GUIDEWIRE

BillingCenter 10 Configuration: Kickstart

Student Workbook

Labs and Tutorials

Document Published: 2020-07-15

Table of Contents

Introduction

Welcome to the Guidewire BillingCenter 10.0 Configuration Kickstart 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.

Lesson 1

Configuration Basics

1.1 Prerequisites

For this lab, use BillingCenter 10.0 Education Installer, Guidewire Studio, and a supported web browser. http://localhost:8580/bc/BillingCenter.do is the default URL for BillingCenter. To test configuration changes, log in to BillingCenter as Super User. The login/password for Super User is su/gw.

1.2 Load sample data

In this lab, you practice the same steps that you performed during the demo. Here are the steps you need to complete:

1. **Reset your database to an empty BillingCenter database. (HINT: Use Studio to drop your existing BillingCenter database and then start Server.)**
2. **Use Server Tools (Alt+Shift+T) to load the BillingCenter sample data. (Internal Tools → BC Sample Data)**

   a) How many accounts were created when the sample data was loaded?

3. **Use the Quick Jump to view a list of the data builder methods that are available for an account. (Run Account ListAllMethods)**
4. **Use the Quick Jump to create a new account that has one policy and no producer. (Run Account with1PolicyWithNoProducer)**
5. **Reset your BillingCenter database again.**
6. **Use the Excel Data Loader to load the Admin, Plan, and Sample data from the configuration. Refer the Cookbook Recipe: Steps for using the Data Loader if needed.**
7. **Load the additional sample data from TrainingSampleConfigData.xls. This file is in the C:\GW10\BillingCenter\modules\configuration\config\datafiles folder.**

1.2.1 Solution

1. **Reset your database to an empty BillingCenter database. (HINT: Use Studio to drop your existing BillingCenter database and then start Server.)**
   a) Stop server if it is running
   b) In the Studio toolbar, change the 

dropdown to 

   c) From the toolbar, click the  to run the DropDB command.
   d) Once the DropDB command completes, change the dropdown back to Server and click the  to start server.

Note: You may also use the Run file menu to most of these actions.

2. **Use Server Tools (Alt+Shift+T) to load the BillingCenter sample data. (Internal Tools → BC Sample Data)**
   a) How many accounts were created when the sample data was loaded?

      2

3. **Use the Quick Jump to view a list of the data builder methods that are available for an account. (Run Account ListAllMethods)**
4. **Use the Quick Jump to create a new account that has one policy and no producer. (Run Account with1PolicyWithNoProducer)**
5. **Reset your BillingCenter database again.**
6. **Use the Excel Data Loader to load the Admin, Plan, and Sample data from the configuration.**
7. **Load the additional sample data from TrainingSampleConfigData.xls. This file is in the C:\GW10\BillingCenter\modules\configuration\config\datafiles folder.**

1.3 Configure invoice fee behavior

Succeed Insurance uses variable invoice fees for some of their accounts. In this lab, you will implement changes to BillingCenter to support their requirements for variable invoice fees.

1.3.1 Requirements

Spec 1 Variable Invoice fees (on the account billing plan) are calculated as percentage rate of the invoice amount.

Spec 2 The invoice fee rate should be specified on the billing plan

Spec 3 A flag is needed on the billing plan to indicate if the default invoice fee should be overridden by the variable invoice fee

1.3.2 Tasks

1. Add new fields to the billing plan

Be sure to follow best practices, by following the data model extension naming conventions.

a) Extend the BillingPlan entity

b) Add a new percentagedec element to hold the variable invoice fee rate.

c) Add a new bit element to indicate if the default invoice fee should be overridden by the variable invoice fee.

d) Regenerate the data dictionary

e) Deploy data model changes

2. Update the UI to display the new fields on the Billing Plan

Be sure to follow best practices, by creating display keys for the labels.

a) Add the new bit element to the BillingPlanFeeHandlingInputSet.default.pcf. Set the label for this field to "Use Variable Rate Invoice Fee".

b) Add the new percentagedec element. Set the label for this field to "Variable Invoice Fee Rate".

3. Modify the Fees and Thresholds plugin to implement the business requirements

Be sure to follow best practices, by commenting out the existing code.

a) Open the Fees and Thresholds plugin.

b) Update getInvoiceFeeAmountOverride ()method in FeesThresholds.gs plugin.

c) Deploy code changes.

1.3.3 Testing procedure

1. Clone the BP01 billing plan.

Field	Value
Name	Variable Invoice Fee Test
Use Variable Rate Invoice Fee	Yes
Variable Invoice Fee Rate	0.25%

2. Manually create an account that uses your new billing plan.

Field	Value
Account Name	Invoice Fee Test
Billing Plan	Variable Invoice Fee Test
Select any value for remaining required fields	

3. Add a policy to the account.

Field	Value
Policy #	IATT01
Payment Plan	PP02
Premium	$5000

4. Advance the system clock to the invoice date of the first invoice.
5. Run the invoice batch process.
6. In BillingCenter, go to the Account tab → Charges screen and confirm that one invoice fee of $1.25 was added to the billed invoice.

1.3.4 Solution

1. Add new fields to the billing plan

Be sure to follow best practices, by following the data model extension naming conventions.

a) Extend the BillingPlan entity

In Studio, go to configuration → config → Extensions → Entity. Then right click on Entity, select New → Entity Extension. In the Entity Extension window, enter "BillingPlan" and then click the OK button.

b) Add a new percentagedec element to hold the variable invoice fee rate.

Add a new column to the BillingPlan.etx. As listed below:

Name: InvoiceFeeRate_Ext

Type: percentagedec

Nullok: true

Default: 0

c) Add a new bit element to indicate if the default invoice fee should be overridden by the variable invoice fee.

Add a new column to the BillingPlan.etx. As listed below:

Name: VariableInvoiceFee_Ext

Type: bit

Nullok: true

Default: false

d) Regenerate the data dictionary

From a command line, execute the gwb genDataDictionary command.

e) Deploy data model changes

Restart server

2. Update the UI to display the new fields on the Billing Plan

Be sure to follow best practices, by creating display keys for the labels.

a) Add the new bit element to the BillingPlanFeeHandlingInputSet.default.pcf.. Set the label for this field to “Use Variable Rate Invoice Fee”.

Add a new Boolean Radio Button Input, below the PlanMultiCurrencyFeeThresholdInputSet. Set the editable field to call the planNotInUse method. Set the id to UseVariableRateInvoiceFee. Set the label to DisplayKey.get("Ext.UseVariableRateInvoiceFee"). Create a display key for this label. Set the value to billingPlan.VariableInvoiceFee_Ext.

b) Add the new percentagedec element. Set the label for this field to “Variable Invoice Fee Rate”.

Add a new Text Input, below the pcf element you just added. Set the editable field to call the planNotInUse method. Set the id to VariableInvoiceFeeRate. Set the label to DisplayKey.get("Ext.VariableInvoiceFeeRate"). Create a display key for this label. Set the value to billingPlan.InvoiceFeeRate_Ext. Set the valueType to java.math.BigDecimal.

3. Modify the Fees and Thresholds plugin to implement the business requirements

Be sure to follow best practices, by commenting out the existing code.

a) Open the Fees and Thresholds plugin.

In Studio, open the FeesThresholds.gs class.

b) Update getInvoiceFeeAmountOverride ()method in FeesThresholds.gs plugin.

Comment out the existing method and the add the following code to this class.

```
override function getInvoiceFeeAmountOverride(defaultAmount : MonetaryAmount,
                  invoice : AccountInvoice) : MonetaryAmount {
             return invoice.Account.BillingPlan.VariableInvoiceFee_Ext ?
                 invoice.Amount.multiply(invoice.Account.BillingPlan.InvoiceFeeRate_Ext / 100) : defaultAmount
}
```

c) Deploy code changes.

In Studio, select Run → Reload Changed Classes

Lesson 2

Configuring Charge Invoicing Behaviors

2.1 Bill an item immediately

Succeed Insurance wants to customize when deposits are billed based on the policy issuance date.

2.1.1 Requirements

Spec 1 If the effective date of a policy issuance billing instruction is earlier than today, then bill the down payment immediately.

Spec 2 Not applicable to Agency Bill processing

2.1.2 Tasks

1. **Configure the customizeChargeInitializer() method in ChargeInitializer plugin.**
2. **Compile your changes**

2.1.3 Testing procedure

1. **Create a new account**
   a) Use quick jump to execute the Run Account command
2. **Add a policy to the new account with these details: (Actions → Add Policy)**
   a) Policy Number: BillToday 1
   b) Effective Date: 5 days earlier than the BillingCenter clock date
   c) Expiration Date: 1 year after the effective date
   d) Payment Plan: PP03
   e) Premium: $2500
   f) Taxes: $220
3. **Go to the Invoice screen (Account → Invoices) and verify that:**
   a) The premium down payment is invoiced on the same day as the BillingCenter clock date.
   b) The installment premium and taxes are charged on the second invoice (that is, the first regularly scheduled invoice)
4. **Add a policy to the new account with these details: (Actions → Add Policy)**
   a) Policy Number: BillToday 2

b) Effective Date: 5 days after than the BillingCenter clock date

c) Expiration Date: 1 year after the effective date

d) Payment Plan: PP03

e) Premium: $600

f) Taxes: $45

5. Go to the Invoice screen (Account → Invoices) and verify that:

a) Down payment and taxes are on the same invoice.

