GUIDEWIRE

InsuranceSuite 10 Integration: Kickstart

Student Workbook

Labs and Tutorials

Document Published: 2020-09-29

Table of Contents

Introduction

Welcome to the Guidewire InsuranceSuite 10.0 Integration - 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

Introduction to InsuranceSuite Integration

1.1 No lab

Lesson 2

Gosu for Integration

This lab requires that you use TrainingApp, Guidewire Studio. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

2.1 Create a custom class

Succeed Insurance has several integration points that need to make use of ABContact summaries. In general, it is highly desirable for integration points to transmit and work with only the data that is required.

2.1.1 Requirements

Spec 1 Create a package that uses **si** as the customer code, **ta** as the product code, **classes** as the mechanism, and **entity** as the functional area.

Spec 2 Create a class called **ABContactSummary** that represents summary information of an ABContact. Whenever this class is instantiated, the ExternalID property should be set.

Spec 3 Create the following class properties:

- ExternalID (int)
  - If the summary's ExternalID is 0, set the ExternalID to a unique integer no less than 1000, otherwise do nothing.
- ContactID (String)
- Name (String)
- NumCheckingAccounts (int)

Spec 4 Create a function called **loadSummaryData**.

- Takes an input parameter of the type entity.ABContact.
- Set the summary's ContactID to the ABContact's public ID.
- Set the summary's Name to the ABContact's DisplayName field.
- Set the summary's NumCheckingAccounts to the number of ABContact.BankAccounts where the type is checking. Use the appropriate array function to determine the number of checking accounts.

Spec 5 Create a function called **buildConcatenatedSummary**.

- Returns a String that contains the value of all properties as a comma-delimited list.

2.1.2 Tasks

1. **Create a new package.**
2. **Create a new class.**

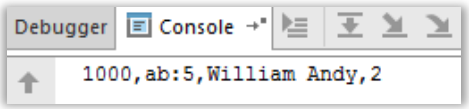
3. **Create new properties and functions.**
4. **Deploy code changes.**
5. **Perform verification steps.**

Hints

1. **For the ExternalID property:**
   - Use a getter and setter rather than shorthand notation.
   - Use a constructor to initialize the property.
   - Use the sequence utility.
   - Don't use Integer because it will initialize as null.
2. **Properly comment and annotate code.**

2.1.3 Verification steps

1. **Generate console output for ABContactSummary using Gosu Scratchpad.**
   - In Studio, open Gosu Scratchpad by clicking **Tools → Gosu Scratchpad**.
   - Write code that will test the solution:
   - Using **trainingapp.base.QueryUtil.findContact** method, create a variable that references contact William Andy whose publicID is **ab:5**.
     - Create a new ABContactSummary object.
       - Initialize the ExternalID sequence counter.
     - Execute loadSummaryData() using the given contact.
     - Retrieve and print the output of the buildConcatenatedSummary () function.
   - Verify the output
2. **Verify that the output in the Debug Console has four delimited values with correct values:**
   - Sequence number
   - Public ID
   - Contact name
   - Number of checking account



2.1.4 Solution

1. **Create a new package.**
   - Right-click on **gsrc** folder and select **New → Package**.

- Enter **si.ta.classes.entity** as the new package name.

2. Create a new class.

- Right click on the **entity** package and select **New → Gosu Class**.
- Enter **ABContactSummary** as the new Gosu class name**.**

3. Create new properties and functions.

```
package si.ta.classes.entity

uses gw.api.system.database.SequenceUtil

/**
 * Training lab
 */
class ABContactSummary {

  construct(i: int) {
    ExternalID = i
  }

  // Declare ExternalID property using getter and setter
  var _externalID: int  // Don't use Integer because it will initialize as null.

  property get ExternalID(): int {
    return _externalID
  }

  property set ExternalID(externalId: int) {
    if(externalId == 0) {
      _externalID = SequenceUtil.next(1000, "externalID") as int
    } else
      _externalID = externalId
  }

  // Declare properties using shorthand syntax
  var _contactID: String as ContactID
  var _name: String as Name
  var _numCheckingAccounts: int as NumCheckingAccounts

  // Create functions

  /**
   * Function that initializes ABContact properties.
   */
  @Param("contact", "Input parameter of type entity.ABContact")
  function loadSummaryData(contact: ABContact): void {
    this._contactID = contact.PublicID
    this._name = contact.DisplayName
    this._numCheckingAccounts = contact.BankAccounts.countWhere(\account ->
        account.AccountType == typekey.BankAccountType.TC_CHECKING)
  }

  /**
   * Function that builds a comma-delimited list.
   */
  @Returns("A comma-delimited list of property values")
  function buildConcatenatedSummary(): String {
```

```
    return String.format("%s,%s,%s,%s", {_externalID, _contactID, _name, _numCheckingAccounts})
  }

}
```

4. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
5. **Perform verification steps.**

```
uses trainingapp.base.QueryUtil
uses si.ta.classes.entity.ABContactSummary

var testContact = QueryUtil.findContact("ab:5")
var newSummary = new ABContactSummary(0)
newSummary.loadSummaryData(testContact)
print(newSummary.buildConcatenatedSummary())
```

Lesson 3

Gosu Queries

This lab requires that you use TrainingApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Alice Applegate whose login/password is aapplegate/gw.

3.1 Query the assigned user contact

Succeed Insurance wants to modify and enhance the **ABContactSummary** class with a query that meets the requirements listed below.

3.1.1 Requirements

Spec 1 Add an additional property to **ABContactSummary**.

- AssignedUserWorkload (an integer).

Spec 2 Modify the **loadSummaryData** method that meets the following criteria:

- Query the total number of Contacts assigned to the current AssignedUser of the displayed contact.
- Set the summary's AssignedUserWorkload to this total.
- If the contact's assigned user is null, then set the AssignedUserWorkload field to 0.

Spec 3 Modify the **buildConcatenatedSummary** method that meets the following criteria:

- Returns a String that contains the value of all five properties as a comma-delimited list.

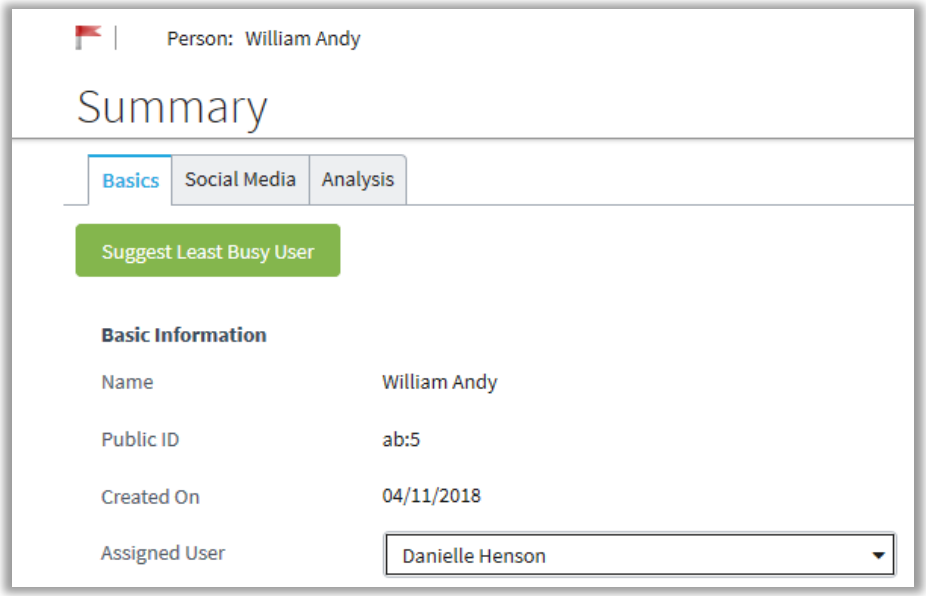
3.1.2 Tasks

1. **Modify ABContact Summary class.**
   - Add new AssignedUserWorkload property.
   - Modify loadSummaryData method.
   - Modify buildConcatenatedSummary method.
2. **Deploy code changes.**
3. **Perform verification steps.**

3.1.3 Verification steps

1. **Assign a User to an ABContact.**
   - In TrainingApp, navigate to the contact that you worked on in the previous lab, e.g. William Andy.
   - On the summary screen, select **Edit** and select an **Assigned User** as Danielle Henson.

- Click **Update**.



2. **Generate console output for ABContactSummary using Gosu Scratchpad.**
   - In Studio, open Gosu Scratchpad by clicking **Tools → Gosu Scratchpad**.
   - Write code that will test the solution:
     - Create a query that references contact William Andy whose publicID is **ab:5**.
       - Make sure only one result is retrieved, otherwise throw an exception.
     - Create a new ABContactSummary.
       - Initialize the **ExternalID** sequence counter.
     - Execute the **loadSummaryData** method using the given contact.
     - Retrieve and print the output of the **buildConcatenatedSummary** method.
3. **Verify the output**
   - Verify that the output in the Debug Console has four delimited values with correct values:
     - Sequence number
       - **Note**: this number can vary based on how many times the sequence utility was executed in the previous lab
     - Public ID
     - Contact name
     - Number of checking account
     - Assigned User Workload

3.1.4 Solution

1. **Modify ABContact Summary class.**
   - Add new AssignedUserWorkload property.
   - Modify loadSummaryData method.
   - Modify buildConcatenatedSummary method.

```
package si.ta.classes.entity

uses gw.api.database.Query
uses gw.api.database.Relop
uses gw.api.system.database.SequenceUtil

/**
 * Training lab
 */
class ABContactSummary {

  construct(i: int) {
    ExternalID = i
  }

  // Declare ExternalID property using getter and setter
  var _externalID: int  // Don't use Integer because it will initialize as null.

  property get ExternalID(): int {
    return _externalID
  }

  property set ExternalID(externalId: int) {
    if(externalId == 0) {
      _externalID = SequenceUtil.next(1000, "externalID") as int
    } else
      _externalID = externalId
  }

  // Declare properties using shorthand syntax
  var _contactID: String as ContactID
  var _name: String as Name
  var _numCheckingAccounts: int as NumCheckingAccounts
  var _assignedUserWorkload: int as AssignedUserWorkload

  // Create functions

  /**
   * Function that initializes ABContact properties.
   */
  @Param("contact", "Input parameter of type entity.ABContact")
  function loadSummaryData(contact: ABContact): void {
    this._contactID = contact.PublicID
    this._name = contact.DisplayName
    this._numCheckingAccounts = contact.BankAccounts.countWhere(\account ->
        account.AccountType == typekey.BankAccountType.TC_CHECKING)
    // check if the contact actually has an assigned user
    if(contact.AssignedUser != null) {
      var queryObj = Query.make(ABContact)
      queryObj.compare(ABContact#AssignedUser, Relop.Equals, contact.AssignedUser)
      this._assignedUserWorkload = queryObj.select().Count
```

```
    } else {
      _assignedUserWorkload = 0
    }
  }

  /**
   * Function that builds a comma-delimited list.
   */
  @Returns("A comma-delimited list of property values")
  function buildConcatenatedSummary(): String {
    return String.format("%s,%s,%s,%s,%s",
        {_externalID, _contactID, _name, _numCheckingAccounts, _assignedUserWorkload})
  }

}
```

2. **Deploy code changes.**
   - From the Studio menu, select **Run → Reload Changed Classes**
3. **Perform verification steps.**

```
uses gw.api.database.Query
uses gw.api.database.Relop
uses si.ta.classes.entity.ABContactSummary

var queryObj = Query.make(ABContact)
queryObj.compare(ABContact#PublicID, Relop.Equals, "ab:5")
var targetContact = queryObj.select().AtMostOneRow

var newSummary = new ABContactSummary(0)
newSummary.loadSummaryData(targetContact)
print(newSummary.buildConcatenatedSummary())
```

Lesson 4

Bundles and Database Transactions

This lab requires that you use TrainingApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Alice Applegate whose login/password is aapplegate/gw.

4.1 Create a contact summary note

Succeed Insurance wants to create and save contact notes with summary information for a given contact.

4.1.1 Requirements

Spec 1 Create the **saveSummaryNote** method that meets the following criteria:

- If the ABContactSummary's ContactID is null, do nothing, otherwise, for a non-null ContactID, create a contact note.
- Set the note subject to ABContact Summary.
- Set the note contact type to General.
- Where <x> is the appropriate value, the note body must contain the following lines (use "\n" for line breaks):
  - External ID: <x>
  - Name: <x>
  - Number of checking accounts: <x>

Spec 2 Query for the ABContact whose public ID matches the ABContactSummary's ContactID.

Spec 3 Add the contact note to the ABContact's ContactNotes array.

Spec 4 Commit the new note to the database.

4.1.2 Tasks

1. **Modify ABContact Summary class by inserting a new saveSummaryNote method.**
2. **Deploy code changes.**
3. **Perform verification steps.**

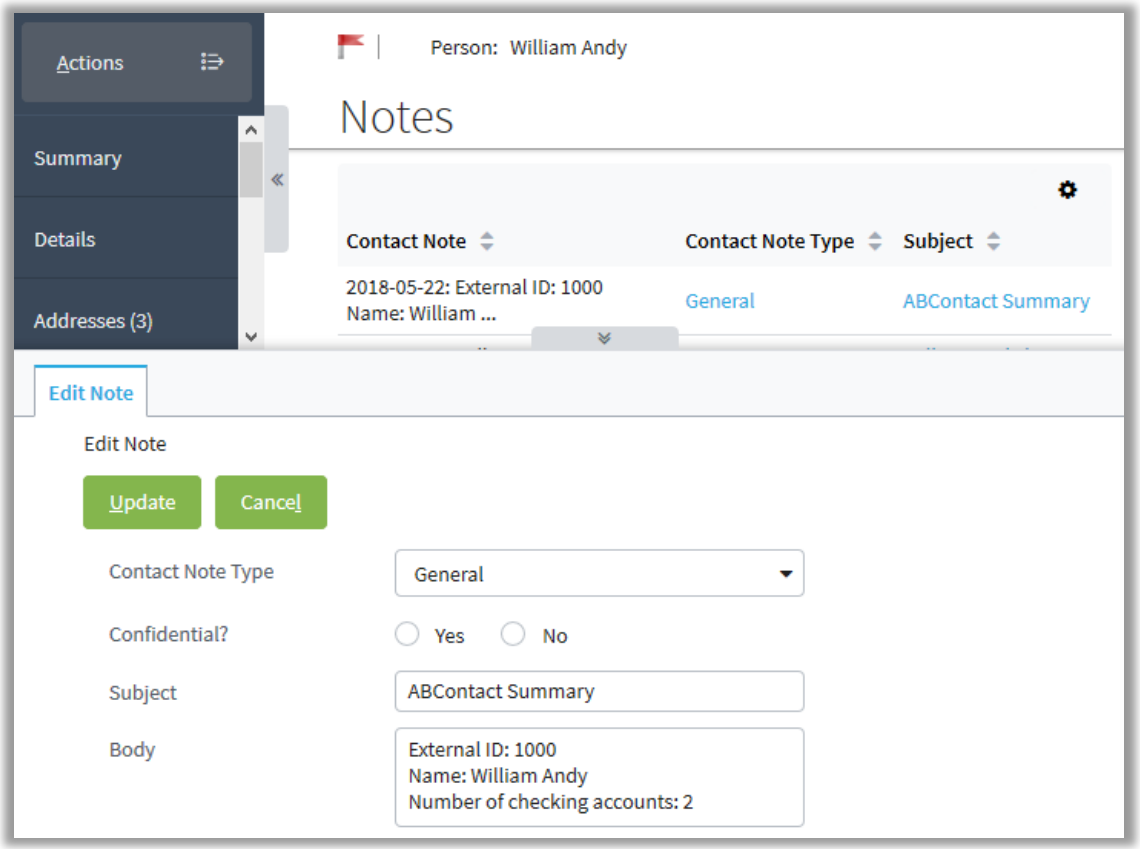
4.1.3 Verification steps

1. **Generate console output for ABContactSummary using Gosu Scratchpad**
   - In Studio, open Gosu Scratchpad by clicking **Tools → Gosu Scratchpad**.
   - Write code that will test the solution:

- Create a query that references contact William Andy whose publicID is **ab:5**.
  - Make sure only one result is retrieved, otherwise throw an exception.
- Create a new ABContactSummary.
  - Initialize the ExternalID sequence counter.
- Execute the loadSummaryData method using the given contact.
- Execute the saveSummaryNote method.

