previously assigned user. The method tracks the current group. If there is no current group defined, the method does nothing and logs a warning. Although not officially deprecated, Guidewire recommends that you not use this method.

Claim-based assignment

Guidewire provides several assignment methods that assign an activity based on the claim owner.

assignToClaimOwner

```
void assignToClaimOwner()
```

This method assigns the assignable entity to the user and group of the associated claim. *It is your responsibility to verify that the entity has a link to an assigned claim.* If the resulting assignment is invalid for any reason, the method throws `IllegalAssignmentException`. Currently, Guidewire only supports this method for use with an activity, exposure, or matter.

The following example assigns an exposure to the claim owner.

```
(Exposure.CurrentAssignment as CCAssignable).assignToClaimOwner()
```

Guidewire recommends, before attempting to assign the assignable, that you test if it is possible to assign the assignable using this method. Use the following property on the assignable to determine the owning claim for the assignable.

```
entity.OwningClaim
```

If the return value is `null`, then it is not possible to assign the assignable using the `assignToClaimOwner` method. Otherwise, it returns the owning claim object.

assignToClaimUserWithRole

```
boolean assignToClaimUserWithRole(userRole)
```

This method assigns the assignable entity based on the role associated with that entity. The method performs the following steps in searching for a user with a matching role:

1. If assigning an activity associated with an exposure, the method looks at roles associated with Exposures first.

2. It next attempts to match the supplied `userRole` with a a Claim `UserRole`.
3. If not assigning an activity associated with an exposure, the method then searches for a match with a `UserRole` associated with an Exposure.

During the search:

- If the search finds a match at any step, it stops.
- If the match is unique, the method performs the assignment and returns `true`.
- If the match is not unique, the method returns `false`.
- If the search goes through all steps without finding a match, the method returns `false`.

In the base configuration, you can assign the entity to any one of the following roles:

- Attorney
- Doctor
- Independent Appraiser
- Nurse Case Manager
- Police
- Property Inspector
- Related User
- Repair Shop
- Reporter
- Salvage Owner
- SIU
- SIU Investigator
- Subrogation Owner
- Supervisor
- Vendor
- Vehicle Inspector

Note: Guidewire defines the roles that a user can have on an assignable object in typelist `UserRole`.

The following example assigns an exposure to a claim owner with a user role of `siuinvestigator`, a claim fraud specialist that investigates potential claim fraud.

```
(Exposure.CurrentAssignment as CCAssignable).assignToClaimUserWithRole("siuinvestigator")
```

Condition-based assignment

Condition-based assignment methods follow the same general pattern:

- They use a set of criteria to find a set of qualifying users, which can span multiple groups.
- They perform round-robin assignment among the resulting set of users.

It is important to note:

- ClaimCenter ties the round-robin sequence to the set of criteria, not to the set of users. Thus, using the same set of restrictions to find a set of users re-uses the same round-robin sequence. However, two different sets of restrictions can result in *distinct* round-robin sequences, even if the set of resulting users is the same.
- ClaimCenter does not use this kind of round-robin assignment with the load factors maintained in the ClaimCenter **Administration** screen. Those load factors are meaningful only within a single group, and condition-based assignment can span multiple groups.

assignByUserAttributes

```
boolean assignByUserAttributes(attributeBasedAssignmentCriteria, includeSubGroups,
currentGroup)
```

This method assigns an assignable item to the user who best matches the set of user attribute constraints defined in the `attributeBasedAssignmentCriteria` parameter.

If no user matches the criteria specified by the method, the method assigns the item to the item owner. For example, if the method cannot determine a user from the supplied criteria, it assigns an activity to the owner of the claim associated with the activity.

The `AttributeBasedAssignmentCriteria` object contains two fields:

`Group`	If set, restricts the search to the indicated group. This can be `null`.

`AttributeCriteria`	An array of `AttributeCriteriaElement` entities.

The `AttributeCriteriaElement` entities represent the conditions to be met. If more than one `AttributeCriteriaElement` entity is present, the method attempts to assign the assignable entity to those users who satisfy all of them. In other words, the method performs a Boolean AND operation on the restrictions.

The `AttributeCriteriaElement` entity has a number of fields, which are all optional. These fields can interact in very dependent ways, depending on the value of `UserField`.

Field	Description
`UserField`	The the `AttributeCriteriaElement` behaves differently depending on whether the `UserField` property contains an actual value: • If set, then `UserField` must be the name of a property on the `User` entity. The method imposes a search restriction using the `Operator` and `Value` fields to find users based on their value for this field. • If `null`, then the method imposes a search restriction based on attributes of the user. The exact restriction imposed can be more or less strict based on the other fields set:
`AttributeField`	If set, this is the name of a property on the `Attribute` entity. The method imposes a search restriction using the `AttributeValue` field to find users based on the user having the appropriate value for the named field for some attribute.
`AttributeType`	If set, then the method tightens the `AttributeField`-based restriction to `Attributes` only of the indicated type.
`AttributeValue`	If set, then the method restricts the search to users that have the specified `AttributeValue` only.
`State`	If set, then the method restricts the search to users that have an `Attribute` with the indicated value for `State`.
`Value`	If set, then the method restricts the search to users that have the specified `Value` for an `Attribute` that satisfies the other criteria.

The assignByUserAttribute group parameters

You use the `currentGroup` and `includeSubGroups` parameters to further restrict the set of users under consideration to certain groups or subgroups. The `currentGroup` parameter can be `null`. If it is non-null, the assignment method uses the parameter for the following purposes:

1. The assignment method maintains separate round-robin states for the search criteria within each group. This is so that ClaimCenter can use the method for group-specific assignment rotations.
2. If the method selects a user, it uses the supplied group to determine the best group for the assignment:
   - If the user is as a member of multiple subgroups under the supplied group, the method assigns the object to the closest group in the hierarchy to the supplied group.
   - If the user is a member of multiple groups at the same level of the group hierarchy, the method assigns the object to one of these groups randomly.
   - If the user is not a member of any subgroup under the supplied group, the method assigns the object to the supplied group.
   - If the supplied group value is `null`, the method assigns the object to the first group it finds of which the user is a member. The user must belong to at least one group for this assignment to succeed.

Assign by attribute example

At times, it is important that you assign a particular claim to a specific user, such as one who speaks French or one who has some sort of additional qualification. It is possible that these specially qualified users exist across an organization, rather than concentrated in a single group.

The following example searches for all of `User` entities who have an `AttributeType` of language and `Attribute` value of French.

```
var attributeBasedAssignmentCriteria = new AttributeBasedAssignmentCriteria()
var frenchSpeaker= new AttributeCriteriaElement()
frenchSpeaker.AttributeType = UserAttributeType.TC_LANGUAGE
frenchSpeaker.AttributeField = "Name"
frenchSpeaker.AttributeValue = "French"
attributeBasedAssignmentCriteria.addToAttributeCriteria( frenchSpeaker )
activity.CurrentAssignment.assignByUserAttributes(attributeBasedAssignmentCriteria , false,
activity.CurrentAssignment.AssignedGroup )
```

assignUserByLocation

```
boolean assignUserByLocation(address, includeSubGroups, currentGroup)
```

This method uses a location-based assigner to assign an assignable entity based on a given address. This is useful, for example, in the assignment of adjusters and accident appraisers, which is often done based on geographical territory ownership.

If no user matches the criteria specified by the method, the method assigns the item to the item owner. For example, if the method cannot determine a user from the supplied criteria, it assigns an activity to the owner of the claim associated with the activity.

The method matches users first by zip, then by county, then by state. The first match wins. If one or more users match at a particular location level, then assignment performs round-robin assignment through that set, ignoring any matches at a lower level. For example, suppose the method finds no users that match by zip, but a few that match by county. In this case, the method performs round-robin assignment through the users that match by county and it ignores any others that match by state.

The `assignUserByLocation` method bases persistence in the round-robin assignment state on the specified location information. For this reason, it is preferable to use a partially completed location, such as one that includes only the zip code, rather than a specific house.

The following example searches on the claimant's primary address location.

```
Claim.assignUserByLocation(Claim.Claimant.PrimaryAddress, true, Claim.AssignedGroup)
```

assignUserByLocationAndAttributes

```
boolean assignUserByLocationAndAttributes(address, attributeBasedAssignmentCriteria,
includeSubGroups, currentGroup)
```

The `assigUserByLocationAndAttribute` method is a combination of the `assignUserByLocation` and `assignByUserAttributes` methods. You can use it apply both kinds of restrictions simultaneously. In a similar fashion to the `assignUserByLocation` method, you can use this method in situations in which the assignment needs to take a location into account. (This is the claim loss location, for example.) You can then impose additional restrictions, such as the ability to handle large dollar amounts or foreign languages, for example.

If no user matches the criteria specified by the method, the method assigns the item to the item owner. For example, if the method cannot determine a user from the supplied criteria, it assigns an activity to the owner of the claim associated with the activity.

As with the `assignByUserAttribute` assignment method, if no user matches the criteria specified by the `attributeBasedAssignmentCriteria`, the method assigns the object to the owner of the object being assigned.

The following example searches for a French speaker that is closest to a given address.

```
var attributeBasedAssignmentCriteria = new AttributeBasedAssignmentCriteria()
var frenchSpeaker = new AttributeCriteriaElement()
frenchSpeaker.AttributeType = UserAttributeType.TC_LANGUAGE
frenchSpeaker.AttributeValue = "french"
attributeBasedAssignmentCriteria.addToAttributeCriteria( frenchSpeaker )

activity.CurrentAssignment.assignUserByLocationAndAttributes(claim.LossLocation,
        attributeBasedAssignmentCriteria , false, activity.CurrentAssignment.AssignedGroup)
```

Proximity-based assignment

Guidewire provides several proximity-based assignment methods. Each method provides a slightly different functionality. All proximity-based assignment methods are functions that the `CCAssignable` delegate interface defines.

IMPORTANT: To use the proximity assignment methods, you must integrate ClaimCenter with a geocoding server and geocode all users. For more information on proximity searches and geocoding, see the *Contact Management Guide*.

assignUserByLocationUsingProximitySearch

```
boolean assignUserByLocationUsingProximitySearch(address, includeSubGroups, currentGroup)
```

This method assigns the entity the user with the closest address to the specified address, as measured by a great-circle distance along the surface of the earth. To use this method, you must first geocode all user addresses in the database. You must also geocode the supplied address that serves as the center of the search, either in advance or at the time of the search. If the geocoding attempt fails for the supplied location, the method logs an error and returns `false`. (It also returns `false` if it unable to make an assignment.)

