Developer Guide | PUBLIC
2019-09-20

Risk-based Development Guide

SAP Quotation and Underwriting for Insurance 1.1

THE BEST RUN SAP

Content

1 Introduction. . . . 4
1.1 About This Document. . . . 4
1.2 Audience. . . . 4
1.3 Source Code Terms and Conditions. . . . 4
1.4 Supported Clients. . . . 4
1.5 Substitution Variable Placeholders. . . . 5

2 Application Development in FS-QUO. . . . 6
2.1 Terminology. . . . 6
2.2 Customer Views. . . . 8
2.3 Overview of an FS-QUO Runtime Deployment 10
2.4 References. . . . 13
2.5 Development Components. . . . 13
2.6 Understanding How the Components Work Together. . . . 14
2.7 What is the Difference Between Coverage-Based and Risk-Based Product Architectures?. . . . 15

3 Understanding Extensibility. . . . 16
3.1 Understanding Design Time. . . . 16
3.2 Customization Site Map. . . . 17
3.3 Extensibility Scenarios by Area. . . . 18
Example: Modifying the General Information Screen in the FS-QUO Fiori App. . . . 19
Example: Creating a Custom eApp Renderer. . . . 20
3.4 Adding a Custom Project. . . . 21
3.5 Unsupported Extension Types. . . . 22

4 Core FS-QUO. . . . 23
4.1 Business Services. . . . 23
Accessing the javadoc. . . . 23
Business Service Components and Methods. . . . 24
Understanding the Core Service Components. . . . 24
Preserving Extended Services After Upgrading. . . . 25
4.2 Underwriting Application Configuration. . . . 25
Locating the Standalone Underwriting Application Configuration. . . . 25
Extending the Underwriting Application Configuration Standalone Object. . . . 26
Modifying Application Configuration Settings. . . . 27
Folders in the Underwriting Application Configuration Object. . . . 28

5 Risk-Based Architecture. . . . 37

5.1 Extending Fiori Apps UI. 37
5.2 Working with Roles and the Permissions Framework. 38
5.3 Understanding Visitor Patterns. 38
Model Visitor. 39
SubmissionVisitor Class. 39
Policy Traversal. 40
Visitor Patterns. 41
5.4 Domain Layers. 43
Domain Layer and Database Tables. 43
Modifying the Domain Layer. 43
Policy Database Tables. 46
Supporting Database Tables. 78
5.5 Product and Reference Services. 82
User-Defined Services. 83
Location of the Product Services. 83
Location of the Reference Services. 84
Service API for the Commercial P and C Product Template. 84
Service API for the `Commercial P and C Reference Object Template`. 88
5.6 Using Stems. 91
Stem Classes and Functions. 92
Accessing the javadoc. 93
Extending Stem Implementation Classes and Stem Classes. 94
Testing the Extended Stem Classes in FS-QUO. 95
Uploading New Stem Classes to FS-PRO. 96
Registering Stems in the FS-PRO Function Library. 97
5.7 Selecting a Ratebook. 98
Configuring Ratebook Selection. 100
Example of Ratebook Selection Configuration. 105
5.8 Additional Configuration Procedures. 107
Configuring Forms Attachment. 107
Configuring Tax Rules. 107
Configuring Validation for Flat Cancellation Type. 108
Referrals Configuration Note. 109

6 Unsupported Features. 110

1 Introduction

1.1 About This Document

This document describes the following information:

- The SAP Quotation and Underwriting for Insurance (FS-QUO) architecture and the objects used to create the application
- How to extend the default FS-QUO behavior
- How to modify the FS-QUO configuration (for example, the application configuration and flows)

1.2 Audience

This document is intended for technical consultants and developers who modify FS-QUO functionality. This guide is also intended for business analysts and administrators who configure FS-QUO.

The prerequisites for using this guide are the following:

- You are familiar with FS-QUO.
- You have experience developing applications using SAP UI5 and Java.

1.3 Source Code Terms and Conditions

The FS-QUO source code described in this guide is the intellectual property (IP) of SAP SE. The source code is available for customers to support extension.

Any changes that you make to the core source code voids part of the warranty or the entire warranty.

For more information about IP terms and conditions, see the license agreement.

1.4 Supported Clients

The FS-QUO client is browser-based.

For information about the browsers that are supported, see Client System Requirements.

1.5 Substitution Variable Placeholders

The following table defines the substitution variables used in this guide. Gather the information listed prior to beginning.

Substitution Variable	Description
`<sp>`	The support package version you are installing.
`<pl>`	The patch level of you are installing.
`<rel>`	The FS-QUO release version you are installing.
`<CSI_HOME>`	The installation directory of the file system portion of an FS-PRO or FS-QUO installation.
`<quo_app_url>`	The URL to the FS-QUO approuter application.

