GUIDEWIRE

PolicyCenter 10 Configuration: Essentials

Student Workbook

Labs and Tutorials

Document Published: 2019-08-27

Table of Contents

Introduction

Welcome to the Guidewire PolicyCenter 10 Configuration - Essentials course.

The Student Workbook will lead you through the course labs. The lesson numbers correspond to the lesson numbers in your training. Complete the assigned labs to the best of your ability.

Sample Accounts

If the large sample data is loaded in PolicyCenter, it contains the following sample accounts for you to use to complete the exercises. It is recommended that you follow all the labs to create your own new accounts.

Account No.	**Type**	**First Name**	**Last Name**
C000143542	Person	Ray	Newton
B000457766	Person	Erica	Billings
C000456353	Person	Mary	Taylor
C000148456	Person	Bill	Kinman
A000377766	Person	Dave	Worthington
Account No.	**Type**	**Company Name**	
C000212105	Company	Wright Construction	
C000478975	Company	Big Lake Bakery	
C000478567	Company	Calloway Cheese Factory	
A000377655	Company	Dominion Logistics	
B000467655	Company	Earth Tech	

Open navigation pane

Tip

Open navigation pane in a PDF file

If you do not see the left navigation pane, you can turn it on. In a PDF file, click the bookmark icon on the left. Or click the Adobe Acrobat menu item View → Show/Hide → Navigation Panes → Bookmarks.

Lesson 1

Concepts of Revisioning Contact and Location Information

1.1 Labs

1.1.1 Lab 1

Investigation

This is an investigation exercise where you will explore the behavior of revisioned and non-revisioned fields and the advantage of revisioned fields.

Edit name and phone number of an existing contact on an account and see how this affects existing policies and new policies.

1. **Create an account and issue a policy, based on the following information:**

   "Good morning. My name is Pat Penguin. I'm not a customer of Acme Insurance. I live at 1212 Dayton Street in Portland, Oregon. My zip code is 97201. My home phone number is 555-555-5555. I'm interested in getting a personal auto policy for my auto whose VIN number is 11122233344455566. The cost when it was new was $25000."

2. **Make a policy change.**

   a) After two months, Pat calls to inform that she has gotten married and would like to change her last name to Puffin.

   b) She also has a new phone number.

   c) Change the Contact name and phone number on the account.

3. **Open the policy.**

   a) Has the name changed? Why or why not?

   b) Has the phone number changed? Why or why not?

4. **After another 2 months, Pat calls to renew the policy for 6 months.**

   What is the name on the Renewal job? Why?

1.1.2 Lab 2

Investigation

This exercise is divided into three parts. Observe how the contact information is updated based on each change.

1) Create a policy in the first part.
2) Make a future dated change on that policy in the second part.
3) Make a back-dated change in the third part.

Part 1: Create a policy

1. **Create a new account for Jane Smith, who is single and born on 4/1/1967.**
2. **Create an annual personal auto policy.**
   a) For her car which cost $30,000 when she first bought it.
   b) She is the only driver and owns only one vehicle.

Part 2: Future dated change

After two months, Jane calls to inform that she is getting married in a week and would like to add her husband John Doe as a driver on her policy.

1. **Start a Policy Change transaction and make the following changes:**
   - Calculate and enter Effective Date as: today's date + 2 months.
   - Description: Getting married.
   - Note down the effective date.
   - Add John Doe as a driver.
   - She would also like to change her last name to "Doe".
   - Her status to "Married".
   - Fill in the other details as required.
2. **Quote and issue the policy change.**
3. **On the account, what is the last name of Jane? Why?**
4. **After issuing the Policy Change, execute the change by advancing the system clock and run the batch process by following the steps below:**
   a) Press Alt + Shift + T
   b) Navigate to Internal Tools → Testing System Clock

c) Click **Change Date** to set the clock to the date Jane is getting married, which is also the effective date of your policy change.

d) Go to Server Tools → Batch Process Info

e) Run the batch process ***Apply Pending Account Data Updates*** by clicking the **Run** button in the **Action** column.

f) Click **Return to PolicyCenter** in the top right corner of the window Under the Options menu.

5. **Answer the following question.**

   After advancing the system clock to the policy change's effective date and running the batch process, what is the last name of Jane on the account? Why?

Part 3: Back dated change

After three months, Jane calls to inform that the birth year on her policy is wrong and needs to be corrected to 1965.

1. **Start a policy change.**

   a) This change should take effect from the beginning of the policy.

   b) This is a back dated change so you will get a warning about an out-of-sequence transaction. Click OK to continue.

   c) You must resolve the out-of-sequence conflicts on the Policy Review screen's Change Conflicts tab.

   d) Quote and issue the policy.

2. **The policy shows the updated birth year.**
3. **The back dated changes do not flow back up to the Account. The Contact birth date is not updated in the Account file.**
4. **Activities and Notes are created on the Account to inform the user about the contact changes that may need to be updated on the account.**

1.2 Solutions

1.2.1 Lab 1 Solution

Solution

Edit name and phone number of an existing contact on an account and see how this affects existing policies and new policies.

1. **Create an account and issue a policy, based on the following information:**

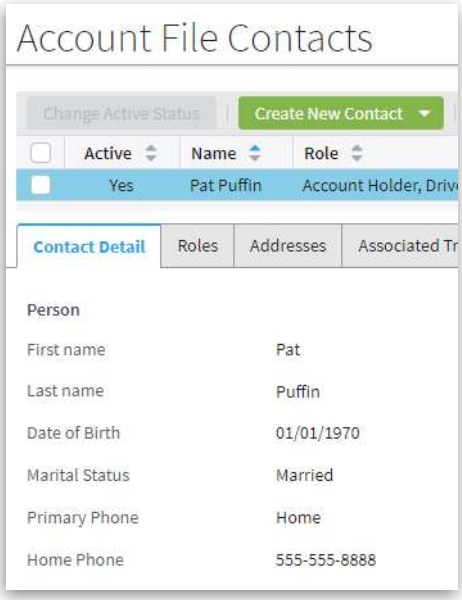
   "Good morning. My name is Pat Penguin. I'm not a customer of Acme Insurance. I live at 1212 Dayton Street in Portland, Oregon. My zip code is 97201. My home phone number is 555-555-5555. I'm interested in getting a personal auto policy for my auto whose VIN number is 11122233344455566. The cost when it was new was $25000."

2. **Make a policy change.**

   a) After two months, Pat calls to inform that she has gotten married and would like to change her last name to Puffin.

   b) She also has a new phone number.

   c) Change the Contact name and phone number on the account



3. **Open the policy**

   a) Has the name changed? Why or why not?

   The policy contract has already been bound and the contact name is a revisioned field so the name of the insured is not changed on the policy.

   b) Has the phone number changed? Why or why not?

   The phone number on the policy, however, has the new value that was changed on the account, since it is not a revisioned field.

The reason the name is revisioned and the phone number is not is because the name is part of the policy contract data, whereas the phone number is not and they are configured accordingly.

4. After another 2 months, Pat calls to renew the policy for 6 months.

What is the name on the Renewal job? Why?