The `assignUserByLocationUsingProximitySearch` method performs the following search algorithm:

1. For each user in the specified group, the method computes the distance from each user location to the supplied location. (It uses the primary address of the `Contact` entity for each user.) The search begins with the users of the current group (`currentGroup`). It is an error if there is no current group.
2. If the `includeSubGroups` parameter is `true`, the method repeats this process with all of the descendant groups of the specified group.
3. The method then returns the user who is closest to the specified location. It does not perform round-robin assignment among the set of users.

This method takes the following parameters:

Parameter	Description
`address`	An address to use as the center of the search.
`includeSubGroups`	if `true`, then include users in any subgroups of the supplied group, and the users in the current group.
`currentGroup`	The group whose members to consider for assignment. This cannot be `null`.

The following example assigns a claim to someone that is closest (in distance) to the claimant's primary address:

```
Claim.assignUserByLocationUsingProximitySearch(Claim.claimant.PrimaryAddress,
      true, Claim.AssignedGroup)
```

assignUserByLocationUsingProximityAndAttributes

```
boolean assignUserByLocationUsingProximityAndAttributes(address,
attributeBasedAssignmentCriteria, includeSubGroups, currentGroup)
```

This method is similar to the `assignUserByLocationAndAttributes` method, except that it also takes user attributes into account in the search algorithm. See "assignUserByLocationAndAttributes" on page 147.

The method uses a distance calculation and one or more attributes to select the closest user. It does not perform round-robin assignment among the set of users. You use this method, therefore, if you want to find the closest user with certain attributes.

This method takes the following parameters:

Parameter	Description
`address`	The location to use as the center of the search.
`attributeBasedAssignmentCriteria`	The user attributes to match against.
`includeSubGroups`	if `true`, then include users in any subgroups of the supplied group, and the users in the current group.
`currentGroup`	The group that includes the users to consider for the assignment

The following example searches for a French speaker that is closest to a given address. The example code first creates a new `AttributeCrieriaElement` object and assigns it a value of French. It then adds the new object to the `attributeBasedAssignmentCriteria` object and uses that object as part of the search criteria. The assignment method then tries to assign the activity to a French speaker that closest, in straight-line distance, to the primary address of the claimant on a claim.

```
var attributeBasedAssignmentCriteria = new AttributeBasedAssignmentCriteria()
var frenchSpeaker = new AttributeCriteriaElement()
frenchSpeaker.AttributeType = UserAttributeType.TC_LANGUAGE
frenchSpeaker.AttributeValue = "french"
attributeBasedAssignmentCriteria.addToAttributeCriteria( frenchSpeaker )
Activity.CurrentAssignment.assignUserByLocationUsingProximityAndAttributes(Activity.Claimant.Person
      .PrimaryAddress, attributeBasedAssignmentCriteria, false, Activity.CurrentAssignment.AssignedGroup )
```

assignUserByProximityWithSearchCriteria

```
public boolean assignUserByProximityWithSearchCriteria(usc, cap, includeSubGroups,
currentGroup)
```

This method assigns the entity to a user based on a *user search by proximity*. It uses the following search algorithm:

1. The method first geocodes the location that serves as the center of the search (if it is not already geocoded). You can then access this location through the following code:

   `usc.getContact().getProximitySearchCriteria()`

2. The method compiles a list of users who satisfy the user search criteria. Use the specified `cap` to limit the number of users in this list. If you include proximity restrictions on the search, the method discards the users farthest from the search center.

3. The method uses the round-robin algorithm to pick one of the found users. This means that repeated, identical calls to this method rotate through the resulting set of users to find the user to return. The method bases the round-robin rotation on the exact `UserSearchCriteria` that you supply. Guidewire recommends that you use as general a location as possible (for example, a city, state or postal code, rather than a specific street address). This maximizes the benefit of the round-robin processing and reduces the load on the system.

This method takes the following parameters:

Parameter	Description
`usc`	An object of type UserSearchCriteria that sets the starting location for the proximity search. This cannot be `null`. The `UserSearchCriteria` object has a foreign key to a `ContactSearchCriteria` entity, which, in turn, has a foreign key to `Address`.
`cap`	Integer value that determines the number of users to return. • If greater than zero, it sets the maximum number of users to include in the round-robin assignment algorithm from the search results. • If zero or less, the method includes all users from the search results in the round-robin assignment algorithm. To always assign the entity to the closest user, set `cap` to 1.

Parameter	Description
`includeSubGroups`	if `true`, then include users in any subgroups of the supplied group, and the users in the current group. The method ignores this parameter if the `currentGroup` parameter is `null`.
`currentGroup`	The group whose members to consider for assignment. This can be `null`. If it is non-null, then the method uses it as part of the search.

IMPORTANT: This assignment method can be very slow compared to other assignment methods. Guidewire recommends that you consider its performance implications before using it.

Setting Up the Proximity Search

To avoid errors in setting up the proximity search, Guidewire recommends that you use the following method:

```
gw.api.geocode.GeocodeScriptHelper.setupUserProximitySearch
```

This method sets up a number of specific fields on the search criteria. The `GeocodeScriptHelper` method returns a `UserSearchCriteria` object that other proximity assignment methods can use.

The `GeocodeScriptHelper` method has the following signature and parameters:

```
setupUserProximitySearch(searchCenter, isDistanceBased, number, unitOfDistance )
```

Parameter	Description
`searchCenter`	The location to use as the center of the search. This is an `Address` object.
`isDistanceBased`	• **Distance-based searches** – Set to `true` if you want to perform a distance-based search. This type of search finds users within n miles (kilometers) of the `searchCenter` location. • **Ordinal-based searches –** Set to `false` if you want to perform an ordinal-based search. This type of search finds the nearest n users to the `searchCenter` location (computed as the straight-line distance).
`number`	• **Distance-based searches –** The search radius in miles or kilometers for distance-based searches • **Ordinal-based searches –** The maximum number of results to return for an ordinal search.
`unitOfDistance`	• **Distance-based searches –** Indicates the unit (miles or kilometers) to use for distance-based searches. • **Ordinal-based searches –** Determines how ClaimCenter displays the results, ignored otherwise. This parameter takes its value from `UnitOfDistance` typelist. • `UnitOfDistance.TC_KILOMETER` • `UnitOfDistance.TC_MILE`

Distance-based Proximity Assignment Example

The following example sets up the proximity information to use in the search using the `Claim.LossLocation` as the supplied address, starting with a search radius of 10 miles. This is a distance-based search as the `isDistanceBased` parameter is set to `true`. Notice that this code also logs whether the assignment succeeded or failed.

```
uses gw.api.geocode.GeocodeScriptHelper

// Set up a distance-based proximity search for claim assignment
if (claim.LossLocation != null) {

  var usc = GeocodeScriptHelper.setupUserProximitySearch(claim.LossLocation, true, 10,
        UnitOfDistance.TC_MILE)

  if (claim.CurrentAssignment.assignUserByProximityWithSearchCriteria(usc, -1, true, claim.AssignedGroup )) {
    actions.exit()  //Needed if using this method within the context of the assignment rules
  }
}
```

Ordinal-based Proximity Assignment Example

The following example sets up the proximity information to use in the search using a supplied address. This is a ordinal-based search as the `isDistanceBased` parameter is set to `false`. It returns the closest three users to the supplied address.

```
uses gw.api.geocode.GeocodeScriptHelper

// Set up a ordinal-based proximity search for claim assignment
var usc = GeocodeScriptHelper.setupUserProximitySearch(claim.LossLocation, false, 3, UnitOfDistance.TC_MILE)

//Set user city and postal code of search center location
usc.Contact.Address.City = claim.LossLocation.City
usc.Contact.Address.State = claim.LossLocation.State
>usc.Contact.Address.PostalCode = claim.LossLocation.PostalCode

if (claim.CurrentAssignment.assignUserByProximityWithSearchCriteria(usc, -1, true, claim.AssignedGroup )) {
  actions.exit()  //Needed if using this method within the context of the assignment rules
}
```

assignUserByProximityWithAssignmentSearchCriteria

```
boolean assignUserByProximityWithAssignmentSearchCriteria(asc, cap, includeSubGroups,
currentGroup)
```

This method is nearly identical to the `assignUserByProximityWithSearchCriteria` method, except that it uses an `AssignmentSearchCriteria` object rather than a `UserSearchCriteria` object. See that method for more information. See “assignUserByProximityWithSearchCriteria” on page 149.

This method takes the following parameters:

Parameter	Description
`asc`	An object of type `gw.api.assignment.AssignmentSearchCriteria` that sets the starting location for the proximity search and the user attribute to match.
`cap`	Integer value that determines the number of users to return. • If greater than zero, it sets the maximum number of users to include in the round-robin assignment algorithm from the search results. • If 0 or less, the method includes all users from the search results in the round-robin assignment algorithm.
`includeSubGroups`	if `true`, then include users in any subgroups of the supplied group, and the users in the current group. The method ignores this parameter if the `currentGroup` parameter is `null`.
`currentGroup`	The group whose members to consider for assignment. This can be `null`. If it is non-null, then the method uses it as part of the search.

IMPORTANT: This assignment method can be very slow compared to other assignment methods. Guidewire recommends that you consider its performance implications before using it.

The following example attempts to assign a claim to a French speaker within 10 mile radius of the claim loss location. Notice that the code first uses the `CCGeocodeScriptHelper.setupAssignmentProximitySearch` method to set up the parameters for the proximity search. It then sets up the French-speaking attribute on the `CCUserSearchCriteria` object.

```
uses gw.api.geocode.CCGeocodeScriptHelper

// Set up an distance-based proximity search for claim assignment
if (claim.LossLocation != null) {

  var asc = CCGeocodeScriptHelper.setupAssignmentProximitySearch(Claim.LossLocation, true, 10,
         UnitOfDistance.TC_MILE )

  // Set the custom attribute match.
  asc.CCUserSearchCriteria.AttributeName = "French"
  asc.CCUserSearchCriteria.AttributeValue = 1
  if (claim.assignUserByProximityWithAssignmentSearchCriteria(asc, -1, true,
          claim.CurrentAssignment.AssignedGroup )) {
    actions.exit()  //Needed if using this method within the context of the assignment rules
  }
}
```

Round-robin assignment

The round-robin algorithm rotates through a set of users, assigning work to each in sequence. It is important to understand that round-robin assignment is distinct from workload-based assignment. Round-robin assignment does not take the current number of entities assigned to any of the users into account.

You can use load factors (in some circumstances) to affect the frequency of assignment to one user or another. For example, suppose that person A has a load factor of 100 and person B has a load factor of 50. In this case, the round-robin algorithm selects person A for assignment twice as often as person B.

See also

- See “Secondary (role-based) assignment entities” on page 132 for a discussion on secondary assignment and round-robin states.
- See “assignGroupByRoundRobin” on page 141 for a discussion of assigning a group through round-robin assignment.

assignUserByRoundRobin

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

This method uses the round-robin user selection to choose the next user to receive the assignable from the current group or group tree. If the `includeSubGroups` parameter is `true`, the selector performs round-robin assignment not only through the direct children of the current group, but also through all of the parent group’s subgroups.