2 Application Development in FS-QUO

2.1 Terminology

While this Developer Guide focuses on FS-QUO, any customer wishing to customize this solution should be familiar with how FS-QUO works as an add-on to the FS-PRO product. The following terms and labels are meant to help with this understanding:

- A Program is an overall software offering/solution.
- A Program is made up of software Components, which are separately released and installed.
- A Component contains Modules, which are logical areas of functionality within the Component. Modules can be either horizontal (frameworks) or vertical (features).

The following table describes the Components and Modules in FS-PRO:

Item		Type	Description
SAP Quotation and Underwriting for Insurance		Program	Product Model Definition
FS-PRO		Component	Product Model Definition
	Administrative Console	Module (Framework)	An application for managing the Design Time FS-PRO system.
	Product Modeler (FS-PRO)	Module (Framework)	An application for managing products.
	Product Web Services (PWS)	Module (Framework)	A framework for exposing product services as web services.
	IFBC Integration	Module (Feature)	Ability to export or import a product's IFBC definitions to or from FS-PM.

Item		Type	Description
	Product Architectures	Module (Framework)	[FS-PRO content] Represents a top-level architecture (mode) for the Program. Out of the box, two product architectures are supported: • Coverage-Based • Risk-Based These terms are explained in more detail in a later topic.
	Sample Products	Module (Feature)	[FS-PRO content] Out of the box examples of functioning products, for both product architectures.
	Product Stems	Module (Framework)	[Java] A set of specialized functions callable from product script rules.
	Product Runtime Framework	Module (Framework)	A platform for product-driven applications. Includes libraries and engines to operationalize products (for example, federated data model, product services, eApps).

The following table describes the Components and Modules in FS-QUO:

Item		Type	Description
SAP Quotation and Underwriting for Insurance		Program	Product, Quotation and Underwriting for Insurance
FS-QUO		Component	Quotation and Underwriting for Insurance
	Administrative Console	Module (Framework)	An application for managing the Runtime FS-QUO system.
	Application Framework	Module (Framework)	Includes utilities for OData, Business Services, and Data Model.
	Quotation Web Application	Module (Feature)	Back-end for Quotation Fiori applications. Includes implementation for OData, and Business Services

Item		Type	Description
	Underwriting Web Application	Module (Feature)	Back-end for Underwriting Fiori applications. Includes implementation for OData, and Business Services
	FS-PM Integration	Module (Feature)	Interfaces for searching and creating Business Partners and Insurable Objects, and issuing new insurance policies.
	Fiori Launchpad Configuration	Module (Feature)	Enables application tiles and related authorization.
	Quotation Applications	Module (Feature)	Front-end for Quotation Fiori applications, implemented in SAPUI5.
	Underwriting Applications	Module (Feature)	Front-end for Underwriting Fiori applications, implemented in UI5.
FS-IPW		Component	Workplaces for Insurance
	Fiori Launchpad Configuration	Module (Feature)	Enables application tiles and related authorization.
	Quotation Applications	Module (Feature)	Front-end for Quotation Fiori applications, implemented in SAPUI5.
	Underwriting Applications	Module (Feature)	Front-end for Underwriting Fiori applications, implemented in UI5.

Fiori refers to a set of usability guidelines for applications.

In the context of FS-QUO, a Fiori Application is any application that conforms to Fiori guidelines or is implemented using a standard architecture consisting of the following items:

- Fiori launchpad
- UI5
- OData services

2.2 Customer Views

From the customer's perspective, there are three views of FS-QUO software:

Installable View Refers to all FS-QUO-related artifacts available for the customer to download from SAP and install.

Design Time View	Refers to set of systems that exist while the customer is developing their insurance solution, such as configuring their products and customizing the application for their use: • FS-PRO Design Time (Administrative Console, Product Modeler) • FS-QUO Toolkit These systems are created from installable artifacts.
Runtime View	Refers to set of systems that exist while the customer's solution is deployed and testable (for example, in QA or Production): • FS-QUO Runtime These systems are created from installable artifacts.

These three views apply to all three components (FS-PRO, FS-QUO, and FS-IPW). The following table outlines the artifacts/concepts relevant to each view-component combination.

> **Note**
>
> The details in this table aren't meant to be exhaustive, but rather provide orientation for where certain concepts and technologies fall in the customer project lifecycle.