2.1.4 Solution

1. Configure the customizeChargeInitializer() method in ChargeInitializer plugin.


```
package gw.plugin.charge

uses gw.api.util.DateUtil

@Export
public class ChargeInitializer implements IChargeInitializer {

  public override function customizeChargeInitializer(initializer :
gw.api.domain.charge.ChargeInitializer) {
    // Invoice Plugins - Bill an Item Immediately lab
    var isBackDatedPolicy =
DateUtil.compareIgnoreTime(initializer.BillingInstruction.EffectiveDate,
                           DateUtil.currentDate()) < 0
    var isDirectBillIssuance = !initializer.AgencyBill and initializer.BillingInstruction typeis
Issuance
    if (isBackDatedPolicy and isDirectBillIssuance) {
      var deposit = initializer.Entries.firstWhere(\entry -> entry.InvoiceItemType ==
InvoiceItemType.TC_DEPOSIT)
      if(deposit != null)
        deposit.billToday()
    }
  }

}
```

2. Compile your changes

a) Execute Run → Reload Changed Classes

2.2 Demo code: Splitting the down payment

The following code is from the instructor demo:

```
package gw.plugin.charge
@Export
public class ChargeInitializer implements IChargeInitializer {

  public override function customizeChargeInitializer(initializer :
gw.api.domain.charge.ChargeInitializer) {
    //Insert Code here to modify the ChargeInitializer's Entries that will be used for creation
and placement of InvoiceItems.

    var isDirectBillIssuance = !initializer.AgencyBill and initializer.BillingInstruction typeis
Issuance
    if(isDirectBillIssuance) {
      var paymentPlan = initializer.PolicyPeriod.PaymentPlan
      if(paymentPlan.SplitDownPayment_Ext
          and paymentPlan.DownPaymentPercent >= 20
          and paymentPlan.Periodicity == Periodicity.TC_MONTHLY) {
        // find amounts
        var firstDownPaymentItem = initializer.Entries.firstWhere(\entry -> entry.InvoiceItemType
== InvoiceItemType.TC_DEPOSIT)
        var totalDownPaymentAmount = firstDownPaymentItem.Amount
        var newFirstDownPaymentAmount = totalDownPaymentAmount / 2
        var newSecondDownPaymentAmount = totalDownPaymentAmount - newFirstDownPaymentAmount
        var newSecondDownEventDate = firstDownPaymentItem.EventDate.addMonths(1)
        // set new amounts
        firstDownPaymentItem.Amount = newFirstDownPaymentAmount
        initializer.addEntry(newSecondDownPaymentAmount, InvoiceItemType.TC_DEPOSIT,
newSecondDownEventDate)
      }
    }
  }
}
```

Lesson 3

Configuring Invoice Streams

3.1 Implement a new payment interval

Succeed Insurance wants the ability to invoice some customers every five months.

3.1.1 Requirements

Spec 1 Policies are invoiced every five months.

3.1.2 Tasks

1. **Add *everyfivemonths* to Periodicity typelist.**
   a) Restart server to deploy database changes.
2. **Configure createPeriodicSequenceWith() method in *DateSequence* plugin to add new periodicity.**
3. **Configure getInvoiceStreamPeriodicityFor() method in *InvoiceStream* plugin.**
   a) Compile code changes

3.1.3 Testing procedure

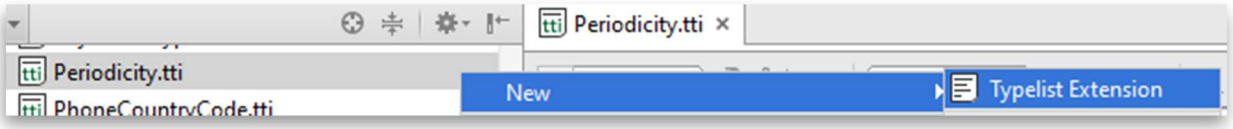
1. **Clone the PP03 payment plan. *(Administration → Business Settings → Payment Plans)***
   a) Name: Every Five Months Plan
   b) Payment Interval: Every Five Months
2. **Create an account. (QuickJump: Run Account)**
3. **Add a policy to the new account with these details: *(Actions → Add Policy)***
   a) Policy Number: EFM
   b) Expiration Date: 2-years after effective date
   c) Payment Plan: Every Five Months Plan
   d) Premium: $1300
4. **Go to the Invoice screen (Account → Invoices) and verify that:**
   a) The monthly invoice stream was used for the invoices.
   b) The invoice periodicity is *every five months*.

3.1.4 Solution

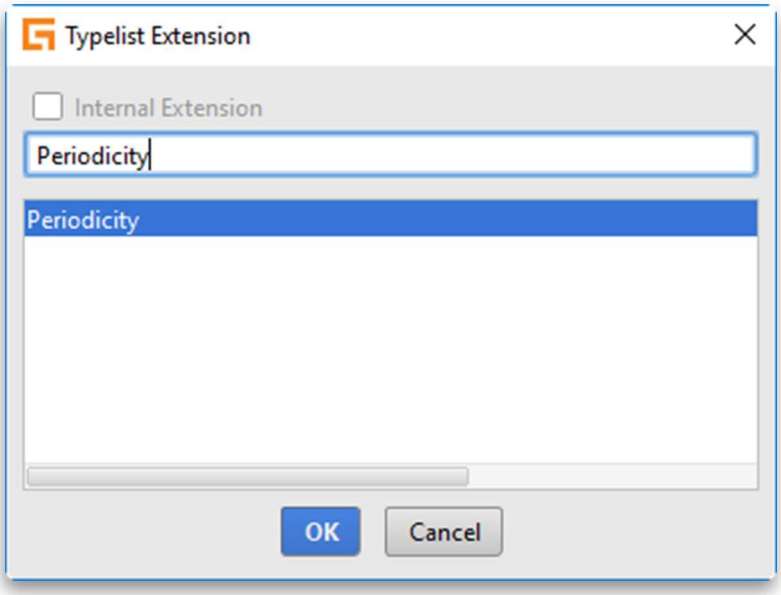
1. Add everyfivemonths to Periodicity typelist.

a) In Studio, go to configuration → config → Extensions → Typelist

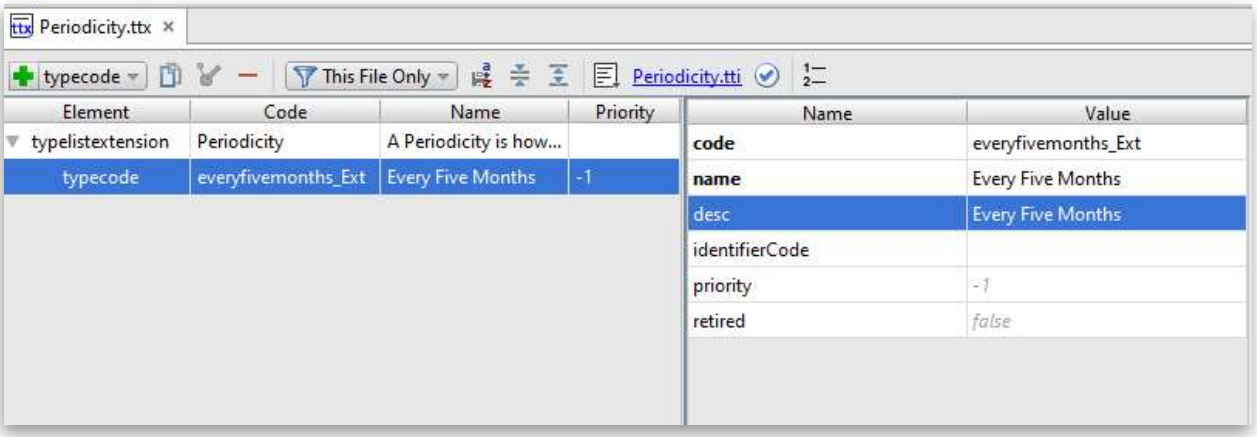
b) Right click on Typelist, select New → Typelist Extension



c) In the Typelist Extension window, enter "Periodicity" and then click the OK button.

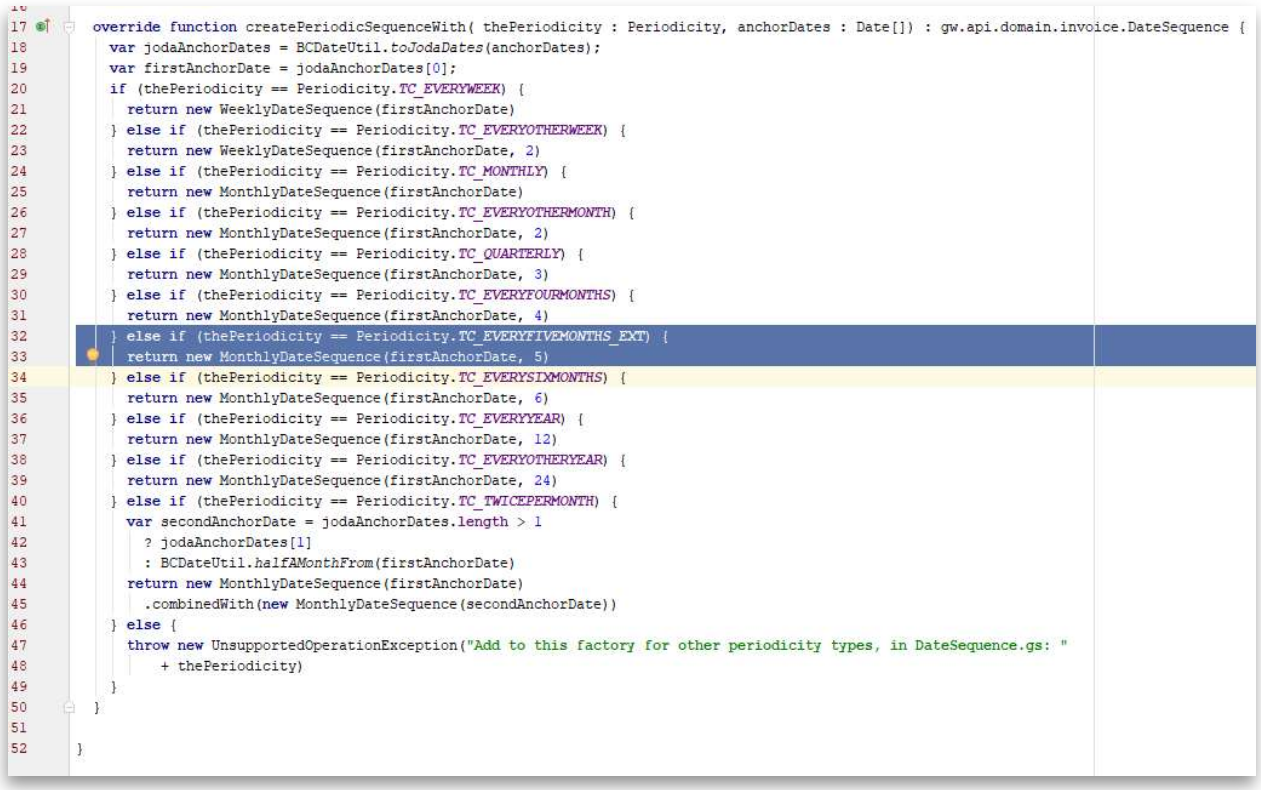


d) Add new typecode value



e) Restart server to deploy database changes.