To give a concrete example, suppose that you have a set of claims that you want to assign to a member of a particular group. If you want all of the users within that group and all of its descendent groups to share the workload, then you set the `includeSubGroups` parameter to `true`.

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
```

Dynamic 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. You must lock the rows in the database for these entities.
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. That method returns a `GroupUser` entity representing the user and group that you need to assign.
6. Commit the transaction, which results in the lock entities being updated and unlocked.

   Dynamic assignment is not complete after these steps. Often, such as during FNOL intake or creating a new claim in a wizard, ClaimCenter performs assignment and updates workload information well before it saves the claim. If ClaimCenter cannot save the claim, the database still shows the increase in the user's workload. The interface methods allow for the failure of the commit operation by adding one last final step.

   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 later, during a future save operation.

Dynamic assignment – required methods implementation

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

`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 that you must lock. By locked, Guidewire means that the current server node obtains the database rows corresponding to those entities, which must be persistent entities. Any other server nodes that needs to access these rows must wait until the assignment process finishes. Round-robin and dynamic assignment require this sort of locking to mandate that multiple nodes do not perform simultaneous assignments. This ensures that multiple nodes 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.) This 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 ClaimCenter 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 caused a database rollback. However, at this point, ClaimCenter has already updated the locked objects and committed the objects to the database (as in "step 6" in the assignment flow).

If the bundle commit does not succeed, ClaimCenter 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 ClaimCenter 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 it is not possible to commit an assignment, your logic does one of the following:

- ClaimCenter rolls back any database changes that you made.
- ClaimCenter preserves the assignment in the event that you invoke the assignment logic again.

Dynamic assignment – DynamicUserAssignmentStrategy implementation

As a very simple implementation of this API, Guidewire includes the following in the base configuration:

`gw.api.LeastRecentlyModifiedAssignmentStrategy`

The following code shows the 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, the class 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.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 = candidates as Set<User>
    var oldestModifiedUser = 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
  }
}
```

Manual assignment

Manual assignment is slightly different from other assignment scenarios. Instead of assigning the assignable entity, manual assignment creates a new activity directing the specified user to manually perform (or review) the assignment. To be eligible to receive a manual assignment activity, a user must have the Review Assignment (`actreviewassign`) permission.

In the base configuration of ClaimCenter, the following roles all have the Review Assignment permission.

- Claim Supervisor
- New Loss Processing Supervisor
- Manager

IMPORTANT: Only use manual assignment methods, such as `assignManually`, in the Assignment rule set.

assignManually

```
boolean assignManually(responsibleUser)
```

This method designates a user to look at the assignable and manually assign it to someone. Use this method to flag the assignable for manual review and assignment. If you call this method, ClaimCenter does not actually assign the assignable entity *until* the designated user takes action to do so.

The following example assigns the claim to the group supervisor to assign manually.

```
Claim.CurrentAssignment.assignManually( Claim.CurrentAssignment.AssignedGroup.Supervisor )
```

confirmManually

```
boolean confirmManually(responsibleUser)
```

This method requests manual confirmation of the assignable entity's current assignment. It is roughly equivalent to `assignManually`, except that it assumes that a provisional assignment has already been made. ClaimCenter does not consider the assignment to be final until someone reviews and approves it.

Note: Items that you mark for manual confirmation appear in the **Pending Assignment** list of the supervisor of the assigned group.

The following example first assigns a claim provisionally to a user using round-robin assignment, then creates an activity for that user’s supervisor to review the assignment and approve it.

```
Claim.CurrentAssignment.assignUserByRoundRobin( false, Claim.CurrentAssignment.AssignedGroup)
Claim.CurrentAssignment.confirmManually( Claim.CurrentAssignment.AssignedGroup.Supervisor )
```

assignManuallyByRoundRobin

```
boolean assignManuallyByRoundRobin(group)
```

This method is similar to `assignManually`. However, this method shifts the responsibility for manual assignment to members of the specified group who have the `actreviewassign` permission. Use this method if the workload of manual assignment is too large for a single user. You would use this method to distribute the workload among members of a group that have the requisite permission.

For example, suppose that you have a team of supervisors responsible for assigning certain sensitive tasks to a set of Claims adjusters. You want manual assignment because you want the supervisors to select someone manually. But, you also want to rotate the responsibility for making assignment through the group of supervisors. This is different from the standard round-robin assignment, which rotates the assignment itself. This method rotates the designation of the person responsible for completing the manual assignment.

The following example moves the assignment responsibility among members of a particular group.

```
Claim.CurrentAssignment.assignManuallybyRoundRobin( Claim.CurrentAssignment.AssignedGroup)
```

confirmManuallyByRoundRobin

```
boolean confirmManuallyByRoundRobin(group)
```

This method is similar to `confirmManually`, except that it determines the responsibility for assignment confirmation through round-robin assignment among users in the specified group. Again, you would use a method of this type if the workload of assignment confirmation is too large for a single user.

The following example first assigns a claim to a user using round-robin assignment through members of a group. It then assigns responsibility for assignment confirmation to a member of that group.

```
Claim.CurrentAssignment.assignUserByRoundRobin( false, Claim.CurrentAssignment.AssignedGroup)
Claim.CurrentAssignment.confirmManuallybyRoundRobin( Claim.CurrentAssignment.AssignedGroup)
```

Workload Count

In many cases, a ClaimCenter user can be a member of multiple work teams or groups. This means that you can assign a user to a claim review activity as a member of one group. You can then also assign that user a claim as a member of a different group. Therefore, in determining workload assignments, it is important to take the total (global) number of items assigned to an individual into account.

To this end, ClaimCenter provides a count of the total number of activities, claims, exposures, and matters assigned to an individual. You can access these total counts both through the ClaimCenter interface and through fields on the `User` object in Gosu.

Accessing workload count on the Team tab

The **Team** tab in ClaimCenter displays item counts for each user who is part of that team, organized into columns by each item type—claim, activity, and so on. However, these numbers are simply the item counts for that user in that group and do not include items assigned to that user as a member of another group. To identify the total number of items assigned to a user, ClaimCenter displays a global total for an item type in parentheses next to the subtotal number. You must be an administrator or a group supervisor to access the **Team** tab.

Accessing workload count in Gosu code

There are fields on the entities `User` and `Group` that determine the total number of items of a given type assigned to an individual or group. You can access these fields in Gosu code.

The fields are:

- `OpenActivityCount`
- `OpenClaimCount`
- `OpenExposureCount`
- `OpenMatterCount`

For example, the following Gosu code returns the total number of activities across all groups for the current owner of a claim:

```
var total = Claim.AssignedUser.OpenActivityCount
```

ClaimCenter updates the workload count numbers hourly while running statistics batch processing.

See also

Administration Guide

chapter 9

Rules-based validation

Note: Although the terms *object* and *entity* are not entirely identical, this topic uses these terms interchangeably. See “Rules: A background” on page 13 for a description of each.

ClaimCenter runs preupdate and validation rules every time that it commits data to the database. Guidewire calls a commit of data to the database a *database bundle commit.* The validation rules execute after ClaimCenter runs all preupdate callbacks and preupdate rules. ClaimCenter runs the validation rules as the last step before ClaimCenter actually writes data to the database.

Before applying rules during the bundle commit operation, ClaimCenter builds a validation object graph according to configuration settings. The graph contains possible targets for rule execution and both the preupdate and validation rules use the validation object graph.

Preupdate rules, unlike validation rules, can, and do, modify additional objects during execution. If a preupdate rule does modify additional objects, it is often necessary to construct these objects after the preupdate rules run and before the validation rules run.

If the preupdate rules:

- Modify an entity that is not in the commit bundle, ClaimCenter can possibly run additional validation rules if the entity being modified also triggers validation.
- Modify only entities that are already in the commit bundle, ClaimCenter runs the validation rules for the same set of entities on which the preupdate rules ran.
- Modify something that simply triggers validation for one of the top-level entities, there is no noticeable difference from those validation rules. ClaimCenter was already going to run these rules anyway.

The set of entities that the rules validate can be a superset of the entities for which preupdate rules are run. The preupdate rules are non-recursive—they do not run a second time on objects modified during execution of the preupdate rules. ClaimCenter adds any objects modified by the preupdate rules to the list of objects needing validation as described by the validation graph.

See also

- For information about customizing the processing of preupdate and validation rules, see the *Integration Guide*.

See also

- “Validation” on page 98

Overview of rules-based validation

For an entity to have preupdate or validation rules associated with it, the entity must be validatable. To be *validatable*, the entity must implement the `Validatable` delegate, defining an `implementsEntity` element with `name` set to `Validatable`.

In the base configuration, ClaimCenter provides a number of high-level entities that are validatable:

- `Claim`
- `Exposure`
- `Matter`
- `Policy`
- `RIAgreement`
- `RITransactionSet`
- `ServiceRequest`
- `TransactionSet` (and all its subclasses)
- `Activity`
- `Contact`
- `Group`
- `Region`
- `Person`
- `User`

See also

- "Top-level entities that trigger validation" on page 162
- *Configuration Guide*

Entity validation order

ClaimCenter evaluates entities in the order set in the `config.xml` configuration parameter `EntityValidationOrder`. Use this parameter to specify a comma-separated list of the validatable entities in the order in which you want ClaimCenter to perform validation. If you do not change this parameter, ClaimCenter performs entity validation by using the default order defined in the base configuration:

- `Policy`
- `Claim`
- `Exposure`
- `Matter`
- `TransactionSet`

ClaimCenter validates all other validatable entities not specified in the list after evaluating the listed entities in configuration parameter `EntityValidationOrder`, in no particular order.

IMPORTANT: ClaimCenter does not use any predetermined order to validate entities of the same type, such as the exposures on a claim.

About the validation graph

During database commit, ClaimCenter performs validation on the following:

- Any validatable entity that ClaimCenter updated or inserted into the validation graph.
- Any validatable entity that refers to an entity that ClaimCenter updated, inserted, or removed from the validation graph.

ClaimCenter constructs a virtual graph of the entities that reference a changed entity. This graph maps all the paths from each type of entity to the top-level validatable entities, such as `Claim` and `Exposure`. ClaimCenter queries these paths in the database or in memory to determine which validatable entities, if any, reference the entity that ClaimCenter inserted, updated, or removed from the validation graph.

ClaimCenter builds the validation graph by traversing the set of foreign keys and arrays that trigger validation. For example, if the data model marks the `Exposures` array on `Claim` as triggering validation, any change made to an exposure causes ClaimCenter to validate the claim as well.