2. Verify the note was created.

- In TrainingApp, navigate to the contact.
- In the sidebar menu, click the Notes menu link.
- Click Contact Note to view the entire note.
- Verify the note was created with the correct subject and body.



4.1.4 Solution

1. Modify ABContact Summary class by inserting a new saveSummaryNote method.

```
package si.ta.classes.entity

uses gw.api.database.Query
```

```
uses gw.api.database.Relop
uses gw.api.system.database.SequenceUtil
uses gw.transaction.Transaction

/**
 * Training lab
 */
class ABContactSummary {

  construct(i: int) {
    ExternalID = i
  }

  // Declare ExternalID property using getter and setter
  var _externalID: int  // Don't use Integer because it will initialize as null.

  property get ExternalID(): int {
    return _externalID
  }

  property set ExternalID(externalId: int) {
    if(externalId == 0) {
      _externalID = SequenceUtil.next(1000, "externalID") as int
    } else
      _externalID = externalId
  }

  // Declare properties using shorthand syntax
  var _contactID: String as ContactID
  var _name: String as Name
  var _numCheckingAccounts: int as NumCheckingAccounts
  var _assignedUserWorkload: int as AssignedUserWorkload

  // Create functions

  /**
   * Function that initializes ABContact properties.
   */
  @Param("contact", "Input parameter of type entity.ABContact")
  function loadSummaryData(contact: ABContact): void {
    this._contactID = contact.PublicID
    this._name = contact.DisplayName
    this._numCheckingAccounts = contact.BankAccounts.countWhere(\account ->
        account.AccountType == typekey.BankAccountType.TC_CHECKING)
    // check if the contact actually has an assigned user
    if(contact.AssignedUser != null) {
      var queryObj = Query.make(ABContact)
      queryObj.compare(ABContact#AssignedUser, Relop.Equals, contact.AssignedUser)
      this._assignedUserWorkload = queryObj.select().Count
    } else {
      _assignedUserWorkload = 0
    }
  }

  /**
   * Function that builds a comma-delimited list.
   */
  @Returns("A comma-delimited list of property values")
  function buildConcatenatedSummary(): String {
```

```
    return String.format("%s,%s,%s,%s,%s",
        {_externalID, _contactID, _name, _numCheckingAccounts, _assignedUserWorkload})
  }

  /**
   * Function that create a summary note.
   */
  function saveSummaryNote(): void {
    if (this._contactID != null) {
      var queryObj = Query.make(ABContact)
      queryObj.compare(ABContact#PublicID, Relop.Equals, this._contactID)
      var targetContact = queryObj.select().AtMostOneRow
      Transaction.runWithNewBundle(\newBundle -> {
        var newNote = new ContactNote()
        newNote.Subject = "ABContact Summary"
        newNote.ContactNoteType = ContactNoteType.TC_GENERAL
        newNote.Body = "External ID: " + this._externalID + "\n" +
            "Name: " + this._name + "\n" +
            "Number of checking accounts: " + this._numCheckingAccounts
        targetContact.addToContactNotes(newNote)
      }, "su")
    }
  }

}
```

2. **Deploy code changes.**
   - From the Studio menu, select **Run → Reload Changed Classes**
3. **Perform verification steps.**

```
uses gw.api.database.Query
uses gw.api.database.Relop
uses si.ta.classes.entity.ABContactSummary

var queryObj = Query.make(ABContact)
queryObj.compare(ABContact#PublicID, Relop.Equals, "ab:5")
var targetContact = queryObj.select().AtMostOneRow

var newSummary = new ABContactSummary(0)
newSummary.loadSummaryData(targetContact)
newSummary.saveSummaryNote()
```

Lesson 5

Gosu Templates

This lab requires that you use TrainingApp and Guidewire Studio. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

5.1 Generate a String message payload

For existing contacts with a modified tax ID and for newly created contacts of the type ABPerson, TrainingApp must determine if the given contact has been involved with a previous act of insurance fraud. The external system that checks for fraud requires that there is specific format for message payloads for various types of entities.

5.1.1 Requirements

Spec 1 The external system requires the following format for a String message payload for a contact of the type **ABPerson**:

- The names in each name/value pair must be taxID and fullName.
- A comma must separate the name and value.
- A semi-colon must delimit each name/value pair.
- For example: **taxID,999-99-9999;fullName,John Doe**

Spec 2 Create a gosutemplate package with the fully qualified name of: **si.ta.gosutemplate**

Spec 3 Create a gosutemplate called **FraudTemplate** in the gosutemplate package.

5.1.2 Tasks

1. **Create a gosutemplates package.**
2. **Create a Gosu Template.**
3. **Deploy code changes.**
4. **Perform verification steps.**

5.1.3 Verification steps

1. **Generate debug console output using Gosu Scratchpad.**
   - In Studio, open Gosu Scratchpad by clicking **Tools → Gosu Scratchpad**.
   - Write code that will test the solution:
     - Create a query that references contact William Andy whose PublicID is **ab:5**.
       - Make sure only one result is retrieved, otherwise throw an exception.
     - Create a new payload variable the generates String output utilizing the Gosu Template.
     - Output the payload to the debug console.
2. **Verify that the output in the debug console has the two name/value pairs in the correct format.**

```
taxID,123-45-6793;fullName,William Andy
```

5.1.4 Solution

1. **Create a gosutemplates package**
   - Right-click on the **gsrc** folder and select **New → Package**.
   - Enter **gosutemplate** as the new package name.
2. **Create a Gosu Template**
   - Right-click on the **gosutemplate** package and select **New → Gosu Template**.
   - Enter **FraudTemplate** as the new Gosu Template name.

```
<%@ params( anABPerson: ABPerson ) %>
taxID,${anABPerson.TaxID};fullName,${anABPerson.FullName}
```

3. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
4. **Perform verification steps.**

```
uses gw.api.database.Query
uses gw.api.database.Relop

var queryObj = Query.make(ABPerson)
queryObj.compare(ABPerson#PublicID, Relop.Equals, "ab:5")
var targetPerson = queryObj.select().AtMostOneRow

var payload = si.ta.gosutemplate.FraudTemplate.renderToString(targetPerson)
print(payload)
```

Lesson 6

XML Modeler and Strongly Typed XML

This lab requires that you use TrainingApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Alice Applegate whose login/password is aapplegate/gw.

6.1 Use an XML model to generate an XML message payload

For existing contacts with a modified tax ID and for newly created contacts of the type ABPerson, TrainingApp must determine if the given contact has been involved with a previous act of insurance fraud. The external system that checks for fraud requires that there is specific format for message payloads for various types of entities.

6.1.1 Requirements

Spec 1 The external system requires the following format for an XML message payload for a contact of the type **ABCompany**:

- Parent XML element for ABContact.
- Sub element for PublicID as Key property.
- Sub elements for DisplayName and TaxID as Normal properties.
- Sub elements contain respective values for TaxID and DisplayName.

```
<?xml version="1.0"?>
<ABContact xmlns="si.ta.xmlmodels.abcontactxmlmodel">
  <PublicID>java.lang.String</PublicID>
  <DisplayName>java.lang.String</DisplayName>
  <TaxID>java.lang.String</TaxID>
</ABContact>
```

Spec 2 Create an xmlmodel package with the fully qualified name of: **si.ta.xmlmodel**

Spec 3 Create an XML model called **ABContactModel** in the xmlmodel package.

Spec 4 Verification requirements:

- Export properties even if their value is null.
- Export properties even if their value has not changed.
- XML serialization should not perform sort or validation.

6.1.2 Tasks

1. **Create an xmlmodel package.**
2. **Create an XML model.**
3. **Deploy code changes.**
4. **Perform verification steps.**

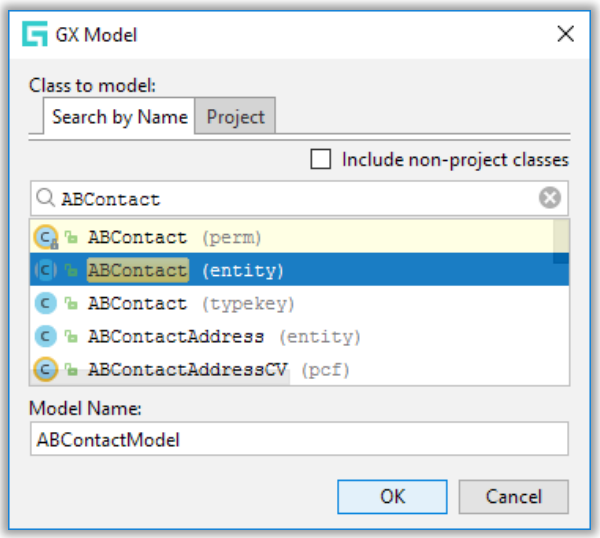
6.1.3 Verification steps

1. **Generate debug console output using Gosu Scratchpad.**
   - In Studio, open Gosu Scratchpad by clicking **Tools → Gosu Scratchpad**.
   - Write code that will test the solution:
     - Create a query that references contact Willam Andy whose publicID is **ab:5**.
       - Make sure only one result is retrieved, otherwise throw an exception.
     - Create a new payload variable the generates XML output utilizing the XML model.
     - Configure GXOptions.
     - Configure XML serialization options.
     - Output the payload to the debug console.
   - Verify that the output in the debug console has the correct format.

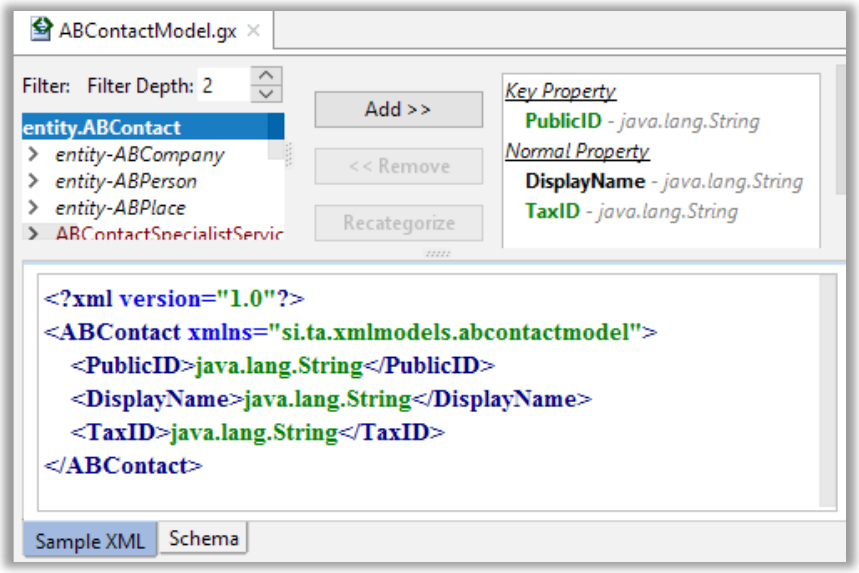
```
<?xml version="1.0"?>
<ABContact xmlns="http://guidewire.com/ab/gx/si.ta.xmlmodels.abcontactxmlmodel">
  <PublicID>ab:5</PublicID>
  <DisplayName>William Andy</DisplayName>
  <TaxID>123-45-6793</TaxID>
</ABContact>
```

6.1.4 Solution

1. **Create an xmlmodel package.**
   - Right-click on **si.ta** folder and select **New → Package.**
   - Enter **xmlmodel** as the new package name.
2. **Create an XML model.**
   - Right-click on the **xmlmodel** package and select **New → GX Model.**
   - Search for the ABContact entity.
   - Enter **ABContactModel** as the model name.



- Click **OK**.
- Select **PublicID** and add as a **Key Property**.
- Select **DisplayName** and add as a **Normal Property**.
- Select **TaxID** and add as **Normal Property**.



3. Deploy code changes.

- From the Studio menu, **Restart** the server.

4. Perform verification steps.

```
uses gw.api.database.Query
uses gw.api.database.Relop
uses gw.api.gx.GXOptions
uses gw.xml.XmlSerializationOptions
uses si.ta.xmlmodel.abcontactmodel.ABContact
```

```
var queryObj = Query.make(ABPerson)
queryObj.compare(ABPerson#PublicID, Relop.Equals, "ab:5")
var targetPerson = queryObj.select().AtMostOneRow
// Configure GXOptions
var gxOpts = new GXOptions() {
  :Incremental = false,
  :Verbose = true
}
// Configure serialization options
var sOpts = new XmlSerializationOptions() {
  :Sort = false,
  :Validate = false
}
// Create payload and display to console
var xml = new ABContact(targetPerson, gxOpts)
var payload = xml.asUTFString(sOpts)
print(payload)
```

6.2 Generate an XML message payload using an XSD

Succeed Insurance has an integration point that needs to create XML about policy holders for an external system. In the external system, policy holder information is structured differently than it is in TrainingApp. The external system has a policyholder XSD that describes how the information should be structured. You need to create code that maps TrainingApp data into this XML structure.

6.2.1 Requirements

Spec 1 Create a new package called **xml**.

Spec 2 Create a new package called **xsd**.

Spec 3 Create a new Gosu class called **PolicyPersonXML**.

Spec 4 Create a new function called **generateXML** for the PolicyPersonXML class that meets the following specifications:

- It takes a public ID as an input parameter.
- It queries for the ABPolicyPerson with that public ID.
- If there is no ABPolicyPerson with that ID, then it prints to console ABPerson with PublicID of <x> not found.
- If there is one ABPolicyPerson with that ID, then it prints to the console the XML for that person. This XML should be created using the **policyholder.xsd** file found at <root>\training.

Spec 5 The following information should be included in the XML:

- Full name
- Tax ID
- Risk assessment
- Set to **high** if the total claim payments made is greater than the total premium paid; otherwise, set to **low**

Hint

Keep in mind that premium amounts are not on ABPolicyPerson directly, but rather on ABPolicyPerson.FinancialSummary. If the ABPolicyPerson has no financial summary, then output ABPolicyPerson with PublicID <x> does not have a Financial Summary.

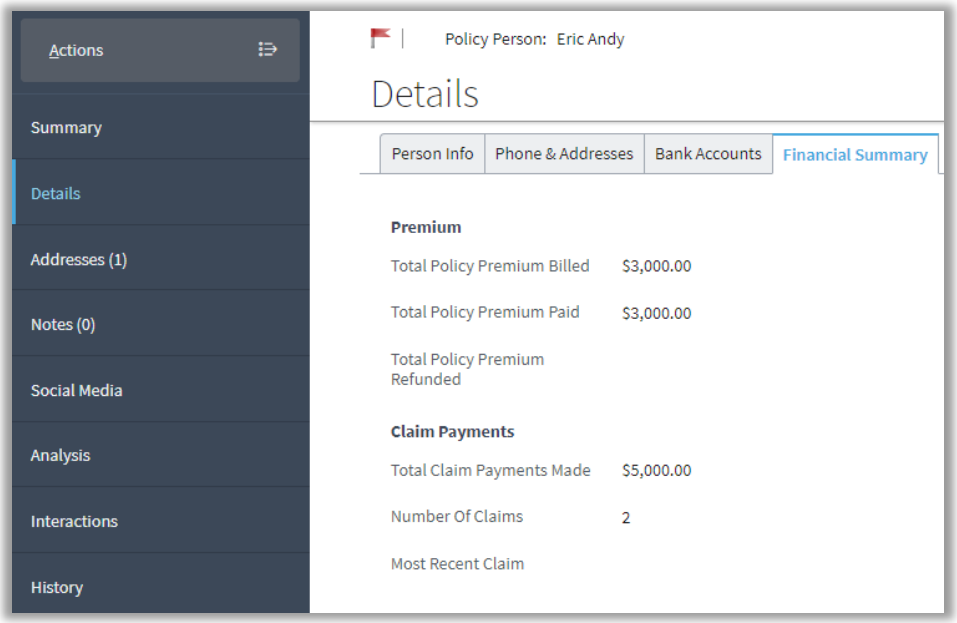
Spec 6 XML serialization should not perform sort or validation.