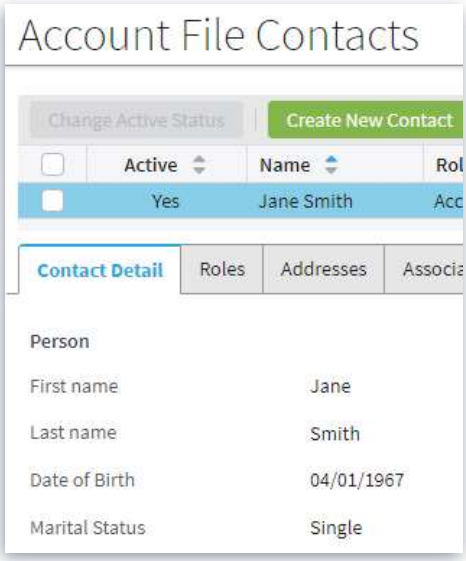
The most recent information is picked up from account when a new job begins. Hence at renewal the updated name is picked up from the account.

1.2.2 Lab 2 Solution

Solution

Part 1: Create a policy

1. Create a new account for Jane Smith, who is single and born on 4/1/1967.

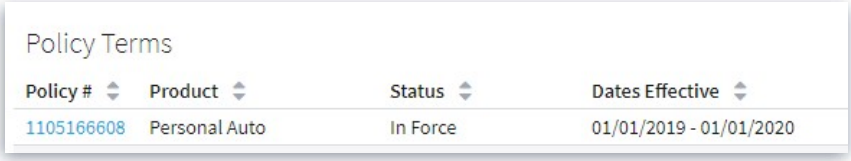


2. Create an annual personal auto policy.

a) For her car which cost $30,000 when she first bought it.

b) She is the only driver and owns only one vehicle.

(Your dates may be different from the dates in the screen shot below

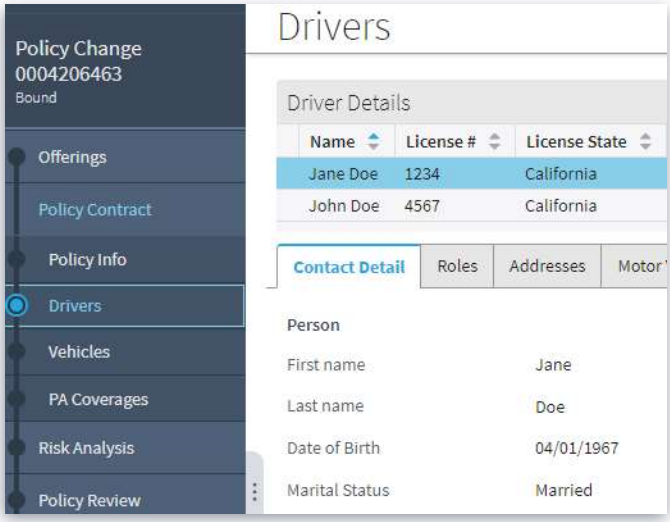


Part 2: Future dated change

After two months, Jane calls to inform that she is getting married in a week and would like to add her husband John Doe as a driver on her policy.

1. Start a Policy Change transaction and make the following changes:

- Calculate and enter Effective Date as: today's date + 2 months + 1 week.
- Description: Getting married.
- Note down the effective date.
- Add John Doe as a driver.
- She would also like to change her last name to "Doe".
- Her status to "Married".
- Fill in the other details as required.



2. **Quote and issue the policy change.**
3. **On the account, what is the last name of Jane? Why?**

   The last name of Jane is Smith. Future dated contact data changes at the policy level do not flow back up to the account because the contact at the account level is considered to be current data and cannot be future dated.

4. **After issuing the Policy Change, execute the change by advancing the system clock and run the batch process by following the steps below:**

   a) Press Alt + Shift + T

   b) Navigate to Internal Tools → Testing System Clock

   c) Click **Change Date** to set the clock to the date Jane is getting married, which is also the effective date of your policy change.

   d) Go to Server Tools → Batch Process Info

   e) Run the batch process ***Apply Pending Account Data Updates*** by clicking the **Run** button in the **Action** column.

   f) Click **Return to PolicyCenter** in the top right corner of the window Under the Options menu.

5. **Answer the following question.**

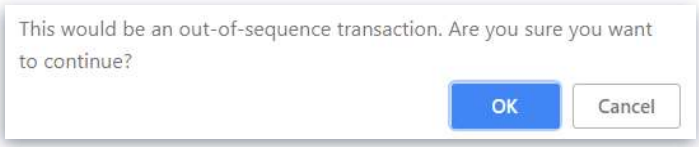
   After advancing the system clock to the policy change's effective date and running the batch process, what is the last name of Jane on the account? Why?

   *The last name of Jane is Doe. Because the batch process **Apply Pending Account Data Updates** automatically updates the account contact data from the policy changes on the job effective date.*

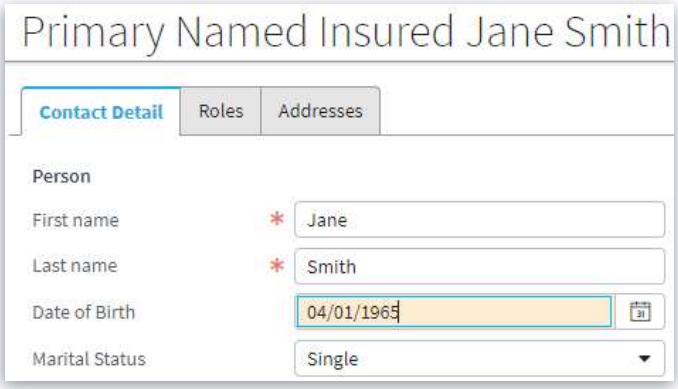
Part 3: Back dated change

After three months, Jane calls to inform that the birth year on her policy is wrong and needs to be corrected to 1965.

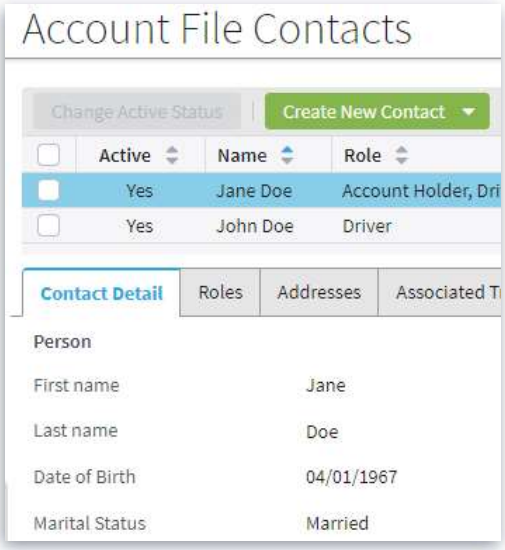
1. **Start a policy change.**
   a) This change should take effect from the beginning of the policy.
   b) This is a back dated change so you will get a warning about an out-of-sequence transaction. Click OK to continue.



   c) Change Jane's birth year by editing either the Primary Named Insured on the Policy Info page or the Drivers page.