ClaimCenter follows foreign keys and arrays that trigger validation through any links and arrays on the referenced entities down the object tree. For example, you might end up with a path like `Claim` → `ClaimContact` → `ContactAddress` → `Address`.

> **Note:** To actually trigger validation, you must set each link in the chain—`Address`, `ContactAddress`, and `ClaimContact`—as triggering validation and set `Claim` as validatable.

ClaimCenter stores this path in reverse in the validation graph. Thus, if an address changes, ClaimCenter traverses `Address` → `ContactAddress` → `ClaimContact` → `Claim` to find any claims that reference a changed address.

How ClaimCenter traverses the validation graph

If a entity is validatable, ClaimCenter applies the preupdate and validation rules whenever you modify the entity contents directly. Suppose that you update an object linked to an object. For example, a foreign key links each of the following objects to the `User` object:

- `Contact`
- `Credential`
- `UserSettings`

If you update a user's credential, you might expect the preupdate and validation rules to execute before ClaimCenter commits the updates to the database. However, updating a user’s credentials does not normally trigger rules for the container object, `User` in this case. The reason is that ClaimCenter implements a user’s credentials as a foreign key from a `User` entity to a `Credential` entity.

To trigger preupdate and validation rules on linked objects, set the `triggersValidation` attribute of the foreign key to the object in the file defining the entity. For example, in `User.eti`, the following `foreignkey` elements have their `triggersValidation` attributes set to `true`:

```
Contact
Credential
```

If the container object implements `Validatable`, setting the `triggersValidation` attribute to `true` ensures that ClaimCenter runs the preupdate and validation rules on the linked object whenever you modify the object.

The `triggersValidation` attribute works from the direction of the element that the foreign key references. If `triggersValidation` is set to `true` on a foreign key or array element of entity `Alpha` that links to entity `Beta`, a change to `Beta` triggers validation in `Alpha`. For example, on the `Policy` object, `triggersValidation` is set to `true` on the array of `ClaimContact` objects. Therefore, a change to a `ClaimContact` object also causes ClaimCenter to validate the `Policy` object.

In the base configuration, ClaimCenter sets the validation triggers so that modifying a validatable entity in a bundle causes ClaimCenter to validate that entity and all its revisioning parents.

You can use the `triggersValidation` attribute on foreign keys and arrays that you add to custom subtypes, to custom entities, or to core entities by using extensions. The default value of this attribute is `false` if you do not set it specifically. If you want a new foreign key or array element of a validatable entity to trigger validation whenever its linked entities change, set `triggersValidation` to `true`.

How ClaimCenter works with owned arrays

In the ClaimCenter data model, an *owned array* is an array that is attached to a validatable entity and which has it `triggersValidation` attribute set to `true`.

ClaimCenter performs validation based on the validation graph. If an entity changes, ClaimCenter follows all the arrays and foreign keys that reference that entity and for which `triggersValidation` is `true`. It then uses the graph to find the validatable entities that ultimately reference the changed entity. In the case of owned arrays, ClaimCenter considers a change to the array as a change to the parent entity.

For example, the following data model illustrates this behavior:

- Entity `Alpha` has a foreign key to `Beta`
- Entity `Beta` has an owned array of `Chiron`
- Array `Chiron` has a foreign key to `Delta`

Both `Alpha` and `Beta` are validatable. Essentially, these relationships look like the following:

```
Alpha → Beta → Chiron[] → Delta
```

If you also mark both `Alpha` → `Beta` and `Chiron` → `Delta` as triggering validation, the following happens with the `Beta` → `Chiron` link:

- If the `Beta` → `Chiron` link has its `triggersValidation` element set to `false`, changes to `Chiron` cause ClaimCenter to validate `Beta` and `Alpha`. This evaluation occurs because ClaimCenter treats the change as a change to `Beta` directly. A change to `Delta`, however, does not cause ClaimCenter to validate anything—the graph stops at `Chiron`. Because there is no change to `Chiron`, ClaimCenter does not consider `Beta` to have changed and performs no validation.
- If the `Beta` → `Chiron` link has its `triggersValidation` element set to `true`, changes to either `Chiron` or `Delta` cause ClaimCenter to validate both `Beta` and `Alpha`.

Top-level entities that trigger validation

To be validatable, an entity, subtype, delegate, or base object must implement the `Validatable` delegate by adding the following to the entity definition:

```
<implementsEntity name="Validatable"/>
```

The following table lists the entities that trigger validation in the base ClaimCenter configuration. All these entities implement the `Validatable` delegate. If a table cell is empty, there are no additional entities associated with that top-level entity that trigger validation in the base configuration.

Validatable entity	Foreign key entity	Array name	One-to-one entity
`Activity`			
`Claim`	`Address`	`Activities`	`PropertyFireDamage`
	`ClaimWorkComp`	`ConcurrentEmpl`	`PropertyWaterDamage`
	`Policy`	`Contacts`	`SubrogationSummary`
	`PolicyLocation`	`ContribFactors`	
		`Documents`	
		`Evaluations`	
		`Exposures`	
		`Incidents`	
		`Matters`	
		`MetroReports`	
		`Negotiations`	
		`Notes`	
		`Officials`	
		`OtherBenefits`	
		`RIAgreementGroups`	
		`RoleAssignments`	
		`ServiceRequests`	
		`SIAnswerSet`	
		`Text`	

Validatable entity	Foreign key entity	Array name	One-to-one entity
`Contact`	`Address`	`ContactAddresses` `Reviews`	

`Exposure`	`Address`
	`Benefits`
	`Coverage`
	`EmploymentData`
	`Incident`
	`RIAgreementGroup`
	`StatCode`
		`BenefitPeriods`
		`Documents`
		`IMEPerformed`
		`MedicalActions`
		`Notes`
		`OtherCoverageDet`
		`Roles`
		`RoleAssignments`
		`Settlements`
		`Text`	

`Group`			
`Matter`		`Roles` `StatusTypeLines`	
`Person`	`Address`	`ContactAddresses` `Reviews`	

`Policy`		`ClassCodes`
		`Contacts`
		`Coverages`
		`Endorsements`
		`PolicyLocations`
		`RiskUnits`
		`Roles`
		`StatCodes`	

`Region`			
`RIAgreement`			
`RITransactionSet`		`RITransactions`	

`ServiceRequest`		`History`
		`Invoices`
		`Messages`
		`Quotes`	

`TransactionSet`			

`User`	`Credential`
	`UserContact`
	`UserSettings`
		`Attributes`	

In many cases, an entity several levels down triggers validation on one of the top-level entities. For example, `Claim` preupdate and validation rules run after you update the address for a contact. The `Contact` entity definition has a `triggersValidation` foreign key attribute to the `Address` entity. The `Claim` entity also has a `triggersValidation` foreign key relationship with the `Address` entity. ClaimCenter follows the chain of triggering validations during a database commit.

Overriding validation triggers

In the base configuration, Guidewire sets validation to trigger on many top-level entities and on many of the arrays and foreign keys associated with them. You cannot modify the base metadata XML files in which these configurations are set. However, in some cases you can override the default validation behavior. For example:

- You want to trigger validation upon modification of an entity in the ClaimCenter base data model that does not currently trigger validation.
- You do not want to trigger validation on entities on which ClaimCenter performs validation in the base configuration.

You can override validation only on certain objects associated with a base configuration entity. In your own extension entities, you simply set `triggersValidation` to the appropriate value.

The following table lists the data objects for which you can override validation, the XML override element to use, and the location of additional information.

Data field	Override element	Topic
`<array>`	`<array-override>`	*Configuration Guide*
`<foreignkey>`	`<foreignkey-override>`	*Configuration Guide*
`<onetoone>`	`<onetoone-override>`	*Configuration Guide*

See also

- *Configuration Guide*

Validation performance issues

There are three ways in which validation can cause performance problems:

- The rules themselves have performance issues.
- There are too many objects being validated.
- The queries used to determine which objects to validate take too long.

The following list summarizes some of the common validation issues and how to mitigate each one.

Validating large amounts of administration objects

Guidewire recommends that you never mark foreign keys and arrays that point to administration objects, such as `User`, `Group`, and `Catastrophe` objects, as triggering validation. An administration object is any object that a large number of claims can reference. For example, suppose that you set the catastrophe field on `Claim` to trigger validation. Editing one catastrophe can then bring the entire application to a crawl as ClaimCenter validated hundreds or even thousands of claims. Setting this particular validation trigger would make editing catastrophe objects nearly impossible.

Length of query paths

In some cases, an entity can have a large number of foreign keys pointing at it. Triggering validation on the entity can cause performance problems, as ClaimCenter must follow each of those chains of relationships during validation. The longer the paths through the tables, the more expensive the queries that ClaimCenter executes whenever an object changes. Having a consistent direction for graph references helps to avoid performance problems.

Triggering validation on parent-pointing foreign keys on entities that have many possible owners, like `ClaimContact`, can result in much longer and unintended paths. For example, a number of entities point to `ClaimContact`, including `Claim`, `Check`, and `MedicalTreatment`. Each of these entities has links to other entities. Validating the entire web of relationships can have a serious negative performance impact.

In the default configuration, ClaimCenter minimizes this issue. However, if you modify the default configuration and add too many validation trigger overrides, you can introduce unintended performance issues.

Links between top-level objects

It is legal to have top-level entities trigger validation on each other. This use of validation triggers, however, can unnecessarily increase the number of paths and the number of objects that ClaimCenter must validate on any particular commit.

Graph direction inconsistency

In general, Guidewire strongly recommends that you consistently order the validation graph to avoid problems:

- ClaimCenter consider arrays and foreign keys that represent some sort of containment as candidates for triggering validation. For instance, you can logically consider contacts and vehicles on a claim as part of the claim.
- ClaimCenter does not consider foreign keys that you reference as arrays or that are merely there to indicate association as candidates for triggering validation. For instance, an `Incident` on a `ServiceRequest` entity merely indicates association, rather than indicating that the incident is logically part of the service request.

Illegal links and arrays

Virtual foreign keys cannot trigger validation because ClaimCenter does not store them in the database. Attempting to use a virtual foreign key to trigger validation causes ClaimCenter to report an error at application startup. Similarly, it is not possible to set any array or link property defined at the application level (such as the `Driver` link on `Claim`) to trigger validation.

ClaimCenter considers an array to be in the database if the foreign key that forms the array is in the database.

See also

- “How ClaimCenter traverses the validation graph” on page 161
- “View the validation graph in the server log” on page 165

View the validation graph in the server log

About this task

It is possible to view a text version of the ClaimCenter validation graph for aid in debugging validation issues.