View	FS-PRO	FS-QUO	FS-IPW

Installable	Database (ZIP)
	`CSI_HOME` (ZIP)
	MTAR
	• FS-PRO `.ear`
	  • 3rd party libraries
	  • `camilionlib.jar`
	  • `camilionejb.jar`
	  • `camilionweb.war`
	  • `PSRepo.war`
	  • `PSRuntime.war`
	  • `ProductExplorer.war`
	  • `ProductBuilder.war`
	Install Options (Usage Types)
	• Product Modeler
	• Product Web Services (PPMS)
		Database (ZIP) - includes FS-PRO database, plus additional content for FS-QUO
		`CSI_HOME` (ZIP) - includes FS-PRO `CSI_HOME`, plus additional content for FS-QUO
		MTAR - includes FS-PRO MTAR contents, plus additional components for FS-QUO
		• FS-QUO JAR files
		• `quo-web.war`
		• `uw-web.war`
		Toolkit (ZIP)
			ABAP Package (add-on)
			• QUO Fiori App (XML, JS, BSP, and so on)
			• UW Fiori App (XML, JS, BSP, and so on)
			• Fiori launchpad (FLP) configuration (tiles, roles, and so on)

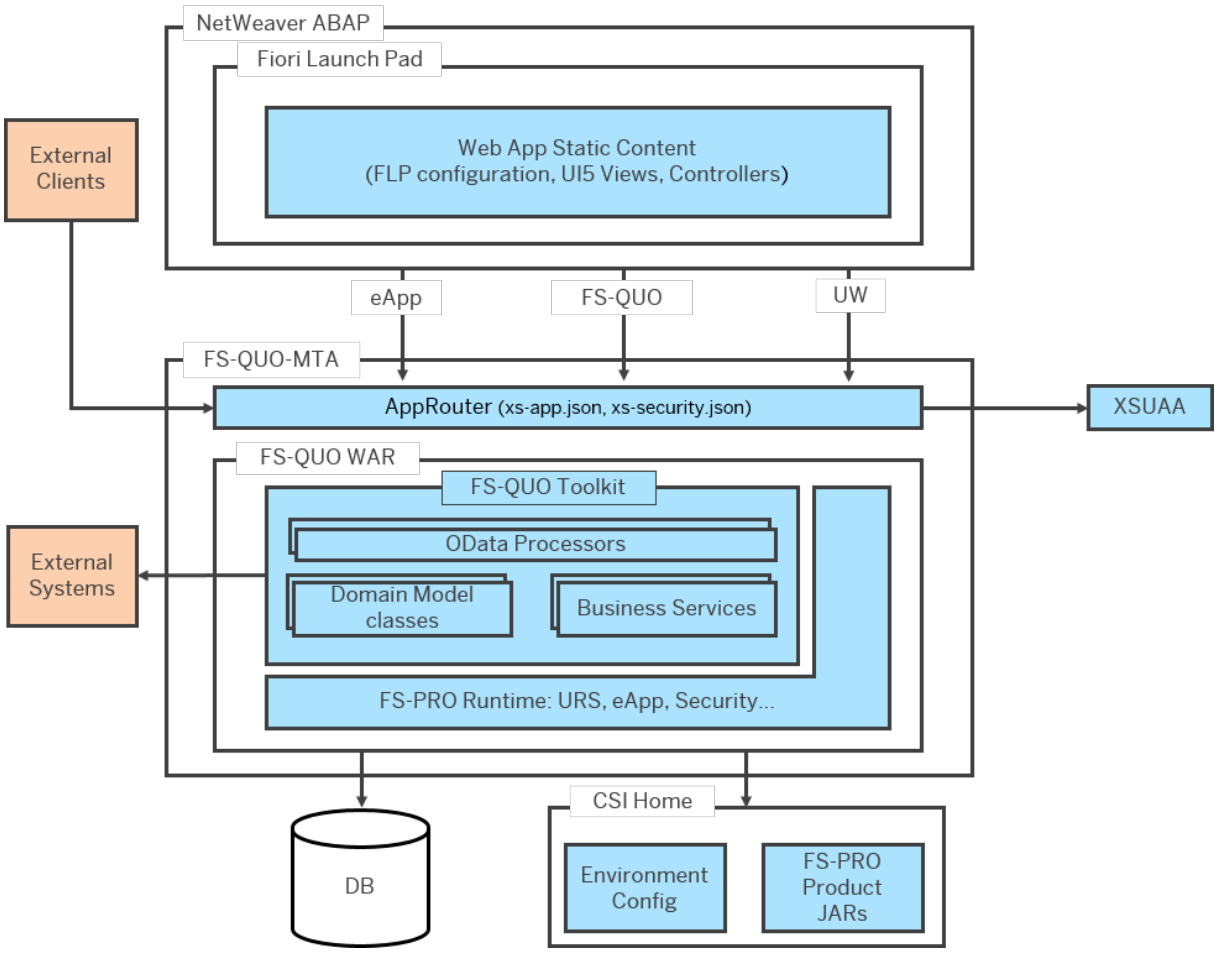
View	FS-PRO	FS-QUO	FS-IPW

Design Time	Database
	File System (`CSI_HOME`)
	Web Applications
	• `camilionweb.war`
	  • Administrative Console (for Operations)
	  • Product Modeler
		Database
		File System (`CSI_HOME`)
		Web Applications
		• `camilionweb.war`
		  • Administrative Console (for Development)
		Toolkit (for Eclipse)
		• Includes FS-IPW Toolkit, plus additional projects for FS-QUO
		• Uses TomEE+ for local testing
		• Contains projects with build scripts and source code
			Toolkit (for Eclipse)
			• Plugin to remotely manage FLP content
			• One project for each app:
			  • *My Insurance Worklist*
			  • *Create Insurance Quote*
			  • *My Underwriting Worklist*

