Guidewire training materials contain Guidewire proprietary information that is subject to confidentiality and non-disclosure agreements. You agree to use the information in this manual solely for the purpose of training to implement Guidewire software solutions. You also agree not to disclose the information in this manual to third parties or copy this manual without prior written consent from Guidewire. Guidewire training may be given only by Guidewire employees or certified Guidewire partners under the appropriate agreement with Guidewire.

Lesson objectives

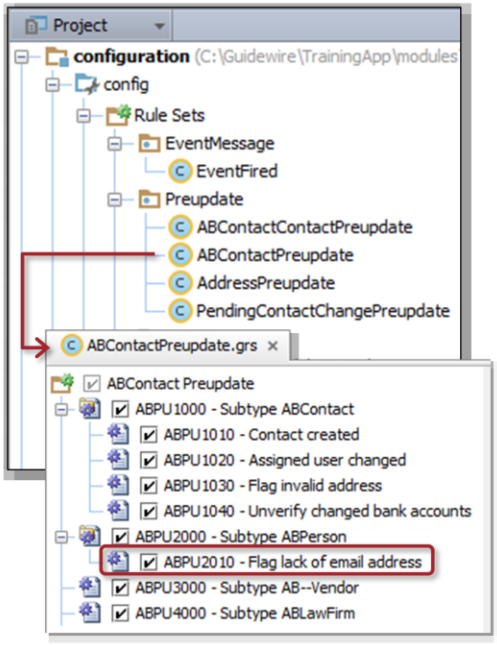
- By the end of this lesson, you should be able to:
  - Explain the functionality of business rules
  - Describe the Gosu techniques unique to business rules
  - Write business rules
  - Use Studio debugger to debug business rules

This lesson uses the notes section for additional explanation and information.
To view the notes in PowerPoint, select View → Normal or View → Notes Page.
When printing notes, select Note Pages and Print hidden slides.

Lesson outline

- Business rules overview
- Rules-specific Gosu
- Working with rules
- Debugging rules

Business rules



- A **business rule** is performs an action based on a condition for a given entity when a specific event occurs to an instance of that entity
  - Example: ABPU2010 – Flag lack of email address
- Rules set organize rules
  - Example: ABContactPreupdate
- Rule set categories organize rule sets
  - Example: Preupdate

A rule is a single decision in the following form of testing a condition and then taking an action: if {some conditions} then {take some action}. A business rule is Gosu code that accomplishes some task for given entity, and executes when specific event occurs to instance of that entity

A rule set combines many individual rules into a useful set to consider as a group.

A rule set category organizes rule sets.

In TrainingApp, Preupdate is a rule set category that organizes various rule sets. The ABContactPreupdate rule set organizes various rules for the ABContact entity. When an update occurs to an ABContact, a Guidewire application executes these rules, such as:

- Creating history entry to record change of assigned user
- Setting a company's Primary Contact to null if the original Primary Contact no longer works for the company

Examples in Guidewire applications

BillingCenter

- Account validation rules
  - Verify that account is valid before committing it to database
  - Triggered when account is created or modified

ClaimCenter

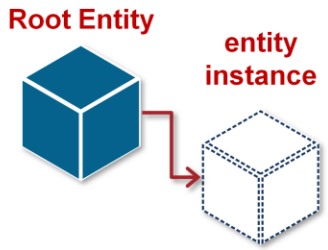
- Activity escalation rules
  - Escalate activity that is open for too long
  - Triggered when activity is open past its escalation date

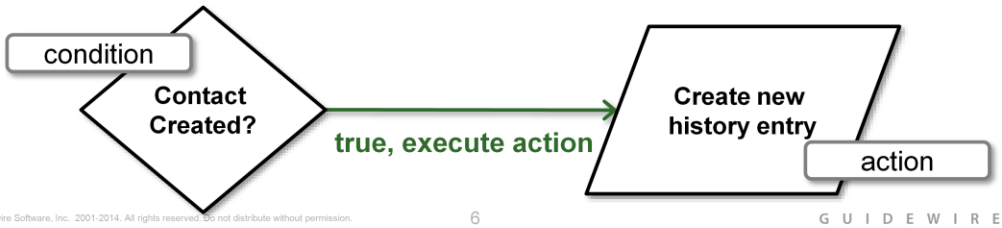
PolicyCenter

- Claim assignment rules
  - Assign given claim to a group and user
  - Triggered when claim is created or needs reassignment

Business rule anatomy

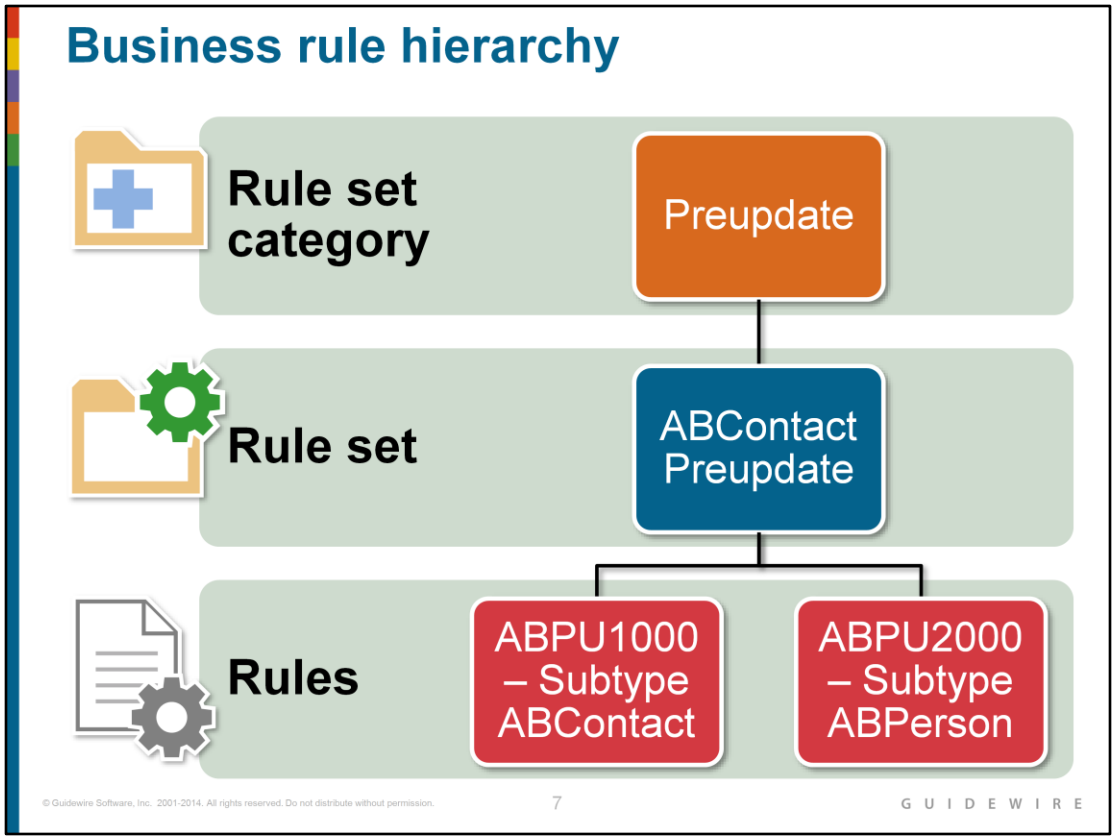
- Root entity instance
  - Object instance of root entity
- Name
  - Unique and follows naming convention
  - Example: ABPU1010 – Contact Created
- Condition
  - Expression that resolves to true or false
- Action
  - If condition is true, Guidewire executes action





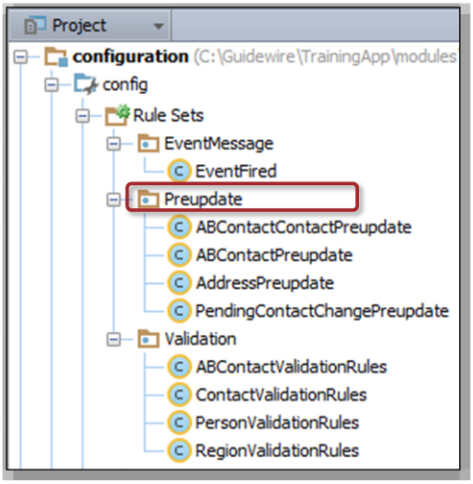
Setting a rule condition to "true" means that the action will always performed whenever the rule is evaluated.

Notice that there is no "if" in the condition. This is assumed. The only valid condition is a boolean expression.



Guidewire applications organize rules into a hierarchy. Rule set categories are at the top of hierarchy. For a given rule set category there are one or more rule sets. For each rule set there is one or more rules. In the slide example, Preupdate is the rule set category. The child rule set is ABContact Preupdate. ABPU1000 and ABPU2000 are child rules of ABContact Preupdate. The example is specific to TrainingApp.