Procedure

1. In the Guidewire Studio **Project** window, expand **configuration** > **config** > **logging**.
2. Open file `log4j2.xml` for editing.
3. Add the following entry to this file:

```
################################################################################
# ClaimCenter Claim Validation Graph
################################################################################
log4j.category.com.guidewire.pl.system.bundle.validation.ValidationTriggerGraphImpl=
      DEBUG, Console, ClaimValidationGraphLog
log4j.appender.ClaimValidationGraphLog=org.apache.log4j.DailyRollingFileAppender
log4j.appender.ClaimValidationGraphLog.encoding=UTF-8
log4j.appender.ClaimValidationGraphLog.File=/tmp/gwlogs/ClaimValidationGraphLog.log
log4j.appender.ClaimValidationGraphLog.DatePattern=.yyyy-MM-dd
log4j.appender.ClaimValidationGraphLog.layout=org.apache.log4j.PatternLayout
log4j.appender.ClaimValidationGraphLog.layout.ConversionPattern=%-10.10X{server}
%-8.24X{userID}
      %d{ISO8601} %p %m%n
```

4. Redeploy logging configuration file and restart the ClaimCenter application server.

Results

This procedure causes the validation graph to print during application deployment both as text in the system console and to log file `/tmp/gwlogs/ClaimValidationGraphLog.log`.

chapter 10

Detecting claim fraud

This topic discusses how you can use Guidewire ClaimCenter to detect claim fraud and how to create Gosu rules to trigger a special investigation of a suspicious claim.

Claim fraud

Guidewire provides a number of Gosu rules to assist in determining claim fraud, which you can use to trigger a special investigation of a suspicious claim. These Gosu rules identify certain characteristics of a claim that increase the suspicion of fraud and assign points to each of these characteristics. Depending on your business logic, you can:

- Assign different point values to these fraud characteristics based on their “severity”
- Set fraud triggers based on the presence of data, the absence of data, or the value of certain data

Using these Gosu rules, you can identify a point threshold after which the claim merits human review. A human reviewer (typically, the claim owner’s supervisor) can then determine whether to assign the claim to a Special Investigation Unit for further investigation.

Using Gosu rules to trigger claim fraud investigations provides a number of benefits, including:

- Providing a standardization of the process
- Enforcing business processes across the organization
- Assigning standardized weight to every characteristic evaluated
- Providing transparency of process
- Providing a consistent evaluation of all claims, rather than using intuition
- Keeping a record of the claim review, why the claim was chosen for review and the nature of the review

Many of these benefits can be important from a legal perspective.

The Special Investigation Details screen

In Guidewire ClaimCenter, you use the **Special Investigation Details** screen to view and track details of suspicious claims. This screen contains:

- A read-only list of conditions—Special Investigation (SI) triggers—that this claim violates
- An SI score field, which is a “counter” that tracks the SI points accumulated on this claim
- Fields to track escalation of the claim to the Special Investigation Unit (SIU), the fraud detection team

You control the user interface elements that a specific user can access and view through ClaimCenter permission settings.

Special Investigation trigger rules

Special Investigation (SI) trigger rules flag suspicious information or lack of information on claims. For example, the following occurrences all trigger SI rules:

- All the telephone fields for the contact on the exposure are null.
- The claim file does not record a police report or on-scene report.
- There was an unreasonable delay in reporting the loss, such as a delay of more than 30 days between the loss date and the claim report date.

During evaluation of the SI trigger rules, if a trigger rule evaluates to `true` and the rule has not yet been logged on the claim, then ClaimCenter:

- Adds the rule to the SI Triggers array
- Increments the SI score by the value specified in the rule actions

If the SI score reaches a defined threshold, ClaimCenter creates an activity for the claim handler's supervisor to review this particular claim. You can control this threshold value, which defaults to 5, by setting the value of the script parameter `SpecialInvestigation_CreateActivityForSupervisorThreshold`.

On receipt of the activity, a claim supervisor reviews the contents of the SI **Details** subtab and the details of the claim. The supervisor can choose to escalate the claim to the Special Investigation Unit. If the claim already has an associated investigator, ClaimCenter sends the activity to that individual.

See also

- "SI rule evaluation sequence" on page 168
- "Using the Special Investigation rules" on page 169

SI rule evaluation sequence

The following sequence of events relating to special investigations occurs whenever you modify a claim or exposure.

1. ClaimCenter runs the Claim Preupdate or Exposure Preupdate rules.
2. ClaimCenter evaluates the SI-specific trigger rules to determine the following:
   - Are the conditions on the rule met?
   - Does the claim already contain a log of a particular trigger? If not already logged, ClaimCenter writes the trigger to the SI Triggers array and increments the SI score field on the claim in ClaimCenter.
3. ClaimCenter determines if the current SI score exceeds the threshold value set by script parameter `SpecialInvestigation_CreateActivityForSupervisorThreshold`.
   - If so, it creates a Special Investigations review activity for the supervisor of the claim owner.
   - If not, it starts the cycle again.
4. The claim supervisor reviews the claim and decides whether to escalate the claim to the Special Investigation Unit.
5. After ClaimCenter creates an SI escalation activity for the SI team, it assigns the activity to the SI investigator assigned to the claim if there is one. If a SI investigator is not currently assigned to this claim, then ClaimCenter does the following:
   a. It calls the SI assignment rules to assign the SI escalation activity to the SI group.
   b. It then uses round-robin assignment to assign the activity to a SI team member.

6. Finally, ClaimCenter adds the SI escalation activity owner to the claim (in the role of SIU Investigator).

See also

- "Claim Preupdate rules" on page 78.
- "Exposure Preupdate rules" on page 85

Using the Special Investigation rules

It is necessary to evaluate the SI trigger rules at the correct stage of the claim's life cycle. For example, it is not possible to truly evaluate at the first report of the claim whether the claim is missing the police report or whether there are no witnesses. It is possible that this information is only available after a certain number of days have passed.

To handle these situations, the SI rules break the claim life cycle into stages. Different ClaimCenter rule sets handle different stages in the claim life cycle. The following list describes the order in which the rule sets run.

Rule set	Rules	More information
Claim Preupdate Rules	CPU03000 - Set SIU Life Cycle	"Set the initial life cycle stage" on page 169
Claim Exception Rules	CER01000 - Setting SIU Life Cycle State	"Advance the life cycle stage" on page 170
Claim Preupdate Rules Exposure Preupdate Rules	CPU04100 - SIU Life Cycle Stage 1 CPU04200 - SIU Life Cycle Stage 2 EPU03000 - SI Triggers	"Evaluating SI triggers" on page 170
Business Rules Editor Default Group Activity Assignment Rules	CPU5000 - Create Supervisor Review Activity DGA01000 - SI Assign claim review activity to supervisor	"Escalating a claim to a supervisor" on page 172
Global Activity Assignment Rules	GAA01000 - SI - Assign Claim Review to Claim Owner's Group GAA02000 - SI - Assign SIU escalation activity to SIU group	"Assigning an activity to a SIU group" on page 172
Default Group Activity Assignment Rules	DGA02000 - SI Assign Escalation Activity to named SIU user DGA03000 - SI Default SI Escalation activity routing	"Assigning an activity to a SIU group" on page 172
Activity Postsetup Rules	APS01000 - SI - Add user in SIU Role to claim if necessary	"Adding an assigned user in a SIU role" on page 172

Set the initial life cycle stage

ClaimCenter uses claim property `SIULifeCycleState` to track a claim's stage.

Claim Preupdate rule CPU03000 - Set SIU Life Cycle sets the initial claim stage to 1 (`step1`) the first time that ClaimCenter runs the preupdate rules against a claim. To see this rule, navigate to the following location in the Guidewire Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ClaimPreupdate**

ClaimCenter uses the value of a claim's state to determine which SIU (Special Investigation Unit) rules to run against the claim.

```
CONDITION (claim : entity.Claim):
return claim.SIULifeCycleState == null and Claim.State !="draft"

ACTION (claim : entity.Claim, actions : gw.rules.Action):
claim.SIULifeCycleState="step1"
```

Advance the life cycle stage

A rule in the Exception rule set category advances the claim stage based on the number of days that have passed since claim inception. ClaimCenter automatically runs the claim exception rules on a daily basis. In the base configuration, these rules run at 2:00 a.m. server time.

Claim exception rule CER01000 - Setting SIU Life Cycle State calls the ClaimCenter library method `SetSIULifeCycleState`, which actually sets the state. To see this rule, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Exception** > **ClaimExceptionRules**

The following table lists the default configuration for the various stages. You can customize this process by setting new values in the library method.

Stage	Number of days since creation
1	Less than 5
2	More than 5 and less than 20
3	More than 20

ClaimCenter evaluates claims not touched today on the standard rule schedule in the Exception rule set category.

See also

- “Exception” on page 61

Evaluating SI triggers

Each time that ClaimCenter runs the Claim Preupdate rules for a claim or the Exposure Preupdate rules for an exposure, ClaimCenter evaluates SI (Special Investigation) rules. For example, suppose that ClaimCenter pushes a claim to stage 2 during the evening’s Exception process. Then, the next time that ClaimCenter evaluates the Claim Preupdate rules, it also evaluates the SI rules associated with stage 2.

See also

- “Claim Preupdate rules” on page 78
- “Exposure Preupdate rules” on page 85

Claim Preupdate SI rules for stage 1

Guidewire provides several sample stage 1 SI (Special Investigation) rules in the Claim Preupdate rule set. These rules add SIU life cycle points under various conditions. To work with these Gosu rules, navigate to the following location in the Guidewire Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ClaimPreupdate** > **CPU04000 - SI Triggers**

The following list describes the base configuration SI rules.

CPU04100 - SIU LIfe Cycle Stage 1

This rule determines if the SIU Life Cycle stage is `step1`. If so, ClaimCenter evaluates the Stage 1 child rules that follow.

CPU04110 - SI - Unreasonable delay in reporting loss

If the claimant filed the claim more than 30 days after the reported loss date, this SI trigger rule increases the trigger value by a point value of 1. You can change the default number of days, the point value, the trigger description, and the additional information.

CPU04120 - SI - Claim within 15 days of policy inception

If the claimant filed the claim within 15 days of the policy issuance, this SI trigger rule increases the trigger value by a default value of 1. You can change the default number of days, the point value, the trigger description, and the additional information.

CPU04130 - SI - Claim within 30 days of policy inception

If the claimant filed the claim within 30 days of the policy issuance, this SI trigger rule increases the trigger value by a default value of 1. You can change the default number of days, the point value, the trigger description, and the additional information.

CPU04140 - SI - Claim loss is theft related

If the claim loss is theft-related, this SI trigger rule increases the trigger value by a default value of 1. You can change the point value, the trigger description, and the additional information.

Claim Preupdate SI rules for stage 2

Guidewire provides several sample stage 2 SI (Special Investigation) rules in the Claim Preupdate rule set. These rules add SIU life cycle points under various conditions. To see these Gosu rules, navigate to the following location in the Guidewire Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ClaimPreupdate** > **CPU04000 - SI Triggers**

The following list describes the base configuration SI rules.