6.2.2 Tasks

1. **Create new packages.**
2. **Copy policyholder.xsd file located at <root>\training folder to xsd package.**
3. **Create new Gosu class and method.**
4. **Deploy code changes.**
5. **Perform verification steps.**

6.2.3 Verification steps

1. Add financial summary information to Eric Andy in TrainingApp.



2. Generate debug console output using Gosu Scratchpad.

- In Studio, open Gosu Scratchpad by clicking **Tools → Gosu Scratchpad**.
  - ab:98 (Eric Andy)

```
<?xml version="1.0"?>
<PolicyHolder xmlns="http://acmePolicyPerson/ab/gx/sandbox.PolicyHolderModel">
  <FullName>Eric Andy</FullName>
  <TaxID>123-84-7704</TaxID>
  <RiskAssessment>high</RiskAssessment>
</PolicyHolder>
```

  - ab:145 (Betty Yalobusha)

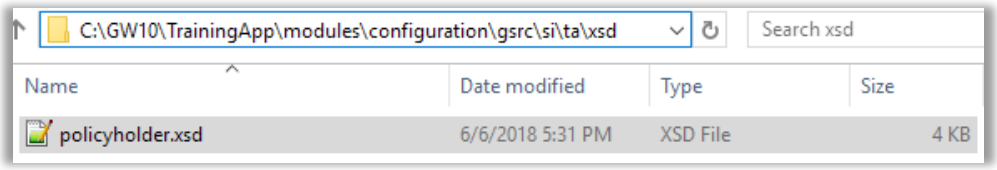
```
ABPolicyPerson with PublicID ab:145 does not have a Financial Summary
<?xml version="1.0"?>
<PolicyHolder xmlns="http://acmePolicyPerson/ab/gx/sandbox.PolicyHolderModel">
  <FullName>Betty Yalobusha</FullName>
  <TaxID>123-84-7751</TaxID>
</PolicyHolder>
```

  - ab:100000 (No contact exists)

```
ABPolicyPerson with PublicID of ab:100000 not found
```

6.2.4 Solution

1. **Create new packages.**
   - Right-click on **si.ta** folder and select **New → Package.**
   - Enter **xml** as the new package name.
   - Right-click on **si.ta** folder and select **New → Package.**
   - Enter **xsd** as the new package name.
2. Copy the **policyholder.xsd** file located at **<root>\training** folder to the xsd package**.**



   - From the Studio menu, **Restart** the server.
3. **Create new Gosu class and method.**
   - Right-click on **xml** package and select **New → Gosu Class.**
   - Enter **`PolicyPersonXML`** as the new Gosu class name.
   - Add the new method to the new class.

```
package si.ta.xml

uses gw.api.database.Query
uses gw.api.database.Relop
uses gw.xml.XmlSerializationOptions
uses si.ta.xsd.policyholder.PolicyHolder

class PolicyPersonXML {

  /**
   * Function will take the publicID of an ABPolicyPerson and generate output.
   */
  @Param("publicID", "Public ID of an ABPolicyPerson contact")
  static function generateXML(publicID: String) {
    var output = ""
    var queryObj = Query.make(ABPolicyPerson)
    queryObj.compare(ABPolicyPerson#PublicID, Relop.Equals, publicID)
    var anABPolicyPerson = queryObj.select().AtMostOneRow
    // create XML output
    if (anABPolicyPerson != null) {
      var xml = new PolicyHolder()
      xml.FullName = anABPolicyPerson.FullName
      xml.TaxID = anABPolicyPerson.TaxID
      // check for financial summary information
      if (anABPolicyPerson.FinancialSummary != null) {
        var totalClaimPayments = anABPolicyPerson.FinancialSummary.TotalClaimPaymentsMade
        var totalPremiumPaid = anABPolicyPerson.FinancialSummary.TotalPolicyPremiumPaid
        if (totalClaimPayments > totalPremiumPaid) {
          xml.RiskAssessment = "high"
        } else {
          xml.RiskAssessment = "low"
        }
      } else {
```

```
        output += String.format("ABPolicyPerson with PublicID %s does not have a Financial
Summary \n", {publicID})
      }
      // Configure serialization options
      var sOpts = new XmlSerializationOptions() {
        :Sort = false,
        :Validate = false
      }
      output += xml.asUTFString(sOpts)
    } else {
      output += String.format("ABPolicyPerson with PublicID of %s not found", {publicID})
    }
    print(output)
  }

}
```

4. Deploy code changes.

- From the Studio menu, **Restart** the server.

5. Perform verification steps.

```
si.ta.xml.PolicyPersonXML.generateXML("ab:98")
si.ta.xml.PolicyPersonXML.generateXML("ab:145")
si.ta.xml.PolicyPersonXML.generateXML("ab:100000")
```

Lesson 7

Integration Views

This lab requires that you use TrainingApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Alice Applegate User whose login/password is aapplegate/gw.

7.1 Create a Contact Integration View

Succeed Insurance has an integration point that needs to create JSON output based on the existing TrainingApp contact Integration View. They want to add new fields and the ability to export only full name and tax id.

7.1.1 Requirements

Spec 1 Extend the existing TrainingApp Integration View located at **integration.schemas.trn.ta** package.

Spec 2 Add the following new fields to the existing output:

- TaxID
- MaritalStatus
- AssignedUser
- BankAccounts
  - BankName
  - BankAccountType

Spec 3 Add a filter called **fraud_check** that exports the following fields:

- Name
- Tax ID

7.1.2 Tasks

1. **Extend the integration.schemas.trn.ta.contact-1.0.schema.json file and add new fields.**
2. **Extend the integration.mappings.trn.ta.contact-1.0.mapping.json file and add new fields.**
3. **Create a new filter.**
4. **Generate wrapper classes.**
5. **Perform verification steps.**

7.1.3 Verification steps

1. **Make sure the following fields have data for William Andy:**
   - Assigned User

- Date of Birth
- Gender
- Marital Status
- Tax ID

2. Generate debug console output using Gosu Scratchpad.

- In Studio, open Gosu Scratchpad by clicking **Tools → Gosu Scratchpad**.
- Write code that will generate JSON output for William Andy whose PublicID is **ab:5**.

```
{
  "AssignedUser" : {
    "DisplayName" : "Mary Maples"
  },
  "BankAccounts" : [ {
    "BankAccountType" : "savings",
    "BankName" : "ACME Credit Union"
  }, {
    "BankAccountType" : "checking",
    "BankName" : "National Bank"
  } ],
  "ContactNotes" : [ {
    "Body" : "William Andy has more questions related to data privacy",
    "ContactNoteType" : "general",
    "Subject" : "William Andy has an inquiry"
  }, {
    "Body" : "William Andy has discovered a serious data entry issue",
    "ContactNoteType" : "problem",
    "Subject" : "William Andy reported an issue"
  }, {
    "Body" : "William Andy sent in his new licence information",
    "ContactNoteType" : "license",
    "Subject" : "William Andy has a new licence"
  }, {
    "Body" : "William Andy has a new phone number",
    "ContactNoteType" : "data_update",
    "Subject" : "New phone number"
  }, {
    "Body" : "William Andy has many general questions",
    "ContactNoteType" : "general",
    "Subject" : "William Andy has questions"
  } ],
  "DateOfBirth" : "1980-08-06T04:00:00.000Z",
  "Gender" : "M",
  "MaritalStatus" : "married",
  "Name" : "William Andy",
  "PrimaryAddress" : {
    "AddressLine1" : "345 Fir Lane",
    "AddressType" : "home",
    "City" : "La Canada",
    "PostalCode" : "91352",
    "State" : "CA"
  },
  "TaxID" : "123-45-6793"
}
```

- Write code that will generate fraud_check filtered JSON output for William Andy whose PublicID is **ab:5**.

```
{
  "Name" : "William Andy",
  "TaxID" : "123-45-6793"
}
```

STOP

7.1.4 Solution

1. **Extend the integration.schemas.trn.ta.contact-1.0.schema.json file and add new fields.**
   - Create a new package.
     - Right-click on **config.integration** folder and click **New → Package**.
     - Enter **schemas.si.ta** as the new package name.
   - Create a new schema file.
     - Right-click on **schemas.si.ta** folder and click **New → File**.
     - Enter **contact-1.0.schema.json** as the new file name.
   - Add schema header information and new definitions.

```
{
  "$schema" : "http://json-schema.org/draft-04/schema#",
  "x-gw-combine" : [
    "trn.ta.contact-1.0"
  ],
  "definitions" : {
    "ContactDetails" : {
      "type" : "object",
      "properties" : {
        "TaxID" : {
          "type" : "string"
        },
        "MaritalStatus" : {
          "type" : "string",
          "x-gw-type": "typekey.MaritalStatus"
        },
        "AssignedUser" : {
          "$ref" : "#/definitions/AssignedUser"
        },
        "BankAccounts" : {
          "type" : "array",
          "items" : {
            "$ref" : "#/definitions/BankAccountDetails"
          }
        }
      }
    },
    "AssignedUser" : {
      "type" : "object",
      "properties" : {
        "DisplayName" : {
```

```
          "type" : "string"
        }
      }
    },
    "BankAccountDetails" : {
      "type" : "object",
      "properties" : {
        "BankName" : {
          "type" : "string"
        },
        "BankAccountType" : {
          "type" : "string",
          "x-gw-type" : "typekey.BankAccountType"
        }
      }
    }
  }
}
```

3. Extend the integration.mappings.trn.ta.contact-1.0.mapping.json file and add new fields.

- Create a new package.
  - Right-click on **integration** folder and click **New → Package**.
  - Enter **mappings.si.ta** as the new package name.
- Create a new mapping file.
  - Right-click on **mappings.si.ta** folder and click **New → File**.
  - Enter **contact-1.0.mapping.json** as the as the new file name.
- Add mapping header information and new mappers.

```
{
  "schemaName" : "si.ta.contact-1.0",
  "combine" : [
    "trn.ta.contact-1.0"
  ],
  "mappers" : {
    "ContactDetails" : {
      "schemaDefinition" : "ContactDetails",
      "root" : "entity.ABContact",
      "properties" : {
        "TaxID" : {
          "path" : "ABContact.TaxID"
        },
        "MaritalStatus" : {
          "path" : "(ABContact as ABPerson).MaritalStatus",
          "predicate" : "ABContact typeis ABPerson"
        },
        "AssignedUser" : {
          "path" : "ABContact.AssignedUser",
          "mapper" : "#/mappers/AssignedUser"
        },
        "BankAccounts" : {
          "path" : "ABContact.BankAccounts",
          "mapper" : "#/mappers/BankAccountDetails"
        }
      }
    },
    "AssignedUser" : {
      "schemaDefinition" : "AssignedUser",
      "root" : "entity.User",
```

```
      "properties" : {
        "DisplayName" : {
          "path" : "User.DisplayName"
        }
      }
    },
    "BankAccountDetails" : {
      "schemaDefinition" : "BankAccountDetails",
      "root" : "entity.BankAccount",
      "properties" : {
        "BankName" : {
          "path" : "BankAccount.BankName"
        },
        "BankAccountType" : {
          "path" : "BankAccount.AccountType"
        }
      }
    }
  }
}
```

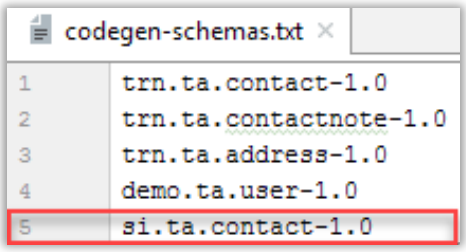
4. Create a new filter.

- Create a new package.
  - Right-click on **integration** folder and click **New → Package**.
  - Enter **filters.si.ta** as the new package name.
- Create a new filter file.
  - Right-click on **filters.si.ta** folder and click **New → File**.
  - Enter **fraud_check-1.0.gql** as the as the new file name.
- Add filtered fields.

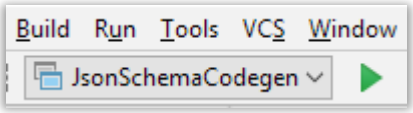
```
{
    Name,
    TaxID
}
```

5. Generate wrapper classes.

- Add the fully-qualified schema name, **si.ta.contact-1.0**, in the **codegen-schemas.txt** file.



- Run JsonSchemaCodegen.



```
Run  JsonSchemaCodegen
C:\jdk1.8.0_152\bin\java -server -ea -Xdebug -Djava.awt.headless=true -Dgw.port=8880 -Xmx4g -Dgw.server.mode=dev -Dgwdebug=true -Dgw.webapp.dir=idea/webapp -
Looking for the devroot starting from C:\GW10\TrainingApp\.
Detected CONFIG environment, root: C:\GW10\TrainingApp
Classpath modules: [configuration]
Module path: [configuration]
--- Json Schema Wrapper Code Generator Tool ---

--- Initializing ---

--- Initialization Finished ---

2018-08-06 20:11:00,843 INFO Generating jsonschema wrapper classes for all modules
2018-08-06 20:11:01,509 INFO ExternalConfigurationProviderPlugin is disabled. All configuration substitutions without default values will be left unchanged.
2018-08-06 20:11:01,732 INFO Parsing C:\GW10\TrainingApp\modules\configuration\config\config.xml for registry. No substitution is supported at that level.
2018-08-06 20:11:01,843 INFO Generating wrapper classes for schema demo.ta.user-1.0 in module C:\GW10\TrainingApp\modules\configuration
2018-08-06 20:11:01,878 INFO Generating wrapper classes for schema trn.ta.contactnote-1.0 in module C:\GW10\TrainingApp\modules\configuration
2018-08-06 20:11:01,882 INFO Generating wrapper classes for schema trn.ta.address-1.0 in module C:\GW10\TrainingApp\modules\configuration
2018-08-06 20:11:01,888 INFO Generating wrapper classes for schema si.ta.contact-1.0 in module C:\GW10\TrainingApp\modules\configuration
2018-08-06 20:11:01,906 INFO Generating wrapper classes for schema trn.ta.contact-1.0 in module C:\GW10\TrainingApp\modules\configuration

Process finished with exit code 0
```

6. Perform verification steps.

- Write code that will generate JSON output for William Andy whose PublicID is **ab:5**

```
uses gw.api.database.Query
uses gw.api.database.Relop
uses gw.api.json.JsonConfigAccess

// Query for Contact
var queryObj = Query.make(ABContact)
queryObj.compare(ABContact#PublicID, Relop.Equals, "ab:5")
var targetObj = queryObj.select().AtMostOneRow
// Create JsonMapper object
var jsonMapper = JsonConfigAccess.getMapper("si.ta.contact-1.0", "ContactDetails")
// Create TransformResult object
var transformResult = jsonMapper.transformObject(targetObj)
// Create output
var payloadJSON = transformResult.toPrettyJsonString()
print(payloadJSON)
```

- Write code that will generate fraud_check filtered JSON output for William Andy whose PublicID is **ab:5**

```
uses gw.api.database.Query
uses gw.api.database.Relop
uses gw.api.json.JsonConfigAccess
uses gw.api.json.mapping.JsonMappingOptions

// Query for Contact
var queryObj = Query.make(ABContact)
queryObj.compare(ABContact#PublicID, Relop.Equals, "ab:5")
var targetObj = queryObj.select().AtMostOneRow
// Create JsonMapper object
var jsonMapper = JsonConfigAccess.getMapper("si.ta.contact-1.0", "ContactDetails")
// Create JsonMapperOptions object
var mappingOpts = new JsonMappingOptions().withFilter("si.ta.fraud_check-1.0")
// Create TransformResult object
var transformResult = jsonMapper.transformObject(targetObj, mappingOpts)
// Create output
var payloadJSON = transformResult.toPrettyJsonString()
print(payloadJSON)
```

Lesson 8

RESTful Web Services

This lab requires that you use TrainingApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Alice Applegate User whose login/password is aapplegate/gw.

8.1 Create a Contact REST API

Succeed Insurance wants to create a new contact API that exposes contact information defined in the Integration View created in the Integration View lab.

8.1.1 Requirements

Spec 1 Define a contact API that exposes contact detail information defined in the Integration View.

Spec 2 The basePath must be **/si/contact/v1**.

Spec 3 Define an API handler class whose method is called getContactDetailInformation.

Spec 4 Make the API available to external resources.

8.1.2 Tasks

1. **Define the API schema.**
2. **Define the API handler class.**
3. **Publish the API.**
4. **Deploy code changes.**
5. **Perform verification steps.**

8.1.3 Verification steps

1. **Use swagger-ui distribution to test.**
   - Open a browser and enter the following URL to launch swagger-ui:

     http://localhost:8880/ab/resources/swagger-ui
   - Manually change the endpoint to the new API and click **Explore**.

     http://localhost:8880/ab/rest/si/ta/contact/v1/swagger.json

   - Authorize the external resource.
     - Click the **Authorize** button.