2. Configure createPeriodicSequenceWith() method in *DateSequence* plugin to add new periodicity.



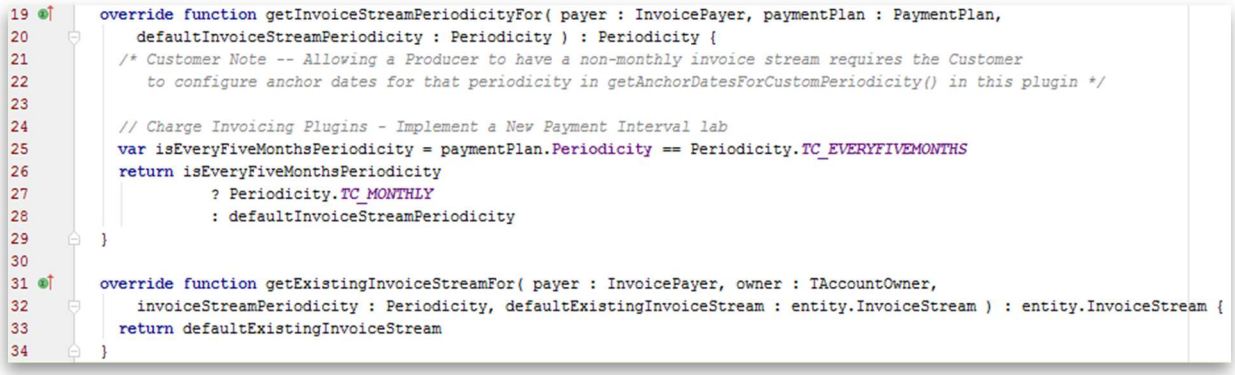
```
override function createPeriodicSequenceWith( thePeriodicity : Periodicity, anchorDates : Date[]) : gw.api.domain.invoice.DateSequence {
  var jodaAnchorDates = BCDateUtil.toJodaDates(anchorDates);
  var firstAnchorDate = jodaAnchorDates[0];
  if (thePeriodicity == Periodicity.TC_EVERYWEEK) {
    return new WeeklyDateSequence(firstAnchorDate)
  } else if (thePeriodicity == Periodicity.TC_EVERYOTHERWEEK) {
    return new WeeklyDateSequence(firstAnchorDate, 2)
  } else if (thePeriodicity == Periodicity.TC_MONTHLY) {
    return new MonthlyDateSequence(firstAnchorDate)
  } else if (thePeriodicity == Periodicity.TC_EVERYOTHERMONTH) {
    return new MonthlyDateSequence(firstAnchorDate, 2)
  } else if (thePeriodicity == Periodicity.TC_QUARTERLY) {
    return new MonthlyDateSequence(firstAnchorDate, 3)
  } else if (thePeriodicity == Periodicity.TC_EVERYFOURMONTHS) {
    return new MonthlyDateSequence(firstAnchorDate, 4)
  } else if (thePeriodicity == Periodicity.TC_EVERYFIVEMONTHS_EXT) {
    return new MonthlyDateSequence(firstAnchorDate, 5)
  } else if (thePeriodicity == Periodicity.TC_EVERYSIXMONTHS) {
    return new MonthlyDateSequence(firstAnchorDate, 6)
  } else if (thePeriodicity == Periodicity.TC_EVERYYEAR) {
    return new MonthlyDateSequence(firstAnchorDate, 12)
  } else if (thePeriodicity == Periodicity.TC_EVERYOTHERYEAR) {
    return new MonthlyDateSequence(firstAnchorDate, 24)
  } else if (thePeriodicity == Periodicity.TC_TWICEPERMONTH) {
    var secondAnchorDate = jodaAnchorDates.length > 1
      ? jodaAnchorDates[1]
      : BCDateUtil.halfAMonthFrom(firstAnchorDate)
    return new MonthlyDateSequence(firstAnchorDate)
      .combinedWith(new MonthlyDateSequence(secondAnchorDate))
  } else {
    throw new UnsupportedOperationException("Add to this factory for other periodicity types, in DateSequence.gs: "
        + thePeriodicity)
  }
}

}
```

```
      } else if (thePeriodicity == Periodicity.TC_EVERYFIVEMONTHS_EXT) {
  return new MonthlyDateSequence(firstAnchorDate, 5)
```

3. Configure getInvoiceStreamPeriodicityFor() method in *InvoiceStream* plugin.

a) Update the code as shown below



```
override function getInvoiceStreamPeriodicityFor( payer : InvoicePayer, paymentPlan : PaymentPlan,
    defaultInvoiceStreamPeriodicity : Periodicity ) : Periodicity {
  /* Customer Note -- Allowing a Producer to have a non-monthly invoice stream requires the Customer
     to configure anchor dates for that periodicity in getAnchorDatesForCustomPeriodicity() in this plugin */

  // Charge Invoicing Plugins - Implement a New Payment Interval lab
  var isEveryFiveMonthsPeriodicity = paymentPlan.Periodicity == Periodicity.TC_EVERYFIVEMONTHS
  return isEveryFiveMonthsPeriodicity
          ? Periodicity.TC_MONTHLY
          : defaultInvoiceStreamPeriodicity
}

override function getExistingInvoiceStreamFor( payer : InvoicePayer, owner : TAccountOwner,
    invoiceStreamPeriodicity : Periodicity, defaultExistingInvoiceStream : entity.InvoiceStream ) : entity.InvoiceStream {
  return defaultExistingInvoiceStream
}
```

```
      override function getInvoiceStreamPeriodicityFor( payer : InvoicePayer, paymentPlan :
PaymentPlan,
    defaultInvoiceStreamPeriodicity : Periodicity ) : Periodicity {
  /* Customer Note -- Allowing a Producer to have a non-monthly invoice stream requires the
Customer
     to configure anchor dates for that periodicity in getAnchorDatesForCustomPeriodicity() in
```

```
this plugin */

  // Charge Invoicing Plugins - Implement a New Payment Interval lab
  var isEveryFiveMonthsPeriodicity = paymentPlan.Periodicity ==
Periodicity.TC_EVERYFIVEMONTHS_EXT
  return isEveryFiveMonthsPeriodicity
            ? Periodicity.TC_MONTHLY
            : defaultInvoiceStreamPeriodicity
}
```

b) Compile code changes. *(Run → Reload Changed Classes)*

3.2 Demo code: Configure separate invoice stream for quarterly policies

The following code is from the instructor demo:

```
       override function getInvoiceStreamPeriodicityFor( payer : InvoicePayer, paymentPlan :
PaymentPlan,
    defaultInvoiceStreamPeriodicity : Periodicity ) : Periodicity {
  /* Customer Note -- Allowing a Producer to have a non-monthly invoice stream requires the
Customer
     to configure anchor dates for that periodicity in getAnchorDatesForCustomPeriodicity() in
this plugin */

  // Invoice Stream Demo
  var isQuarterlyPeriodicity = paymentPlan.Periodicity == Periodicity.TC_QUARTERLY
  return isQuarterlyPeriodicity
            ? Periodicity.TC_QUARTERLY
            : defaultInvoiceStreamPeriodicity
}
       override function getAnchorDatesForCustomPeriodicity( invoicePayer : InvoicePayer,
customPeriodicity : Periodicity )
    : List<AnchorDate> {
  // Invoice Stream Demo
  var strDate = "01/05/2000"
  var dtf = DateTimeFormat.forPattern("MM/dd/YYYY") // format for input
  var jodaDate = dtf.parseDateTime(strDate)         // parsing the date

  return customPeriodicity == Periodicity.TC_QUARTERLY
            ? new ArrayList<AnchorDate>() {AnchorDate.fromDate(jodaDate)}  // year does not
matter
            : new ArrayList<AnchorDate>()                                  // default return
value
}
```

Lesson 4

Configuring Activities

4.1 Create a shared activity

When a producer's tier is changed, create a shared activity to contact the producer to verify the change.

4.1.1 Requirements

Spec 1 Create an activity pattern with the following settings:

- Subject = Contact Producer
- Category = Reminder
- Code = contact_producer
- Priority = Normal
- Mandatory = No
- There is no need to provide any activity pattern dates.

Spec 2 When a producer's tier is changed, create a shared activity that uses the contact_producer activity pattern and includes:

- Subject: Contact <producer_name> re: Change in Tier
- Description: Tier changed from <previous value> to <new value>
- No activity should be created for a new producer.

The project team recommends using a preupdate rule to implement the functionality of this requirement.

4.1.2 Tasks

1. **Create a new activity pattern.**
2. **Create a preupdate rule.**
3. **Restart the server because a new rule was created.**

4.1.3 Testing procedure

1. **Create a new producer and verify that a shared activity was not created on the user desktop.**
   a) QuickJump: Run Producer
2. **Edit the producer and change the producer's tier.**
3. **Verify that a shared activity was created on the user desktop and satisfies the requirements.**

4.1.4 Solution

1. **Create a new activity pattern.**
   a) Subject = Contact Producer
   b) Category = Reminder
   c) Code = contact_producer
   d) Priority = Normal
   e) Mandatory = No
2. **Create a preupdate rule.**

3. **Restart the server because a new rule was created.**

Lesson 5

Configuring Trouble Tickets

5.1 Create a trouble ticket

If a policy is renewed and overdue invoices remain from the prior policy period, a trouble ticket should automatically be created to contact the customer.