CPU04200 - SIU LIfe Cycle Stage 2

Determines if the SIU Life Cycle stage is `step2`. If so, ClaimCenter evaluates the child Stage 2 rules that follow.

CPU04210 - SI - No on-scene reports of incident

If the police report is not available for this claim, this SI trigger rule increases the trigger value by a point value of 1. You can change the point value, the trigger description, and the additional information.

CPU04220 - SI - Minor not on policy

Guidewire disables this rule in the base configuration. If the driver was a minor and not listed on the policy, this SI trigger rule increases the trigger value by a default value of 1. You can change the point value, the trigger description, and the additional information.

Exposure Preupdate SIU rules

ClaimCenter provides sample Exposure Preupdate SI (Special Investigation) rules that you can enable to run every time there is an update or validation of an exposure. To see these Gosu rules, navigate to the following location in the Guidewire Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ExposurePreupdate** > **EPU03000 - SI Triggers**

The following list describes the base configuration SI rules.

EPU03000 - SI Triggers

Basically, a folder to group the Exposure Preupdate SIU rules.

EPU03100 - SI - Claimant has no phone number listed

Guidewire disables this rule in the base configuration. If the claimant has no phone number, this SI trigger rule increases the trigger value by a point value of 1. You can change the point value, the trigger description, and the additional information.

This rule is not active in the base configuration.

EPU03200 - SI - Claimant has POBOX address

Guidewire disables this rule in the base configuration. If the claimant has a primary address that is a post office box, this SI trigger rule increases the trigger value by a default value of 5. You can change the point value, the trigger description, and the additional information.

This rule is not active in the base configuration.

Escalating a claim to a supervisor

In the base configuration, ClaimCenter provides the following business rule to create a supervisor review activity based on the SIU score:

CPU05000 - SI - Create Supervisor Review Activity

You can find this business rule in the ClaimCenter **Administration** screens at the following location:

Business Settings > **Business Rules** > **Activity Rules**

This rule is available as both a Gosu rule in Studio and as a business rule within ClaimCenter. However, in the base configuration, Guidewire disables Gosu Claim Preupdate rule CPU05000 as the ClaimCenter business rule handles this functionality.

To view the disabled Gosu rule in Guidewire Studio, navigate to the following location in the Studio **Project** window:

configuration > **config** > **Rule Sets** > **Preupdate** > **ClaimPreupdate** > **CPU05000- SI - Create Supervisor Review Activity**

The business rule compares the SIU total score for the claim to the script parameter `SpecialInvestigation_CreateActivityForSupervisorThreshold`, which in the base configuration is set to a value of 5.

If the rule creates the activity for the supervisor, this action triggers the following SIU activity assignment rules.

1. A global activity assignment rule, GAA01000 - SI - Assign Claim Review to Claim Owner's Group, assigns the claim review activity to the claim owner's group.
2. A default group activity assignment rule, DGA01000 - SI - Assign Claim Review Activity to Supervisor, assigns the review activity to the supervisor of the group.
3. The group supervisor then reviews the claim and determines whether it requires further action.

Note: You can set the value for a script parameter on the **Administration** tab in Guidewire ClaimCenter.

Assigning an activity to a SIU group

After the escalation of a claim to the reviewing supervisor, as described in "Escalating a claim to a supervisor" on page 172, the supervisor can review the claim. The supervisor can also escalate the claim to a Special Investigation group to investigate the claim further.

In the base cconfiguration, ClaimCenter provides the following assignment rules to assist in this process:

GAA02000 - SI Assign SIU escalation activity to SIU group

If there is a Special Investigator associated with the claim, assign the SI escalation activity to that user's group. Otherwise, in the base configuration, the assignment goes to the Western SIU group.

This rule is in the Global Activity Assignment rule set.

DGA02000 - SI Assign Escalation Activity to named SIU user

If there is a Special Investigator associated with the claim, assign the SI escalation activity to the person with that role.

This rule is in the Default Group Activity Assignment rules.

DGA03000 - Default SI Escalation activity routing

If the previous rule does not result in an assignment, use round-robin assignment to assign the SI escalation activity to a member of the SI group.

This rule is in the Default Group Activity Assignment rules.

Adding an assigned user in a SIU role

As described in "Assigning an activity to a SIU group" on page 172, the ClaimCenter assignment rules assign an escalation activity to a member of the Special Investigation team. After these assignment rules run, ClaimCenter runs an additional Activity Postsetup rule. This rule associates the user with the suspect claim.

APS01000 - SI - Add user in SIU Role to claim if necessary
Determine if the assigned user has the SIU Investigator role. If not, add this user to the claim in that role.

See also

- “Activity Postsetup rules” on page 69.s

chapter 11

Transaction Set validation

Insurance companies typically sell vehicle liability insurance as a package, with the insurer's liability limited to each person, each occurrence (incident), and total property damage. For example, a 100/300/50 package limits the maximum payout in an accident for which an insurer would be liable to the following:

- $100,000 in Bodily Injury (BI) coverage for each person
- $300,000 in Bodily Injury (BI) coverage for all persons
- $50,000 for all Property Damage (PD)

By using the ClaimCenter transaction validation rules and library functions, you can track these limits and raise alerts if a transaction exceeds the exposure or incident limit. You can set these rules either to block the transaction from going through or to allow the transaction to continue, but with warnings.

> **Note:** Guidewire provides the Transaction Set Validation rules as sample rules only. Customize these rules to meet your business logic as appropriate (for example, by modifying the reserve definition to meet your business needs).

Transaction Set validation rules

ClaimCenter provides sample transaction validation rules that block or warn the user from executing a financial transaction that exceeds the policy limits. There is another rule that is specific to PIP (Personal Injury Protection) that blocks the user from making payments that exceed the PIP aggregate limit.

To access the Transaction Set Validation Rules, navigate to the following location in the Guidewire Studio **Project** window:

configuration > **config** > **Rules Sets** > **Validation** > **TransactionSetValidationRules**

The sample Transaction Set Validation rules execute correctly only if both of the following are true:

- The created exposure is tied to the coverage, which is visible in the **Exposure** screen.
- The corresponding coverages are defined at the vehicle level on the policy's **Vehicles** screen.

ClaimCenter runs the validation rules whenever it updates a ClaimCenter business entity (`Claim`, `Exposure`, or `TransactionSet`, for example) and commits data to the database.

These sample rules calculate payments values using *claim cost*, which includes only costs directly associated with the claim, and not ancillary costs such as expenses. The sample rules also calculate all values by using the financial calculations `RemainingReserves` value. The rule rules obtain this value by calling the `getRemainingReserves` method:

```
var RemainingReservescache = gw.api.financials.FinancialsCalculationUtil.getRemainingReserves()
```

To work with a different type of reserve value, use a different getter method. For example, to access the `AvailableReserves` value, use the `getAvailableReserves` method:

```
var AvailableReservescache = gw.api.financials.FinancialsCalculationUtil.getAvailableReserves()
```

TXV01000 – Total payments not greater than exposure limit

ClaimCenter executes the TXV01000 rule whenever you attempt to create a new check.

```
transactionset.Subtype == "CheckSet" &&
      transactionset.new &&
      transactionset.ApprovalStatus == "unapproved"
```

Rule	Condition	Action
TXV01000	**1.** (New payment + Current Total Payments already made) > Exposure Limit	**1.** Error, block
	2. Above condition false *and* (New payment + Current Total Payments already made + Remaining Reserves) > Exposure Limit	**2.** Warn, but permit

Use rule TXV01000 to either block or permit with warnings a payment that would cause one of the following conditions:

- Making a new payment and adding it to the payments already made causes the total payments to exceed the Exposure limit. See "TXV01000 – Total payments not greater than exposure limit" on page 176.
- Making a new payment and adding it to the payments already made plus the remaining reserves causes the total payments to exceed the exposure limit. For this condition to be true, the previous condition must be false. Thus, this condition assumes that the remaining reserves is greater than the exposure limit. See "TXV01000 – Total payments not greater than exposure limit" on page 176.

The rule declares two variables, one for each listed condition, that determine the outcome of that condition.

Variable	Effect
`warning1`	`true` – warn, but permit; `false` – reject and block
`warning2`	`true` – warn, but permit; `false` – reject and block

Example 1: Payment Would Cause Total Payments to Exceed Exposure Limit

Suppose that you have created a claim with a Vehicle Damage exposure. The exposure limit is $10,000 with a reserve of $8,000. As you make payments (transactions 1 and 2), you deplete the reserve, but the total payout is still below both the exposure limit and the reserve limit. An attempted third payment now brings the payment total to $10,900, which is over the exposure limit of $10,000

Exposure Limit	Payment	Amount	Rule action
$10,000	1	$2,000	Permit the transaction as payment of $2,000 < Exposure Limit of $10,000
	2	$5,500	Permit the transaction as $2,000 + $5,500 < $10,000
	3	$3,400	Block the transaction as $2,000 + $5,500 + $3,400 > $10,000
	Total	$10,900	

Depending on your business needs, you might want to raise an alert if a payment causes the payment total to exceed a reserve or exposure limit. In some cases, you might want to block the transaction altogether. The sample rules in the Transaction Set Validation rules handle either of these situations easily.

Example 2: Payment Plus Remaining Reserves Would Cause Total Payments to Exceed Exposure Limit

Exposure Limit	Remaining Reserves	Payment	Amount	Rule action
$10,000	$8,000	1	$1,500	Permit the transaction as the payment of $1,500 + $8,000 (Remaining Reserves) < $10,000 (Exposure limit)
	$6, 500			
		2	$1,000	Permit the transaction as $1,000 + $1,500 + $6,500 < $10,000
	$5,500			
		3	$3,000	Warn the user as $1,500 + $1,000 + $3,000 + $5,500 > $10,000, but permit the user to execute the transaction if desired

See also

- “TransactionSet Validation rules” on page 117

TXV02000 – Check exposure limit when increasing reserves

ClaimCenter executes rule TXV02000 whenever you attempt to create a new reserve or, more specifically, if you attempt to create a new `TransactionSet` of type `ReserveSet`.

```
transactionset.Subtype == "ReserveSet" && transactionset.new
```

Rule	Condition	Action
TXV02000	(New reserve amount + Current Total Payments + Remaining Reserves) > Exposure Limit	Warn, but permit

You can use rule TXV02000 to either block (reject completely) or permit (with warnings) a reserve increase. For example, you can reject a reserve increase that would cause the sum of the combined current payments, the new reserve amount, and remaining reserve to exceed the exposure limit. To illustrate, suppose that the exposure limit is $10,000 and the initial reserve limit is $8,500, and you are attempting to make a payment of $1,000. The remaining reserve after this transaction equals $7,500. Increasing the reserve by $2,000 would create a combined total of $10,500 ($7,500 + $2,000 + $1,000), which exceeds the exposure limit.