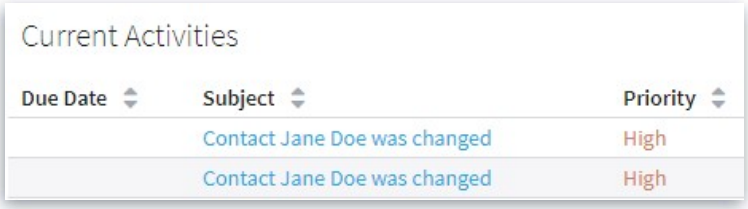


   d) You must resolve the out-of-sequence conflicts on the Policy Review screen's Change Conflicts tab.
   e) Quote and issue the policy.
2. **The policy shows the updated birth year.**
3. **The back dated changes do not flow back up to the Account. The Contact birth date is not updated in the Account file.**

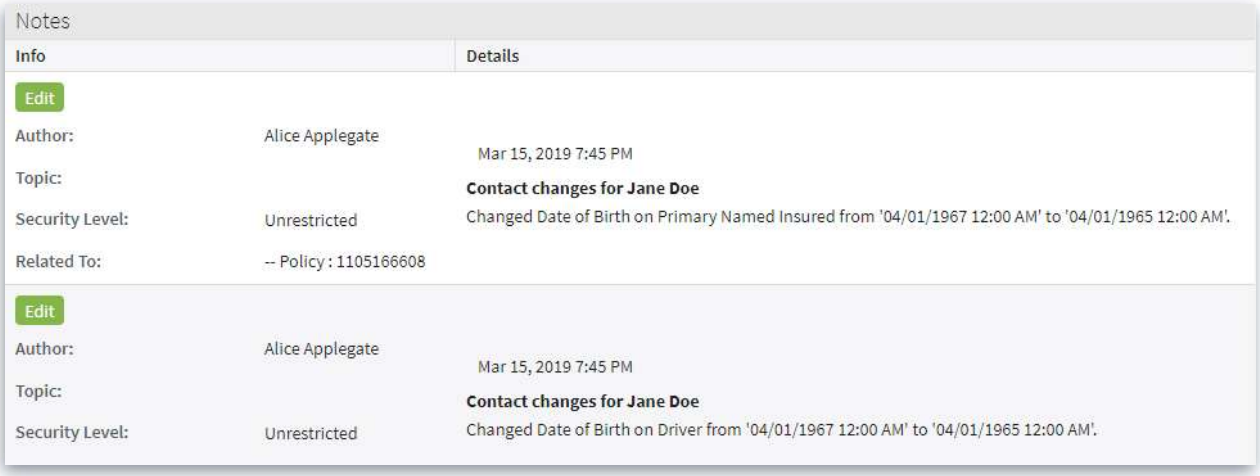


4. Activities and Notes are created on the Account to inform the user about the contact changes that may need to be updated on the account.

Account → Summary page → Current Activities section:



Account → Notes page:



1.3 References

1.3.1 Sharing and revisioning information

Contact and location information is shared between accounts and policies. Revisioned fields are required on contacts or locations when some information is considered part of the policy contract and thus cannot be changed once the policy is bound and the contract written. If the field is revisioned, it can be changed on the account level even after the policy is bound. The information is updated on the account and can be obtained by policies for future jobs.

Share: Information can be kept at the Account level, so it can be shared, to avoid duplication. For example, people move, get married and change their names, get new drivers licenses, etc. Benefits of sharing information are:

- Avoiding data re-entry and errors
- Allowing the same contact to play multiple roles on the account and policy, such as account holder or named insured
- Associating Contact to other entities such as Location or Vehicles

Revision: Information that is part of policy contract on an issued PolicyPeriod should be revisioned

- When a Policy is issued, we want to keep a version of it at that point in time
- It can change anytime a job is executed on it

Data inside and outside PolicyPeriod graph:

Data outside PolicyPeriod	Data inside PolicyPeriod
Lives only on Account	Lives on both Policy and Account
Shared but not revisioned because we do not need to know how data has varied over time	Shared and revisioned because we need to know when it changed
When data changes, change is made to the Account only	Change is made to objects on the Policy
History is not available	History is available

PolicyCenter determines whether to pick up the current contact data from the account based on the effective date of the job. Location information is picked up from the account irrespective of the effective date on the job.

1.3.2 Revisioning contact information

Contacts are always synced to account for submissions and issuances even for future dated changes.

A contact's last update time is updated each time the account level contact information gets modified:

- If it is modified in the context of a policy job, it is set to the effective date of that job.
- If it is modified at the account or cross-account level, it is set to the current date.

1. **To handle changes to current contact data:**

   At the beginning of a job, PC checks if contact data is

   Current: `Last update of contact <= Job Eff Date (Job is not in future).`

   - The contact data is synced to the account data throughout the job because both the policy and the account contact data are considered "current" so changes to either one should be reflected in the other one. Data is copied to and from account and policy.
   - Modify contact's last updated time.

2. **To handle a future dated change:**

   At the beginning of a job, PC checks if contact data is

   Future dated: `current date < Job Eff Date.`

   - Copy down current contact information from Account to the policy.
   - Account and Policy not sync'ed during policy change job. Changes do not flow back up to the account because the contact at the account level is considered to be current data and cannot be future dated.
   - Contacts on submission and issuance jobs are always sync'ed to the account, even when the job is future dated. In that case, the contact update time is set to the current date, not the effective date of the job.
   - Contact data changes creates a work item for the batch process Apply Pending Account Data Updates.
   - The batch process automatically updates account-level contact data on the job effective date.

3. **To handle a back dated change:**

   At the beginning of a job, PC checks if contact data is

   Back dated: `Last update of contact > Job Eff Date.`

   - Use policy-level data and make changes at policy level only.
   - The contact information on the policy is neither copied down from the account, nor does it automatically copy any changes back up to the account.
   - For contact changes:
     - PolicyCenter creates activities suggesting changes at account level.
     - Creates notes to specify the changes.

1.3.3 Revisioning location information

The concept of location revisioning is similar as contact revisioning. The synchronizing of the revisioned fields is the same as in case of current dated changes for contact data. The future dated and back dated change scenarios do not apply to locations because locations should not really change over time; changes to location information should generally be corrections which you'd want to apply throughout the life of the policy.

In the default configuration, PolicyCenter revisions most of the location information. Because most location information is part of the policy contract, PolicyCenter must store that information about the location exactly as it was at the time that the policy was bound.

Location number is a special case. Locations are numbered separately at the account and policy level anyway so you neither want to synchronize that field nor look at the account location number when viewing the policy. The location number at the policy level is revisioned (in the same way policy information is normally revisioned), but not synced because it is not shared between the account and policy.

Changing account location information affects:

Pending Policy Transactions: The changes are immediately apparent when you view pending policy transactions because those policy transactions always display the up-to-date information.

Quoted Policy Transactions: When you try to bind a quoted but not bound submission, the application checks to see if the policy location information matches the account location information. The information will not match if someone made a change on the account location to a revisioned field after the policy transaction was quoted. If the locations do not match, you will see a validation error and will need to requote. This is because the change could influence the quote. When you quote the policy again, the application synchronizes the policy location with the account location.

Bound Policies or Completed Policy Transactions: Bound policies or completed policy transactions have copies of the synchronized location information at the time of binding. This information is included on the associated policy revision. Each bound policy or completed policy transaction is a separate policy revision.