Hierarchy: rule set category (1)



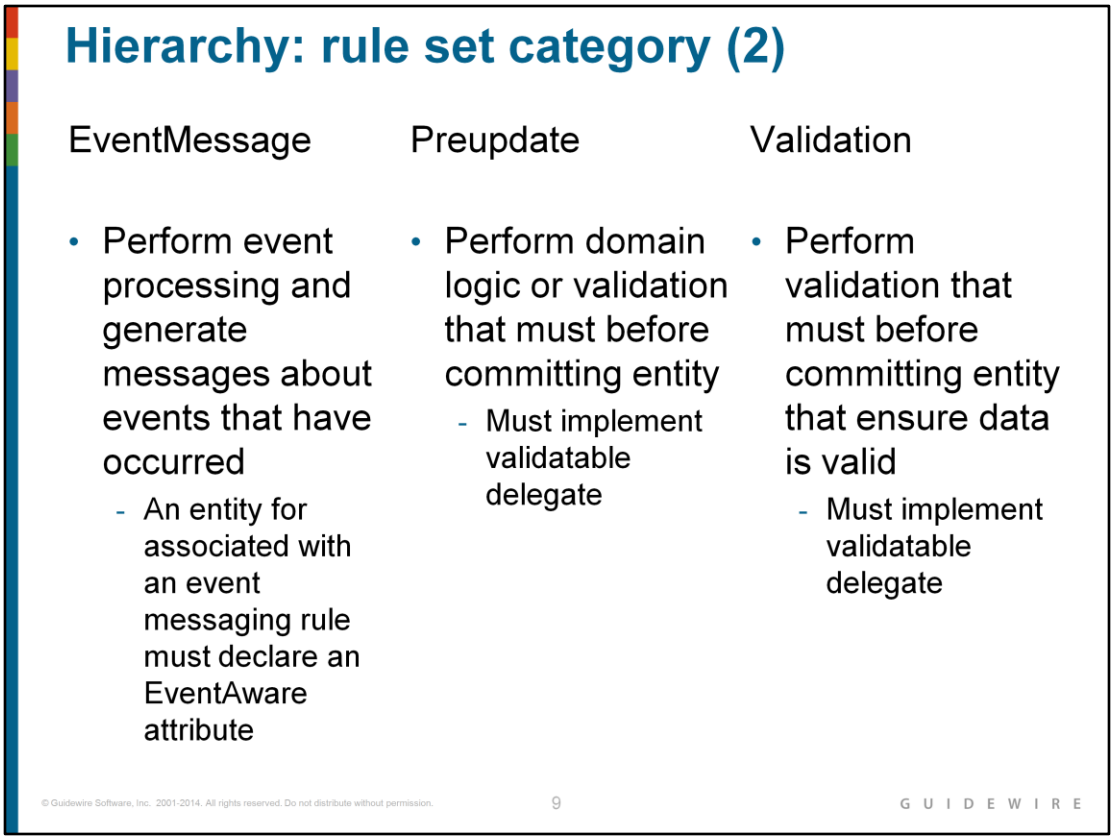
- A **rule set category** is a collection of rules sets that have common high-level business purpose
- Examples:
  - Event messaging
  - Preupdate
  - Validation

In Project View, the top level node is Rule Sets, a misnomer. The physical folder is …\config\rules\.

GUIDEWIRE

You can create new rule set categories. This is recommended only in the uncommon situation of an implementation that needs a new type of rule set category that is completely unlike any existing rule set category, however. To create a new rule set category, right-click the Rule Sets node and select New > Rule Set Category. You can also create a rule set category as a child node of an existing rule set category if there is a need to subgroup rule sets.

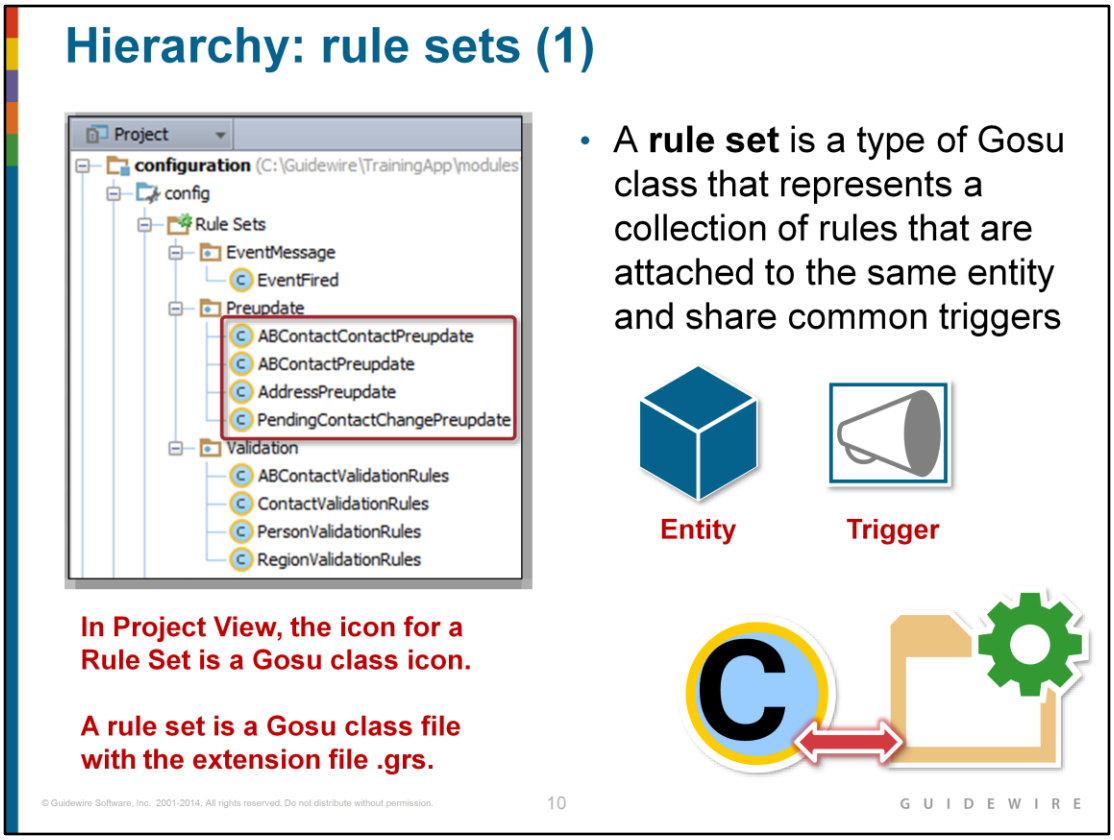
Guidewire applications contain the triggers for event messaging, preupdate, and validation. These triggers are associated with their respective rule set categories. This is true of all Guidewire applications and not just TrainingApp. If you create a new rule set category, you must also write the code necessary to trigger the rule sets in the rule set category. For more information, consult the Rules Guide in documentation.



In the base configuration for a Guidewire application, there is the Event Fired rule set in the Event Message rule set category. The rules in this rule set perform event processing and generate messages about events that have occurred. An entity for associated with an event messaging rule must declare an EventAware attribute. In addition, an active message destination must subscribe to the entity event. Guidewire applications call the Event Fired rules if an entity involved in a bundle commit triggers an event for which a message destination has subscribed.

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 can trigger a preupdate rule.

For an entity to trigger the validation rules, it must implement the Validatable delegate. This is true for an entity in the base configuration or for a custom entity that you create. Guidewire applications runs the validation and preupdate rules:
a) when an instance of a validatable entity is created, modified, or retired
b) when a subentity of a validatable entity is created, modified, or retired and the validatable entity is connected to the subentity through a triggersValidation="true" attribute