Runtime	Database
	File System (`CSI_HOME`)
	Web Applications
	• `camilionweb.war`
	  • Administrative Console (for Operations)
	  • Product Web Services
		Database
		File System (`CSI_HOME`)
		Web Applications
		• `camilionweb.war`
		  • Administrative Console (for Operations)
		  • Product Web Services
		  • eApp service
		• `QUO.war`
		  • OData
		• `UW.war`
			Fiori Apps
			• QUO - *My Work List (Quotes)*
			• QUO - *New Business*
			• UW - *My Work List + Case Details (Cases)*

2.3 Overview of an FS-QUO Runtime Deployment

The following diagram depicts a Runtime deployment of FS-QUO. The subsequent sections describe each concept in more detail.



Fiori Applications

Fiori applications for Quotation and Underwriting are based on the Fiori application architecture, whereby the implementation is separated across front-end and back-end servers.

- Front-end server (SAP HANA XSA ABAP with Fiori launchpad; FS-QUO Fiori application content installed)
  - The front-end server provides Fiori launchpad functionality, as well as the browser-side content for each Fiori application.
  - The tile used to launch each application is configured in Fiori launchpad, along with any users, roles and permissions for the launchpad.
  - UI5 content (View XML and Controller JS) is uploaded to the front-end server, and associated to a tile.
  - At runtime, the front-end accesses the back-end through RESTful URLs (OData) specified in the Controller JS files.
- Back-end server (SAP HANA XSA TomEE; FS-QUO installed)
  - The back-end server provides OData CRUD (Create Retrieve Update Delete business object interface operations) services to support the front-end.
    These services represent the MVC Model for the Fiori application.
  - The OData services are implemented using the Apache Olingo framework.

    - The OData services are implemented as Processor classes, which call FS-QUO Business Services.
- AppRouter
    - Node.js server which handles HTTP request routing, URL authentication, and CSRF token management.

eApps

eApps work with Fiori applications:

- The eApp runtime engine has a clean separation between front-end and back-end.
    - The renderer is implemented in UI5 and is customizable.
    - The front-end makes requests to a back-end servlet using JSON (rather than OData).
    - The front-end eApp libraries convert the results to UI5 for display.
- The eApp is integrated into Fiori applications through standard UI5 hooks.

Refer to the Product Modeler User Guide for more information.

J2EE Deployment

FS-QUO deployment more closely follows J2EE standards.

- Java code isn't deployed in the `<CSI_HOME>` directory and loaded through custom classloader.
- Quotation and Underwriting modules each build as its own WAR.

Security

Authentication and user management are deferred to SAP HANA XS Advanced:

- User information is managed in XSUAA.
  In addition, for SSO between the ABAP Front-End Server and SAP HANA XS Advanced, and external SAML identity provider (IdP) is required.
- AppRouter performs a first-level authentication and authorization check.
  If an unauthenticated HTTP request is received, AppRouter forwards the request to XSUAA for authorization. Once a request has been authenticated, AppRouter performs an authorization check on the requested URL. Once the request is authorized, AppRouter generates a security header in JSON Web Token (JWT) format, and forwards the request to the back-end server.
- The back-end server only accepts requests with a valid JWT.
  In the back-end server, the JWT is processed by the UAASecurityFilter, which populates the FS-QUO ClientSessionState object (needed for internal processing) by calling the SAP HANA XS Advanced Security API to get User information from the JWT.

Any custom URLs added to Quotation or Underwriting must also have these filters applied.

Globalization

- A product in FS-PRO can contain value row data for a non-English language (only one language per product).
- Static UI5 content in Fiori apps supports multiple languages using the Language Pack capability of UI5. When a user logs in to the Fiori launchpad, they can specify a language other than English. If Language Pack content is available for that language, it is loaded by UI5.

2.4 References

Fiori documentation	SAP Fiori
Globalization	Refer to the Product Modeler User Guide.

2.5 Development Components

The FS-QUO application is developed using components that are exposed as SAP UI5, Java classes, Product Modeler objects, and XML files.