New Policy Transactions: When new policy transactions on existing policies begin, the locations on those policy transactions always display the most recent location information from the account. It does not matter what the location information is in the revision they are based on.

Note: This is the behavior in the default configuration and is recommended by Guidewire. However, carriers can change at what point revisioned information is synchronized. This is discussed in the next lesson.

Lesson 2

Introduction to Permission Configuration

2.1 Requirements

Enigma Fire & Casualty Insurance would like to add some restrictions on which users can edit the Account Locations. Configure the Account Locations in PolicyCenter such that only users with the proper permission can edit the account locations, for example, producers and underwriters.

2.2 Configuration

Activity

1. **Verify permissions for Account Location Information in the application before configuration.**

   The current behavior for account locations is that any users can edit account location information.

   a) Log in as different users. Verify that they all can edit account location information for the account Ray Newton.

      - aapplegate (underwriter)
      - aarmstrong (producer)
      - aauditor (auditor).

2. **In Studio, create the following system permission to edit account locations:**

Code	Name	Description
editlocation_Ext	Edit account locations	Permission to edit locations on an account

3. **Configure the `AccountLocationPopup.pcf` page so that only users with the appropriate system permission can edit the page.**
4. **Restart the server as required by additions to a typelist.**

Tip

Invalidate caches

If you still don't see the new typecode in Studio, then you might need to invalidate your caches and restart Studio.

`File → Invalidate Caches / Restart → Invalidate and Restart`

Tip

Generate dictionary

The security dictionary must be regenerated after a typelist is modified. It also helps to validate any entity and typelist changes. Command = `gwb genDataDictionary`

5. **In PolicyCenter, login as su/gw.**
6. **Grant the new edit account location permission to the Producer and Underwriter roles.**

2.3 Verification

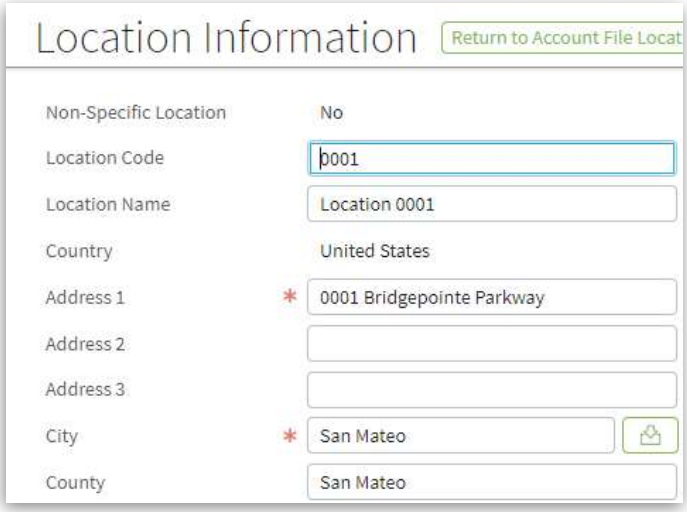
1. **Log in to Guidewire PolicyCenter as the following users and repeat the steps.**
   - Alice Applegate (an underwriter) with `aapplegate/gw`
   - Archie Armstrong (a producer) with `aarmstrong/gw`
   - Adam Auditor (a premium auditor) with `aauditor/gw`

   a) Navigate to the Account Ray Newton.

   b) Click on Locations in the side bar to display the Account File Locations list.

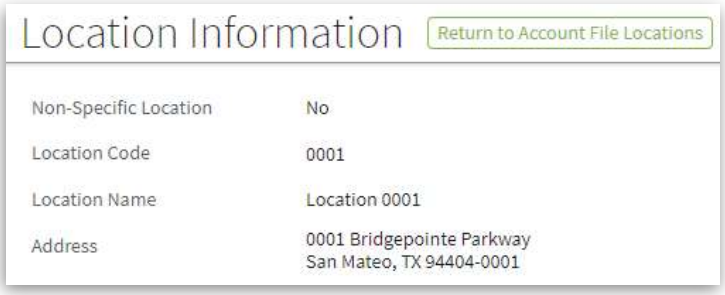
   c) Click on one of the hyperlinks (Loc.#, Location Code and Location Name columns) associated with one of the locations.

2. **Results**
   - User Alice Applegate and Archie Armstrong can edit the Account Location information in the popup.



GUIDEWIRE

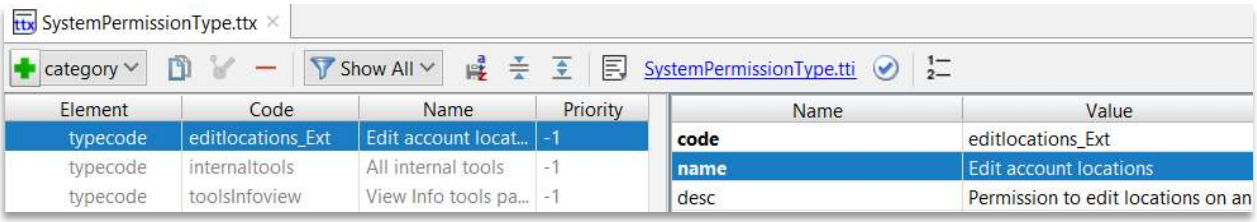
- User Adam Auditor cannot edit the Account Location information in the popup.



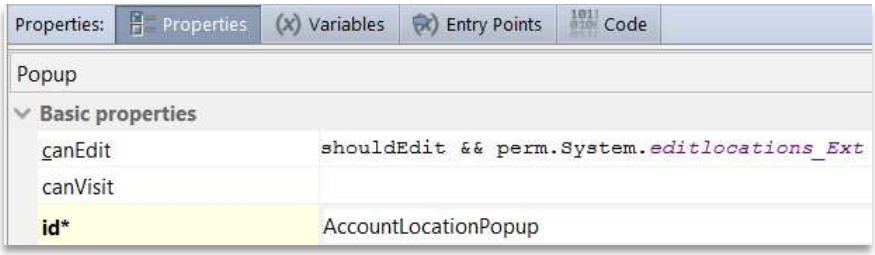
2.4 Solution

1. **In Studio, modify `SystemPermissionType.ttx` to add a new typecode.**

Code	Name	Description
editlocations_Ext	Edit account locations	Permission to edit locations on an account