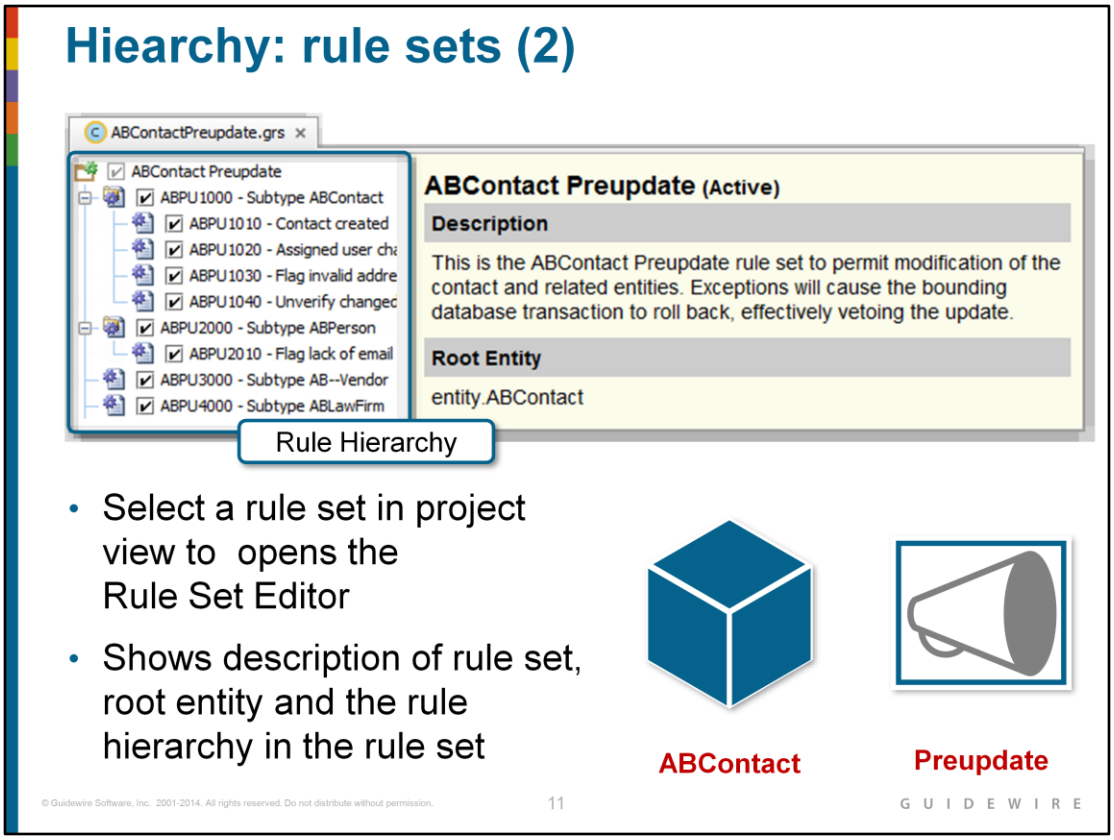


You can think of a rule set as a logical grouping of rules that are specific to a business function. You organize the rules in a rule set into a hierarchy that fits your business model. In other words, a rule set combines many individual rules into a useful set to consider as a group. One way that Guidewire helps you define the organization of rules is with a root entity and common triggers. A rule set is a collection of rules that share the same root entity and share common triggers. You write code for new rule set categories to define the triggers associated with a rule set. If you create rule sets for an existing rule set category (Event Messaging, Pre-update, and Validation), you do not need write trigger code.

Execution of rules always occurs at the rule set level. Unless the rules engine encounters an exit() command, all rules in the rule set are executed.

TrainingApp is built by heavily customizing an instance of ContactManager, and therefore has the same rule sets as ContactManager. These include:

- The Event Fired rule set, which is associated with MessageContext and triggers when a MessageContext object fires an integration event.
- The ABContact Validation rule set, which is associated with ABContact and triggers when an ABContact object is created or modified. ABContact Validation rules are designed to identify invalid changes to ABContacts. ABContact Pre-update rules are designed to take actions required because of changes to an ABContact.
- The Region Validation rule set, which is associated with Region and triggers when a Region object is created or modified.

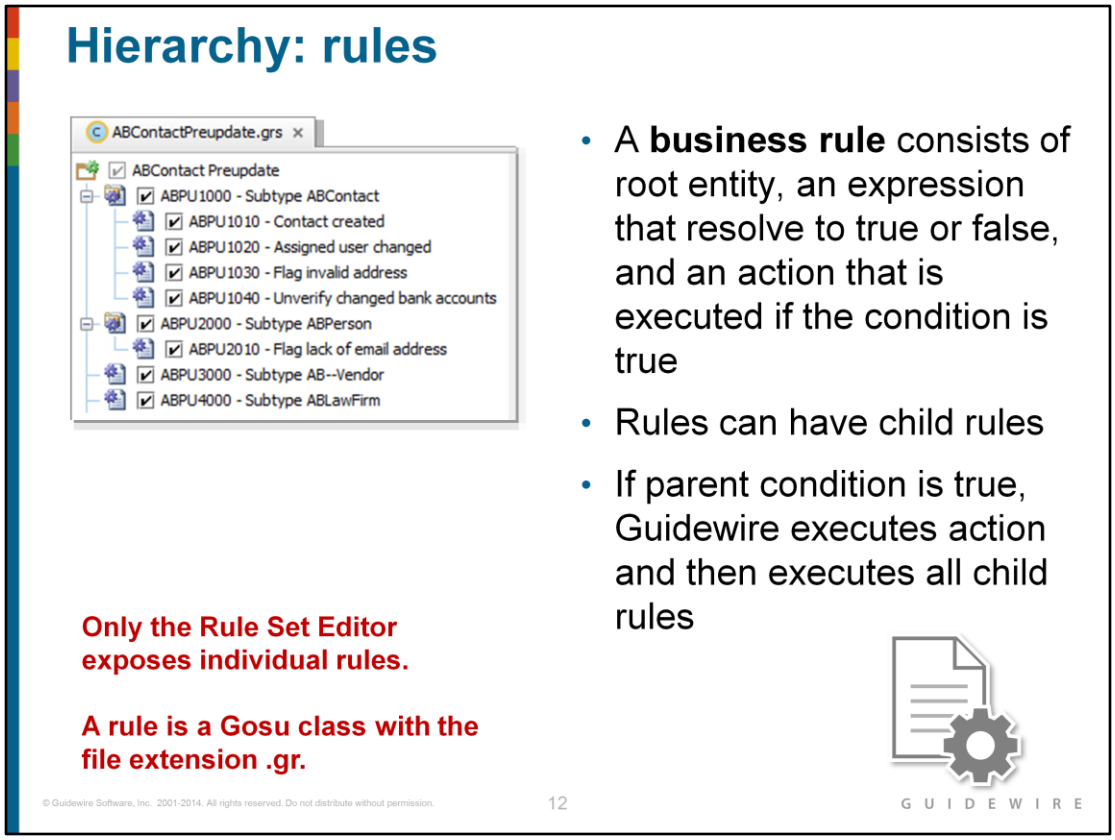


In the slide example, the root entity of the ABContact Preupdate rule set is an ABContact entity and a common trigger is already associated with the rule set category.

You can create new rule sets. To create a new rule set, right-click the appropriate rule set category node, select New → Rule Set, and associate an entity type.

For custom entities, you can create new pre-update and validation rule sets that are triggered whenever an instance of that entity is created, changed, removed, or retired. If an entity has an associated child entity that is set to trigger validation, then changes to a child object trigger validation and pre-update rules for the parent object.

You can also create non-pre-update, non-validation rule sets. These rule sets will not be triggered automatically, however. You must also write the code necessary to trigger the rule sets. For more information on creating new pre-update, validation, or other rule sets, consult the Rules Guide.



If a rule has child rules, but the parent rule condition is false, neither the parent action nor the child rules are executed. In the slide example, the active (checked check box) rules are executed in the following hierarchy:

- ABPU1000 gathers together the rules relevant for all contacts. The child rules are executed only if this condition is true.
  - ABPU1010 creates a new history event when a contact is created.
  - ABPU1020 executes the necessary actions when a contact's assigned user changes (such as creating a note to record the change).
  - ABPU1030 executes the necessary actions when a flagged contact is unflagged (such as creating a note to record who unflagged the contact).
- ABPU2000 checks to see if the contact is an ABPerson. The child rules are executed only if this condition is true.
  - ABPU2010 flags any person who does not have an email address.
  - ABPU2020 sets a company's Primary Contact to null if the original Primary Contact no longer works for the company.
- ABPU3000 checks to see if the contact is an ABPersonVendor or ABCompanyVendor.
- ABPU3010 (not shown) sets the preferred vendor flag if the contact's score is not null.

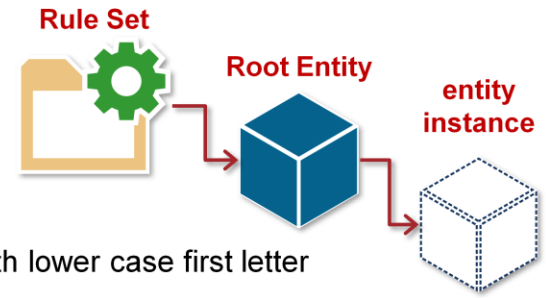
You can create new business rules. By providing an appropriate rule condition, you can also configure whether a given rule's action is executed or not.

Lesson outline

- Business rules overview
- Rules-specific Gosu
- Working with rules
- Debugging rules

Root entity object instance

- Rule set associated with a root entity
- Every rule in rule set has access to entity instance that triggers the rule
  - Object is root entity name with lower case first letter
- Example
  - aBContact is of type ABContact



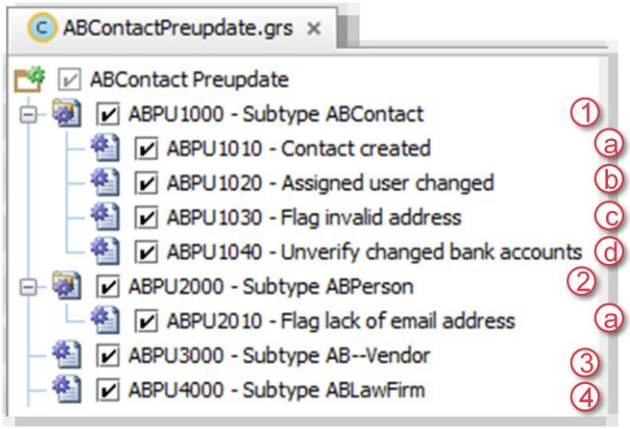

```
USES:
CONDITION (aBContact  :  entity.ABContact):
return true
ACTION (aBContact  :  entity.ABContact, actions : gw.rules.Action):
// This rule is used simply to gather all ABContact rules together.
END
```

Entity instance

Root Entity

The root entity identifies the entity with which the rule set is associated. Every rule has access to the object that triggered the rule set. The object always has the same name as the root entity. For example, all ABContact preupdate rules have access to an object named "ABContact", which is the ABContact that has just been created or modified and which triggered the pre-update rule set.