Exposure Limit	Remaining Reserves	Payments	Rule Action
$10,000	$8,500	$1,000	Permit the transaction as $1,000 + $8,500 < $10,000, leaving a Remaining Reserves of $7,500
	$7,500		
	$7,500 + $2,000 = $9,500		Block (or permit with warnings) as attempting to increase the reserves by $2,000 to $9,500 fails ($9,500 + $1,000 > $10,000)

You set the outcome of rule TXV02000, to block or warn, by setting the variable `warning` in the rule.

Variable	Value
`warning`	`true` – warn, but permit; `false` – reject and block

TXV03000 – Total payments not greater than incident limit

ClaimCenter executes rule TVX03000 if you attempt to create a new check.

```
transactionset.Subtype == "CheckSet" &&
      transactionset.new &&
      transactionset.ApprovalStatus == "unapproved"
```

Rule	Condition	Action
TXV03000	**1.** (New payment + Current Total Payments already made for all exposures tied to the same coverage) > Incident Limit **2.** Above condition false *and* (New payment + Current Total Payments for all exposures tied to the same coverage + Remaining Reserves) > Incident Limit	**1.** Error, block **2.** Warn, but permit

This sample rule works in a similar manner to rule TXV01000, except that it applies to incident limits, not exposure limits.

Use rule TXV03000 to either block or permit, with warnings, a payment that would cause one of the listed conditions.

The rule declares two variables, one for each listed condition, that determine the outcome of that condition.

Variable	Value
`warning1`	`true` – warn, but permit; `false` – reject and block
`warning2`	`true` – warn, but permit; `false` – reject and block

Note that rule TXV03000 handles one vehicle per incident only.

See also

- "TXV01000 – Total payments not greater than exposure limit" on page 176

TXV04000 – Check incident limit when increasing reserves

ClaimCenter executes rule TXV04000 whenever you attempt to create a new reserve.

```
transactionset.Subtype =="ReserveSet" && transactionset.new
```

Rule	Condition	Action
TXV04000	(New reserve amount + Current Total Payments for all exposures tied to the same coverage + Remaining Reserves) > Incident Limit	Warn, but permit

This sample rule works in a similar manner to TXV02000, except that it applies to incident limits, not exposure limits.

You set the outcome of rule TXV04000, either block or warn, by setting the variable `warning` in the rule.

Variable	Value
`warning`	`true` – warn, but permit; `false` – reject and block

Note that rule TXV04000 handles one vehicle per incident only.

See also

- "TXV02000 – Check exposure limit when increasing reserves" on page 177

TXV05000 – PIP

ClaimCenter executes rule TVX05000 whenever it validates a check.

```
TransactionSet.Subtype == "checkset"
```

Rule	Condition	Action
TXV05000	Payments on PIP coverages exceed one of the four defined PIP Aggregate Limits	Warn, but permit

This sample rule determines whether adding this new amount to the payments already made for this exposure would cause the total payments to exceed one of the following limits.

Limit	Description
`PIPNonmedAgg`	Non-medical aggregate limit
`PIPReplaceAgg`	Lost wages and replacement services aggregate limit
`PIPPersonAgg`	PIP per person aggregate limit
`PIPClaimAgg`	PIP per claim aggregate limit

Note: ClaimCenter calculates these aggregate limits in Claim enhancement method `sumForPIPAgg`.

Gosu code within rule TXV05000 looks at each limit and determines whether the new payment total exceeds a specific aggregate limit. If so, the rule permits the transaction, but raises a warning within ClaimCenter. The user must then explicitly cancel or continue the transaction. For example, if the new payment causes the payment total to exceed the `PIPPersonAgg` limit, it raises the following warning

```
TransactionSet.reject( null, null, "external", "The PIP Per Person Aggregate Limit ($"
  + (each.Exposure.Coverage as VehicleCoverage).PersonAggLimit
  + ") has been exceeded for this coverage, " + each.Exposure.Coverage.Type
  + ". Do you still want to continue? )
```

At the same time, the rule sets a `Boolean` flag to indicate that the sum of all payments, including this new one, now exceeds one of the aggregate limits. There are four of these flags, corresponding to the four aggregate limits:

- `PIPNonmedAggLimitReached`
- `PIPReplaceAggLimitReached`
- `PIPPersonAggLimitReached`
- `PIPClaimAggLimitReached`

Other Gosu rules can access these values as properties on the Exposure entity. For example, a rule in the Exposure Preupdate rule set could use the following code to determine if the aggregate limit for each person has been reached for this exposure:

```
var PersonAggLimitReached = exposure.PIPPersonAggLimitReached
```

See also

- “sumForPIPAgg enhancement methods” on page 181
- “Exposure Preupdate rules” on page 85

TXV06000 – Reserve threshold

In the base configuration, the TXV06000 rule imposes a threshold of 1,000,000 in the claim currency.

Note: The rule provides a hard-coded limit of 1,000,000 in claim currency for creating a single reserve.

TXV08000 – Check aggregate limits

ClaimCenter executes the TXV08000 rule to notify users whenever it determines that an amount exceeds an aggregate limit, or it deems the value invalid. In the base configuration, ClaimCenter displays a warning in these cases, but allows the transaction to go through.

Rule	Condition	Action
TXV08000	Transaction results in an aggregate limit being exceeded or marked invalid	Warn, but permit

It is possible to mark aggregate limits as invalid in the following ways:

- Whenever a staging table load brings in new transactions that impact an existing or newly loaded aggregate limit. In this case, ClaimCenter marks the aggregate limits on the corresponding policy period as invalid.
- Whenever a system administrator requests a recalculation of all aggregate limit values and the `AggLimitCalc` batch process, which runs in the background, is not complete. In this scenario, ClaimCenter initially marks all aggregate limits as invalid. As the rebuild process progresses through the database, ClaimCenter marks each aggregate limit as valid, one by one.

TXV13000 – Warning if adverse party pays non-subrogation recovery

The TXV13000 rule validates transactions. It prevents ClaimCenter from saving a new transaction if both of the following are true:

- The transaction involves subrogation.
- Any of its adverse party contacts has made a recovery payment that is not for subrogation.

Rule	Condition	Action
TXV13000	Check's whether the following conditions are all true: • Is the transaction new? • Is the transaction a recovery? • Does the transaction involve subrogation.	Error, block

If all the listed conditions are true, ClaimCenter collects all the adverse party contacts and determines for each of these contacts if it has the subrogation recovery category. If any of these contacts does not have this category, ClaimCenter blocks the transaction and sends a message to the user.

TXV17000 – Reserve lines compatible with service requests

ClaimCenter executes the TXV17000 rule whenever it validates a check. It validates all checks linked to a service request invoice to determine if the check's set of reserve lines is compatible with at least one of the service request invoices.

It is compatible if one of the following conditions is met:

- One of the associated service requests is claim-level.
- Each exposure-level payment shares an incident with one of the associated service requests.

Rule	Condition	Action
TXV17000	Check's reserve line is compatible with one of the service request's invoices.	Error, block

TXV18000 – Currencies compatible with service requests

The TXV18000 rule validates currencies associated with a service request invoice.

Rule	Condition	Action
TXV18000	Check and service request invoice currencies match.	Error, block

sumForPIPAgg enhancement methods

ClaimCenter Gosu rules use the `sumForPIPAgg` enhancement methods on the `Claim` entity in working with PIP-related transactions. To access the `sumforPIPAgg` enhancement class, navigate to the following location in the Guidewire Studio **Project** window:

configuration > **gsrc** > **libraries**

Several of the `sumForPIPAgg` enhancement methods, in turn, contain method calls to other enhancement methods. In the base configuration, these enhancement methods calculate the aggregates sums of the following coverage types per incident:

Aggregate method	Sums these coverage types
`sumForPIPClaimAgg`	• PIP Medical • PIP Rehabilitation • PIP Lost Wages • PIP Funeral • PIP Death • PIP Extraordinary Medical
`sumForPIPNonmedAgg`	• PIP Rehabiliation • PIP Lost Wages • PIP Funeral • PIP Death
`sumForPIPPersonAgg`	• PIP Medical • PIP Rehabiliation • PIP Lost Wages • PIP Funeral • PIP Death • PIP Extraordinary Medical
`sumForPIPReplaceAgg`	PIP Lost Wages

The TXV05000 sample rule uses these summing methods as methods on the Exposure entity. For example:

```
each.Exposure.Claim.sumForPIPNonmedAgg( Claimant ) > (each.Exposure.Coverage as
VehicleCoverage).NonmedAggLimit)
```

See also

- “TXV05000 – PIP” on page 178

chapter 12

Working with queues

This topic describes the ClaimCenter activity queues. It also provides several examples of how to use an activity queue.

About activity queues

An *activity queue* is a list of activities that are ready to be chosen and performed by users of ClaimCenter. An activity must have an assigned owner before it is possible to complete or skip the activity.

The following topics provide examples of how to use queues in Guidewire ClaimCenter:

- "Example: Using a queue to handle FNOL claims" on page 183
- "Example: Using a queue to void/stop checks" on page 187

Assignment review activities

ClaimCenter does not route activities of type *assignment review* to a queue, even if you explicitly call for this behavior. Therefore, these types of actives do not show in the **Queue** tab in ClaimCenter. Instead, ClaimCenter displays these activities in the **Pending Assignments** screen.

See also

- *Application Guide*

Example: Using a queue to handle FNOL claims

The following example shows how to use a queue to handle FNOL claims. If a claim enters ClaimCenter as an FNOL (First Notification of Loss), it triggers the ClaimCenter Global Assignment rules. However, in this example, the assignment rules do not assign the claim. Instead, these rules generate review activities, which ClaimCenter then places in a queue for response by a team of people who manually assign incoming claims.

ClaimCenter associates every queue with a single group, or potentially with a group and its subgroups. ClaimCenter automatically creates a default FNOL queue for each group in the hierarchy. This queue is not visible to the subgroups, by default.

This example performs the following sequence of actions, using a specific Gosu rule at each step to perform the necessary functionality.

Action	More information
Assign the incoming FNOL claim to an intake group to make an assignment decision on that claim.	"Assign a claim to an intake group" on page 184
Assign the claim to the group's supervisor. This "parks" the claim until the final assignment determination.	"Assign a claim to a group supervisor" on page 185
Create the FNOL review activity.	"Create a FNOL review activity" on page 185
Assign the FNOL review activity to the intake group designated in the first assignment rule.	"Assign a FNOL review activity to an intake group" on page 186
Place the review activity on the FNOL queue of the designated intake group, from which members of the group then pull a review activity	"Assign a FNOL review activity to a queue" on page 186

Assign a claim to an intake group

Before you begin

Before proceeding, review "Example: Using a queue to handle FNOL claims" on page 183.