The following FS-QUO components are located in the Product Modeler:

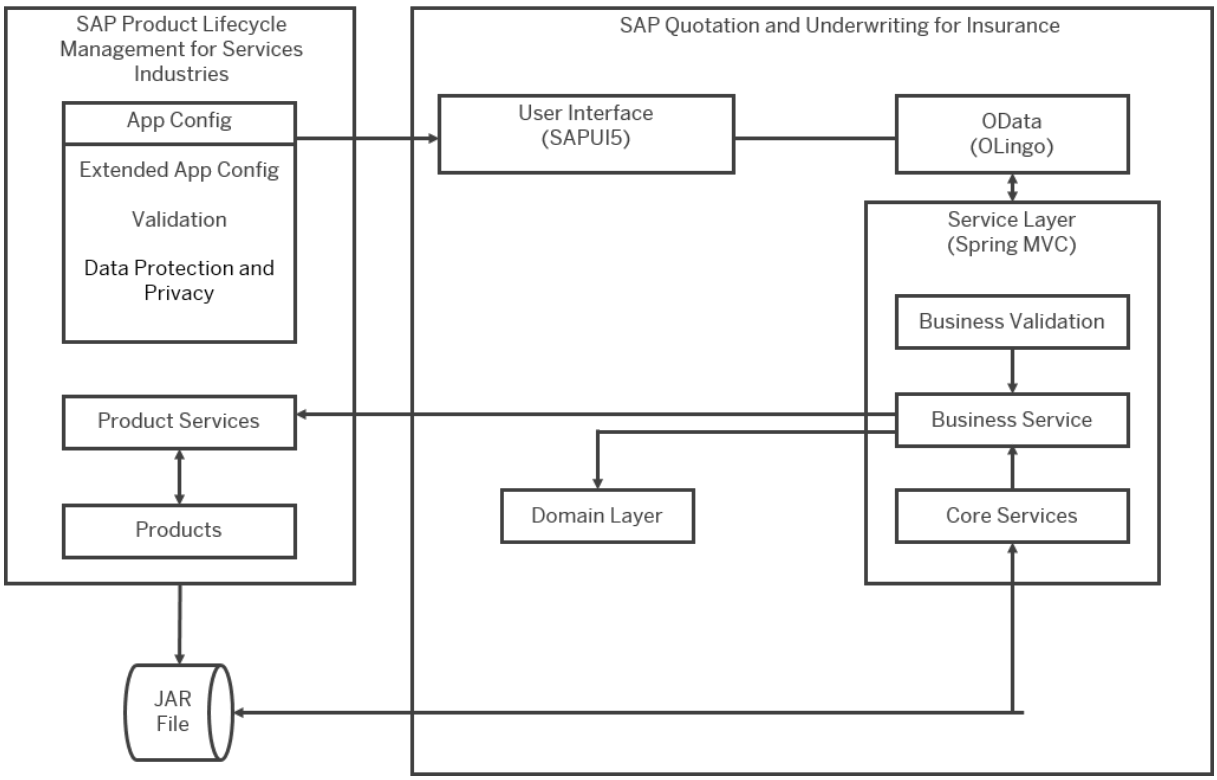
- Underwriting Application
- Permissions Framework
- Validation Framework

The following components require programming:

- User interface JavaScript classes
- Process configuration files
- Handlers and Handler Beans
- Business Services
- Product and Reference Services
- Visitor Patterns
- Policy Models

2.6 Understanding How the Components Work Together

The following diagram shows an overview of the components and technology used to develop and extend FS-QUO:



The components work together in the following way:

- The `Underwriting Application Configuration` object contains the information used to configure the FS-QUO application. This object defines default roles and permissions, search behavior, validation, and business questions.
- The extended `Underwriting Application Configuration` object (typically named Extended Application Configuration) lets you define configuration settings that are specific to your FS-QUO application. The extended object inherits from the Underwriting Application Configuration object.
- The FS-QUO user interface is developed using SAP UI5.
- When a user performs an action on the user interface (for example, click search), an OData call translates the data between the user interface and the business services.
- Business Services are methods that perform operations on policies (for example, create a new account or add a form). Business services aren't specific to products.
- Core services are the underlying layer for business services.
- Product Services are methods that perform operations on products (for example, return a list of coverages in a specific product). Business services can invoke Product Services to get product-specific data for a particular policy.
- The Domain Layer is a representation of the data for a specific policy transaction. The Domain Layer lets you access data without querying the FS-QUO database tables and views. Business services can get policy information from the Domain Layer.

2.7 What is the Difference Between Coverage-Based and Risk-Based Product Architectures?

You can use SAP Quotation and Underwriting for Insurance to create products based on either a coverage-based or a risk-based architecture.

The following table outlines the differences between the two standard product architectures

	Coverage-Based Model	**Risk-Based Model**
Focused Lines of Business	Personal/Simple P&C Life	Commercial/Complex P&C
FS-QUO Responsibilities	Quotation and Underwriting (NB)	Quotation and Underwriting (NB, CB) Policy Management
FS-PM Responsibilities	Policy Management	N/A
Standard FS-QUO Integrations	FS-PM ERP-BP	N/A

3 Understanding Extensibility

Extensibility generally refers to the ability to customize an SAP system in a supported way.