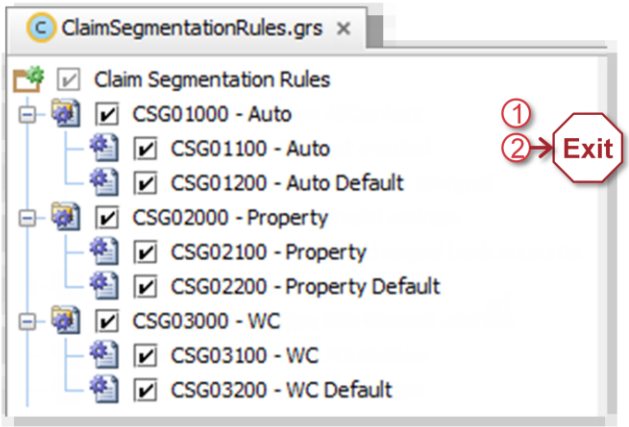
Rule execution: execute all



- When triggering occurs, Guidewire executes **ALL** rules in rule set
  - Executed in order of hierarchy
  - If parent rule condition is true, parent action executed first, then child rules executed

- Example: Validation Rules in ContactManager
  - All validation conditions are checked

Rule execution: exit after first action

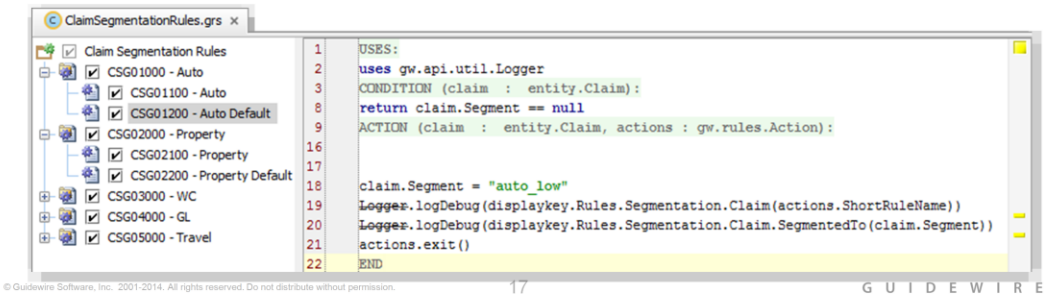


- When triggering occurs, Guidewire executes rules in rule set until **exit**
  - Executed in order of hierarchy
  - If parent rule condition is true, parent action executed first, then child rules executed

- Example: Claim Segmentation Rules in ClaimCenter
  - Segment the claim in the manner of the exposure and then exit rule set

Exiting a rule set

- Exit after first action rules
- Syntax: `actions.exit()`
- Example of ClaimCenter Claim Segmentation rule set rules
  - Condition:
    - If claim.Segement is null
  - Action:
    - Assign "auto_low" to Segement
    - Exit rule set



The actions object actually has multiple exit methods, though exit commands almost always need to exit the entire rule set, and therefore exit() is used almost exclusively. The available exit methods include:

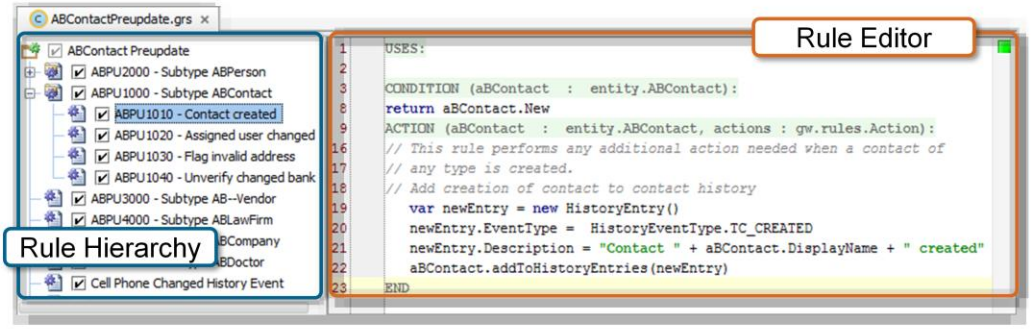
- actions.exit() - Exit the entire rule set
- actions.exitAfter() - Execute this rule's child rules and then exit the rule set
- actions.exitToNextParent() - Skip to the next rule that is at the same level as this rule's parent
- actions.exitToNextRoot() - Skip to the next top-level rule
- actions.exitToNext() - Stop processing the current rule and immediately go to the next peer rule (the next rule at the same level in the hierarchy)

Lesson outline

- Business rules overview
- Rules-specific Gosu
- Working with rules
- Debugging rules

Rule set editor

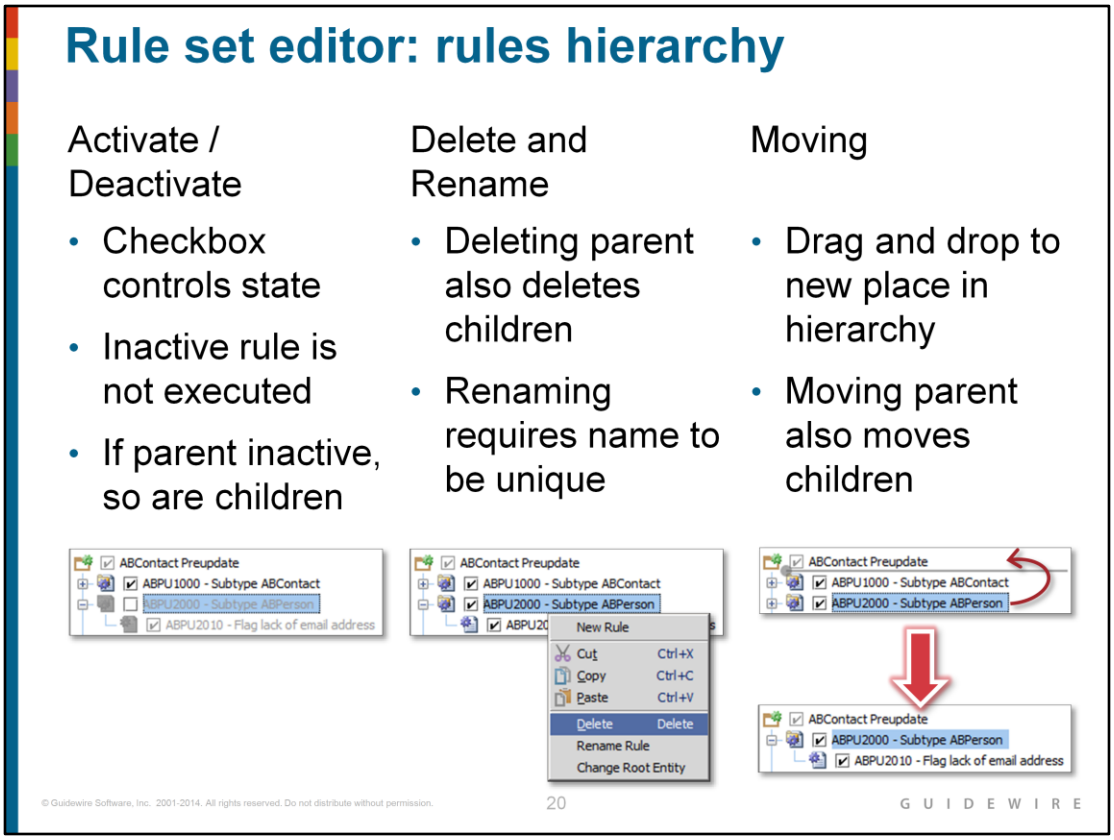
- Select Rule Set in Project View to open Rule Set Editor
- Select Rule in rule hierarchy to edit rule in rule editor



- Rule Hierarchy
  - Context menu
  - Activate/Deactivate checkbox
  - Drag-and-drop ordering
- Rule Editor
  - Overlays segment Uses, Condition, and Action
  - Gosu code for condition and actions

Individual rule files have the file extension .gr. For each rule, there is a condition and action. The Rule Editor provides overlays to segment the condition and action. When a rule condition evaluates to true, the Guidewire application executes the rule action.

An inactive rule is not executed. You can activate and deactivate rules to get the rules engine to selective process and ignore rules. To deactivate a rule, disable the check box that appears to the left of the rule name. Inactive rules appear in gray. To reactivate the rule, enable the check box. To delete or rename a rule, right-click the rule and select the appropriate command from the context menu.