5.1.1 Requirements

Spec 1 A trouble ticket should be created when a policy period is renewed and the prior policy period is delinquent.

Spec 2 This applies to policies that are not agency bill.

Spec 3 The trouble ticket should:

- Be associated with the account that owns the policy
- Be of type “Automatically Generated”
- Have “Urgent” priority
- Have a title stating “Policy <policy number> is being renewed and there are overdue invoice items"
- Have a detailed description stating “Customer has overdue installments on current policy period. Please follow up with the customer regarding renewing the policy”
- Have a due date set to 10 business days from the creation of the ticket
- Have an escalation date set to 3 business days from the due date of the ticket
- Be auto-assigned

5.1.2 Tasks

The implementation team recommends creating the trouble ticket in the BillingInstruction Preupdate rule set.

1. **Create a new BillingInstruction preupdate rule called BIPU1000 – Create TroubleTicket on Delinquent Renewal that implements the specifications.**
2. **Restart the server because a new rule was created.**

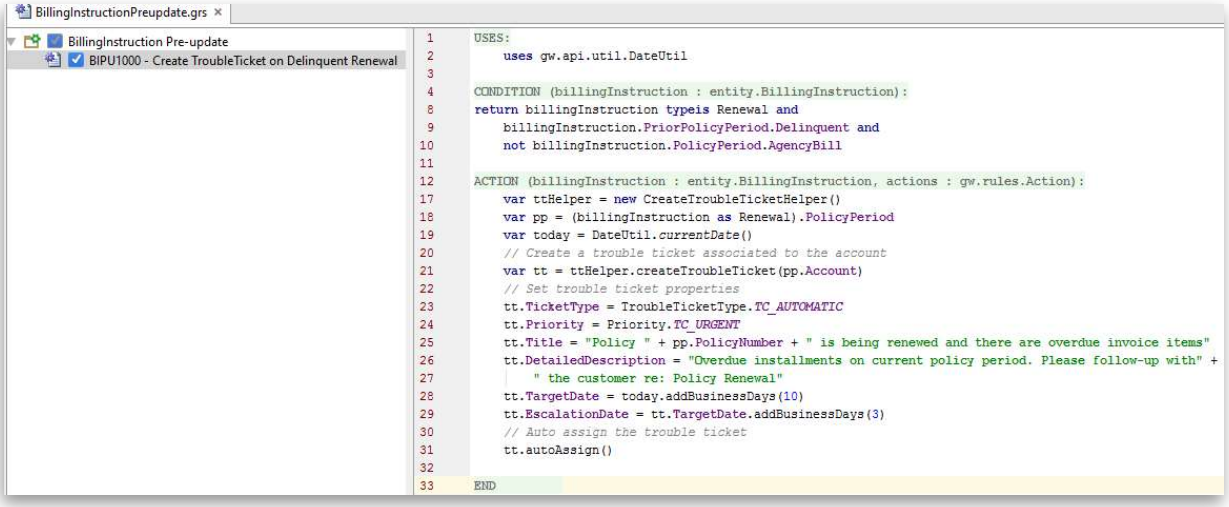
5.1.3 Testing procedure

1. **Create an account with one policy.**
   a) QuickJump: Run Account with1PolicyWithNoProducer

2. **Change the delinquency plan on the account to DP03. (The reason for this is to avoid automatic policy cancellation due to non-payment.) Ignore the warning when saving.**
3. **Bill the first invoice and then make it past due.**
   a) Advance the system date to the bill date of the first invoice
   b) Run the Invoice batch process
   c) Confirm that the invoice is in a billed status
   d) Advance the system date to one day after the due date of the first invoice
   e) Run the Invoice Due batch process
   f) Confirm that the invoice is now in a due status
   g) Confirm that the account and policy are now in a past due delinquency
4. **Renew the policy**
   a) Go to the Policy→Summary screen and renew the policy with a $1200 premium.
   b) Policy Tab: Actions→Renew Policy
5. **Confirm that a trouble ticket was created.**
   a) Account Tab: Trouble Tickets

5.1.4 Solution

1. **Create a new BillingInstruction preupdate rule called BIPU1000 – Create TroubleTicket on Delinquent Renewal that implements the specifications.**



2. **Restart the server because a new rule was created.**

Lesson 6

Workflow Processes

6.1 Create a new workflow

This is the first of two labs that build a workflow based on the Qualify Producer Workflow scenario. The customer wants to verify that a producer has achieved a certain performance level before they promote the producer to Gold tier.

6.1.1 Requirements

Spec 1 Create a new workflow type for the Qualify Producer scenario.

Spec 2 The workflow must be able to access the producer that is being qualified by the workflow.

Spec 3 The Producer entity needs to know about any associated Qualify Producer workflows.

6.1.2 Tasks

1. **Create new workflow type for the Qualify Producer workflow.**
2. **Add a foreign key to the producer entity.**
3. **Add an array from the Producer entity to the new workflow.**
4. **Restart the server.**
5. **Specify context symbol(s).**
6. **OPTIONAL: Regenerate the data dictionary.**
7. **OPTIONAL: Using the data dictionary, verify the new workflow type and the new array in the producer entity were created.**

6.1.3 Solution

Solution

Exact details on how to complete the lab.

1. **Create new workflow type for the Qualify Producer workflow.**
   a) In Studio, right-click Workflow and then select New → Workflow
   b) In the entity field, enter “QualifyProducer_Ext”
   c) In the description field, enter “Verifies a producer’s performance prior to promoting to Gold tier”
   d) Select BCWorkflow as the supertype. To do this click the button at the right of the supertype field and then select BCWorkflow. Click the OK button.

e) On the workflow window, click the next button.

2. Add a foreign key to the producer entity

a) Click the plus (+) button to add the foreign key

b) In the fkey Name field, enter "ProducerID"

c) In the fkey Entity field, click the button next to the field and select Producer.

d) In the description field, enter "Associated producer"

e) Click OK to create the new workflow

3. Add an array from the Producer entity to the new workflow.

a) Find and open the Producer.etx file

b) Add a new array

c) In the name field, enter QualifyProducerWorkflows_Ext

d) In the arrayentity, select the QualifyProducer_Ext workflow you just created.

e) In the desc field, enter "Associated qualify producer workflows"

4. Restart the server.

5. Specify context symbol(s).

a) In the QualifyProducer_Ext workflow, select the "<Context>" element

b) In the properties pane, click the plus (+) button

c) In the name field, enter "producer"

d) In the type field, enter "Producer"

e) In the value field, enter "Workflow.ProducerID"

6. OPTIONAL: Regenerate the data dictionary.

a) Open a command prompt

b) Run the following command `gwb genDataDictionary`

This step may take 10 to 15 minutes depending on the speed of your machine.

7. OPTIONAL: Using the data dictionary, verify the new workflow type and the new array in the producer entity were created.

a) Open the data dictionary, by navigating to C:\GW10\BillingCenter\build\dictionary\data and select the index.html file

b) On the data dictionary page, click the "Data Entities" link

c) In the left panel, find and select the "QualifyProducer_Ext" entity

d) Notice that this entity has the foreign key to the Producer. Click the Producer link to jump to the Producer entity.

e) On the Producer entity, scroll to the arrays section and confirm that there is a new array named "QualifyProducerWorkflows_Ext".

Lesson 7

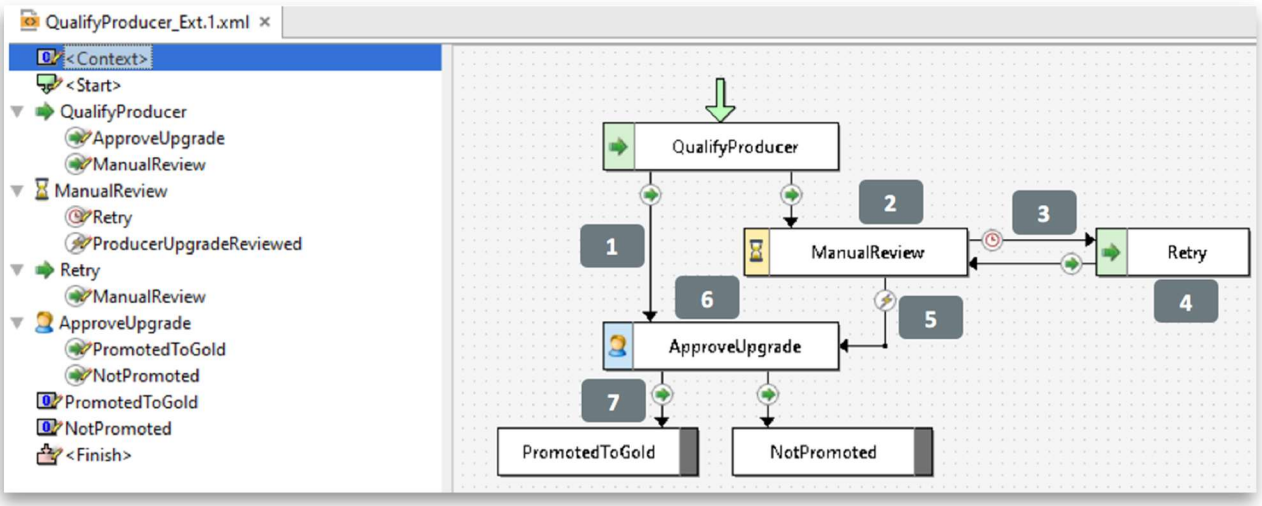
Workflow Elements

7.1 Add elements to workflow

This is the second of two labs that build a workflow based on the Qualify Producer Workflow scenario. The customer wants to verify that a producer has achieved a certain performance level before they promote the producer to Gold tier.

7.1.1 Requirements

The completed workflow to manage this business process is illustrated in the following graphic. The numbers are callouts to the specifications below the graphic. They do not indicate the order of



execution.

Spec 1 If the producer has a minimum of 5 policy periods, no manual review is required, and the workflow progresses directly to the ApproveUpgrade activity step. Otherwise, the ManualReview step is processed next.