Note the following:

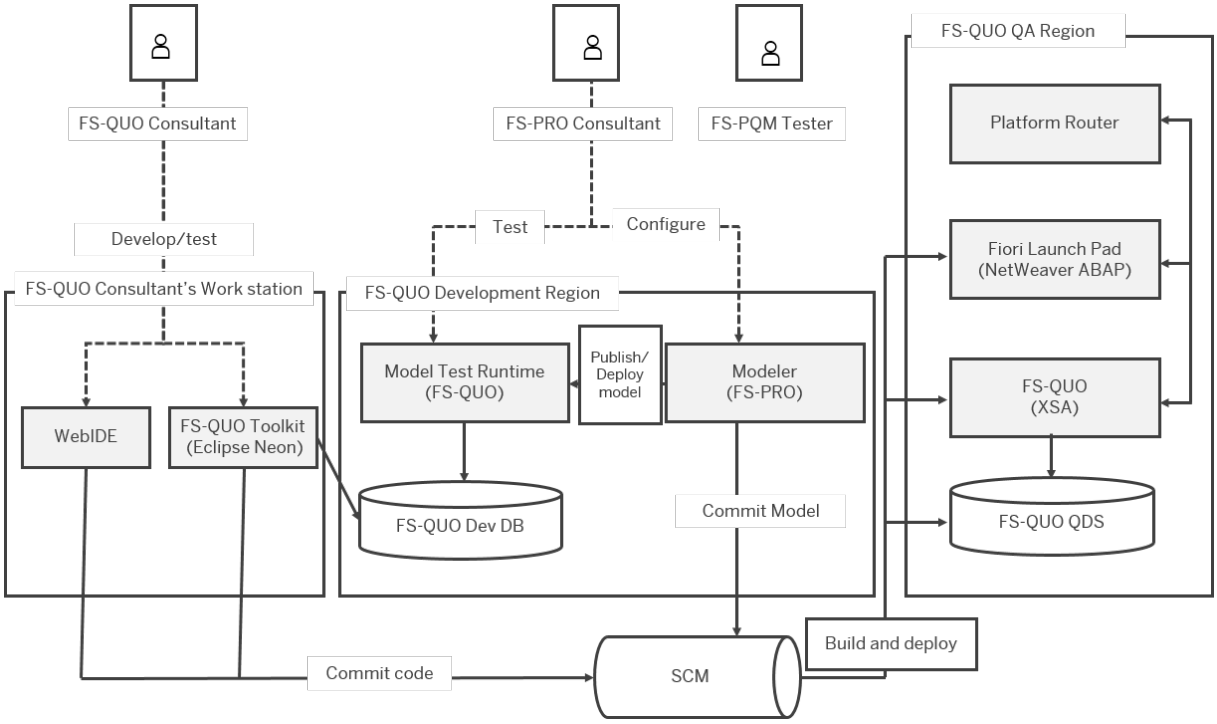
- Any configurability for a feature will be described as part of that feature's documentation (as opposed to being covered under Extensibility).
- The Toolkit includes all source code for the FS-QUO implementation, available for you to analyze or customize.
- During an upgrade of FS-QUO, you are responsible for reapplying any customizations that you have implemented. Depending on the invasiveness of the customization, this may involve re-work for the customization.
- The recommended best practice is to minimize customizations in the FS-QUO code itself, and to keep the majority of custom logic separate (such as in a separate Toolkit project) in order to reduce the effort of reapplying customizations during upgrade.

The following topics in this section describe each part in more detail.

It should be noted that, as with any SAP system, customers should seek guidance from SAP before making a significant customization to FS-QUO. This is to ensure that the customization is in line with the architecture of the system and will continue to be supported in future releases.