    - Enter TrainingApp credentials:
        - Username: **su**
        - Password: **gw**
    - Click **Authorize** button and close window.
  - Test **GET** operation for **/contacts/{contactId}.**
    - Click **GET** button.
    - Click **Try it out** button.
    - Enter **ab:5** as the contactId.
    - Click **Execute**.
  - Server code response should be 200 along with contact detail information.

Server response

Code	Details
200	**Response body**


```
{
  "AssignedUser": {
    "DisplayName": "Mary Maples"
  },
  "BankAccounts": [
    {
      "BankAccountType": "savings",
      "BankName": "ACME Credit Union"
    },
    {
      "BankAccountType": "checking",
      "BankName": "National Bank"
    }
  ],
  "ContactNotes": [
    {
      "Body": "William Andy has more questions related to data privacy",
      "ContactNoteType": "general",
      "Subject": "William Andy has an inquiry"
    },
    {
      "Body": "William Andy has discovered a serious data entry issue",
      "ContactNoteType": "problem",
      "Subject": "William Andy reported an issue"
    },
    {
      "Body": "William Andy sent in his new licence information",
      "ContactNoteType": "license",
      "Subject": "William Andy has a new licence"
    },
    {
      "Body": "William Andy has a new phone number",
      "ContactNoteType": "data_update",
      "Subject": "New phone number"
    },
    {
      "Body": "William Andy has many general questions",
      "ContactNoteType": "general",
      "Subject": "William Andy has questions"
    }
  ],
  "DateOfBirth": "1980-08-06T04:00:00.000Z",
  "Gender": "M",
  "MaritalStatus": "married",
  "Name": "William Andy",
  "PrimaryAddress": {
    "AddressLine1": "345 Fir Lane",
    "AddressType": "home",
    "City": "La Canada",
    "PostalCode": "91352",
    "State": "CA"
  },
  "TaxID": "123-45-6793"
}
```


STOP

8.1.4 Solution

1. **Define the API schema.**
   - Create a new package.
     - Right-click on **apis** package and click **New → Package**.
     - Enter **si.ta** as the new package name.
   - Create a new API schema file.
     - Right-click on **apis.si.ta** package and click **New → File**.
     - Enter **contact-1.0.swagger.yaml** as the new file name.
   - Add schema header information and new paths.

```
swagger: '2.0'
info:
  version: '1.0'
  title: "Contact API"
  description: "Contact API"
basePath: /si/ta/contact/v1
x-gw-schema-import:
 contact: si.ta.contact-1.0
x-gw-apihandlers:
- si.ta.restapi.ContactAPIHandler
produces:
- application/json
consumes:
- application/json
paths:
  /contacts/{contactId}:
    get:
      summary: "Retrieves a contact details"
      operationId: getContactDetailInformation
      parameters:
      - $ref: "#/parameters/contactId"
      responses:
        '200':
          description: "Contact details returned"
          schema:
            $ref: "contact#/definitions/ContactDetails"
parameters:
  contactId:
    name: contactId
    in: path
    type: string
    required: true
```

2. **Define the API handler class.**
   - Create a new package.
     - Right-click on **si.ta** package and click **New → Package**.
     - Enter **restapi** as the new package name.
   - Create a new Gosu class.
     - Right-click on **restapi** package and click **New → Gosu Class**.
     - Enter **ContactAPIHandler** as the new Gosu class name.

   - Configure the API method.

```
package si.ta.restapi

uses gw.api.database.Query
uses gw.api.database.Relop
uses gw.api.json.JsonConfigAccess
uses gw.api.json.mapping.TransformResult

class ContactAPIHandler {

  function getContactDetailInformation(contactId : String) : TransformResult {
    // Query for contact
    var user = findContactById(contactId)
    // Create JsonMapper object
    var jsonMapper = JsonConfigAccess.getMapper("si.ta.contact-1.0", "ContactDetails")
    return jsonMapper.transformObject(user)
  }

  private function findContactById(id : String) : ABContact {
    // Query for contact
    var queryObj = Query.make(ABContact)
    queryObj.compare(ABContact#PublicID, Relop.Equals, id)
    var targetObj = queryObj.select().AtMostOneRow
    return targetObj
  }

}
```

3. **Publish the API.**
   - Update the **published-apis.yaml** file with the fully-qualified name of the new API.

```
published-apis.yaml
1  apis:
2    - name: gw.pl.framework.api_list-1.0
3    - name: trn.ta.contact-1.0
4    - name: demo.ta.user-1.0
5    - name: si.ta.contact-1.0
6  defaultTemplate:
7    - name: gw.pl.framework.dev_template-1.0
```

4. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
5. **Perform verification steps.**

Lesson 9

SOAP Web Services

This lab requires that you use TrainingApp, ExternalApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'. Start ExternalApp using the **Start ExternalApp** shortcut.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Alice Applegate User whose login/password is aapplegate/gw.

9.1 Consume a SOAP web service

Succeed Insurance needs to connect to a WSI web service hosted on an external system. The web service API offers a method that takes a VIN number as a string argument and returns a string that contains the vehicle's color, year, make, and model.

9.1.1 Requirements

Spec 1 The web service collection package should be called vehiclevin.

Spec 2 ExternalApp's VehicleWsiAPI is accessible at: http://localhost:8890/ab/ws/externalapp/webservice/VehicleAPI?WSDL

Spec 3 Connect to the web service using HTTP authentication with externalappuser/gw as the username/password.

Spec 4 Specify a parameter that limits server response time to 30000 ms.

9.1.2 Tasks

1. **Create a new package called webservice.vehicle.**
2. **Create a web service collection to access the VIN service called vehiclevinwsc.**
3. **Deploy code changes.**
4. **Perform verification steps.**

9.1.3 Verification steps

1. **In Gosu Scratchpad, execute code that calls the external system web service to verify each VIN number in the following table:**

Vehicle Identification Number (VIN)	Result
12345678901234567	Blue 1991 Dodge Challenger
ABCDEFGHIJKLMNOPQ	Black 2007 BMW X3
111	Vin is not 17 characters
$$$$$$$$$$$$$$$$$	Vin has invalid characters

   - Verify the output.

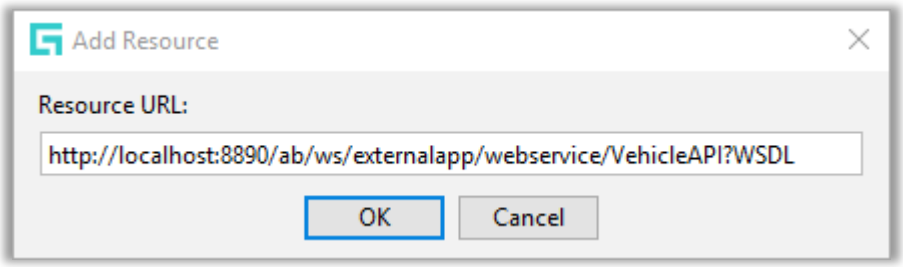
```
gger   Console
c6273923961          01ca56bd-df
Blue 1991 Dodge Challenger
c6273923961          11ea105a-cf
Black 2007 BWM X3
c6273923961          0ee2bfa2-35
VIN is not 17 characters
c6273923961          9b099b4f-85
VIN has invalid characters
```

STOP

9.1.4 Solution

1. **Create a new package called webservice.vehicle.**
   - Right-click on **si.ta** package and select **New → Package**.
   - Enter **webservice.vehicle** as the new package name.
2. **Create a web service collection to access the VIN service.**
   - Right-click on the **vehicle** package and select **New → Web Service Collection**.
   - Enter **vehiclevinwsc** as the new Web Service Collection name.
   - Click the **Add New Resource** button.

   - Enter the ExternalApp VehicleAPI WSDL URL: http://localhost:8890/ab/ws/externalapp/webservice/VehicleAPI?WSDL



   - Click the **Fetch** button.

3. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
4. **Perform verification steps.**

```
uses si.ta.webservice.vehicle.vehiclevinwsc.vehicleapi.VehicleAPI

var api = new VehicleAPI()
// set API properties
api.Config.Http.Authentication.Basic.Username = "externalappuser"
api.Config.Http.Authentication.Basic.Password = "gw"
api.Config.CallTimeout = 30000
// verify output
print(api.verifyVehicle("12345678901234567"))
print(api.verifyVehicle("ABCDEFGHIJKLMNOPQ"))
print(api.verifyVehicle("111"))
print(api.verifyVehicle("$$$$$$$$$$$$$$$$$$"))
```

9.2 Publish a SOAP web service

Succeed Insurance needs to publish a WS-I web service that will allow an external system to retrieve company contact information. The API must contain three functions that will: verify if a company exists given a tax identification (tax ID), create a contact note, and generate an employee summary.

9.2.1 Requirements

Spec 1 Create a WS-I web service named CompanyAPI in a package called company.

Spec 2 Create a method called doesCompanyExist:

- Verify that a company exists given a tax identification (tax ID) number.
- Define an input parameter for the tax ID as a String.
- Identify if a company exists for the given tax ID.
- For a company that exists for the given tax ID, return true as a boolean value.

Spec 3 Create a method called createContactNote:

- Create a contact note.
- Define two input parameters: String representing a tax ID and String identifying the body of the note.
- If the tax ID does not correspond to a company, the method does nothing.
- If a company exists for the given tax ID, create a note whose ContactNoteType is **general**, whose subject is **External note**, and whose body is specified by the external system as an input parameter.
- Add the note to the company's ContactNotes array and save the changes.
- The function does not return a value.

Spec 4 Create a method called getEmployeeSummary:

- Generate an Employee Summary return object of type EmployeeSummary.
- Define an input parameter for the tax ID as a String.
- If the tax ID does not correspond to a company, the method does nothing.
- If a company exists with the tax ID, return an EmployeeSummary object.
- The EmployeSummary object must contain the following fields:
  - Number of employees as an integer.
  - Employee score as an integer.
  - Headquarters location as a comma-separated string created by concatenating the city, state, and country of the company's primary address.

9.2.2 Tasks

1. **Create a web service package.**
2. **Create EmployeeSummary class return type.**
3. **Create CompanyAPI Gosu class.**
4. **Create CompanyAPI methods.**

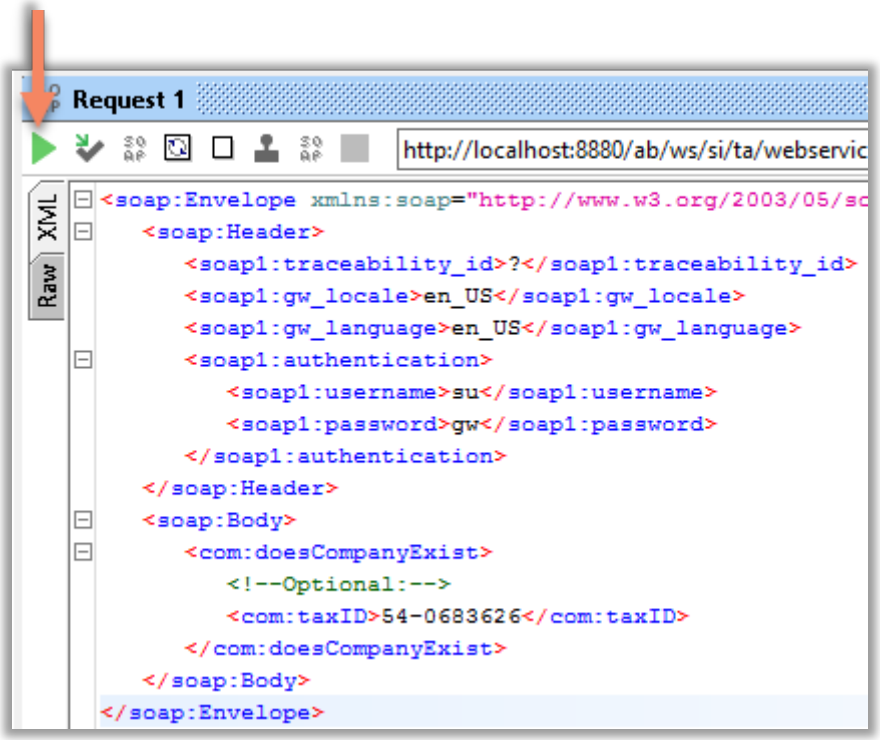
Tip

Create a helper method called **findCompanyByTaxID** that performs a company query.

5. **Deploy code changes.**
6. **Perform verification steps.**

9.2.3 Verification steps

1. **Obtain the CompanyAPI WSDL URL.**
   - Launch a web browser.
   - Enter the following URL: localhost:8880/ab/ws
   - Navigate to **Document/Literal Web Services.si.ta.webservice.company** and select **CompanyAPI** web service.
     - Record the URL from browser address field.
     - http://localhost:8880/ab/ws/si/ta/webservice/company/CompanyAPI?WSDL
2. **Launch SoapUI to test the CompanyAPI.**
3. **Create a new SOAP project.**
   - File → New SOAP Project
   - Enter **CompanyWSC** as the project name.
   - Enter http://localhost:8880/ab/ws/si/ta/webservice/company/CompanyAPI?WSDL as the initial WSDL.
   - Select **Create Requests** option.
   - Click OK.
4. **Test the doesCompanyExist API.**
   - Double-click on Request 1 under:
     CompanyWSC → CompanyAPISoap12Binding → doesCompanyExist.
   - Enter the following in the request XML:
     - soap1:gw_locale: **en_US**
     - soap1:gw_language: **en_US**
     - soap1:username: **su**
     - soap1:password: **gw**
     - com:taxID: **54-0683626**
     - Click the submit request arrow to execute the request.



- Verify the output

5. **Test the getEmployeeSummaryAPI.**
   - Double-click on Request 1 under:
     CompanyWSC → CompanyAPISoap12Binding → getEmployeeSummay.
   - Enter the following in the request XML:
     - soap1:gw_locale: **en_US**
     - soap1:gw_language: **en_US**
     - soap1:username: **su**
     - soap1:password: **gw**
     - com:taxID: **54-0683626**
     - Click the submit request arrow to execute the request.

Request 1

http://localhost:8880/ab/ws/si/ta/webservice/company/Con

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
   <soap:Header>
      <soap1:traceability_id>?</soap1:traceability_id>
      <soap1:gw_locale>en_US</soap1:gw_locale>
      <soap1:gw_language>en_US</soap1:gw_language>
      <soap1:authentication>
         <soap1:username>su</soap1:username>
         <soap1:password>gw</soap1:password>
      </soap1:authentication>
   </soap:Header>
   <soap:Body>
      <com:getEmployeeSummary>
         <!--Optional:-->
         <com:taxID>54-0683626</com:taxID>
      </com:getEmployeeSummary>
   </soap:Body>
</soap:Envelope>
```

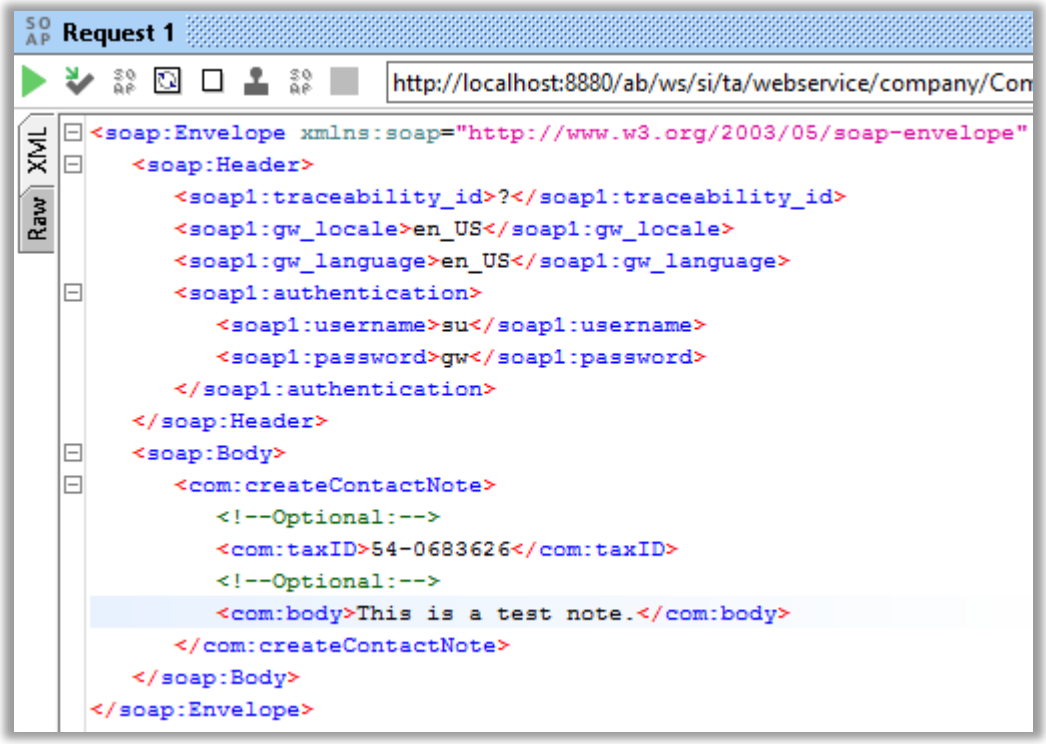
- Verify the output.