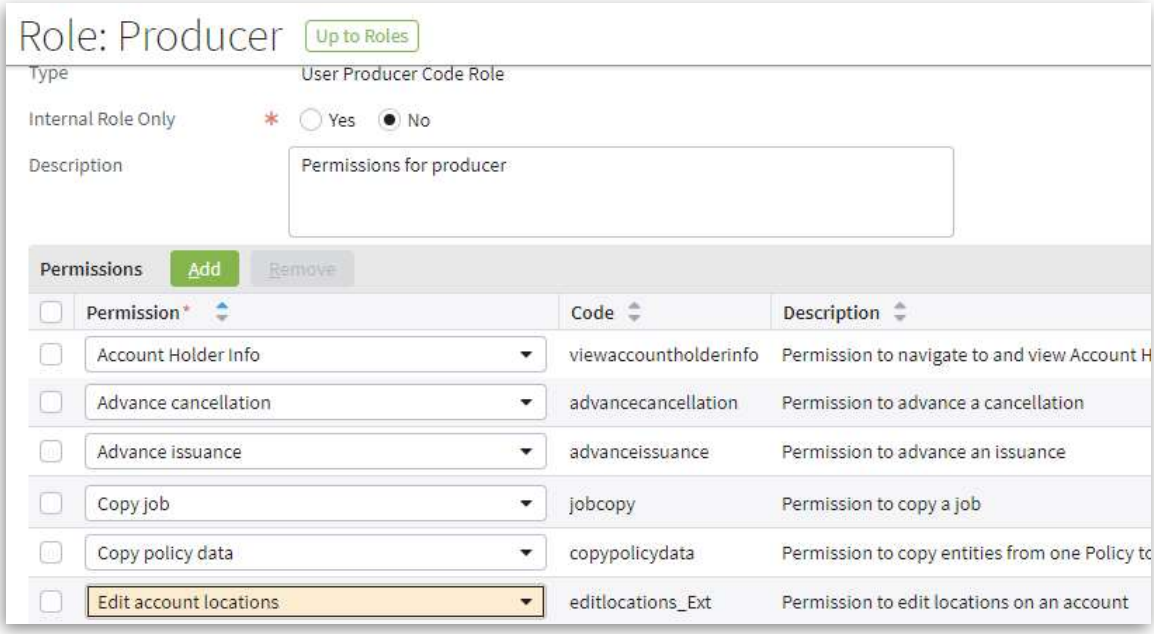
2. **Configure AccountLocationPopup.pcf, so that only users with the appropriate system permission can edit the page.**



3. **Restart server**

4. **In PolicyCenter, login as super user su/gw.**
5. **Go to Administration → Users & Security → Roles**
6. **Grant the new edit location permission to the Producer and Underwriter roles.**

   Edit → Add → Update



2.5 References

2.5.1 Creating new system permissions

System permissions are defined in the `SystemPermissionType` typelist. Add typecodes to create new system permissions.

Best Practice

Typecodes for `SystemPermissionType`

- Permission codes should be all lowercase without white spaces
- Specify one permission per action
- Name permission codes as a verb for the entity name and action
- Permission code: [entity][action] – can be interchanged

2.5.2 Checking permissions with Gosu expressions

perm is a Gosu namespace used to create expressions that determine if current user has a given permission. The expression returns true or false.

- perm.**System**.*permission* is used when user calls a system permission defined in the `SystemPermissionType` typelist.
- perm.**Entity**.*permission* is used when an application permission key is used.

2.5.3 Static and object-based permissions

An **Application Permission Key** (APK) is a set of one or more system permissions defined in security-config.xml.

- APKs use custom handlers to define the mapping of different objects in PolicyCenter to system permissions. For example, `StaticHandlers`, `WrapHandler`, `AccountProducerCodeHandler`, etc.
- PolicyCenter defines APKs for following objects: Account, PolicyPeriod, Jobs, Notes, Documents, Activities, Etc.

Static permissions do not require an object as an argument and it may or may not use APKs.

Object-based permissions always require an object as an argument and use APKs.

Syntax

- Static System Permissions

  perm.System.permission

  - permission: typecode of a permission defined in SystemPermissionType typelist.

- Static Permissions on Entities

  perm.Entity.permission

  - Entity: entity on which the permission is defined.
  - permission: permKey attribute defined in security-config.xml.
  - it is the APK Defined using StaticHandlers in security-config.xml.
- Object Based Permissions

  perm.Entity.permission (object)

  - Entity: entity on which the permission is defined.
  - permission: permKey attribute defined in security-config.xml.
  - object: the current object in memory
  - It is the APK defined using custom Handlers in security-config.xml.

Lesson 3

Creating Activities

3.1 Requirements

Create a new general activity pattern called “Inspect Vehicles”, due in 30 business days (from activity creation date), escalated in 40 business days (from activity creation date), normal priority, and Reminder category. You can reference this activity pattern to all assignable.

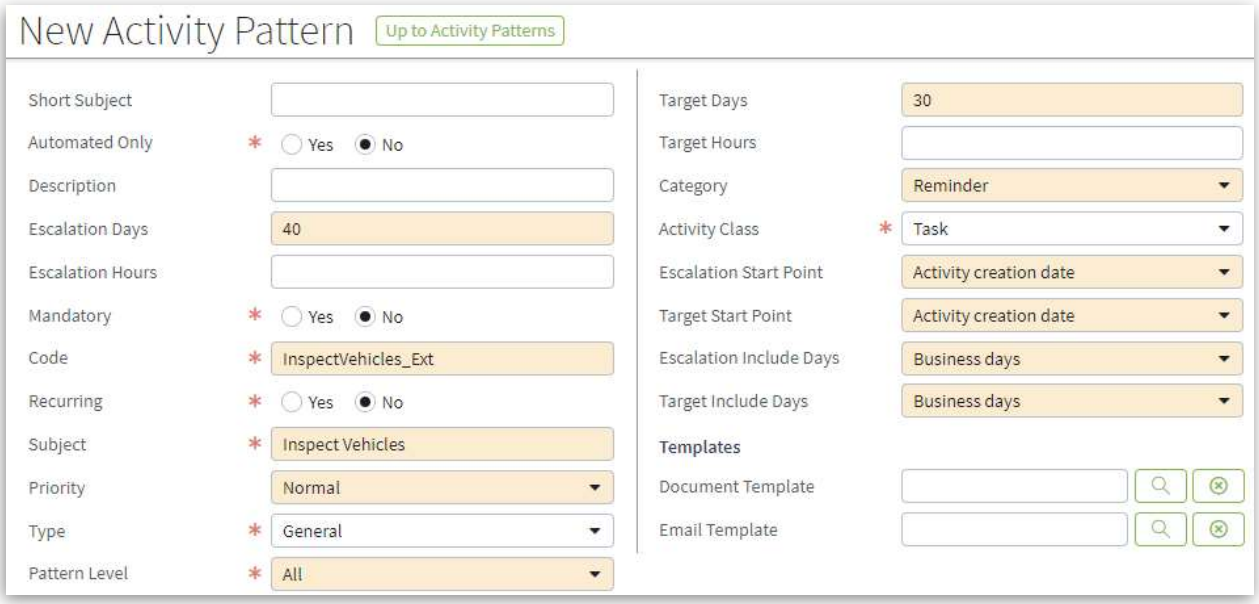
3.2 Configuration

Activity

Creating Activity Pattern

1. **Log in to PolicyCenter as super user (su/gw).**
2. **Go to Administration → Business Settings → Activity Patterns.**
3. **Click New Activity Pattern.**

3.3 Solution



3.4 References

3.4.1 Activity pattern

Activity Patterns are templates that standardize how activities are generated.

Activity patterns are typically created and modified by administrators on the page **Administration tab → Business Settings → Activity Patterns**.

1. **To create a new activity pattern, click the New Activity Pattern button**

   The Code field is a unique internal name for the pattern and is used to reference an activity pattern in Gosu

2. **To edit an existing activity pattern, click the name of the activity pattern in the list**

   Changes to an activity pattern affect only activities from that point forward

Find

Please refer to the PolicyCenter Application Guide for the definitions of each field of the Activity Pattern.