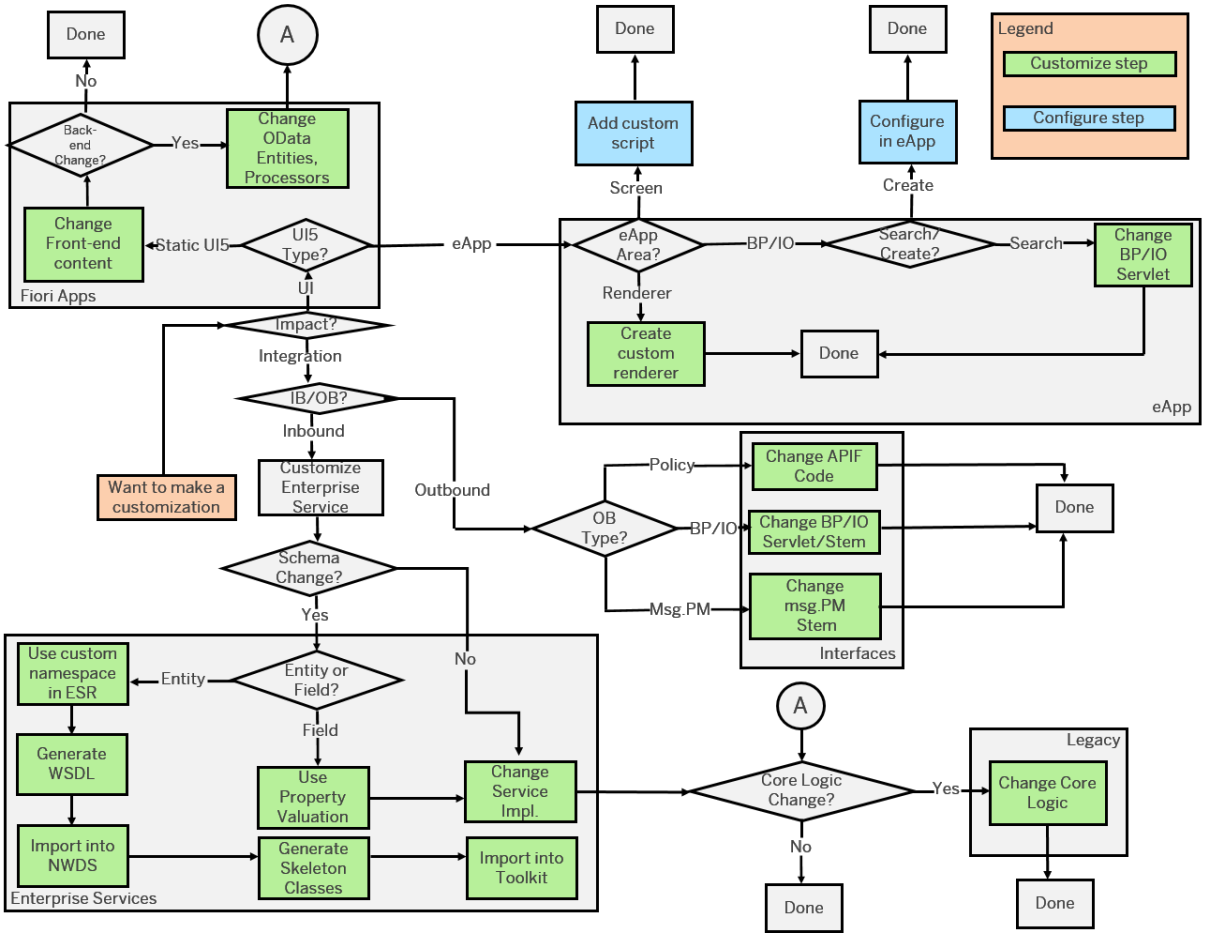
3.1 Understanding Design Time

The following diagram shows the various systems and roles involved during design time for insurance applications. Any customizations are made in this environment.



3.2 Customization Site Map

The following diagram outlines what types of customizations are possible and the steps involved. The next section contains extensibility scenarios that provide examples for making each of these types of customization.



3.3 Extensibility Scenarios by Area

Before you begin, you must have a successfully installed FS-QUO Toolkit in order to perform these scenarios. These scenarios should work both locally in TomEE as well as on SAP HANA XS Advanced.

3.3.1 Example: Modifying the General Information Screen in the FS-QUO Fiori App

This topic details a sample scenario to demonstrate FS-QUO extensibility.

Context

The purpose of this example is to add a new field, *Producer License*, to the *General Information* screen. This field will trigger a message when its value changes. Its value will be populated from the back-end.

Procedure

1. In the `QUOUINewBusiness` project, open the `GeneralInfo.view.xml` file.
2. Find the input element with the id value `inputTextforProducerCode`. After that element, add the following text:

```
<Input id="inputTextForProducerLicense"
enabled="true"
change="handleProducerLicenseChange">
<layoutData>
    <l:GridData span="L3 M3">
</layoutData>
</Input>
<Text text="Producer License"/>
```

3. Open the `GeneralInfo.controller.js` file.

> **Note**
>
> This Controller declares it's binding to the above View

4. Above the function `handleDateChange`, add the following text:

```
handleProducerLicenseChange: function(oEvent) {
    sap.m.MessageBox.show(“Producer License is set to: " +
oEvent.getSource().getValue() ,{
    icon: sap.m.MessageBox.Icon.WARNING,
        title: this.oBundle.getText("WARNING"  ) ,
        actions:
[fs.ipw.insquote.create.util.Formatter.i18nFormatter("BTN_OK")]
    } ) ;
},
```

5. In the `com.sap.fs.quo.web` project, open class `com.sap.fs.quo.web.odata.model.Producer` and create a new field under `producerName`, as follows:

```
@EdmProperty
protected String producerLicense;
```

6. Add `getter` and `setter` methods for this new field and update methods `equals()` and `hashCode()` to include this new field.