```
<soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
   <soap12:Header>
      <gwsoap:traceability_id xmlns:gwsoap="http://guidewire.com/ws/soapheaders">?<
   </soap12:Header>
   <soap12:Body>
      <getEmployeeSummaryResponse xmlns="http://example.com/si/ta/webservice/compan
         <return xmlns:pogo="http://example.com/si/ta/webservice/company">
            <pogo:EmployeeScore>0</pogo:EmployeeScore>
            <pogo:HeadquartersLocation>La Canada,CA,US</pogo:HeadquartersLocation>
            <pogo:NumberOfEmployees>4</pogo:NumberOfEmployees>
         </return>
      </getEmployeeSummaryResponse>
   </soap12:Body>
</soap12:Envelope>
```

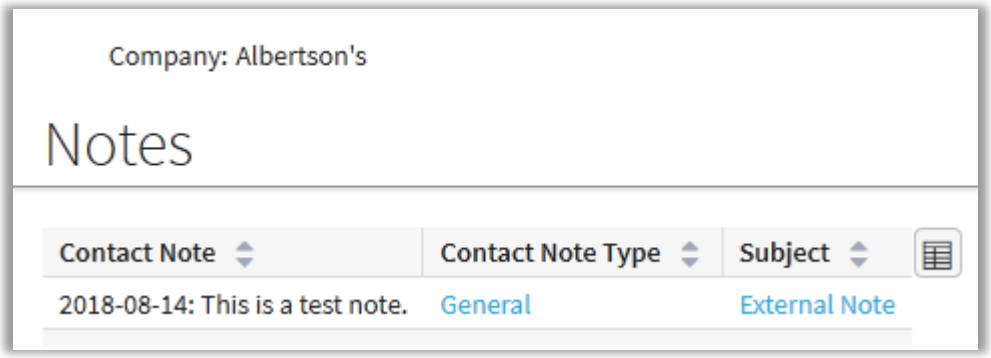
6. Test the createContactNote API.

- Double-click on Request 1 under:
  CompanyWSC → CompanyAPISoap12Binding → createContactNote.
- Enter the following in the request XML:
  - soap1:gw_locale: **en_US**
  - soap1:gw_language: **en_US**
  - soap1:username: **su**
  - soap1:password: **gw**
  - com:taxID: **54-0683626**
  - com:body: This is a test note.

- Click the submit request arrow to execute the request.



- Log in to TrainingApp to verify if the note was added to Albertson's.



9.2.4 Solution

1. **Create a web service package.**
   - Right-click on **si.ta** package and select **New → Package**.
   - Enter **webservice.company** as the new package name.
2. **Create EmployeeSummary class return type.**

- Right-click on the **company** package and select **New → Gosu Class**.
- Enter **EmployeeSummary** as the new Gosu class name.

```
package si.ta.webservice.company

uses gw.xml.ws.annotation.WsiExportable

/**
 * Created by training.
 */
@WsiExportable
final class EmployeeSummary {
  // class properties
  var _numberOfEmployees: int as NumberOfEmployees
  var _employeeScore: int as EmployeeScore
  var _headquartersLocation: String as HeadquartersLocation
}
```

3. **Create CompanyAPI Gosu class.**
   - Right-click on the **company** package and select **New → Gosu Class**.
   - Enter **CompanyAPI** as the new Gosu class name.
4. **Create CompanyAPI methods.**

```
package si.ta.webservice.company

uses gw.api.database.Query
uses gw.api.database.Relop
uses gw.transaction.Transaction
uses gw.xml.ws.annotation.WsiWebService

/**
 * Created by training.
 */
@WsiWebService
class CompanyAPI {

  /**
   * Function verifies if a company exists.
   */
  function doesCompanyExist(taxID: String): boolean {
    // query for Company for a given taxID
    var targetCompany = findCompanyByTaxID(taxID)
    if (targetCompany != null) {
      return true
    } else {
      return false
    }
  }

  /**
   * Function creates a ContactNote for a given company.
   */
  @Param("taxID", "Company taxID")
  @Param("body", "String identifying the body of the note")
  function createContactNote(taxID: String, body: String): void {
    // query for Company for a given taxID
    var targetCompany = findCompanyByTaxID(taxID)
    if (targetCompany != null){
```

```
      // create new bundle
      Transaction.runWithNewBundle(\newBundle -> {
        // add query read-only object to newBundle
        targetCompany = newBundle.add(targetCompany)
        // create new Note and add to Company
        var newNote = new ContactNote()
        newNote.ContactNoteType = typekey.ContactNoteType.TC_GENERAL
        newNote.Subject = "External Note"
        newNote.Body = body
        targetCompany.addToContactNotes(newNote)
      })
    }
  }

  /**
   * Function returns an EmployeeSummary object for a given tax ID.
   */
  @Param("taxID", "Company tax ID")
  @Returns("EmployeeSummary object")
  function getEmployeeSummary(taxID: String): EmployeeSummary {
    // query for Company for a given taxID
    var targetCompany = findCompanyByTaxID(taxID)
    if (targetCompany != null) {
      var anEmployeeSummary = new EmployeeSummary()
      anEmployeeSummary.EmployeeScore = targetCompany.EmployeeScore
      anEmployeeSummary.NumberOfEmployees = targetCompany.NumberOfEmployees
      anEmployeeSummary.HeadquartersLocation = targetCompany.PrimaryAddress.City
          + "," + targetCompany.PrimaryAddress.State
          + "," + targetCompany.PrimaryAddress.Country
      return anEmployeeSummary
    } else {
      return null
    }
  }

  //////    Helper Methods    //////

  /**
   * Method takes taxID and returns company object
   */
  @Param("taxID", "Company tax ID")
  @Returns("Finds company by tax id. Returns type ABCompany")
  private function findCompanyByTaxID(taxID: String): ABCompany {
    // validate all input params sent in by the external system
    if (taxID == null or taxID.Empty) {
      throw new IllegalArgumentException("Invalid input parameter, taxID is null or empty!")
    } else {
      var queryObj = Query.make(ABCompany)
      queryObj.compare(ABCompany#TaxID, Relop.Equals, taxID)
      var resultObj = queryObj.select().AtMostOneRow
      return resultObj
    }
  }
}
```

5. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
6. **Perform verification steps.**

Lesson 10

Plugins

This lab requires that you use TrainingApp, ExternalApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'. Start ExternalApp using the **Start ExternalApp** shortcut.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Super User whose login/password is su/gw.

10.1 Create an exchange rate plugin

Succeed Insurance needs to retrieve exchange rate values from an external system. The exchange rates shall be retrieved from the external system by clicking on the **Invoke ExchangeRateSet Plugin** button in the UI.

10.1.1 Requirements

Spec 1 Configure the rate exchange plugin so that it will use a custom Gosu class.

Spec 2 The rate exchange plugin must pass authentication parameters to the Gosu class.

Spec 3 Create a custom Gosu class that utilizes the given code for the createExchangeRateSet method.

10.1.2 Tasks

1. **Create a plugin package.**
2. **Create a custom rate exchange Gosu class using best practice naming convention.**
   - Implement IExchangeRateSetPlugin and InitializablePlugin interfaces and methods.
   - Copy the following code for the createExchangeRateSet method:

```
override function createExchangeRateSet() : ExchangeRateSet {
  // Create and initialize new exchange rate set
  var erSet = new ExchangeRateSet()
  erSet.Name = "Lab ExchangeRateSet " + gw.api.util.DateUtil.currentDate()
  erSet.Description = "Lab ExchangeRateSet"
  erSet.MarketRates = true
  erSet.EffectiveDate = gw.api.util.DateUtil.currentDate()
  // Create external web service object and set API properties
  var CurrencyAPI = new trainingapp.webservice.currency.exchangeratewsc.currencyapi.CurrencyAPI()
  CurrencyAPI.Config.Http.Authentication.Basic.Username = _username
  CurrencyAPI.Config.Http.Authentication.Basic.Password = _password
  var baseCurrencies = Currency.getTypeKeys(true)
  var priceCurrencies = Currency.getTypeKeys(true)
  // For each base/price currency pair, get exchange rate and add it to set
  for (currentBaseCurrency in baseCurrencies) {
    for (currentPriceCurrency in priceCurrencies) {
      var newExchangeRate = new ExchangeRate()
      newExchangeRate.BaseCurrency = currentBaseCurrency
```

```
        newExchangeRate.PriceCurrency = currentPriceCurrency
        newExchangeRate.Rate = CurrencyAPI.getConversionRate(
            currentBaseCurrency as java.lang.String,
            currentPriceCurrency as java.lang.String)
        erSet.addToExchangeRates(newExchangeRate)
      }
    }
    return erSet
}
```

- Create four class variables for authentication parameters.
  - Two private static final variables for the plugin parameter names.
  - Two private static variables for the authentication values retrieved form plugin registry.
- Configure the **set Parameters** property to get plugin parameters.

3. **Modify the IExchangeRateSetPlugin registry.**
   - Update Gosu class reference.
   - Create two plugin parameters:
     - Name = externalAppUsername Value = externalappuser
     - Name = externalAppPassword Value = gw
4. **Deploy code changes.**
5. **Perform verification steps.**

10.1.3 Verification steps

1. **Start ExternalApp.**
   - Launch **Start ExternalApp** shortcut.
2. **Open TrainingApp.**
   - Navigate to **Administration → Training:Plugins → Predefined Plugins,** and click on the **Invoke ExchangeRateSet Plugin** button.
3. **Verify exchange rates are retrieved.**
   - There are two ways to verify exchange rates are retrieved:
     - The **Retrieved On** column shows the date and time the exchange rates were retrieved.
     - The rate for each USD-to-nonUSD currency is a decimal value that goes to at least 4 digits. The whole number part of the first two digits are always the same each time you run the plugin. The third and fourth digits vary based on the current minute. For example, the USD to EUR rate is 0.75XX, where XX is the current minute.

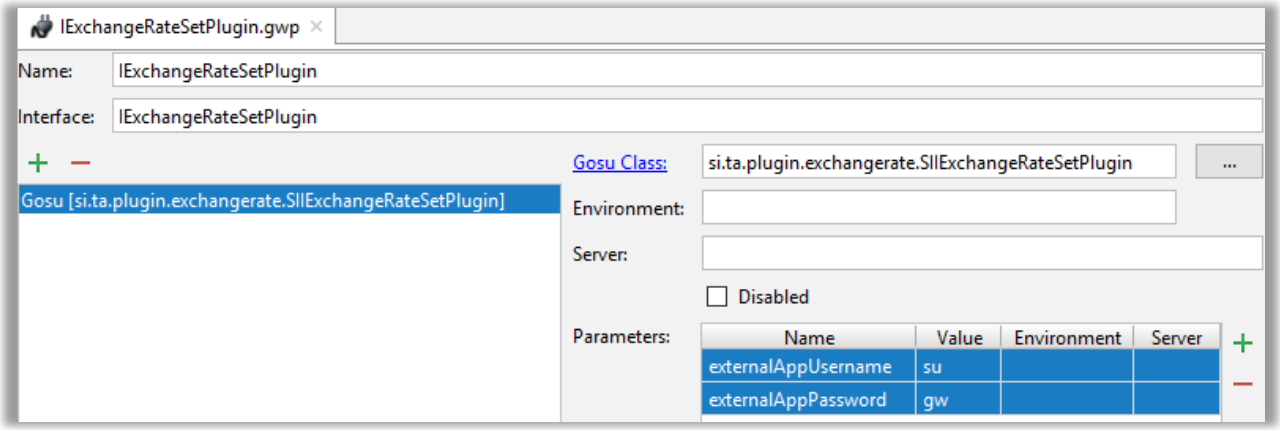
10.1.4 Solution

1. **Create a plugin package.**
   - Right-click on **si.ta** package and select **New → Package**.
   - Enter **plugin.exchangerate** as the new package name.
2. **Create a custom rate exchange Gosu class using best practice naming convention.**
   - Right-click on the **exchangerate** package and select **New → Gosu Class**.

- Enter **SIIExchangeRateSetPlugin** as the new Gosu class name.

```
package si.ta.plugin.exchangerate

uses gw.plugin.InitializablePlugin
uses gw.plugin.exchangerate.IExchangeRateSetPlugin

class SIIExchangeRateSetPlugin implements IExchangeRateSetPlugin, InitializablePlugin{
  // authentication parameters
  private static final var USERNAME = "externalAppUsername"
  private static var _username : String
  private static final var PASSWORD = "externalAppPassword"
  private static var _password : String

  override function createExchangeRateSet() : ExchangeRateSet {
    // Create and initialize new exchange rate set
    var erSet = new ExchangeRateSet()
    erSet.Name = "Lab ExchangeRateSet " + gw.api.util.DateUtil.currentDate()
    erSet.Description = "Lab ExchangeRateSet"
    erSet.MarketRates = true
    erSet.EffectiveDate = gw.api.util.DateUtil.currentDate()
    // Create external web service object and set API properties
    var CurrencyAPI = new
trainingapp.webservice.currency.exchangeratewsc.currencyapi.CurrencyAPI()
    CurrencyAPI.Config.Http.Authentication.Basic.Username = _username
    CurrencyAPI.Config.Http.Authentication.Basic.Password = _password
    var baseCurrencies = Currency.getTypeKeys(true)
    var priceCurrencies = Currency.getTypeKeys(true)
    // For each base/price currency pair, get exchange rate and add it to set
    for (currentBaseCurrency in baseCurrencies) {
      for (currentPriceCurrency in priceCurrencies) {
        var newExchangeRate = new ExchangeRate()
        newExchangeRate.BaseCurrency = currentBaseCurrency
        newExchangeRate.PriceCurrency = currentPriceCurrency
        newExchangeRate.Rate = CurrencyAPI.getConversionRate(
            currentBaseCurrency as java.lang.String,
            currentPriceCurrency as java.lang.String)
        erSet.addToExchangeRates(newExchangeRate)
      }
    }
    return erSet
  }

  override property set Parameters(map : Map<Object, Object>) {
    _username = map.get(USERNAME) as String
    _password = map.get(PASSWORD) as String
  }