3.4.2 Create an activity in Gosu

1. **Retrieve an activity pattern by its pattern code**

```
ActivityPattern.finder.getActivityPatternByCode("pattern_code")
```

2. Create an activity related to an Account, a Policy or a Job

a) create an activity related to an Account

```
activityPattern.createAccountActivity(bundle, activityPattern, Account,
subject, description, approvalIssue, priority, mandatory, targetDate,
escalationDate)
```

or

```
account.newActivity(activityPattern)
```

b) create an activity related to a Policy

```
activityPattern.createPolicyActivity(bundle, policy, subject,
description, approvalIssue, priority, mandatory, targetDate,
escalationDate)
```

c) create an activity related to a Job

```
activityPattern.createJobActivity(bundle, job, subject, description,
approvalIssue, priority, mandatory, targetDate, escalationDate)
```

3. After creating the activity in Gosu, the next step is normally to assign it to a user or a queue.

The activity assignment is discussed in the next lesson.

Lesson 4

Assigning Activities

4.1 Requirements

For Commercial Auto submissions at Acme company, if the agent has not inspected the vehicles, create an activity to remind underwriter to inspect vehicles.

Specifications

Spec 1 In the Commercial Auto qualification questions, the default answer for question "Did agent inspect vehicles?" should be true. If the answer to this question is set to False, automatically generate an Inspect Vehicles and assign it to an underwriter.

Spec 2 The description should state: "For customer, <customer name>, inspect all vehicles."

Spec 3 The activity should be related to the account.

Spec 4 If the activity already exists, don't create another.

Spec 5 Use the activity pattern "Inspect Vehicles" created in the lab for the lesson Creating Activities.

4.2 Configuration

Activity

Creating and Assigning Activity

1. Create an activity and assign it to an underwriter.

a) Create a Job enhancement and add a function called `createInspectVehiclesActivity_Ext()`.

b) The code should only apply when the policy product is `BusinessAuto`.

HINT:

i. The question code for question "Did agent inspect vehicles?" is `AgentInspected`.

ii. Use the `createRoleActivity()` method found in the `JobAssignmentEnhancement` class. This method will create and assign an activity to an Underwriter.

2. Call the new function from the job wizard.

Call `createInspectVehiclesActivity_Ext()` from the `PreQualification` wizard step in the `SubmissionWizard.pcf`. Use the `beforeSave` property of the wizard step to call the code.

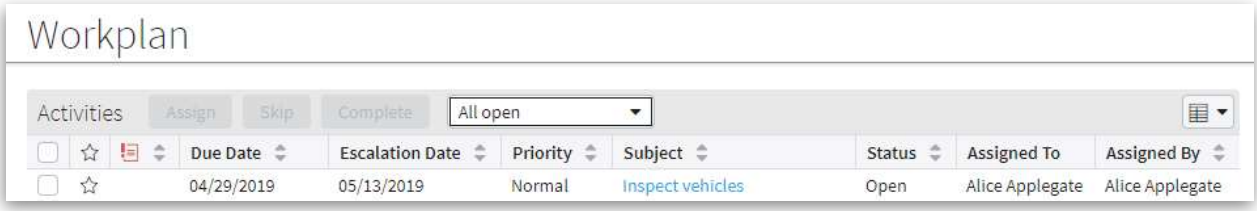
3. **Either reload PCF by clicking ALT + SHIFT + L in the browser or restart the server.**

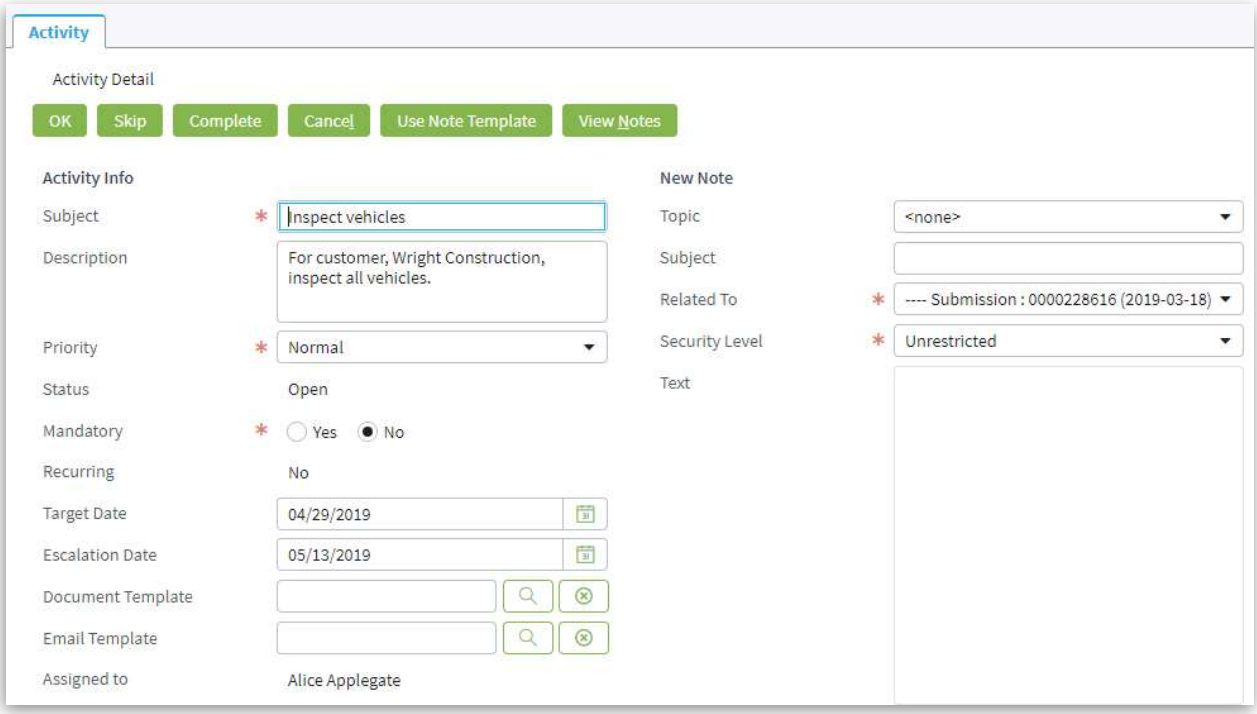
4.3 Verification

1. **Log in to Guidewire PolicyCenter as Alice Applegate aapplegate/gw.**
2. **Select any company account, e.g. Wright Construction, or create a new one.**
3. **Start or copy a Commercial Auto submission.**
4. **Answer "No" to the question "Did agent inspect vehicles?".**
5. **After you click "Next," click on the Workplan link under Tools.**

   The activity should be displayed and assigned to the submission Underwriter. Verify description field is populated correctly.