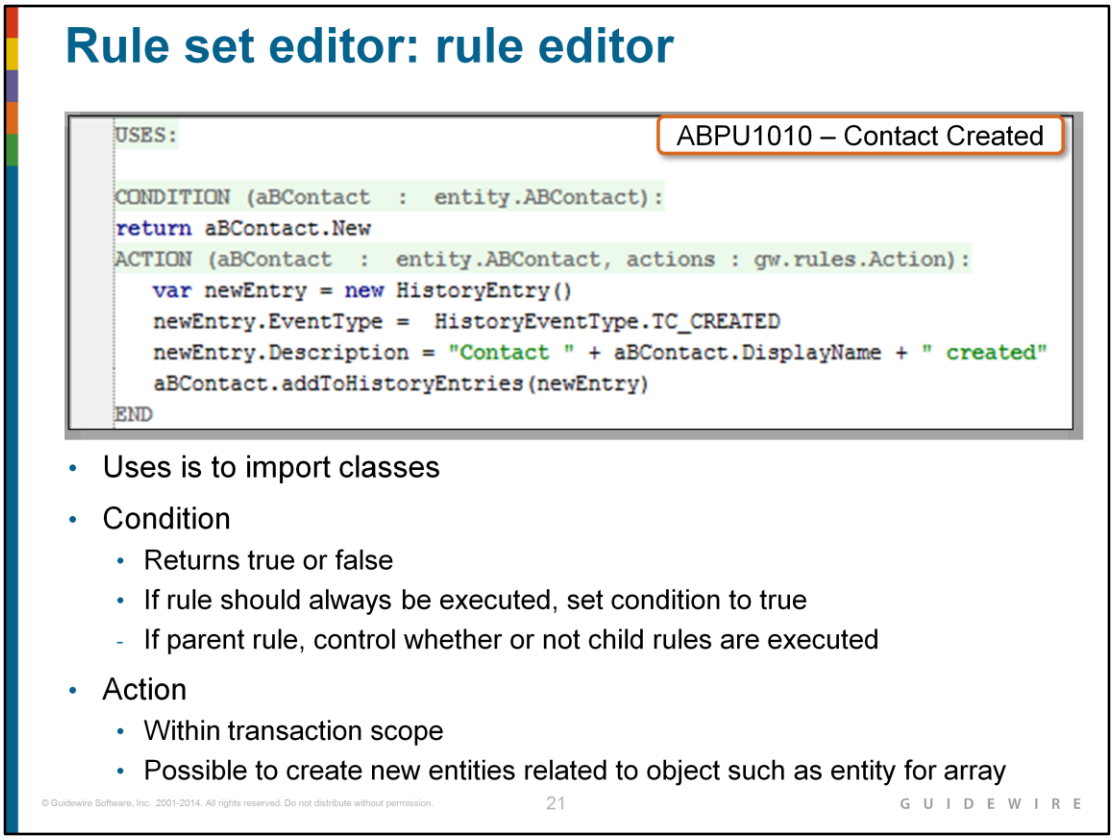


An inactive rule is not executed. You can activate and deactivate rules to get the rules engine to selective process and ignore rules. To deactivate a rule, disable the check box that appears to the left of the rule name. Inactive rules appear in gray. To reactivate the rule, enable the check box.

To delete or rename a rule, right-click the rule and select the appropriate command from the context menu.

You can think of a rule set as a logical grouping of rules that are specific to a business function. You typically organize these rules sets into a hierarchy that fits your business model. Guidewire strongly recommends that you implement a rule-naming scheme that is hierarchal in nature. A hierarchal naming convention will reflect the hierarchal structure when you create rules and organize rules into a hierarchy. However, the naming scheme for rules corresponds to the file structure on disk. So, if it has too many levels customers can run into problems with file names being too long on Windows.

To move a rule, click and drag the rule to the desired position.

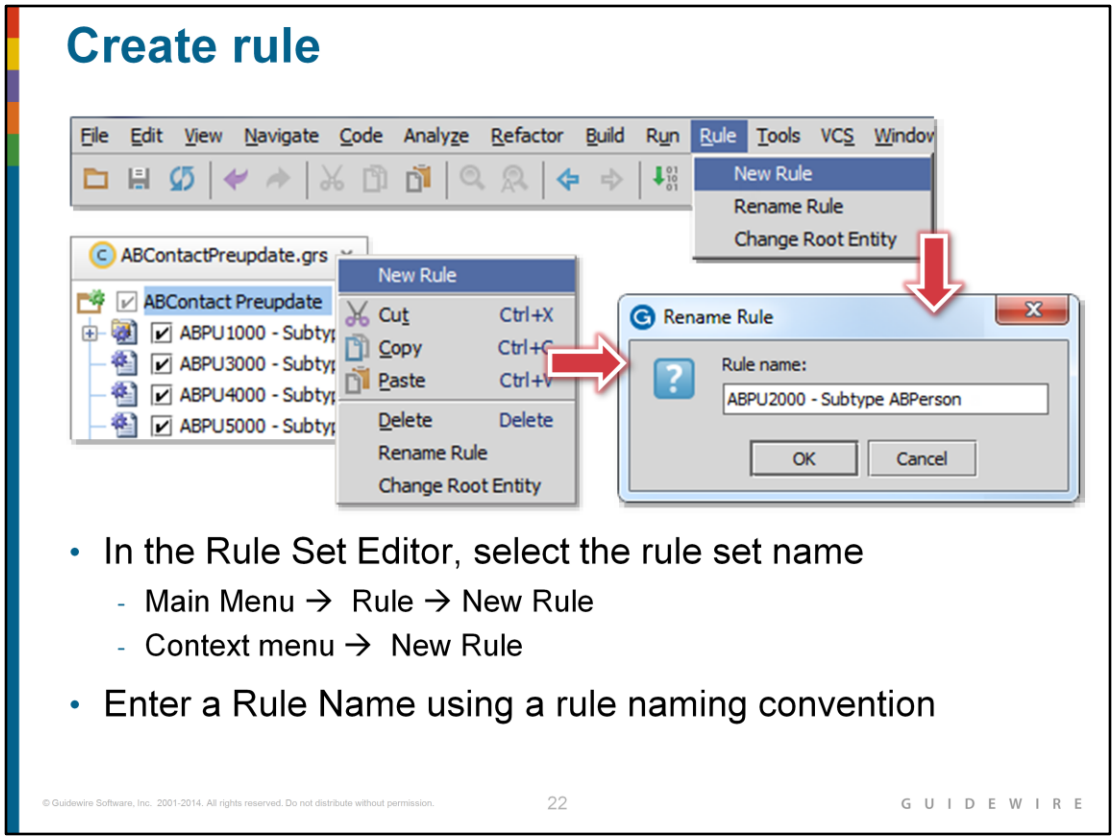


The name of the argument for both the condition and action functions is aBContact. Use a rule condition to indicate if rule action should be executed, control whether or not child rules are executed, and if the rule should always be executed, set the condition to true. If the rule condition is true, then the code in the rule action is executed. This can contain any type of Gosu. For rules without child rules, the rule condition is intended to improve the readability of the code.

You could ignore the condition and put all of the code in the actions.

```
Condition:    ABContact.score != null
Action:       ABContact.preferred = (ABContact.score >= 90)
...could be written as...
Condition:    true
Action:       if ABContact.score != null {
                        ABContact.preferred = (ABContact.score >= 90)
              }
```

Putting the entire condition in the action is not recommended as it makes the purpose of the rule less apparent. That said, you will often see conditions in the rule action so as to specific the control of flow.



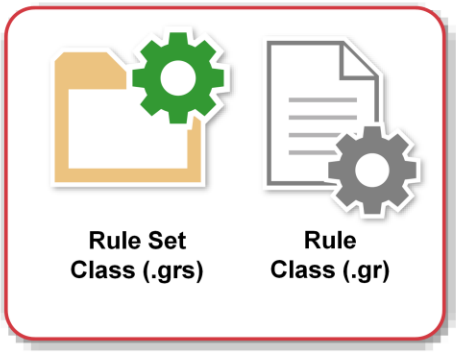
Guidewire strongly recommends that you implement a rule-naming scheme that is hierarchal in nature. A hierarchal naming convention will reflect the hierarchal structure when you create rules and organize rules into a hierarchy. However, the naming scheme for rules corresponds to the file structure on disk. So, if it has too many levels customers can run into problems with file names being too long on Windows.

In a rule set, a Guidewire application evaluates the individual rules recursively in the hierarchy, unless an exit action is encountered. The process starts with the first direct child of the root and then evaluates all children and siblings. All rules in a rule set are processed unless an explicit "exit" is encountered.