}
```

3. Modify the IExchangeRateSetPlugin registry.



4. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
5. **Perform verification steps.**

Lesson 11

Messaging Overview

11.1 No lab

Lesson 12

Triggering Messages

This lab requires that you use TrainingApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Super User whose login/password is su/gw.

12.1 Configure an event aware entity and destination

Note

The comprehensive scenario below is applicable to all the messaging labs. Therefore, the specifications applicable to each messaging lab will be listed in the **Requirements** section.

Succeed Insurance must determine if a given contact has been involved with a previous act of insurance fraud. To implement their fraud prevention system, they must send a message to an external system for every new contact or for an existing contact whose tax ID is updated. The payload must be in XML format and must contain the contact full name, tax ID, and a reference value generated by Guidewire to help identify the message. The external system must respond immediately to the fraud investigation request with a fraud report code.

External system information:

The WSDL URL to the external system is:

http://localhost:8890/ab/ws/externalapp/webservice/FraudReportAPI?WSDL

The authentication parameters are:

- Username: externalappuser
- Password: gw

Do not code the authentication parameters in the implementation code – use plugin parameters that are passed to the code.

API method checkForFraudReport requires the transformed payload as its argument.

The valid acknowledgment report codes are as follows:

- **1** – Request processed, no fraud report found
  - Acknowledge the message.
- **2** – Request processed, fraud report found!
  - Acknowledge the message.
- **4** – Request could not be processed (Payload Format Error)

  - Acknowledge the message with error using error category **Payload Format**.
- **5** – Request could not be processed (Database Unavailable)
  - Acknowledge the message with error using error category **Database Contention**.
- **Default** – Request could not be processed (Acknowledgment Code Invalid). If the error code returned from the external system is not valid, then acknowledge the message with error using a new error category called **Acknowledgement Code Invalid**.

For training purposes, output an acknowledgment message to console using the print statement.

12.1.1 Requirements

Spec 1 Verify ABContact entity is configured to trigger messages based on default events.

Spec 2 Create a new messaging destination called **Fraud Check**.

- ID = 30
- Name = Fraud Check
- Transport Plugin = DefaultPrintToConsoleTransport
- Subscribe to events that trigger for new or modified contacts.

Tip

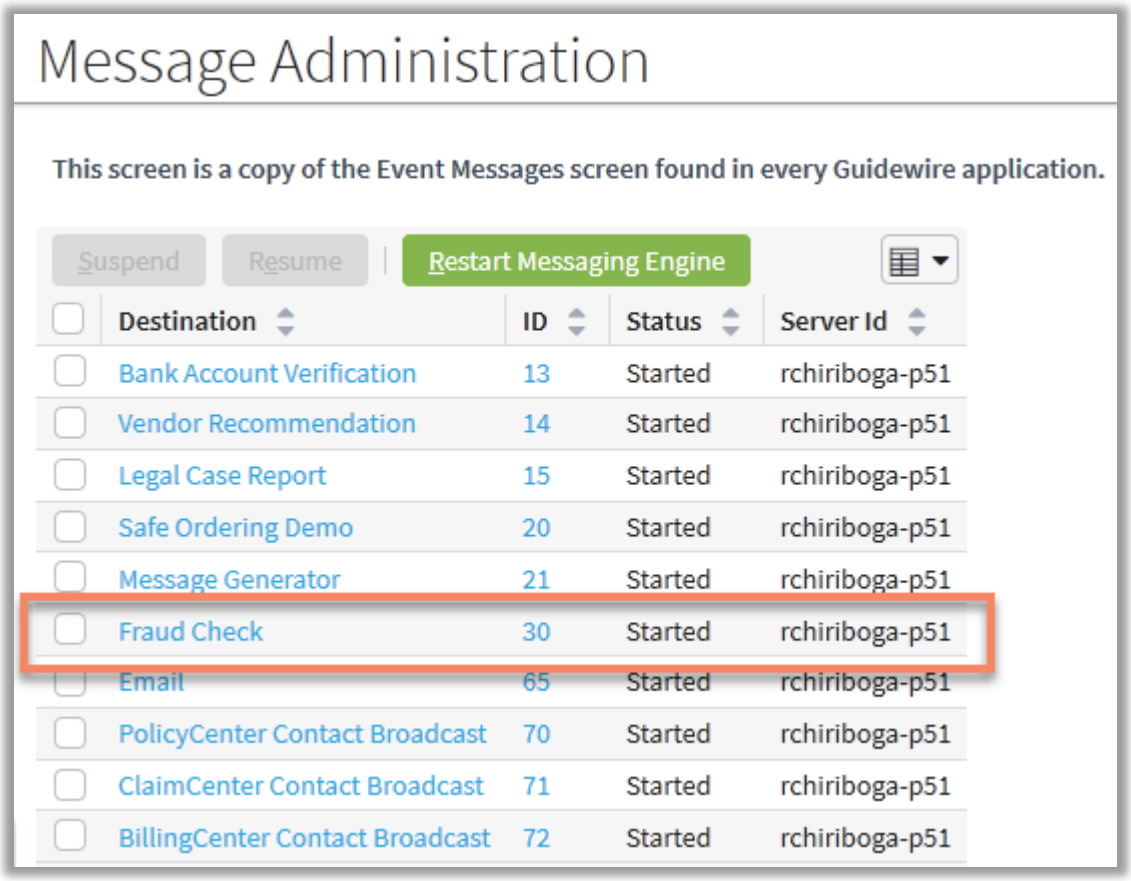
Do not change default connectivity and error parameters.

12.1.2 Tasks

1. **Verify ABContact entity delegates to EventAware delegate.**
2. **Create a new messaging destination called Fraud Check.**
3. **Deploy code changes.**
4. **Perform verification steps.**

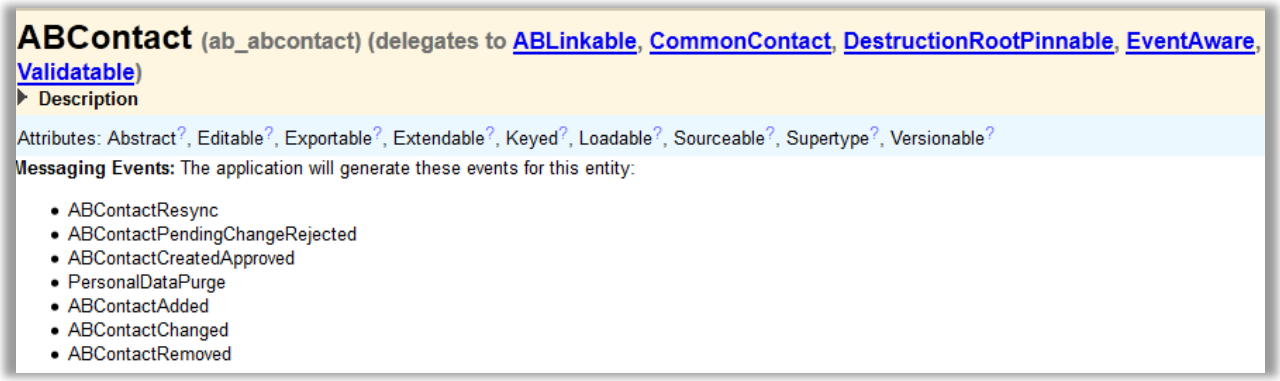
12.1.3 Verification steps

1. **Launch TrainingApp.**
2. **Verify the new destination in Administration screen.**
   - Navigate to **Administration → Training: Messaging**.
   - Click on the **Message Administration** link.
   - Verify the **Fraud Check** destination status is **Started**.

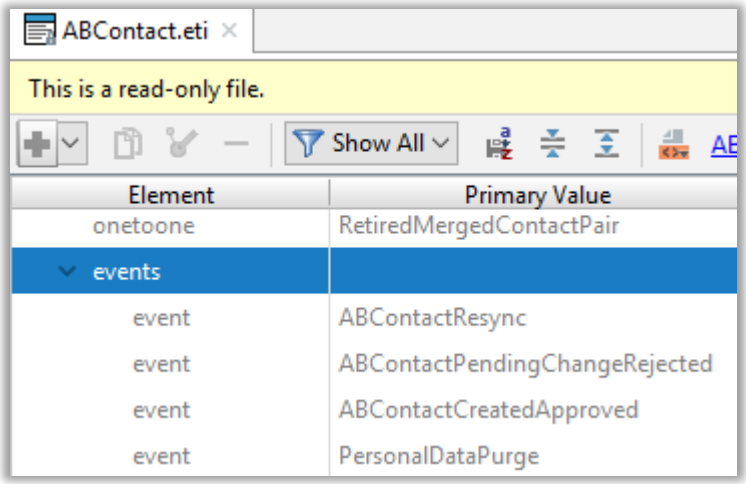


12.1.4 Solution

1. **Verify ABContact entity delegates to EventAware delegate.**
   - There are two approaches to verify that ABContact delegates to EventAware:
     - Use the data dictionary, or



- Verify event delegate was added to ABContact.eti file.



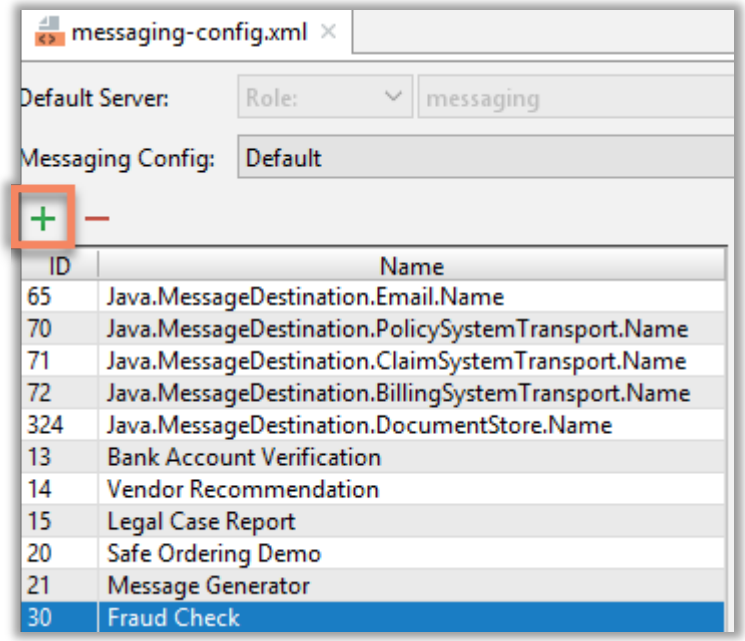
Note

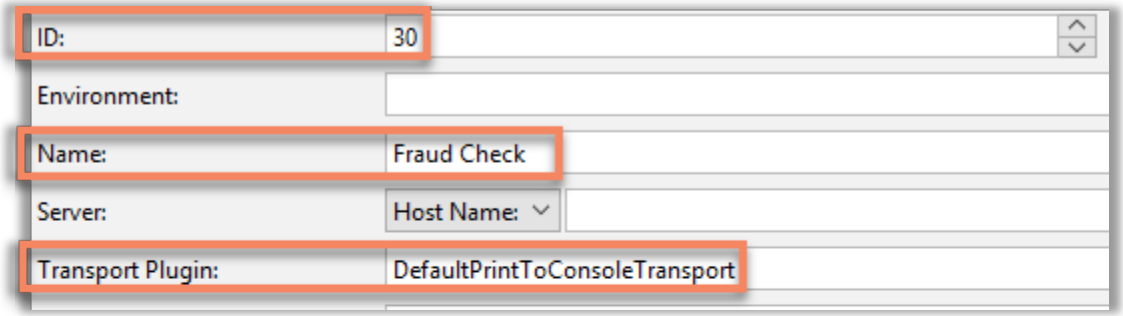
Four custom events also exist.

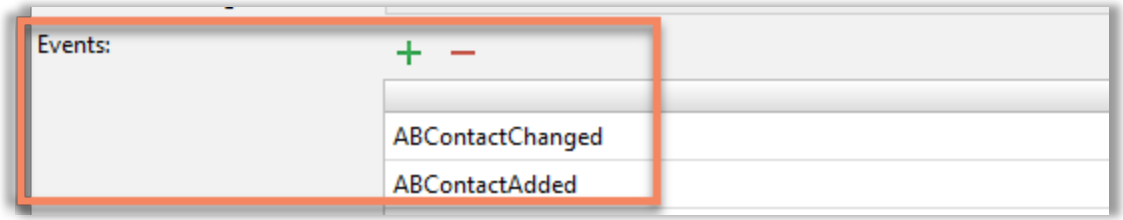
2. **Create a new messaging destination called Fraud Check.**
   - Open **messaging-config.xml** under configuration.config.Messaging.
   - Click the **PLUS** symbol to create a new destination.
   - Populate the required fields:
     - ID = 30
     - Name = Fraud Check
     - Transport Plugin = DefaultPrintToConsoleTransport
     - Events
       - ABContactChanged
       - ABContactAdded

Note

Do not change default connectivity and error parameters.







3. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
4. **Perform verification steps**

Lesson 13

Creating Messages

This lab requires that you use TrainingApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Super User whose login/password is su/gw.

13.1 Configure Event Fired rules

Succeed Insurance must determine if a given contact has been involved with a previous act of insurance fraud. To implement their fraud prevention system, they must send a message to an external system for every new contact or for an existing contact whose tax ID is updated. The payload must be in XML format and must contain the contact full name, tax ID, and a reference value generated by Guidewire to help identify the message. The external system must respond immediately to the fraud investigation request with a fraud report code.

External system information:

The WSDL URL to the external system is:

http://localhost:8890/ab/ws/externalapp/webservice/FraudReportAPI?WSDL

The authentication parameters are:

- Username: externalappuser
- Password: gw

Do not code the authentication parameters in the implementation code – use plugin parameters that are passed to the code.

API method checkForFraudReport requires the transformed payload as its argument.

The valid acknowledgment report codes are as follows:

- **1** – Request processed, no fraud report found
  - Acknowledge the message.
- **2** – Request processed, fraud report found!
  - Acknowledge the message.
- **4** – Request could not be processed (Payload Format Error)
  - Acknowledge the message with error using error category **Payload Format**.
- **5** – Request could not be processed (Database Unavailable)
  - Acknowledge the message with error using error category **Database Contention**.
- **Default** – Request could not be processed (Acknowledgment Code Invalid). If the error code returned from the external system is not valid, then acknowledge the message with error using a new error category called **Acknowledgement Code Invalid**.

For training purposes, output an acknowledgment message to console using the print statement.

13.1.1 Requirements

Spec 1 Trigger a message for new contacts of type ABContact.

Spec 2 Trigger a message for existing contacts that have modified their tax ID.

Spec 3 The XML message payload must contain the following two fields:

- Display name
- Tax ID

Tips

1. Use the Integration View and filter created in the Integration View lab.
2. Use **Bank Account Verification** Event Fired rules as an example for structure best practices.

13.1.2 Tasks

1. **Create the Event Fired rules for Fraud Check.**
2. **Deploy code changes.**
3. **Perform verification steps.**

13.1.3 Verification steps

1. **Launch TrainingApp.**
2. **Create a new contact named John Abercrombie and verify message is created.**
   - From the **Actions** menu, select **New Person → Person**.
   - Complete required fields
   - Edit the **Tax ID** field. Enter **123-45-6789**.
   - Click **Update** button.
   - In Studio, verify the message was created by checking the console output.

```
Payload for message 401: <?xml version="1.0"?>
<ContactDetails xmlns="http://guidewire.com/xsd/si.ta.contact-1.0">
  <Name>John Abercrombie</Name>
  <TaxID>123-45-6789</TaxID>
</ContactDetails>
```

Tip

To further test the same contact, you need to acknowledge any outstanding messages for the contact. This is because the application enforces a concept known as **Safe Ordering** – which means that a new message will not be sent for a specific contact until all previous outstanding messages for the specific contact have been acknowledge. This concept will be covered in detail in the Sending Messages lesson.

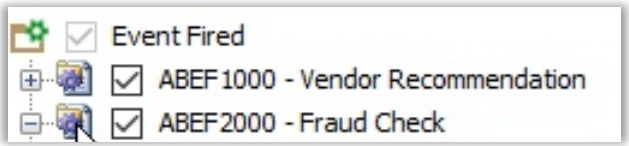
3. **Acknowledge message in Message Table screen.**
   - Navigate to **Administration → Training: Messaging.**
   - Select the new message with **Pending acknowledged** status.
   - Click the **Skip Selected Message(s)** button.
   - Click **OK** to the popup window.
4. **Edit the tax ID of the new contact.**
   - Search for **John Abercrombie** contact.
   - Navigate to the **Details** screen.
   - Edit Tax ID = 222-33-4444.
   - Click the **Update** button.
   - In Studio, verify the message was created in the console output.

```
Payload for message 2402:
<ContactDetails xmlns="http://guidewire.com/xsd/si.ta.contact-1.0">
  <Name>John Abercrombie</Name>
  <TaxID>222-33-4444</TaxID>
</ContactDetails>
```

STOP

13.1.4 Solution

1. **Create the Event Fired rules for Fraud Check.**
   - Navigate to Event Fired rule set.
   - Create the following rules based on best practices structure.
     - Right-click on Event Fired rule set and select **New Rule**.
     - Enter **ABEF2000 – Fraud Check** for the rule name.
     - Move the new rule under ABEF1000 – Vendor Recommendation
     - Rule condition should check for the destination.



```
1   USES:
2
3   CONDITION (messageContext  :  entity.MessageContext):
7   return messageContext.DestID == 30
8   ACTION (messageContext  :  entity.MessageContext, actions : gw.rules.Action):
13  // execute child rules
14  END
```

- Create **ABEF2100 – ABContact** child rule.
- Rule condition should check for entity type.