Spec 2 The ManualReview manual step creates and assigns a notification activity to a user for the purpose of verifying the producer's qualifications for the promotion. The base assignment rules can be used to determine who is assigned the notification activity.

Spec 3 The timeout branch waits for 1 day before execution resumes at the Retry step.

Spec 4 The Retry auto step counts the number of retries. It creates a notification activity with an urgent priority to notify a user of the number of retries. The base assignment rules can be used to determine who is assigned the notification activity. (This notification activity would

normally be assigned to a supervisor, but this requirement is omitted for simplicity in this scenario.)

Spec 5 The ProducerReviewed trigger is invoked by a Producer Reviewed button on the Producer Summary screen.

Spec 6 The ApproveUpgrade activity step creates an approval blocking activity to approve the upgrade of the producer to the gold tier. The subject of the activity should say "Confirm that <producer name> should be promoted to Gold". The activity should be assigned using the assignment rules.

Spec 7 The go branch to the PromotedToGold outcome is executed if the approval activity has been approved. It includes logic to promote the producer to gold. Otherwise, the outcome is NotPromoted.

Spec 8 A BillingCenter user starts the promotion workflow process from the Producer Summary screen by clicking the Promote to Gold button.

7.1.2 Tasks

1. Modify the QualifyProducer_Ext workflow created in the previous lesson.

a) Create two new outcomes.

b) Create activity step.

c) Edit activities tab of ApproveUpgrade step.

d) Add new go branch to PromotedToGold step.

e) Edit PromotedToGold branch condition and execution script.

f) Create QualifyProducer auto step and go branch to ApproveUpgrade activity step.

g) Point <Start> block to QualifyProducer step and delete DefaultOutcome outcome step.

h) Add a new trigger type called ProducerUpgradeReviewed.

i) Create new manual step called ManualReview with a trigger branch to ApproveUpgrade activity step.

j) Add a Count integer column to QualifyProducer_Ext workflow entity and restart the server.

k) Create ManualReview notification activity.

l) Create a go branch from QualifyProducer step to ManualReview step.

m) Initialize counter in ManualReview go branch execution script.

n) Add a condition to ApproveUpgrade go branch.

o) Create an auto step called Retry with go branch to ManualReview step.

p) Create Retry notification activity.

q) Increment counter in Retry step enter script.

r) Create a new timeout branch from ManualReview step to Retry step.

2. **Reload Workflow Engine.**
3. **Add workflow start code and trigger branch code to ProducerDetailScreen.pcf file.**
   a) Add workflow start code and trigger branch code
   b) Create Promote To Gold button on the ProducerDetailScreen.pcf toolbar.
   c) Create Producer Reviewed button on the ProducerDetailScreen.pcf toolbar.
4. **Reload changed classes and reload UI.**

7.1.3 Testing procedure

1. **Test the ManualReview functionality:**
   a) In BillingCenter, add a producer with two policies
   b) QuickJump: Run Producer with2Policies
   c) Modify the producer's tier to Silver.
   d) Click Promote to Gold to start the workflow.
   e) Navigate to Administration → Monitoring → Workflows.
   f) Search for QualifyProducer_Ext workflows.
   g) Confirm that the workflow step is ManualReview.
   h) Execute the Retry timeout branch by selecting Manage Workflows button.
   i) Search for notification activities and look for one with a priority of High and a subject that includes the number of retries.

   j) Execute the Retry timeout branch again.
   k) Find the new notification activity and confirm that the retry count has increased by 1.
   l) On the Producer Summary screen, click Producer Reviewed button to invoke the trigger branch.
   m) Confirm that the workflow step is now ApproveUpgrade.
   n) Click the Workflow Type link for the workflow to verify an approval activity was generated.
   o) Search for the approval activity and Approve the activity and confirm that the outcome is PromotedToGold.
   p) Revisit the Producer Summary screen to confirm that the promotion has taken place.
2. **In BillingCenter, add a producer with ten policies**
   a) QuickJump: Run Producer with10Policies

b) Modify the producer's tier to Bronze.

c) Click Promote to Gold to start the workflow.

d) Navigate to Administration → Monitoring → Workflows.

e) Search for QualifyProducer_Ext workflows.

f) Confirm that the workflow step is ApproveUpgrade.

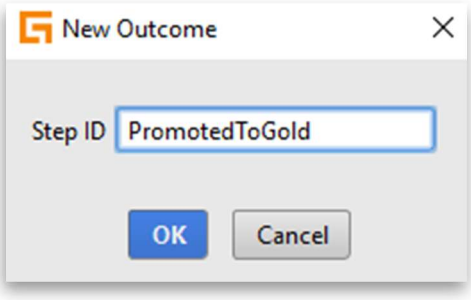
7.1.4 Solution

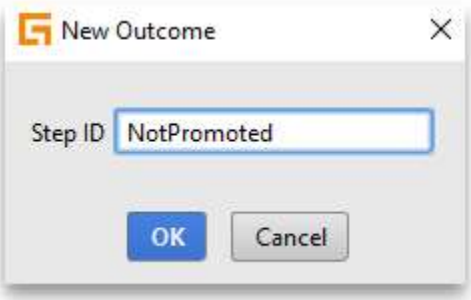
Solution

Exact details on how to complete the lab.

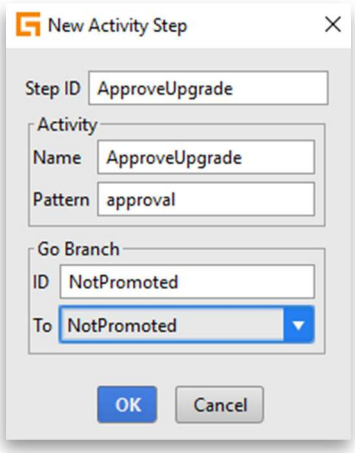
1. Modify the QualifyProducer_Ext workflow created in the previous lesson.

a) Create two new outcomes.

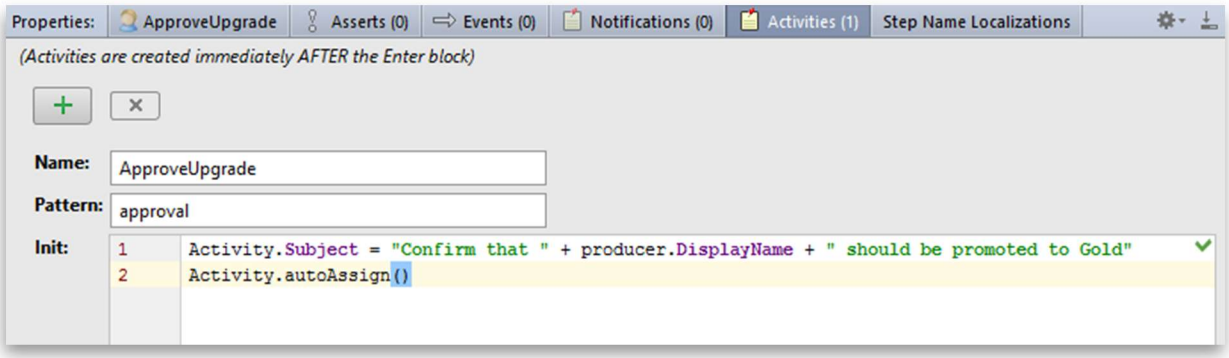




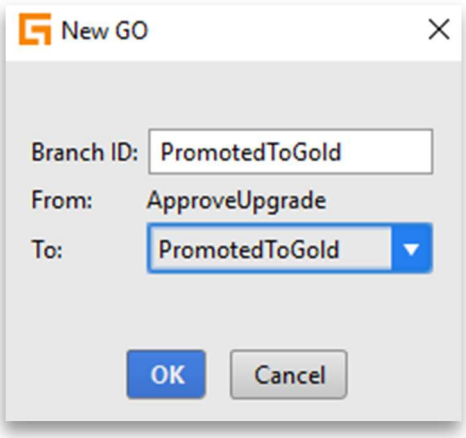
b) Create activity step.



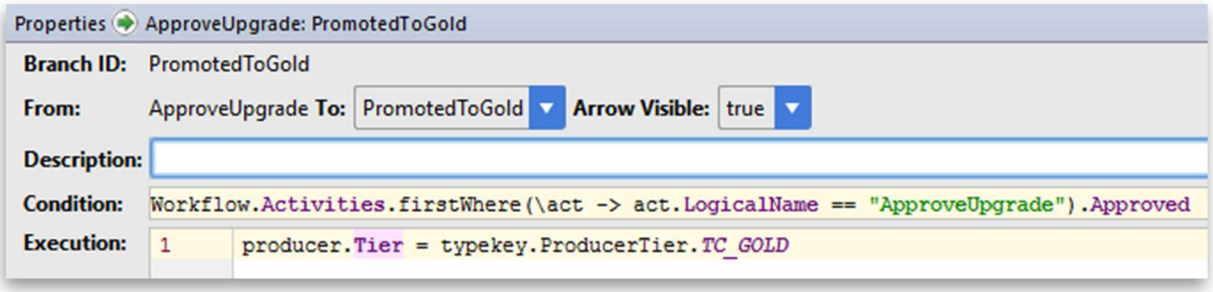
c) Edit activities tab of ApproveUpgrade step.



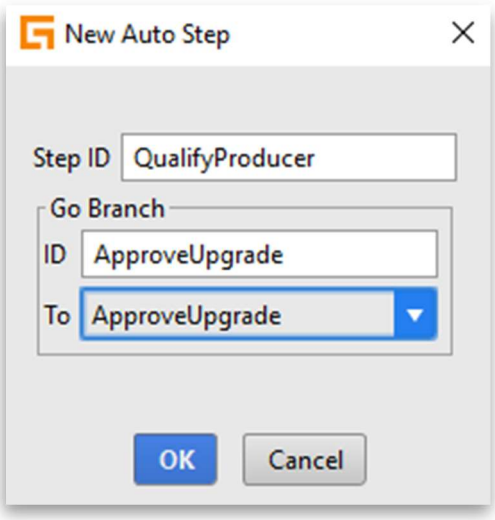
d) Add new go branch to PromotedToGold step.