About this task

Global Claim Assignment rule ASGCDefault - Assign to Claim Intake Units assigns an incoming claim to a specific group or groups. This group consists of users who make the assignment decisions. This example uses two groups:

- FNOL Review Group - WC
- FNOL Review Group - GL

There are no conditions for this rule, so the rule fires for every claim that comes into ClaimCenter.

Procedure

1. In the ClaimCenter **Project** window, navigate to the following location:

   configuration > **config** > **Rule Sets** > **Assignment** > **GlobalClaimAssingmentRules**
2. Create the following new rule:
   ASGCDefault - Assign to Claim Intake Units
3. Ensure that this rule is the first rule in the rule hierarchy.
4. Populate the rule with the following code:

```
CONDITION (claim : entity.Claim) :
  return true

ACTION (claim : entity.Claim, actions : gw.rules.Action) :
  var FNOLreview : Group //Some FNOL review group

  if (claim.LossType == TC_WC ) {
    if ( claim.CurrentAssignment.assignGroup( FNOLreview ) ) {//Assigned to FNOL Review Group - WC
      actions.exit() }
  } else {//Assigned to FNOL Review Group - GL
    if (claim.CurrentAssignment.assignGroup( FNOLreview ) ) {
      actions.exit()
    }
  }
```

What to do next

After completing this procedure, proceed to "Assign a claim to a group supervisor" on page 185.

Assign a claim to a group supervisor

Before you begin

Before proceeding, complete “Assign a claim to an intake group” on page 184.

About this task

Default Group Claim Assignment rule ASDGCDefault - Assign to Claim Group Supervisor assigns the claim to the supervisor of the group. There are no conditions for this rule, so the rule fires for every claim that comes into ClaimCenter.

Procedure

1. In the ClaimCenter **Project** window, navigate to the following location:

   configuration > **config** > **Rule Sets** > **Assignment** > **DefaultGroupClaimAssignmentRules**
2. Create the following new rule:
   ASDGCDefault - Assign to Claim Group Supervisor
3. Ensure that this rule is the first rule in the rule hierarchy.
4. Populate the rule with the following code:

```
CONDITION (claim : entity.Claim)
  return true

 ACTION (claim : entity.Claim, actions : gw.rules.Action)
  claim.CurrentAssignment.assign(claim.AssignedGroup, claim.AssignedGroup.Supervisor)
```

What to do next

After completing this procedure, proceed to “Create a FNOL review activity” on page 185.

Create a FNOL review activity

Before you begin

Before proceeding, complete “Assign a claim to a group supervisor” on page 185.

About this task

Claim Workplan rule WPWC03 - FNOL Review creates the review activity, which ClaimCenter then places in the FNOL queue. This activity tells the user to review and assign the claim. There are no conditions for this rule, so the rule fires for every claim that comes into ClaimCenter.

Procedure

1. Create an activity pattern with the code `fnol_review`. In the ClaimCenter **Administration** tab, navigate to the **Business Settings** > **Activity Patterns** page. For information on how to use and create activity patterns, see the *Application Guide*.
2. In the ClaimCenter **Project** window, navigate to the following location:

   configuration > **config** > **Rule Sets** > **Workplan** > **ClaimWorkplan**
3. Create the following new rule:
   WPWC03 - FNOL Review
4. Ensure that this rule is the last rule in the rule hierarchy.
5. Populate the rule with the following code:

```
CONDITION (claim: entity.Claim) :
  return true
```

```
ACTION (claim : entity.Claim, actions : gw.rules.Action) :
 claim.createActivityFromPattern( null,
       ActivityPattern.finder.getActivityPatternByCode( "fnol_review" ) )
```

What to do next

After completing this procedure, proceed to “Assign a FNOL review activity to an intake group” on page 186.

Assign a FNOL review activity to an intake group

Before you begin

Before proceeding, complete “Create a FNOL review activity” on page 185.

About this task

Global Activity Assignment rule ASGA30 - FNOL Review Assignments assigns the activity to the group that has already been assigned to the claim. The condition on the rule ensures that the rule assigns only this specific type of activity. Otherwise, ClaimCenter would assign all activities that trigger this rule to the queue. Notice that the activity assignment happens as part of the `if` condition, which returns `true` if successful.

Procedure

1. In the ClaimCenter **Project** window, navigate to the following location:

   configuration > **config** > **Rule Sets** > **Assignment** > **GlobalActivityAssignmentRules**

2. Create the following new rule:
   ASGA30 - FNOL Review Assignments
3. Ensure that this rule is the first rule in the rule hierarchy.
4. Populate the rule with the following code:

```
CONDITION (activity : entity.Activity, actions : gw.rules.Action) :
  return activity.ActivityPattern.Code == "fnol_review"

 ACTION (activity : entity.Activity, actions : gw.rules.Action) :
  if ( activity.CurrentAssignment.assignGroup( activity.claim.AssignedGroup ) ) {
    actions.exit()
  }
```

What to do next

After completing this procedure, proceed to “Assign a FNOL review activity to a queue” on page 186.

Assign a FNOL review activity to a queue

Before you begin

Before proceeding, complete “Assign a FNOL review activity to an intake group” on page 186.

About this task

Default Group Activity Assignment rule ASDGA01 - Assign New Claim Activity to Queue adds the review activity to the queue. This rule condition also specifies the correct activity type.

Note: ClaimCenter defines the default FNOL queue as `AssignableQueues[0]`.

Procedure

1. In the ClaimCenter **Project** window, navigate to the following location:

   configuration > **config** > **Rule Sets** > **Assignment** > **DefaultGroupActivityAssignmentRules**

2. Create the following new rule:
   ASDGA01 - Assign New Claim Activity to Queue
3. Ensure that this rule is the first rule in the rule hierarchy.
4. Populate the rule with the following code:

```
CONDITION (activity : entity.Activity) :
  return activity.ActivityPattern.Code == "fnol_review"

 ACTION (activity : entity.Activity, actions : gw.rules.Actions) :
  activity.CurrentAssignment.assignActivityToQueue( activity.AssignedGroup.AssignableQueues[0],
        activity.AssignedGroup)
  actions.exit()
```

Example: Using a queue to void/stop checks

You can configure ClaimCenter to use a queue to handle Void/Stop Payment activity requests. In this scenario, ClaimCenter does not allow random adjusters to void or stop checks. Instead, individual adjusters create an activity requesting a void/stop of a check. The activity enters a queue from which a member of the group of accountants takes the activity and completes it.

This example uses two rules that assign an activity that an end-user created manually. Therefore, there is no need for claim assignment or workplan rules, as there is in “Example: Using a queue to handle FNOL claims” on page 183. All that is necessary are the two activity assignment rules. You must, however, first create a Stop/Void queue if one does not exist.

This example performs the following sequence of actions.

Step action	More information
Create the void/stop queue.	“Create a stop/void queue” on page 187
The first rule assigns an activity of type `void_or_stop_check` to a particular group, the Accountants group.	“Assign an activity to the appropriate group” on page 188
The second rule adds the activity to the void/stop queue.	“Add an activity to a queue” on page 188

Create a stop/void queue

Before you begin

Before proceeding, review “Example: Using a queue to void/stop checks” on page 187.

About this task

ClaimCenter associates each queue with a single group, or potentially with a group and its subgroups. Therefore, the first step in this example is to associate the group that is responsible for voiding/stopping payments with the void/stop queue. If a void/stop queue does not exist, you must create one. If the group responsible for voiding/stopping payments does not exist, you must define it first as well.

If the appropriate queue does not exist, use the ClaimCenter**Administration** tab to create one.

Procedure

1. Log into Guidewire ClaimCenter using an administrative account.
2. Select the **Administration** tab.
3. In the left-hand navigation pane, select a specific group in the group hierarchy tree.
   This example uses the Accountants group to handle the voiding and stopping of checks. The members of this group are the only ones assigned the proper role to void/stop payments.

4. Then, select the **Queues** tab for that group.
5. Click **Edit** and add the queue.

What to do next

After completing this procedure, proceed to “Assign an activity to the appropriate group” on page 188.

Assign an activity to the appropriate group

Before you begin

Before proceeding, complete “Create a stop/void queue” on page 187.

About this task

In this second step of the example of using a queue to void or stop checks, you create a rule that assigns an activity if appropriate. Global Activity Assignment rule ASGA40 - Void/Stop Payments checks to see if the activity is of type `void_or_stop_check`. If it is, the rule assigns the activity to the Accounts group.

Procedure

1. In the ClaimCenter **Project** window, navigate to the following location:

   configuration > **config** > **Rule Sets** > **Assignment** > **GlobalActivityAssignmentRules**
2. Create the following new rule:
   ASDGA01 - Assign New Claim Activity to Queue
3. Ensure that this rule is the first rule in the rule hierarchy.
4. Populate the rule with the following code:

```
CONDITION (activity : entity.Activity) :
  return activity.ActivityPattern.Code == "void_or_stop_check"

ACTION (activity : entity.Activity, actions : gw.rules.Actions) :
  if (activity.CurrentAssignment.assignGroup( Group( "cc:96" /* Accountants */ ) ) ) {
    actions.exit()
  }
```

   This rule example uses a place holder group for demonstration purposes.

What to do next

After completing this procedure, proceed to “Add an activity to a queue” on page 188.

Add an activity to a queue

Before you begin

Before proceeding, complete “Assign an activity to the appropriate group” on page 188.

About this task

In this third step of the example of using a queue to void or stop checks, you create a rule that adds an activity to a queue. Default Group Activity Assignment rule ASDGA40 - Assign Void/Stop Activity to Queue adds the activity to the queue. The rule condition requires that the activity be of the correct activity type.

Note: The rule refers to `AssignableQueues[1]` because this queue is presumably the first new queue added to this group. If you have multiple queues on a group, you need to choose the correct one. `AssignableQueues[0]` refers to the default FNOL queue that ClaimCenter creates whenever you create a new group.

Procedure

1. In the ClaimCenter **Project** window, navigate to the following location:

   configuration > **config** > **Rule Sets** > **Assignment** > **DefaultGroupActivityAssignmentRules**

2. Create the following new rule:
   ASDGA40 - Assign Void/Stop Activity to Queue
3. Ensure that this rule is the first rule in the rule hierarchy.
4. Populate the rule with the following code:

```
CONDITION (activity : entity.Activity) :
  return activity.ActivityPattern.Code == "void_or_stop_check"

ACTION  (activity : entity.Activity, actions : gw.rules.Action) :
  activity.CurrentAssignment.assignActivityToQueue( activity.AssignedGroup.AssignableQueues[1],
        activity.AssignedGroup)
  actions.exit()
```

What to do next

See also

- “Assign a FNOL review activity to a queue” on page 186.