```
1   USES:
2
3   CONDITION (messageContext  :  entity.MessageContext):
7   return messageContext.Root typeis ABContact
8   ACTION (messageContext  :  entity.MessageContext, actions : gw.rules.Action):
13  // execute child rules
14  END
```

- Create **ABEF2110 – Added or Changed** child rule.
- Rule condition should check for event type.


```
1   USES:
2
3   CONDITION (messageContext    :    entity.MessageContext):
7   return messageContext.EventName == "ABContactAdded" or
8          messageContext.EventName == "ABContactChanged"
9   ACTION (messageContext    :    entity.MessageContext, actions : gw.rules.Action):
14  // execute child rules
15  END
```

- Create **ABEF2111 – Field Changed** child rule.
- Rule condition should check if tax ID field has changed or if new contact.
- Rule condition should create a message whose payload is in XML format.


```
1   USES:
2
3   uses gw.api.json.JsonConfigAccess
4   uses gw.api.json.mapping.JsonMappingOptions
5
6   CONDITION (messageContext : entity.MessageContext):
10  return (messageContext.Root as ABContact).isFieldChanged(ABContact#TaxID) or
11      (messageContext.Root as ABContact).New
12  ACTION (messageContext : entity.MessageContext, actions : gw.rules.Action):
17      var aBContact = messageContext.Root as ABContact
18      var jsonMapper = JsonConfigAccess.getMapper("si.ta.contact-1.0", "ContactDetails")
19      // Create JsonMapperOption object
20      var mappingOpts = new JsonMappingOptions().withFilter("si.ta.fraud_check-1.0")
21      // Create TransformResult object
22      var transformResult = jsonMapper.transformObject(aBContact, mappingOpts)
23      // Create message
24      var payload = transformResult.toXmlString()
25      messageContext.createMessage(payload)
26  END
```

```
var aBContact = messageContext.Root as ABContact
    var jsonMapper = JsonConfigAccess.getMapper("si.ta.contact-1.0", "ContactDetails")
    // Create JsonMapperOption object
    var mappingOpts = new JsonMappingOptions().withFilter("si.ta.fraud_check-1.0")
    // Create TransformResult object
    var transformResult = jsonMapper.transformObject(aBContact, mappingOpts)
    // Create message
    var payload = transformResult.toXmlString()
    messageContext.createMessage(payload)
```

2. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
3. **Perform verification steps.**

Lesson 14

Message Payload Transformation

This lab requires that you use TrainingApp, Guidewire Studio, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Super User whose login/password is su/gw.

14.1 Configure message payload transformation

Succeed Insurance must determine if a given contact has been involved with a previous act of insurance fraud. To implement their fraud prevention system, they must send a message to an external system for every new contact or for an existing contact whose tax ID is updated. The payload must be in XML format and must contain the contact full name, tax ID, and a reference value generated by Guidewire to help identify the message. The external system must respond immediately to the fraud investigation request with a fraud report code.

External system information:

The WSDL URL to the external system is:

http://localhost:8890/ab/ws/externalapp/webservice/FraudReportAPI?WSDL

The authentication parameters are:

- Username: externalappuser
- Password: gw

Do not code the authentication parameters in the implementation code – use plugin parameters that are passed to the code.

API method checkForFraudReport requires the transformed payload as its argument.

The valid acknowledgment report codes are as follows:

- **1** – Request processed, no fraud report found
  - Acknowledge the message.
- **2** – Request processed, fraud report found!
  - Acknowledge the message.
- **4** – Request could not be processed (Payload Format Error)
  - Acknowledge the message with error using error category **Payload Format**.
- **5** – Request could not be processed (Database Unavailable)
  - Acknowledge the message with error using error category **Database Contention**.
- **Default** – Request could not be processed (Acknowledgment Code Invalid). If the error code returned from the external system is not valid, then acknowledge the message with error using a new error category called **Acknowledgement Code Invalid**.

For training purposes, output an acknowledgment message to console using the print statement.

14.1.1 Requirements

Spec 1 Transform the payload to include the SenderRefID reference value.

Tips

Use **BankAccountVerificationRequest** class as an example for transforming an XML payload.

14.1.2 Tasks

1. **Create message request plugin class.**
2. **Create and configure request plugin registry.**
3. **Configure the destination.**
4. **Deploy code changes.**
5. **Perform verification steps.**

14.1.3 Verification steps

1. **Launch TrainingApp.**

Tip

To further test the same contact, you need to acknowledge any outstanding messages for the contact. This is because the application enforces a concept known as **Safe Ordering** – which means that a new message will not be sent for a specific contact until all previous outstanding messages for the specific contact have been acknowledge. This concept will be covered in detail in the Sending Messages lesson.

2. **Acknowledge message(s) in Message Table screen.**
   - Navigate to **Administration → Training: Messaging.**
   - Select message(s) with **Pending acknowledged** status.
   - Click the **Skip Selected Message(s)** button.
   - Click **OK** to the popup window.
3. **Edit the tax ID of the contact named John Abercrombie.**
   - Search for **John Abercrombie**.
   - Navigate to the **Details** screen.
   - Edit the **Tax ID** field. Enter **555-55-8888**.
   - Click **Update** button.

- In Studio, verify the message and was created and transformed in the console output. The transformed payload should include the SenderRefID.

```
Payload for message 2403:
<ContactDetails xmlns="http://guidewire.com/xsd/si.ta.contact-1.0">
  <Name>John Abercrombie</Name>
  <TaxID>555-55-8888</TaxID>
  <SenderRefID>ab:2403</SenderRefID>
</ContactDetails>
```

STOP

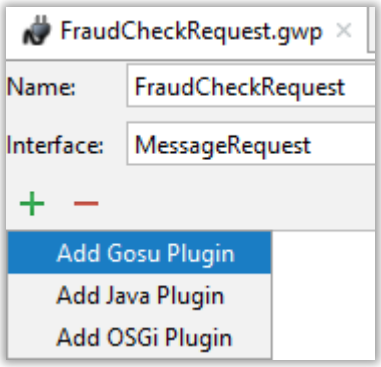
14.1.4 Solution

1. **Create message request plugin class.**
   - Right-click on **si.ta** package and select **New → Package**.
   - Enter **messaging.fraudcheck** as the new package name.
   - Right-click on **fraudcheck** package and select **New → Gosu Class**.
   - Enter **FraudCheckRequest** as the new class name.
   - Implement the **MessageRequest** interface.
   - Configure the **beforeSend** method.

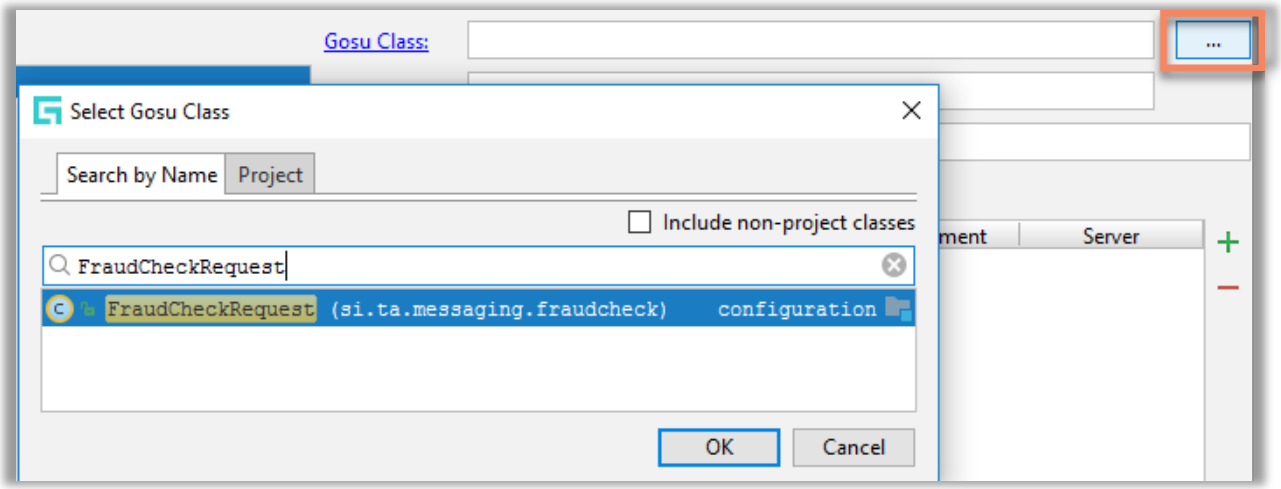
```
package si.ta.messaging.fraudcheck

uses gw.plugin.messaging.MessageRequest
uses gw.xml.XmlElement
uses gw.xml.XmlSerializationOptions

uses javax.xml.namespace.QName

class FraudCheckRequest implements MessageRequest
{

  override function beforeSend(message : Message) : String {
    // Set the SenderRefID
    if (message.SenderRefID == null) {
      message.SenderRefID = message.PublicID
    }
    // Add SenderRefID on the outbound request payload
    var transformedPayload = message.Payload
    var xml = XmlElement.parse(transformedPayload)
    var namespace = xml.$Namespace.NamespaceURI
    var senderRefID = new XmlElement(new QName(namespace,"SenderRefID"))
    senderRefID.set$Text(message.SenderRefID)
    xml.addChild(senderRefID)
    // Create transformedPayload with serialization options
    var opts = new XmlSerializationOptions()
    opts.XmlDeclaration = false
    opts.Sort = false
    opts.Validate = false
    transformedPayload = xml.asUTFString(opts)
    return transformedPayload
```

```
  }

  override function afterSend(message : Message) {

  }

  override function shutdown() {

  }

  override function suspend() {

  }

  override function resume() {

  }

  override property set DestinationID(destinationID : int) {

  }

}
```

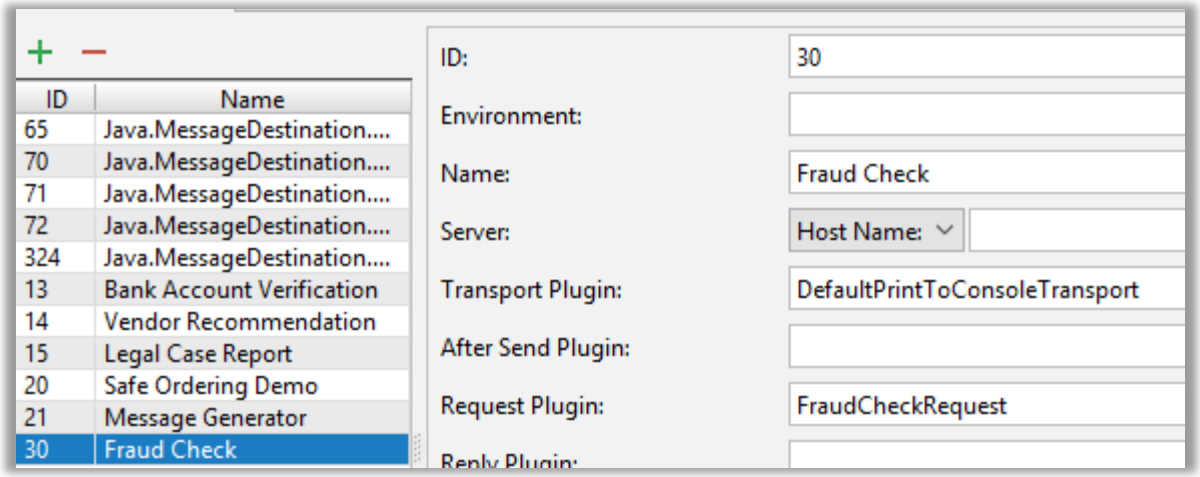
2. **Create and configure request plugin registry.**
   - Navigate to **configuration.config.Plugins.registry** package.
   - Right-click on **registry** and select **New → Plugin**.
   - Enter **FraudCheckReqeust** for the plugin name.
   - Enter **MessageRequest** for the Interface.
   - Configure the plugin.
     - Add a Gosu plugin.



     - Configure the Gosu Class field.



3. **Configure the destination.**
   - Open **messaging-config.xml** file.
   - Select **FraudCheck** destination.
   - Add **FraudCheckRequest** in Request Plugin field.



4. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
5. **Perform verification steps.**

Lesson 15

Sending Messages

This lab requires that you use TrainingApp, Guidewire Studio, ExternalApp, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

Start ExternalApp using the **Start ExternalApp** shortcut.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Super User whose login/password is su/gw.

15.1 Configure sending a message

Succeed Insurance must determine if a given contact has been involved with a previous act of insurance fraud. To implement their fraud prevention system, they must send a message to an external system for every new contact or for an existing contact whose tax ID is updated. The payload must be in XML format and must contain the contact full name, tax ID, and a reference value generated by Guidewire to help identify the message. The external system must respond immediately to the fraud investigation request with a fraud report code.

External system information:

The WSDL URL to the external system is:

http://localhost:8890/ab/ws/externalapp/webservice/FraudReportAPI?WSDL

The authentication parameters are:

- Username: externalappuser
- Password: gw

Do not code the authentication parameters in the implementation code – use plugin parameters that are passed to the code.

API method checkForFraudReport requires the transformed payload as its argument.

The valid acknowledgment report codes are as follows:

- **1** – Request processed, no fraud report found
  - Acknowledge the message.
- **2** – Request processed, fraud report found!
  - Acknowledge the message.
- **4** – Request could not be processed (Payload Format Error)
  - Acknowledge the message with error using error category **Payload Format**.
- **5** – Request could not be processed (Database Unavailable)
  - Acknowledge the message with error using error category **Database Contention**.

- **Default** – Request could not be processed (Acknowledgment Code Invalid). If the error code returned from the external system is not valid, then acknowledge the message with error using a new error category called **Acknowledgement Code Invalid**.

For training purposes, output an acknowledgment message to console using the print statement.

15.1.1 Requirements

Spec 1 Configure the system to send the payload to the external system.

Spec 2 API method checkForFraudReport requires the transformed payload as its argument.

Spec 3 Do not code the authentication parameters in the implementation code – use plugin parameters that are passed to the code.

15.1.2 Tasks

1. **Create a web service collection.**

Tip

Review the **SOAP Web Services** lesson for instructions on how to create a web service collection.

2. **Create message transport plugin class that implements InitializablePlugin interface.**

Tip

Review the **Plugins** lesson for instructions on how to implement the InitializablePlugin interface.

3. **Create and configure transport plugin registry.**
4. **Configure the destination.**
5. **Deploy code changes.**
6. **Perform verification steps.**

15.1.3 Verification steps

1. **Launch TrainingApp.**
2. **Clear pending messages in Message Table screen.**
   - Navigate to **Administration → Training: Messaging → Message Table.**
   - Select the new message with **Pending acknowledged** status.
   - Click the **Skip Selected Message(s)** button.
   - Click **OK** to the popup window.
3. **Clear messages in MessageHistory Table screen.**

- Navigate to **Administration → Training: Messaging → MessageHistory Table.**
- Select all messages.
- Click **Delete Selected Message Histories** button.
- Click **OK** to the popup window.

4. Edit the Tax ID of the contact named John Abercrombie.

- Search for **John Abercrombie**.
- Navigate to the **Details** screen.
- Edit the **Tax ID** field. Enter **766-55-2323**
- In ExternalApp console verify the message received. The payload should include the SenderRefID. The returning value will vary based on the Tax ID number entered.