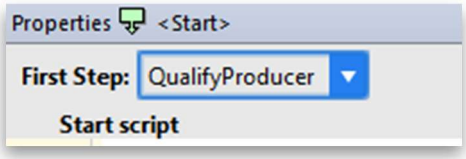
e) Edit PromotedToGold branch condition and execution script.



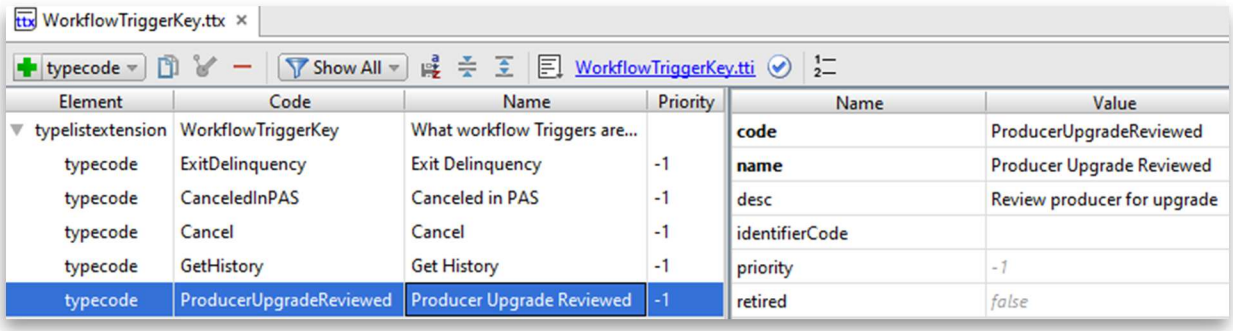
f) Create new QualifyProducer auto step and go branch to ApproveUpgrade activity step.



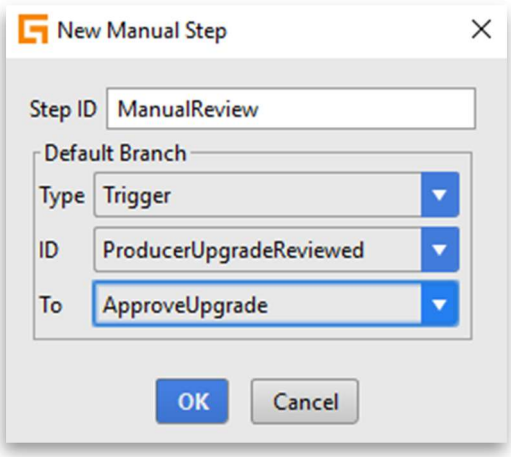
g) Point <Start> block to QualifyProducer step and delete DefaultOutcome outcome step.



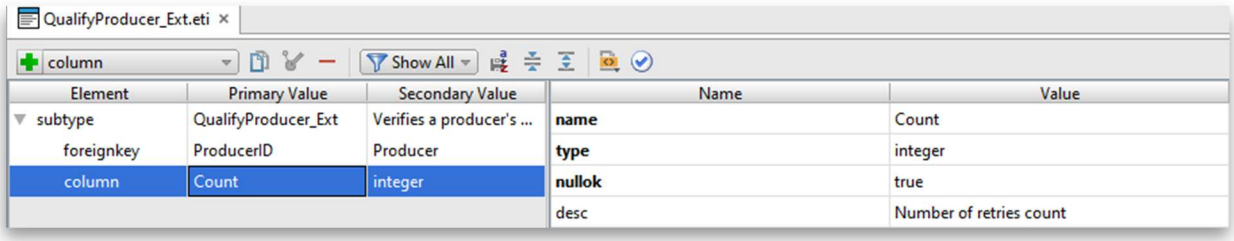
h) Add a new trigger type called ProducerUpgradeReviewed.



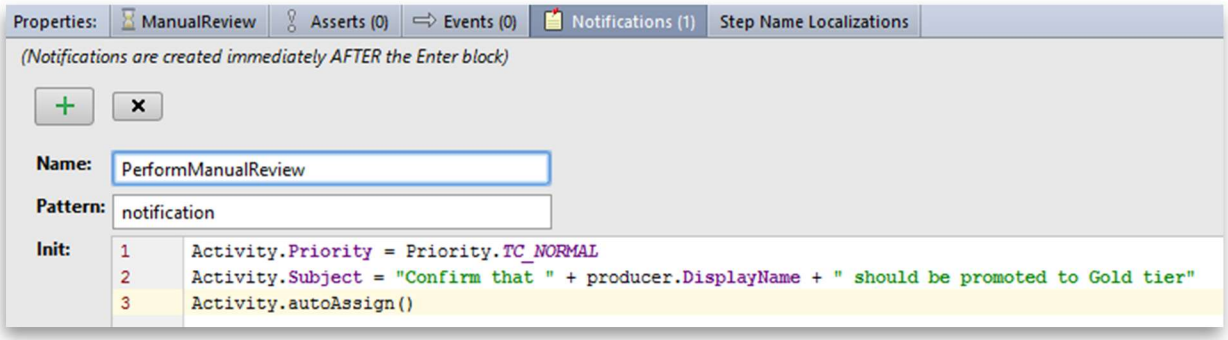
i) Create new manual step called ManualReview with a trigger branch to ApproveUpgrade activity step.



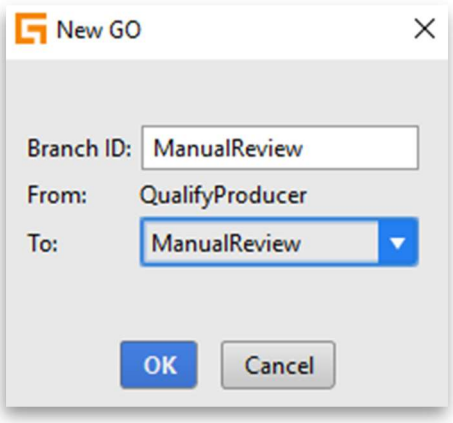
j) Add a Count integer column to QualifyProducer_Ext workflow entity and restart the server.



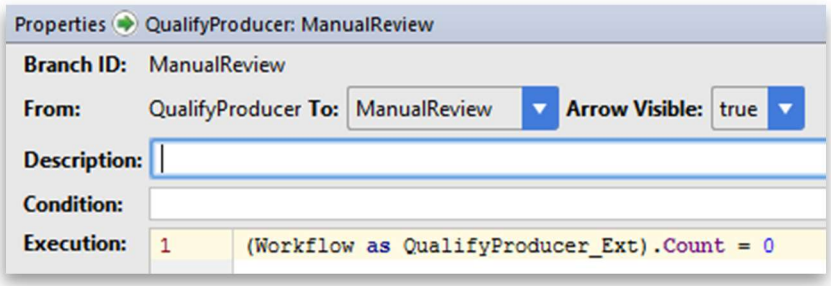
k) Create ManualReview notification activity.



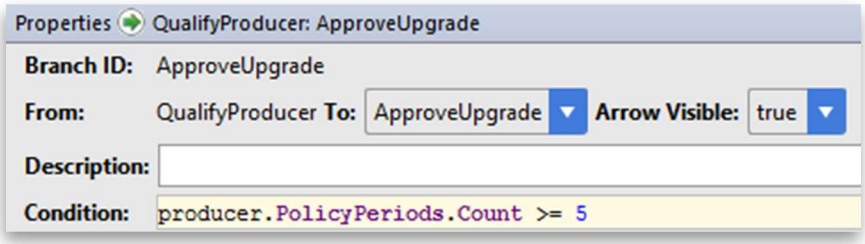
l) Create a go branch from QualifyProducer step to ManualReview step.



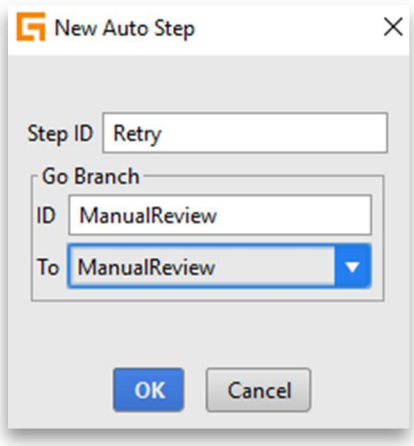
m) Initialize counter in ManualReview go branch execution script.



n) Add a condition to ApproveUpgrade go branch.

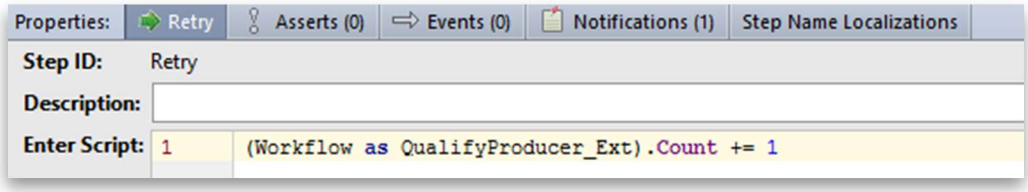


o) Create an auto step called Retry with go branch to ManualReview step.

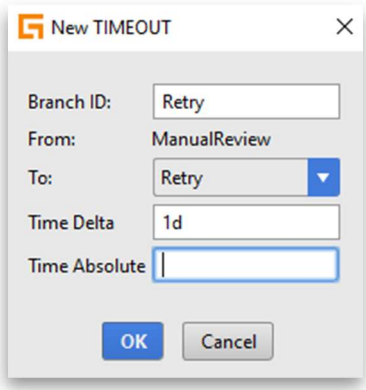


p) Create Retry notification activity.

q) Increment counter in Retry step enter script.



r) Create a new timeout branch from ManualReview step to Retry step.