6. **Go back to Qualification step and click "Next". Make sure a second Inspect Vehicles activity was not created.**





GUIDEWIRE

4.4 Solution

1. Create an activity and assign it to an underwriter in a new Job enhancement.

`JobEnhancement.gsx`

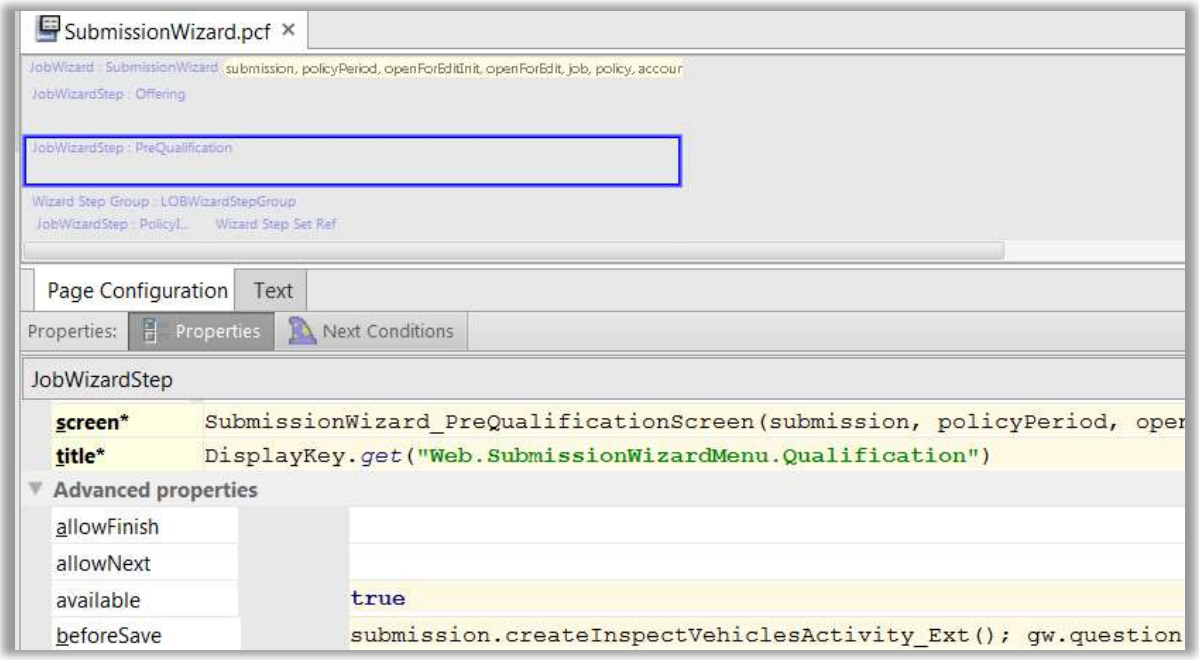
```
package gw.acme.pc.enhancements.entity

uses gw.api.locale.DisplayKey

enhancement JobEnhancement: Job {
 function createInspectVehiclesActivity_Ext() : void {
  var isBusinessAuto = this.Policy.ProductCode == "BusinessAuto"
  var answerToAgentInspected = this.LatestPeriod.PeriodAnswers.firstWhere(\q ->
q.QuestionCode ==  "AgentInspected" ).BooleanAnswer
  var activityExists = this.AllOpenActivities.hasMatch(\act -> act.ActivityPattern.Code
== "Inspect_vehicles_Ext")

  if( isBusinessAuto && !answerToAgentInspected && !activityExists ) {
   var pattern = ActivityPattern.finder.getActivityPatternByCode("Inspect_vehicles_Ext")
   var description = DisplayKey.get("Ext.InspectVehicles",
this.LatestPeriod.PrimaryNamedInsured.DisplayName)

   this.createRoleActivity(typekey.UserRole.TC_UNDERWRITER, pattern, null, description)
  }
 }
}
```

`display.properties`

```
Ext.InspectVehicles = For customer, {0}, inspect all vehicles.
```

2. Call the new function from the job wizard.

Call `createInspectVehiclesActivity_Ext()` from the `PreQualification` wizard step in the `SubmissionWizard.pcf`. Use the `beforeSave` property of the wizard step to call the code.



3. **Either reload PCF by clicking ALT + SHIFT + L in the browser or restart the server.**

4.5 References

4.5.1 Assignment

1. **Guidewire refers to certain business entities as assignable entities. There are two types of assignable entities: Primary vs Role Based.**
   - **Primary** assignment (discussed in this lesson): The entity can have a single owner only. In the base configuration, this applies to Activity entities. To be primarily assignable, an entity must implement Assignable and PCAssignable delegates, which provide methods and fields needed for assignment.
   - **Role-based** assignment: The entity has a set of users – with different roles – assigned to it. In the base configuration, this applies to Account, Job (and its subtypes), and Policy entities.
2. **An activity can be assigned to a user or a queue. Under no circumstances should you configure the application so that an entity is assigned at the same time to both a queue and a user.**

   When an activity is assigned to a user, it is considered owned by both the user and the group. It appears in the user's My Activities desktop list.
3. **An activity is assigned either by Assignment rules or assignment classes or combination of both.**
4. **Assignment rules:**
   - `GlobalActivityAssignmentRules` and `DefaultGroupActivityAssignmentRules`.
   - Calling the `Activity.autoAssign()` and `Activity.autoAssign(group, null)` in the Gosu code will trigger the assignment rules to be run.

Note: Never call the `autoAssign()` methods in the assignment rules. If `userId` is not null in the call to `autoAssign(group, user)`, no assignment rules are run, and the activity is assigned to the specified user directly.

- The assignment rules can then call other Activity assignment methods to construct the assignment logic, such as round robin to a user in a group based on location.

5. **Assignment classes**
   - Some files that contain assignment logics in the default configuration.

     `gsrc/gw/assignment: AssignmentUtil, AuditAssignmentEnhancement, JobAssignmentEnhancement`

   - The assignment classes can call the Activity assignment methods to initiate assignment logic.
6. **Common assignment strategies**
   - Assign to a specific group and then round-robin among users
     - Round robin ignores users that lack permission to own the object
     - Round-robin functionality can be configured to take into account user workload/load factors
   - Assign to a specific group and user
   - Assign to a queue
7. **Common assignment methods on the Activity entity**
   - Assignment methods return boolean values

     True: assignment was successful

     False: unsuccessful, appropriate group (or user) could not be found

Context	Method	Assigns to
Group	assignGroup	Named group
Group	assignGroupByLocation	Group matching specified location and group type
User	assign()	Named user (and group)
User	assignUserAndDefaultGroup()	Named user (default group)
Queue	assignActivityToQueue()	Named Queue

Lesson 5

The Job Lifecycle

5.1 Requirements

Examine the PolicyChange process in the base application.

5.2 Lab

Investigation

1. **Find the components to examine:**
   a) Log in to PolicyCenter as Alice Applegate (aapplegate/gw).
   b) Select any policy and start a Policy Change transaction.
   c) On the very first page, press `Alt+Shift+I` to get information about the PCF files.
   d) What is the name of this page?
   e) Close the Location Info window and click Next. On the Offerings screen, press `Alt+Shift+I` to get information about the PCF files.
   f) What is the name of the wizard?
   g) What is the name of the button set?
   h) Close the Location Info window. How many buttons are visible in the UI?
   i) Withdraw the transaction. Note the message that appears.
2. **Locate these components in Studio:**
   a) If necessary, start Guidewire PolicyCenter Studio.
   b) Navigate to `configuration/config/Page Configuration/pcf/job/policychange`
3. **Examine the `StartPolicyChange` page.**
   a) How many entry points does this page have, and what variable must be passed to it?
   b) Where and how does the `effectiveDate` variable get set? (Two answers.)
   c) What does the `Next` button do?

d) What is its visibility property?