```
c6273923961 externalappuser f00da1ed-916a-41ef-9ea2-6805dda8510b 2018-10-22 15:51:39,324 INFO
 FraudReportAPI.checkForFraudReport():  Received request for fraud report. Payload:

<ContactDetails xmlns="http://guidewire.com/xsd/si.ta.contact-1.0">
  <Name>John Abercrombie</Name>
  <TaxID>766-55-2323</TaxID>
  <SenderRefID>ab:601</SenderRefID>
</ContactDetails>
c6273923961 externalappuser f00da1ed-916a-41ef-9ea2-6805dda8510b 2018-10-22 15:51:39,335 INFO
 FraudReportAPI.checkForFraudReport():  Fraud request processed.
 Returning value: 0
```

15.1.4 Solution

1. Create a web service collection.

- Right-click on **si.ta.messaging.fraudcheck** package and select **New → Web Service Collection**.
- Enter **fraudcheckwsc** as the new WS-I web service collection name.
- Click the **PLUS** symbol to add a new resource.
- Enter the resource URL.
  - http://localhost:8890/ab/ws/externalapp/webservice/FraudReportAPI?WSDL
- Click the **Fetch** icon to retrieve external resources. Make sure ExternalApp is running.

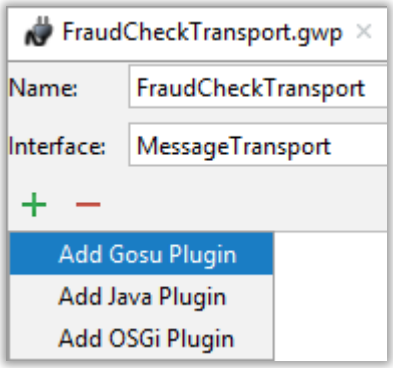
2. Create message transport plugin class that implements InitializablePlugin interface.

- Right-click on **si.ta.messaging.fraudcheck** package and select **New → Gosu Class**.
- Enter **FraudCheckTransport** as the new class name.
- Implement the **MessageTransport and InitializablePlugin** interfaces.
- Configure **set Parameters** property.
- Configure the **send** method.

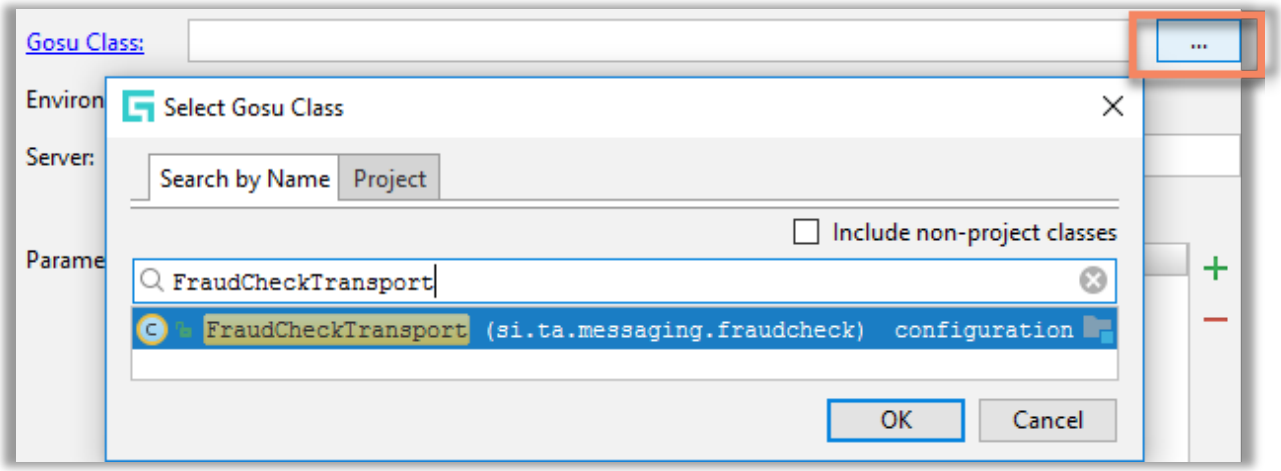
```
package si.ta.messaging.fraudcheck

uses gw.plugin.InitializablePlugin
uses gw.plugin.messaging.MessageTransport
uses si.ta.messaging.fraudcheck.fraudcheckwsc.fraudreportapi.FraudReportAPI

class FraudCheckTransport implements MessageTransport, InitializablePlugin{
  // Plugin parameters
```

```
  private static final var USERNAME = "Username"
  private static final var PASSWORD = "Password"
  // API parameters
  private static var _username : String
  private static var _password : String

  override function send(message : Message, transformedPayload : String) {
    // Create external web service object and set API properties
    var fraudreportAPI = new FraudReportAPI()
    fraudreportAPI.Config.Http.Authentication.Basic.Username = _username
    fraudreportAPI.Config.Http.Authentication.Basic.Password = _password
    var ackCode = fraudreportAPI.checkForFraudReport(transformedPayload)
  }

  override function shutdown() {

  }

  override function suspend() {

  }

  override function resume() {

  }

  override property set DestinationID(destinationID : int) {

  }

  override property set Parameters(map : Map<Object, Object>) {
    _username = map.get(USERNAME) as String
    _password = map.get(PASSWORD) as String
  }

}
```

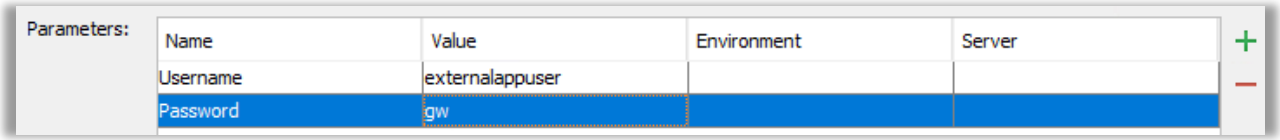
3. **Create and configure transport plugin registry.**
   - Navigate to **configuration.config.Plugins.registry** package.
   - Right-click on **registry** and select **New → Plugin**.
   - Enter **FraudCheckTransport** for the plugin name.
   - Enter **MessageTransport** for the Interface.
   - Configure the plugin.
     - Add a Gosu plugin.



- Configure the Gosu Class field.

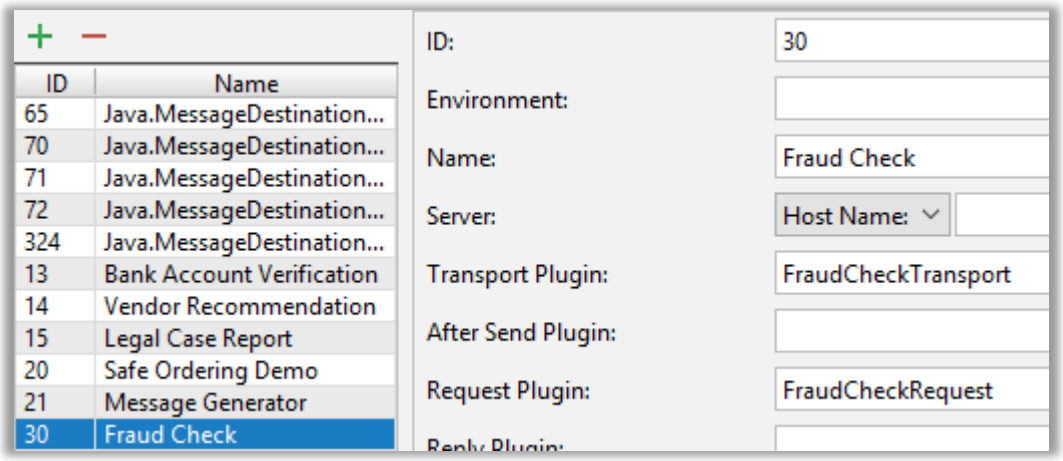


- Add plugin parameters.



4. Configure the destination.

- Open **messaging-config.xml** file.
- Select **FraudCheck** destination.
- Add **FraudCheckTransport** in Transport Plugin field.



5. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
6. **Perform verification steps.**

Lesson 16

Acknowledging Messages

This lab requires that you use TrainingApp, Guidewire Studio, ExternalApp, and a supported web browser. Start Guidewire Studio for TrainingApp. Start the server as Debug 'Server'.

Start ExternalApp using the **Start ExternalApp** shortcut.

The default URL for TrainingApp is: http://localhost:8880/ab/ContactManager.do. Log in to TrainingApp as Super User whose login/password is su/gw.

16.1 Configure synchronous acknowledgment

Succeed Insurance must determine if a given contact has been involved with a previous act of insurance fraud. To implement their fraud prevention system, they must send a message to an external system for every new contact or for an existing contact whose tax ID is updated. The payload must be in XML format and must contain the contact full name, tax ID, and a reference value generated by Guidewire to help identify the message. The external system must respond immediately to the fraud investigation request with a fraud report code.

External system information:

The WSDL URL to the external system is:

http://localhost:8890/ab/ws/externalapp/webservice/FraudReportAPI?WSDL

The authentication parameters are:

- Username: externalappuser
- Password: gw

Do not code the authentication parameters in the implementation code – use plugin parameters that are passed to the code.

API method checkForFraudReport requires the transformed payload as its argument.

The valid acknowledgment report codes are as follows:

- **1** – Request processed, no fraud report found
  - Acknowledge the message.
- **2** – Request processed, fraud report found!
  - Acknowledge the message.
- **4** – Request could not be processed (Payload Format Error)
  - Acknowledge the message with error using error category **Payload Format**.
- **5** – Request could not be processed (Database Unavailable)
  - Acknowledge the message with error using error category **Database Contention**.

- **Default** – Request could not be processed (Acknowledgment Code Invalid). If the error code returned from the external system is not valid, then acknowledge the message with error using a new error category called **Acknowledgement Code Invalid**.

For training purposes, output an acknowledgment message to console using the print statement.

16.1.1 Requirements

Spec 1 Process external system synchronous response code based on the following criteria:

- **1** – Request processed, no fraud report found
  - Acknowledge the message.
- **2** – Request processed, fraud report found!
  - Acknowledge the message.
- **4** – Request could not be processed (Payload Format Error)
  - Acknowledge the message with error using error category **Payload Format**.
- **5** – Request could not be processed (Database Unavailable)
  - Acknowledge the message with error using error category **Database Contention**.
- **Default** – Request could not be processed (Acknowledgment Code Invalid). If the error code returned from the external system is not valid, then acknowledge the message with error using a new error category called **Acknowledgement Code Invalid**.

Spec 2 Add a new **ErrorCategory** typecode named **Acknowledgment Code Invalid**.

16.1.2 Tasks

1. **Configure the ErrorCategory typelist.**
2. **Configure the FraudCheckTransport class.**
3. **Deploy code changes.**
4. **Perform verification steps.**

16.1.3 Verification steps

The external system will take the first digit in the Tax ID field and return that number as the acknowledgment code. To properly test the code, enter Tax ID numbers that starts with 1, 2, 4, 5, and 8. This will completely test all the different acknowledgment scenarios.

1. **Launch TrainingApp.**
2. **Clear pending messages in Message Table screen.**
   - Navigate to **Administration → Training: Messaging → Message Table.**
   - Select all messages with **Pending acknowledged** status.
   - Click the **Skip Selected Message(s)** button.
   - Click **OK** to the popup window.
3. **Clear messages in MessageHistory Table screen.**
   - Navigate to **Administration → Training: Messaging → MessageHistory Table.**
   - Select all messages.
   - Click **Delete Selected Message Histories** button.

- Click **OK** to the popup window.

4. **Edit the Tax ID of the contact named John Abercrombie.**
   - Search for **John Abercrombie**.
   - Navigate to the **Details** screen.
   - Edit the **Tax ID** field.
     - Enter 111-11-1111
     - In TrainingApp Studio console verify the output is correct.

```
Request processed, no fraud report found.
```

     - Verify the message was acknowledged in the MessageHistory screen.

5. **Edit the Tax ID of the contact named John Abercrombie.**
   - Search for **John Abercrombie**.
   - Navigate to the **Details** screen.
   - Edit the **Tax ID** field.
     - Enter 222-22-2222
     - In TrainingApp Studio console verify the output is correct.

```
Request processed, fraud report found!
```

     - Verify the message was acknowledged in the MessageHistory screen.

6. **Edit the Tax ID of the contact named John Abercrombie.**
   - Search for **John Abercrombie**.
   - Navigate to the **Details** screen.
   - Edit the **Tax ID** field.
     - Enter 444-44-4444
     - In TrainingApp Studio console verify the output is correct.

```
Request could not be processed (Payload Format Error)
```

     - Verify the message is in the **Message Table** screen with a status of **Retryable error** and Error Category of **Payload Format**.
     - Select the message and select **Skip Selected Message(s)** button.

7. **Edit the Tax ID of the contact named John Abercrombie.**
   - Search for **John Abercrombie**.
   - Navigate to the **Details** screen.
   - Edit the **Tax ID** field.
     - Enter 555-55-5555
     - In TrainingApp Studio console verify the output is correct.

```
Request could not be processed (Database Unavailable)
```

     - Verify the message is in the **Message Table** screen with a status of **Retryable error** and Error Category of **Database Contention**.
     - Select the message and select **Skip Selected Message(s)** button.

8. **Edit the Tax ID of the contact named John Abercrombie.**
   - Search for **John Abercrombie**.
   - Navigate to the **Details** screen.
   - Edit the **Tax ID** field.
     - Enter 888-88-8888
     - In TrainingApp Studio console verify the output is correct.

```
Request could not be processed (Acknowledgment Code Invalid)
```

     - Verify the message is in the **Message Table** screen with a status of **Retryable error** and Error Category of **Acknowledgment Code Invalid**.
     - Select the message and select **Skip Selected Message(s)** button.

9. **Verify MessageHistory table.**
   - The MessageHistory table should have two acknowledged messages and three errored messages.

Msg ID	Creation Time	Triggering Entity	Event Name	Destination	SenderRefID	Error Category	Status
2701	09/14/2018 5:58 PM	ABPerson(504)	ABContactChanged	30	ab:3001		Acknowledged (10)
2702	09/14/2018 5:58 PM	ABPerson(504)	ABContactChanged	30	ab:3002		Acknowledged (10)
2703	09/14/2018 5:59 PM	ABPerson(504)	ABContactChanged	30	ab:3003	Payload Format	Error cleared (11)
2704	09/14/2018 5:59 PM	ABPerson(504)	ABContactChanged	30	ab:3004	Database Contention	Error cleared (11)
2705	09/14/2018 6:00 PM	ABPerson(504)	ABContactChanged	30	ab:3005	Acknowledgment Code Invalid	Error cleared (11)

STOP

16.1.4 Solution

1. **Configure the ErrorCategory typelist.**
   - Open **ErrorCategory** typelist.
   - Add new typecode.

ErrorCategory.ttx

typecode | Show All | ErrorCategory.tti

Element	Code	Name	Priority
typelistextension	ErrorCategory	The type of error ...	
typecode	no_connection	No Connection	1
typecode	database_content...	Database Content...	2
typecode	user_authenticati...	User Authenticati...	3
typecode	unexpected_error	Unexpected Error	4
typecode	permanent_error	Permanent Error	5
typecode	payload_format	Payload Format	6
typecode	ack_code_invalid	Acknowledgment...	-1

Name	Value
code	ack_code_invalid
name	Acknowledgment Code Invalid
desc	Acknowledgment Code Invalid
identifierCode	
priority	-1
retired	false

2. **Configure the FraudCheckTransport class.**

```
package si.ta.messaging.fraudcheck

uses gw.plugin.InitializablePlugin
uses gw.plugin.messaging.MessageTransport
uses si.ta.messaging.fraudcheck.fraudcheckwsc.fraudreportapi.FraudReportAPI

class FraudCheckTransport implements MessageTransport, InitializablePlugin{
  // Plugin parameters
  private static final var USERNAME = "Username"
  private static final var PASSWORD = "Password"
  // API parameters
  private static var _username : String
  private static var _password : String

  override function send(message : Message, transformedPayload : String) {
    // Create external web service object and set API properties
    var fraudreportAPI = new FraudReportAPI()
    fraudreportAPI.Config.Http.Authentication.Basic.Username = _username
    fraudreportAPI.Config.Http.Authentication.Basic.Password = _password
    var ackCode = fraudreportAPI.checkForFraudReport(transformedPayload)

    // Process acknowledgment code
    switch (ackCode) {
      case 1:
        message.reportAck()
        print("Request processed, no fraud report found.")
        break
      case 2:
        message.reportAck()
        print("Request processed, fraud report found!")
        break
      case 4:
        message.reportError(ErrorCategory.TC_PAYLOAD_FORMAT)
        print("Request could not be processed (Payload Format Error)")
        break
      case 5:
        message.reportError(ErrorCategory.TC_DATABASE_CONTENTION)
        print("Request could not be processed (Database Unavailable)")
        break
      default:
        message.reportError(ErrorCategory.TC_ACK_CODE_INVALID)
        print("Request could not be processed (Acknowledgment Code Invalid)")
        break
    }
  }

  override function shutdown() {

  }

  override function suspend() {

  }

  override function resume() {

  }

  override property set DestinationID(destinationID : int) {
```

```
  }

  override property set Parameters(map : Map<Object, Object>) {
    _username = map.get(USERNAME) as String
    _password = map.get(PASSWORD) as String
  }

}
```

3. **Deploy code changes.**
   - From the Studio menu, **Restart** the server.
4. **Perform verification steps.**