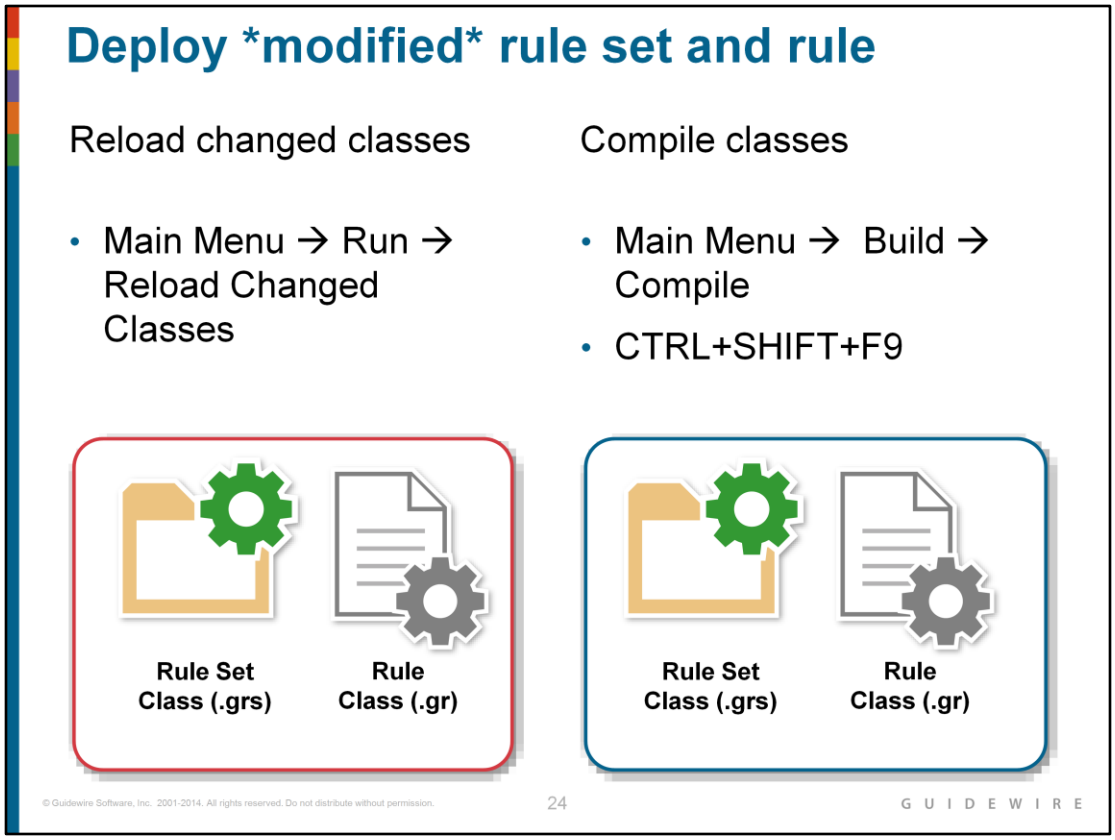
Deploy *new* rule set and rule

Restart Server

- Rule set class and rule class loaded at server startup



If the new rules exists in a new rule set, then you must restart the server.



You can deploy modified rules when the server is running in run or debug server process. If the rules exists in a new rule set, then you must restart the server.

When you activate a deactivated rule in a rule set or deactivate an activated rule in a rule set, you have modified the rule set (.grs) class.

If your modified rule (.gr) contains a new displaykey, then you will also need to reload PCF files using ALT+SHIFT+L, the Guidewire API and/or internal server tools.

Lesson outline

- Business rules overview
- Rules-specific Gosu
- Working with rules
- Debugging rules

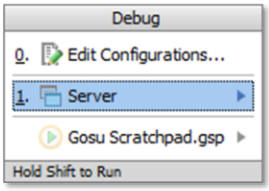
Steps to debug rules

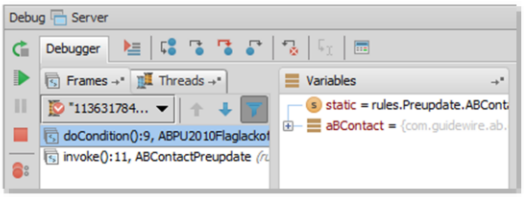
- Set breakpoints in rule
- Debug Server
  - ALT+SHIFT+F9
  - Select Server
- Console tab
  - Verify output reads

```
*** AppName ready ***
```

- Perform actions to hit breakpoint
- In Debug tool window, review Debugger
- Resume program
  - F9

```
1   USES:
2   uses gw.api.util.DateUtil
3
4   CONDITION (aBContact : entity.ABContact):
9   return aBContact.EmailAddress1==null
```





Breakpoints

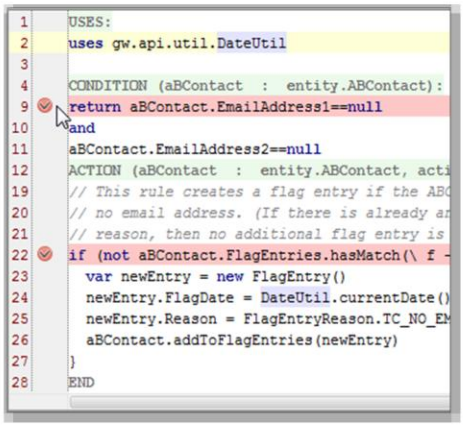
```
1   USES:
2   uses gw.api.util.DateUtil
3
4   CONDITION (aBContact  :  entity.ABContact):
9   return aBContact.EmailAddress1==null
10  and
11  aBContact.EmailAddress2==null
12  ACTION (aBContact  :  entity.ABContact, actions : gw.rules.Action):
19  // This rule creates a flag entry if the ABContact is a person with
20  // no email address. (If there is already an unresolved flag entry for this
21  // reason, then no additional flag entry is created.)
22  if (not aBContact.FlagEntries.hasMatch(\ f -> (f.Reason == FlagEntryReason.TC_NO_EMAIL)
23    var newEntry = new FlagEntry()
24    newEntry.FlagDate = DateUtil.currentDate()
25    newEntry.Reason = FlagEntryReason.TC_NO_EMAIL
26    aBContact.addToFlagEntries(newEntry)
27  }
28  END
```

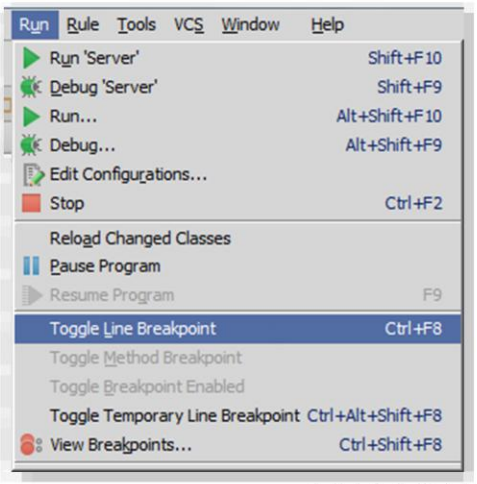
- A **breakpoint** indicates a place where you want to suspend the execution of Gosu code

Because you can step through lines, you normally need only one breakpoint for each section of code you want to investigate.

Setting breakpoints

- Click in the gutter for the given line
- Select line
  - Main menu → Run → Toggle Line Breakpoint
  - CTRL+F8

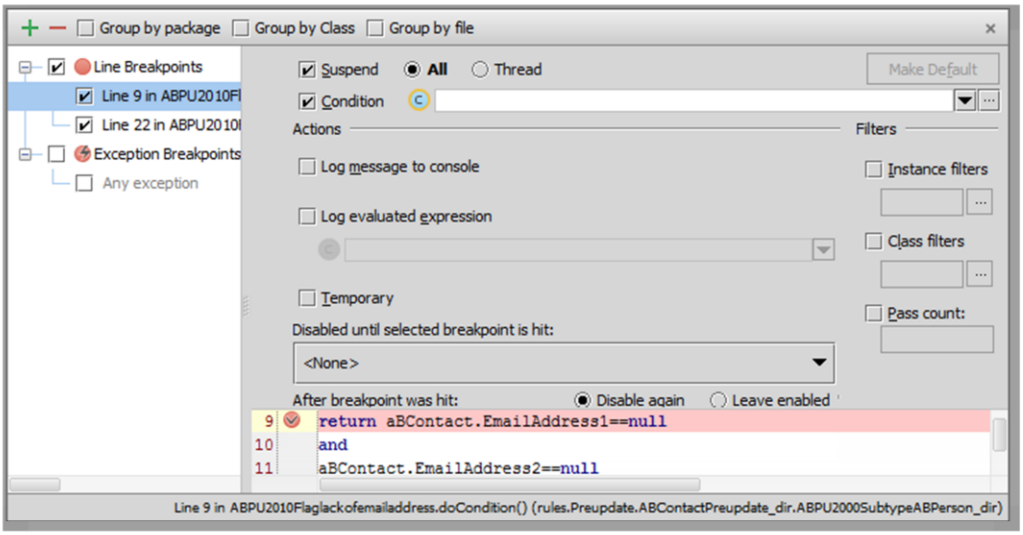




You can set a breakpoint by clicking in gutter for the given line in the editor. You can also select the line and then select Main menu → Run → Toggle Line Breakpoint or use the CTRL+F8 keystroke.

View all breakpoints

- Run → View Breakpoints… or CTRL+SHIFT+F8
  - Enable, disable, define action and conditions



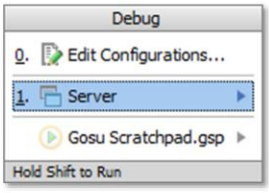
In addition to Ctrl-Shift-F8, right-clicking on the breakpoint also displays a menu with choices to Edit, Disable, Remove, or View Breakpoints. This screen shows the Edit breakpoint options.

The Breakpoint options dialog allows you to review all breakpoints. In the Toolbar, you can add a breakpoint, remove a breakpoint, display breakpoints under their respective packages rather than under their types, display breakpoints under their respective classes, and display breakpoints under their respective files. There are various Breakpoint options to set. You can enable a suspend policy for a breakpoint. For example, for a Thread policy the thread where the breakpoint is hit is suspended. You can also specify a condition in either Gosu or Java. You can also specify actions, such as logging the evaluated expression.