4. Examine the buttons in the Policy Change toolbar button set:

a) What is the PCF file name or the button set?

b) Open the button set. How many buttons are visible?

c) Note that two buttons have the same button text. What is the difference between these two buttons? Will they ever appear at the same time in the UI?

d) What is the action for the Issue Policy button?

e) Drill down on this action. Where does it reside (i.e., what object is it associated with)?

f) When a user clicks the Withdraw Transaction button, what causes the confirmation message to appear? (Hint: Two properties are involved.)

5. Examine how the base application works with the user to handle preemptions:

a) What is the visibility property of the Handle Preemption button?

b) Drill down on the "`can`" function. What conditions does it check for?

c) Return to the toolbar. What is the action for the `Handle Preemption` button?

d) Drill down on the popup. What is the action for the `Apply All Changes` button?

e) Drill down on this action. Where does the script reside?

f) In the `applyChanges` function, locate the `handlePreemptions` function. Where does it reside and what does it do?

5.3 Solutions

Solution

1. Find the components to examine:

a) Log in to PolicyCenter as Alice Applegate (aapplegate/gw).

b) Select any policy and start a Policy Change transaction.

c) On the very first page, press `Alt+Shift+I` to get information about the PCF files.

d) What is the name of this page?

`StartPolicyChange.pcf`

e) Close the Location Info window and click Next. On the Offerings screen, press `Alt+Shift+I` to get information about the PCF files.

f) What is the name of the wizard?

`PolicyChangeWizard.pcf`

g) What is the name of the button set?

`JobWizardTooldbarButtonSet.PolicyChange.pcf`

h) Close the Location Info window. How many buttons are visible in the UI?

Depends on policy – typically, 4 or 5

i) Withdraw the transaction. Note the message that appears.

2. Locate these components in Studio:

a) If necessary, start Guidewire PolicyCenter Studio.

b) Navigate to `configuration/config/Page Configuration/pcf/job/policychange`

3. Examine the `StartPolicyChange.pfc file`.

a) How many entry points does this page have, and what variable must be passed to it?

One entry point, which requires a PolicyPeriod variable

b) Where and how does the `effectiveDate` variable get set? (Two answers.)

i. This variable is set in the Variables tab. The initial value is set to the `EditEffectDate` of the latest bound policy period. (For example, if the `EditEffectiveDate` of the first issued PolicyChange is 3 months after the Policy's effective date, then the second PolicyChange will have the initial `EditEffectiveDate` as 3 months after the Policy's effective date.)

ii. After that the user can change it using the `DateTimeInput` widget "Effective Date". If the date is outside of the policy term's date range, an error message is displayed.

c) What does the `Next` button do?

- *Starts a new PolicyChange job and commits the database transaction*
- *calls the `applyEffectiveTimePluginForPolicyChange` function*
- *then navigates to the PolicyChange Wizard*

d) What is its visibility property?

It is set in the `canVisit` property with permissions to view the policy period and to create policy change.

4. Examine the buttons in the Policy Change toolbar button set:

a) What is the PCF file name or the button set?

`JobWizardToolbarButtonSet.PolicyChange.pcf`

b) Open the button set. How many buttons are visible?

10

c) Note that two buttons have the same button text. What is the difference between these two buttons? Will they ever appear at the same time in the UI?

There are two buttons for "Apply change to renewal".

One applies changes to an unbound renewal, the other to a bound renewal. It is visibility is controlled by conditions in the visible property.

They will not appear at the same time in the UI.

d) What is the action for the Issue Policy button?

`jobWizardHelper.requestIssueJob(policyPeriod)`

e) Drill down on this action. Where does it reside (i.e., what object is it associated with)?

It resides in the `JobWizardHelperEnhancement.gsx` *entity.*

f) When a user clicks the Withdraw Transaction button, what causes the confirmation message to appear? (Hint: Two properties are involved.)

1) A display key listed in the `confirmMessage` *property.*

2) The `showConfirmMessage` *property is set to true.*

5. Examine how the base application works with the user to handle preemptions:

a) What is the visibility property of the Handle Preemption button?

`policyChangeProcess.canHandlePreemptions().Okay`

b) Drill down on the "`can`" function. What conditions does it check for?

`HasUnhandledPreemptions` *and* `BranchNotLocked`

c) Return to the toolbar. What is the action for the `Handle Preemption` button?

`HandlePreemptionPopup.push(wizard, jobWizardHelper, policyPeriod, true)`

d) Drill down on the popup. What is the action for the `Apply All Changes` button?

`applyChanges(wizard, jobWizardHelper, policyPeriod, CurrentLocation)`

e) Drill down on this action. Where does the script reside?

`HandlePreemptionPopupUIHelper.gs`

f) In the `applyChanges` function, locate the `handlePreemptions` function. Where does it reside and what does it do?

It is a method of `JobProcess.gs` *and (according to the comments) it resolves unhandled preemptions and returns the new policy period.*

5.4 References

5.4.1 The job lifecycle

1. **Below are some of the special job components:**

- **<Jobtype> entities**: contain required data for a policy transaction. Job is the parent entity and subtypes include Submission, PolicyChange, Cancellation, Reinstatement, Renewal, Rewrite, Audit, Reinsurance, etc.
- **Branches**: are sets of objects related to one `PolicyPeriod` object. A branch is one version of the Policy.
- **<Jobtype>Process** classes: contain methods required for processing the job. `JobProcess` is the parent class. Sub classes include `SubmissionProcess`, `PolicyChangeProcess`, `CancellationProcess`, `RenewalProcess`, `AuditProcess`, etc. They are located in `configuration/gsrc/gw/job`.
- **<Jobtype>Wizards**: contain screen, tools and buttons that work with the user in moving through the life cycle of the job.
- **Job Wizard Helper**: is a java class that provides functionality for job wizards.
- **Job Wizard Button Set**: contains relevant buttons for a job type in the job wizard.

2. **Jobs can be started in three ways:**
   - By users through the Actions menu
   - By external systems through APIs
   - Through PolicyCenter batch processes
3. **When the job is started in the user interface, the <Jobtype> object is instantiated by the screen that calls the job wizard.**
4. **A series of job conditions can be checked before the business logic is performed.**

   Typically, they are the can methods in the job process classes. The job wizard button visibility is also based on these can methods, which return a `JobConditions` object. The `JobConditions` object contains a string builder called `_message`. If the `_message` string remains empty after all the error condition checks, the derived property `Okay` returns true, otherwise false.
5. **The business logic can be executed, for example, edit, request quote, issue, etc. from the wizard buttons, the job process methods and workflows. The workflow maybe started in the job process methods as well.**
6. **When the job is bound (or issued), withdrawn, discarded, its life ends. The job object remains in the database and can be retrieved for reference. But it cannot be edited.**