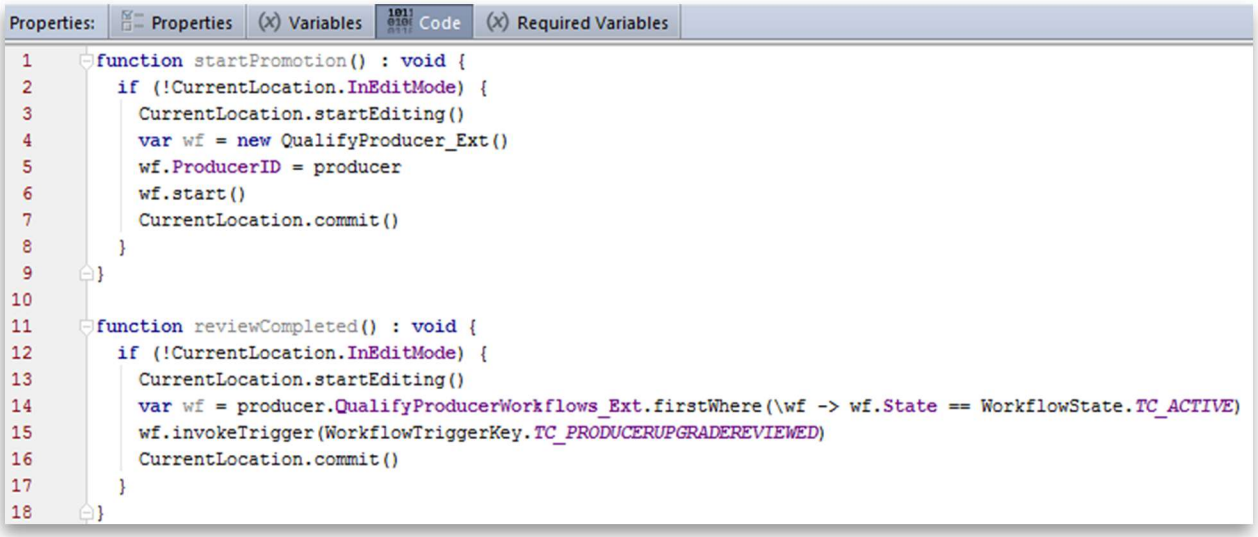
2. Reload Workflow Engine.

a) UI: ALT+SHIFT+T

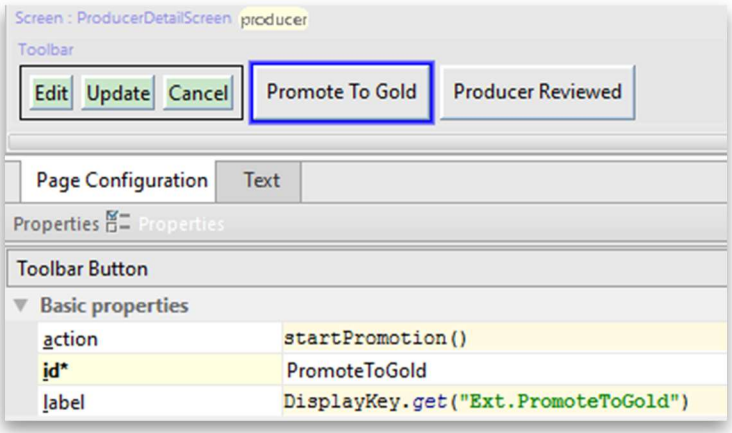
b) UI: Internal Tools → Reload → Reload Workflow Engine

3. Add workflow start code and trigger branch code to ProducerDetailScreen.pcf file.

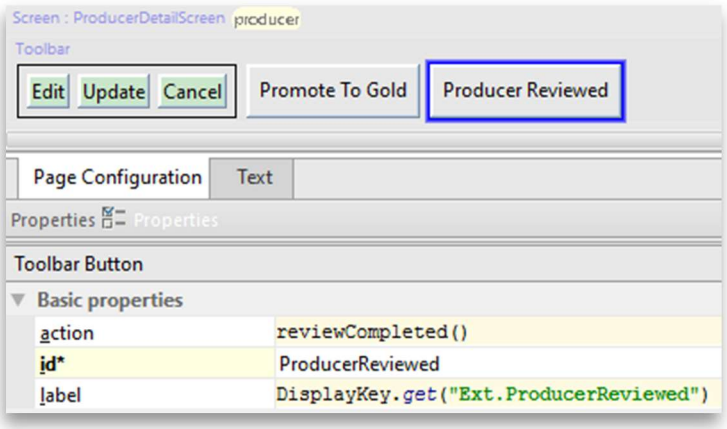
a) Add workflow start code and trigger branch code



b) Create Promote To Gold button on the ProducerDetailScreen.pcf toolbar.



c) Create Producer Reviewed button on the ProducerDetailScreen.pcf toolbar.



4. Reload changed classes and reload UI.

a) Studio: Run → Reload Changed Classes

b) UI: ALT+SHIFT+L

Lesson 8

Configuring Payment Allocation

8.1 Configure a new payment allocation filter

Succeed Insurance wants to customize how the payment allocation filter behaves.

8.1.1 Requirements

Spec 1 When BillingCenter receives a payment targeted for a policy period, restrict the payment allocation to invoice items belonging to the policy only if the policy is delinquent.

Spec 2 Otherwise, the filter does nothing.

8.1.2 Tasks

1. **Add a typecode to the *DistributionFilterType* typelist.**
   a) Extend the *DistributionFilterType.tti* typelist.
   b) Suggested typecode name: PolicyPeriodIfDelinquent
   c) Do not restart the server until all configuration is done; otherwise, the system will throw an exception.
2. **Create a new class based on PolicyPeriodDistributionFilterCriterion.gs.**
   a) Suggested package name: si.bc.classes.paymentallocation
   b) Suggested class name: PolicyPeriodIfDelinquentFilterCriterion
3. **Register the new class in LinkedImplementationLoaderImpl.gs**
4. **Restart the server.**

8.1.3 Testing procedure

1. **Create a new payment allocation plan by cloning the Default Payment Allocation Plan with these details: (Administration → Business Settings → Payment Allocation Plans)**
   a) Name: PP Delinquent Allocation Plan
   b) Remove all existing filters
   c) Add new PolicyPeriodIfDelinquent filter
2. **Move the *PP Delinquent AllocationPlan* to the top priority in the list of payment allocation plans.**
3. **Create a new account.**

a) *QuickJump: Run Account with1PolicyWithNoProducer*

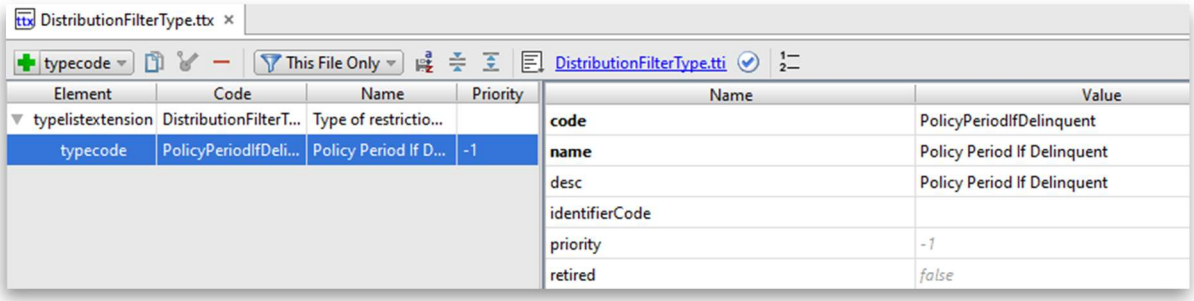
4. **Add a second policy to the new account with these details: *(Actions → Add Policy)***
   a) Policy Number: PPID2
   b) Payment Plan: PP02
   c) Premium: $1000
5. **Make the first invoice *Billed* by advancing the clock forward to the bill date of the first invoice and running the invoice batch process.**
   a) *Open Server Tools (ALT-SHIFT-T)*
   b) *Advance the clock to the first invoice date (if needed) using Internal Tools → Testing System Clock*
   c) *Run the Invoice batch process from Server Tools → Batch Process Info*
6. **From the Multiple Payment Entry screen, make a $300 payment on PPID2 policy.**
   a) Desktop Tab → Actions → New Payment → Multiple Payment Entry
7. **Run the New Payment batch process to allocate the payment.**
   a) *Open Server Tools (ALT-SHIFT-T)*
   b) *Run the New Payment batch process from Server Tools → Batch Process Info*
8. **Go to the *Charges* screen on the account and examine how the money was allocated.**
   a) They should be allocated to both policies since the filter was ignored because neither policy is delinquent.
9. **Advance the clock to one day past the due date of the second invoice.**
   a) *Open Server Tools (ALT-SHIFT-T)*
   b) *Internal Tools → Testing System Clock*
10. **Run the *Invoice* and *Invoice Due* batch processes to cause a past due delinquency on the policies.**
    a) *Server Tools → Batch Process Info*
11. **From the Multiple Payment Entry screen, make a $300 payment on PPID2 policy.**
    a) Desktop Tab → Actions → New Payment → Multiple Payment Entry
12. **Run the New Payment batch process to allocate the payment.**
    a) *Server Tools → Batch Process Info*
13. **Go to the *Charges* screen on the account and examine how the money was allocated.**
    a) The payment should only be allocated to PPID2 policy that is now delinquent. The other policy should be ignored since this was a targeted payment to PPID2 policy.

8.1.4 Solution

1. Add a typecode to the *DistributionFilterType* typelist.

a) Extend the *DistributionFilterType.tti* typelist.

b) Suggested typecode name: PolicyPeriodIfDelinquent



c) Do not restart the server until all configuration is done, otherwise system will throw an exception.