Debug 'Server'

- Start debug Server
  - ALT+SHIFT+F9
  - Select Server
- Console tab
  - Verify output reads

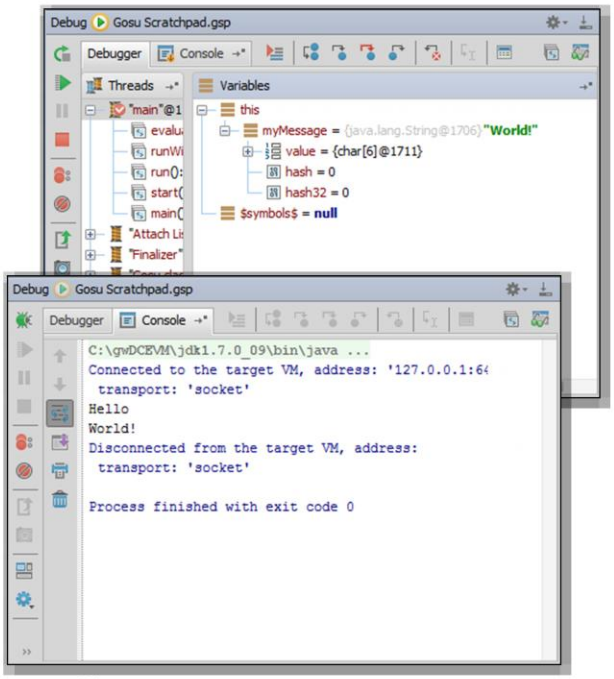
```
*** <GuidewireApplication ready ***
```



To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. In the Debug tools window, confirm that the application is running and is ready (***** AppName ready *****).

Debug tool window

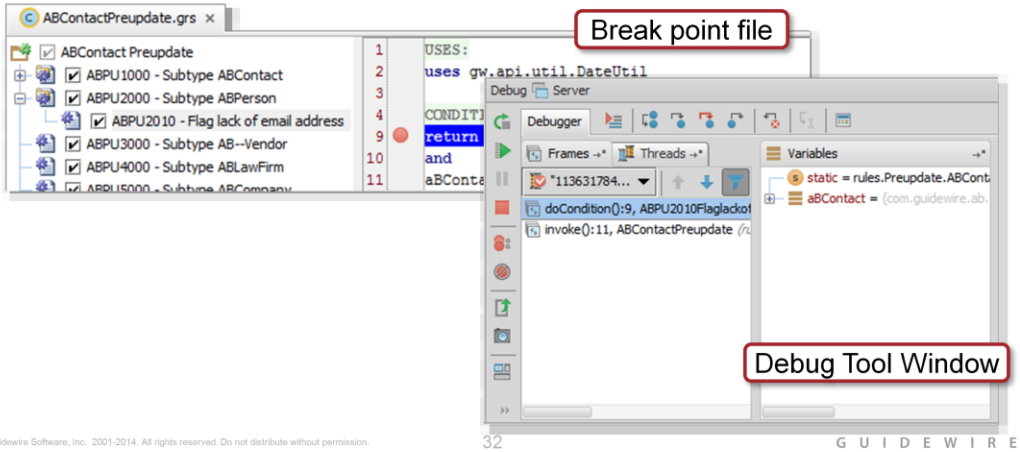
- ALT+5
  - Opens Debug window
- Debugger tab
  - Rerun, Resume, Pause, Stop
  - View breakpoints
  - Step over, into and out
  - Inspect and watch
- Console tab
  - View output



The debugger enables you to execute your application step by step, examine program information related to variables, watches, or threads, and change your program without leaving Guidewire Studio. Gosu Scratchpad

When application hits a breakpoint…

- Guidewire Studio suspends code execution
- File with breakpoint opens in Studio
- Breakpoint line highlighted in blue
- Debug tool window opens



To hit breakpoints, you must run the application in a debug 'server' process.

Because you can step through lines, you normally need only one breakpoint for each section of code you want to investigate. There is no advantage to having breakpoints on multiple consecutive lines because the first breakpoint suspends normal execution and normal execution does not resume until you request it to.

The Frame tab lists the object passed to the code (for business rules, this is the root entity). Variable values at the breakpoint are visible in the Variables window. Frames shows the sequence of actions executed until the breakpoint.

Stepping through code

- Several step tools are available to help in debugging

Step over (F8) advances you to the next line in the file

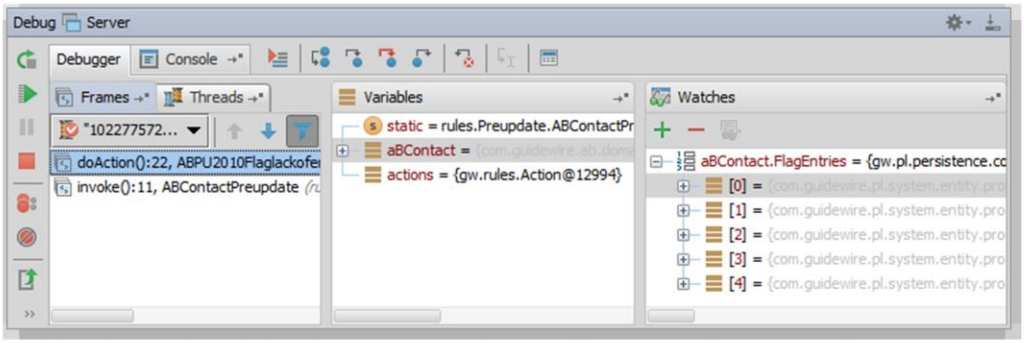
Step into (F7) advances you to the next line executed

Force step into (Alt+Shift+F7) steps into, ignore stepping filters for libraries, constructors, etc.

Step out(Shift+F8) step to the first line executed after returning from this method

Once the debugger pauses execution, you can step through the code in one of several ways. The most basic is to stepping through code one line at a time using Step Over (F8).

Debugger



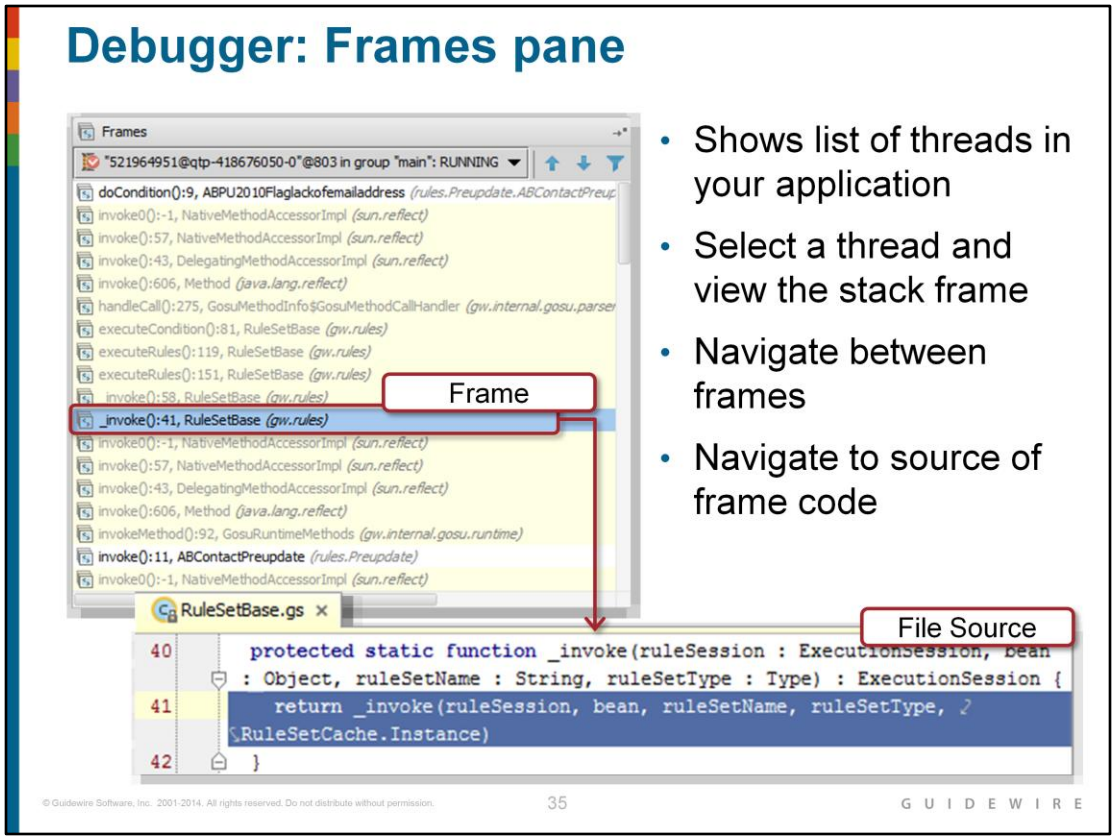
- For breakpoint, debugger shows how code executes
- Debug Gosu code running in the Debug 'Server' process

Class

Enhancement

Rule

When the Debug 'Server' process hits a breakpoint in Gosu code, the debugger suspends the normal execution of code until you resume it. When you resume the debugger, code executes as normal until the debugger hits another breakpoint.

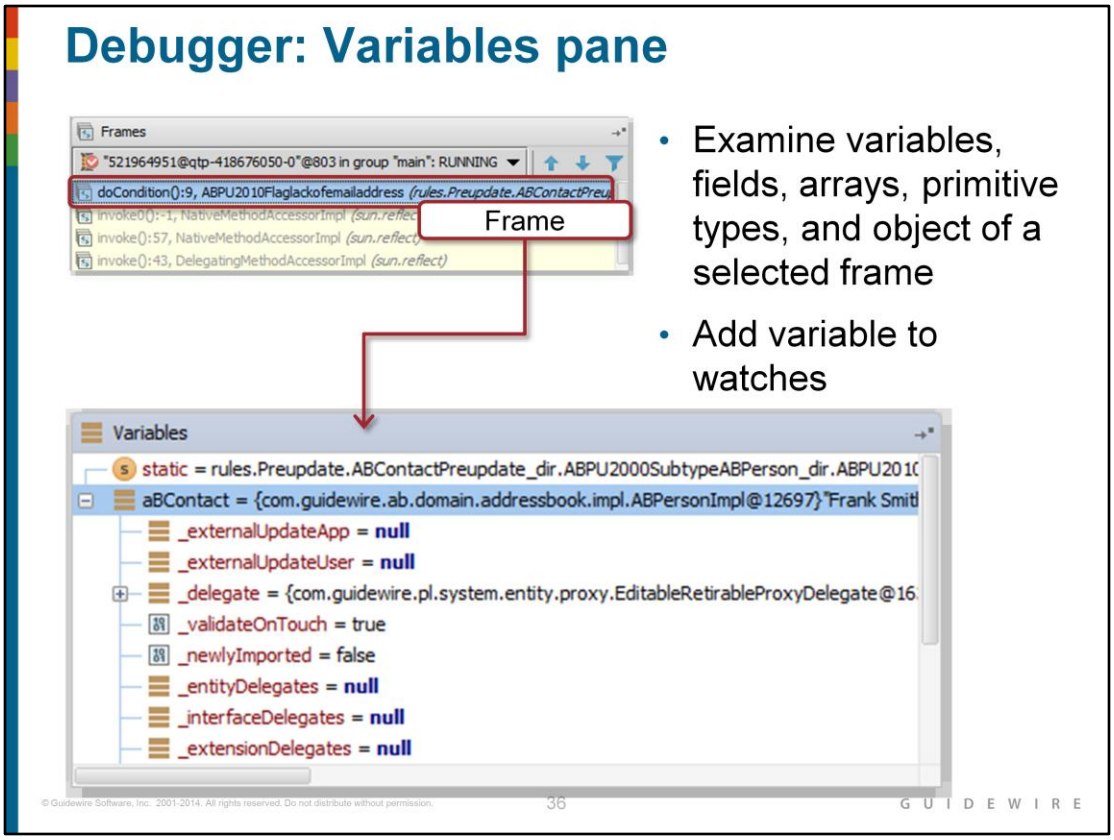


The Frames pane enables you to gain access to the list of threads running in your application, export threads to a text file, and customize the thread presentation.

For each thread, you can view the stack frame, examine frames, navigate between frames, and automatically jump to the source code of a frame in the editor.

A thread can be chosen via a thread selector drop-down list on top of the pane. The status and type of a thread is indicated by the special icon and textual note next to the thread's name.

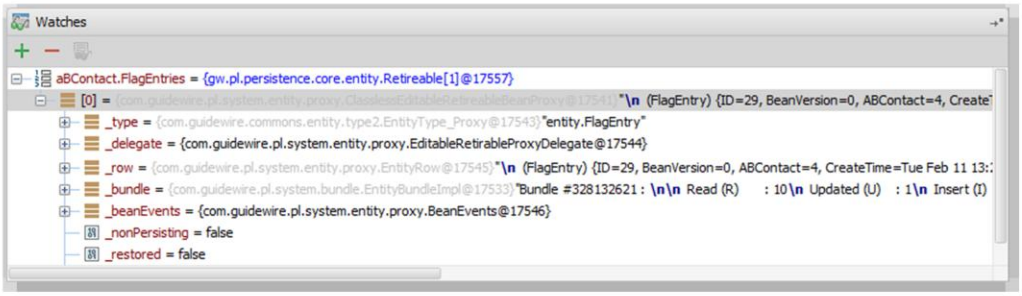
To examine the values stored in a frame, use the Variables pane of the Debug tool window.



When a stack frame is selected in the Frames pane, the Variables pane displays all the data within its scope (method parameters, local and instance variables). In the variables pane you can set labels for the objects, inspect objects, evaluate expressions, add variables to watches and more. You can examine static variables, fields, arrays, primitive types, and objects.

Enable Enhance Entities Visualization in Guidewire Studio to view all entity properties. In Project Settings, in IDE Settings, select Guidewire Studio. Then, in Debugger Settings, check the Enhance Entities Visualization checkbox.

Debugger: Watches pane

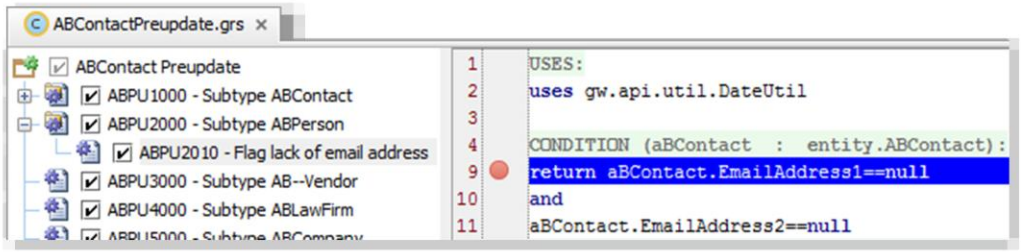


- A **watch** is an expression whose value you wish to observe
- To add a watch, any of the following:
  - Click Add → Enter property or expression
  - Drag object or property from the Variable pane to watches pane
  - Drag variable and property expression from rule editor to watches pane

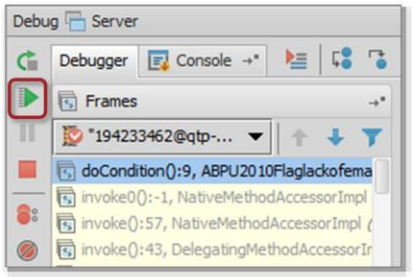
In the Watches pane you can evaluate any number of variables or expressions in the context of the current stack frame. The values are updated with each step through the application and become visible every time the application is suspended.

The easiest way to populate the Watches pane is to run the code to a desired point, then drag the object or property to the Watches pane.

Resume debugging

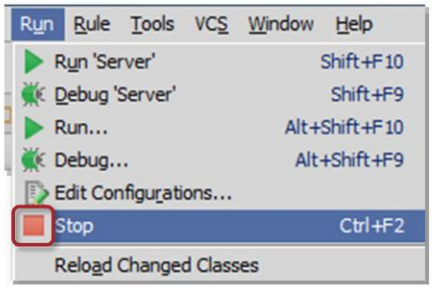


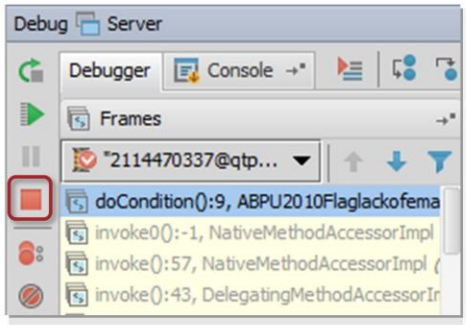
- To resume the execution of code…
  - Main menu → Run → Resume
  - F9
  - Or, in the Debug Tool Window, click Resume
- Code continues to hitting next breakpoint



Stopping debug server

- To stop the debug server process…
  - Main menu → Run → Stop
  - CTRL+F2
  - Or, in the Debug Tool Window, click Stop





Lesson objectives review

- You should now be able to:
  - Explain the functionality of business rules
  - Describe the Gosu techniques unique to business rules
  - Write business rules
  - Use Studio debugger to debug business rules

Review questions

1. What is the significance of a rule set's root entity?
2. What triggers a rule set?
3. What two things are done for a rule whose condition is true that are not done for a rule whose condition is false?
4. What is the difference between an "execute all" rule set and an "exit after first action" rule set?
5. What is the effect of deactivating a rule?
6. How can you deploy a new rule?
7. For a server running a debug 'server' process, how can you deploy changes to a rule?
8. For a server running a debug 'server' process, how can you deploy a newly deactivated rule?

Answers

1) Every rule set is tied to an entity, known as the root entity. This determines which class of objects the rule set is tied to. It also determines the object available to the rules in the rule set when the rule set is triggered.

2) A rule set is triggered when a given event occurs to any instance of the root entity. The actual event varies from rule set to rule set.

3) The rule action is executed and the child rules (if any) are executed.

4) In an "execute all" rule set, all rules are executed. For example, validation rule sets are execute all. All validation conditions are checked, regardless of how many are true. In an "exit after first action" rule set, you do not typically execute all rules. When the first true condition is found, the action associated with that condition is taken and the rule set is exited. For example, assignment rule sets are exit after first action. Once an assignment is made, the rule set exits so that later code doesn't reassign the object.

5) The rule is not executed by the rules engine, which is useful in development for turning rules "on" and "off".

6) Restart the server.

7) Reload changed classes or Compile the rule set.

8) Reload changed classes or Compile the rule set.

Notices