2. Create a new class based on PolicyPeriodDistributionFilterCriterion.gs.

a) Suggested package name: si.bc.classes.paymentallocation

b) Suggested class name: PolicyPeriodIfDelinquentFilterCriterion


```
package si.bc.classes.paymentallocation

uses gw.api.database.Relop
uses gw.api.path.Paths
uses gw.api.restriction.RestrictionBuilder
```

```
uses gw.bc.payment.DistributionFilterCriterion

/**
 * Payment Allocation Lab - Configure a new Payment Allocation filter
 */
class PolicyPeriodIfDelinquentFilterCriterion implements DistributionFilterCriterion{

  override function restrict(restrictionBuilder: RestrictionBuilder<InvoiceItem>,
directBillMoneyRcvd: DirectBillMoneyRcvd) {
    if(directBillMoneyRcvd != null and directBillMoneyRcvd.PolicyPeriod.Delinquent) {
      restrictionBuilder.compare(Paths.make(InvoiceItem#PolicyPeriod), Relop.Equals,
directBillMoneyRcvd.PolicyPeriod)
    }
  }

  override property get TypeKey(): DistributionFilterType {
    return DistributionFilterType.TC_POLICYPERIODIFDELINQUENT
  }
}
```

3. **Register the new class in LinkedImplementationLoaderImpl.gs**

```
66  override function returnDistributionFilterCriteria() : Collection<DistributionFilterCriterion> {
67    return {
68        new PositiveDistributionFilterCriterion(),
69        new InvoiceDistributionFilterCriterion(),
70        new PolicyPeriodDistributionFilterCriterion(),
71        new BilledOrDueDistributionFilterCriterion(),
72        new NextPlannedInvoiceDistributionFilterCriterion(),
73        new PastDueDistributionFilterCriterion(),
74        new PolicyPeriodIfDelinquentFilterCriterion()
75    }
76  }
```

4. **Restart the server.**

8.2 Customize default payment distribution

When a direct bill payment is made, Succeed Insurance wants billed items from workers' compensation policies to be paid before billed items for policies for other LOBs.

8.2.1 Requirements

Spec 1 Distribute direct bill payments to the workers' compensation distribution items first

Spec 2 Distribute the remaining amount (if any) to the other items.

8.2.2 Tasks

1. **Customize the allocatePayment() method in the DirectBillPayment plugin.**
2. **Compile code changes.**

8.2.3 Testing procedure

1. **Verify that the *Default Payment Allocation Plan* is top priority.**

a) *Administration → Business Settings → Payment Allocation Plans*

2. **Create a new account.**

   a) *QuickJump: Run Account with1PolicyWithNoProducer*

3. **Add a second policy to the new account with these details:**

   a) *Actions → Add Policy*

   b) Policy Number: DPD1

   c) Product: Workers' Compensation

   d) Payment Plan: PP02

   e) Premium: $3500

   f) Taxes: $280

4. **Make the first invoice *Billed* by advancing the clock forward and running the invoice batch process.**

   a) *Open Server Tools (ALT-SHIFT-T)*

   b) *Advance the clock to the first invoice date (if needed) using Internal Tools → Testing System Clock*

   c) *Run the Invoice batch process from Server Tools → Batch Process Info*

5. **Go to the Direct Bill Payment screen.**

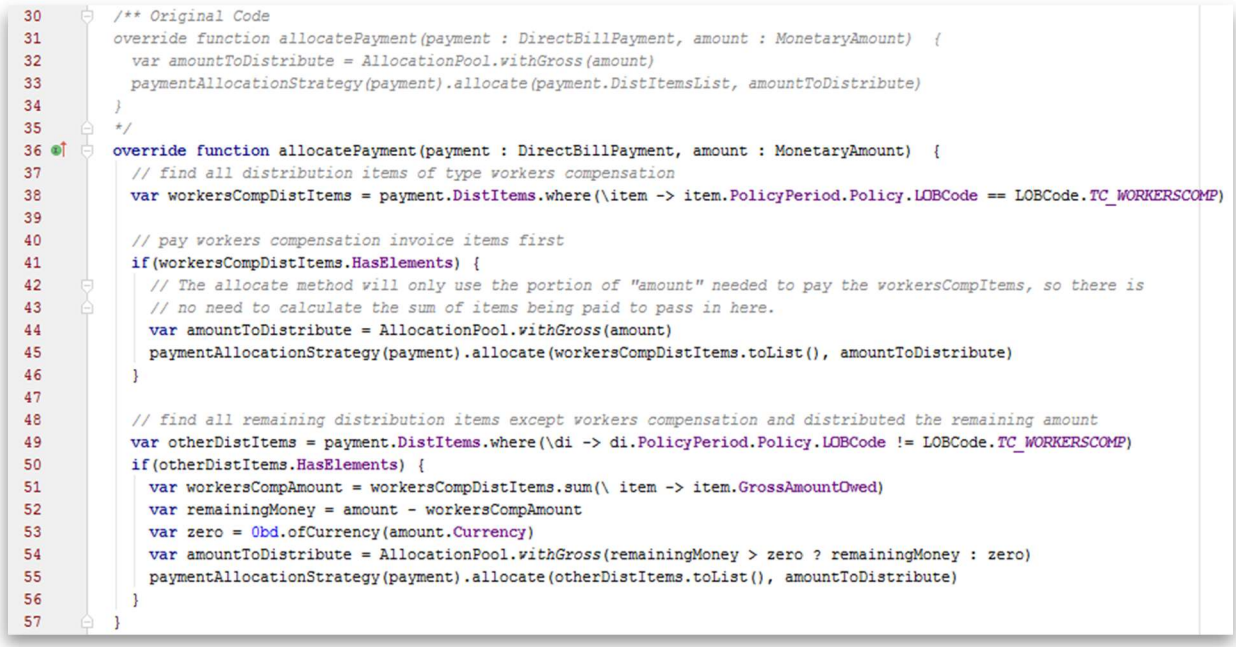
   a) Actions → New Payment → New Direct Bill Payment

6. **Enter $650 as the amount and tab out of the field to observe the customized default allocation.**

   a) The workers' compensation items should be paid first, and any remainder should be paid to the other policy.

8.2.4 Solution

1. **Customize the allocatePayment() method in the DirectBillPayment plugin.**



```
/** Original Code
override function allocatePayment(payment : DirectBillPayment, amount : MonetaryAmount)  {
  var amountToDistribute = AllocationPool.withGross(amount)
  paymentAllocationStrategy(payment).allocate(payment.DistItemsList, amountToDistribute)
}
*/
override function allocatePayment(payment : DirectBillPayment, amount : MonetaryAmount)  {
  // find all distribution items of type workers compensation
  var workersCompDistItems = payment.DistItems.where(\item -> item.PolicyPeriod.Policy.LOBCode == LOBCode.TC_WORKERSCOMP)

  // pay workers compensation invoice items first
  if(workersCompDistItems.HasElements) {
    // The allocate method will only use the portion of "amount" needed to pay the workersCompItems, so there is
    // no need to calculate the sum of items being paid to pass in here.
    var amountToDistribute = AllocationPool.withGross(amount)
    paymentAllocationStrategy(payment).allocate(workersCompDistItems.toList(), amountToDistribute)
  }

  // find all remaining distribution items except workers compensation and distributed the remaining amount
  var otherDistItems = payment.DistItems.where(\di -> di.PolicyPeriod.Policy.LOBCode != LOBCode.TC_WORKERSCOMP)
  if(otherDistItems.HasElements) {
    var workersCompAmount = workersCompDistItems.sum(\ item -> item.GrossAmountOwed)
    var remainingMoney = amount - workersCompAmount
    var zero = 0bd.ofCurrency(amount.Currency)
    var amountToDistribute = AllocationPool.withGross(remainingMoney > zero ? remainingMoney : zero)
    paymentAllocationStrategy(payment).allocate(otherDistItems.toList(), amountToDistribute)
  }
}
```

```
override function allocatePayment(payment : DirectBillPayment, amount : MonetaryAmount)  {
  // find all distribution items of type workers compensation
  var workersCompDistItems = payment.DistItems.where(\item -> item.PolicyPeriod.Policy.LOBCode ==
LOBCode.TC_WORKERSCOMP)

  // pay workers compensation invoice items first
  if(workersCompDistItems.HasElements) {
    // The allocate method will only use the portion of "amount" needed to pay the
workersCompItems, so there is
    // no need to calculate the sum of items being paid to pass in here.
    var amountToDistribute = AllocationPool.withGross(amount)
    paymentAllocationStrategy(payment).allocate(workersCompDistItems.toList(),
amountToDistribute)
  }

  // find all remaining distribution items except workers compensation and distributed the
remaining amount
  var otherDistItems = payment.DistItems.where(\di -> di.PolicyPeriod.Policy.LOBCode !=
LOBCode.TC_WORKERSCOMP)
  if(otherDistItems.HasElements) {
    var workersCompAmount = workersCompDistItems.sum(\ item -> item.GrossAmountOwed)
    var remainingMoney = amount - workersCompAmount
    var zero = 0bd.ofCurrency(amount.Currency)
    var amountToDistribute = AllocationPool.withGross(remainingMoney > zero ? remainingMoney :
zero)
    paymentAllocationStrategy(payment).allocate(otherDistItems.toList(), amountToDistribute)
  }
}
```

2. Compile code changes.

a) Run → Reload Changed Classes

8.3 Demo code: Creating a custom priority

The following code is from the instructor demo.

```
       package demo.bc.classes.paymentallocation

uses com.google.common.collect.Ordering
uses gw.bc.payment.InvoiceItemAllocationOrdering

/**
 * Creating a Custom Priority Demo
 */
class ChargePatternCategoryOrdering  implements InvoiceItemAllocationOrdering {

  override property get TypeKey() : InvoiceItemOrderingType {
    return InvoiceItemOrderingType.TC_CHARGEPATTERNCATEGORY
  }

  override function getInvoiceItemOrdering(directBillMoneyRcvd: DirectBillMoneyRcvd):
Ordering<InvoiceItem> {
    return new Ordering<InvoiceItem>() {
      override function compare(item1: InvoiceItem, item2: InvoiceItem): int {
        var patternListByPriority = {
            ChargeCategory.TC_PREMIUM,
            ChargeCategory.TC_TAX,
            ChargeCategory.TC_FEE
        }.reverse()

        var leftPriority = patternListByPriority.indexOf(item1.Charge.ChargePattern.Category)
        var rightPriority = patternListByPriority.indexOf(item2.Charge.ChargePattern.Category)

        return Integer.compare(rightPriority, leftPriority)
      }
    }
  }
}
```