   > **Note**
   >
   > This can be done automatically in Eclipse by going to *Source* > *Generate hashCode() and equals()*.

7. Open class `com.sap.fs.quo.web.odata.mapper.ProducerMapper` and modify method `mapModelToEntity(2 args)` – under the line containing `producer.getAgencyName()`, add the following line: `mappedEntity.setProducerLicense("XYZ");`
8. Build the toolkit.
9. Test the change in runtime:
   a. Start a new quote.
   b. Search for and select a *Producer*.
   c. The *Producer Details* should be populated on the *General Information* screen.
   d. Check that the new *Producer License* field appears as part of the *Producer Details*.
   e. Check that an alert appeared, due to the fact that the *Producer License* field value changed when the *Producer* was selected.

3.3.2 Example: Creating a Custom eApp Renderer

This topic details a sample scenario to demonstrate FS-QUO extensibility.

Procedure

1. In the FS-QUO-install project (CSI_HOME contents), create a folder named `/src/main/resources/ppms/app/as/custom/web/eapp/renderer/myCustomRenderer1`.
2. Open the `camilionweb.war` archive and copy the contents of the `/resources/eapp/ui5/renderer/splitApp` folder and paste them into the folder you created in the previous step.
3. Open the `…/myCustomRenderer1/view/Detail.view.xml` file. Add the following text in the file; place the new text before the Button element with the id = dtlSave, : `<Button id="customButton" text="My Custom Button" press="customAlert" />`
4. Open the `…/myCustomRenderer1/view/Detail.controller.js` file. Add the following text in the file; place it before the definition for function handleNext: `customAlert : function(oEvent) { alert("Hello, this is a customization."); },`
5. Build the toolkit.
6. Apply your custom renderer in the Product Modeler. Open the `Data List Base` product. In the `DLRendererSource` component, add a data value row with the following values:

   ```
   Name = Custom Renderer 1
   Class = myCustomRenderer1
   Type = EAPP_LAYOUT
   ```

7. Publish the `Data List Base` product, and open the desired test product.

8. In the `eApp Layout` component, locate the data value row with the following information: `eApp Name = ProducerApplicationPQM30:`. Change `Renderer` to `Custom Renderer 1`.
9. Publish and deploy your desired product.
10. Test the change in runtime:
    a. Start a New Quote, selecting your desired product, and go into the eApp.
    b. Select *Policy* from navigation panel on the left side.
    c. On the detail panel on the right side, in the footer bar, check that the *My Custom Button* button appears to the left of the *Save* button
    d. Choose *My Custom Button*.

Results

You should receive an alert with message `Hello, this is a customization`.

3.4 Adding a Custom Project

You can add a custom project.

Procedure

1. In *Project Explorer*, right-click on FS-QUO-*toolkit* and select *New* > *Other...* .
2. Select *Maven* > *Maven Module* and choose *Next*.
3. Select the *Create a simple project* checkbox.
4. Enter the module name and then choose *Next*.
5. Change the packaging depending on what type of module it is:
    - Java utility: jar
    - EJB: ejb
    - Web: war
6. Choose *Finish*.
7. Open the reactor `pom.xml` file (that is, the FS-QUO-`toolkit pom.xml` file) and re-arrange the `<module>` definition of the new module if needed.
8. If you are using a source control system such as SVN or Git, add the following directories/files in the new module to the ignore list:
    - `/.settings`
    - `/.classpath`
    - `/.project`
    - `/target/`

9. Open the new module's `pom.xml` file and add dependencies as appropriate.
10. Add the new module to `FS-QUO-webapp` by adding the following XML element in the `FS-QUO-webapp pom.xml` file, below the comment of `"<!-- FS-QUO module dependencies -->"` in the `pom.xml`:

    The sample below uses "custom-project" as the product/module name:

    ```
    <dependency>
            <groupId>${project.groupId}</groupId>
            <artifactId>custom-project</artifactId>
            <version>${project.version}</version>
                   <exclusions>
                        <exclusion>
                             <groupId>*</groupId>
                             <artifactId>*</artifactId>
                        </exclusion>
                    </exclusions>
    </dependency>
    ```

    This allows the custom project to be packaged in the MTAR file.
11. Repeat step 10 for `FS-QUO-dev-webapp`. This will allow the custom project to be packaged/deployed to the local TomEE server artifacts.

3.5 Unsupported Extension Types

There are limits to the extensibility allowed in FS-QUO application development.

The following types of extensions aren't supported:

- Changing policy model object relationships
- Adding transaction types or statuses
- Batch or offline processing
- Making existing processes "headless"
- Externalizing persistence (for example, for Producer or Account data)
- Managing uploads through the browser
- Changes in core logic, such as Premium Handling and Out-of-Sequence
- Changing Spring aspects

4 Core FS-QUO

4.1 Business Services

Business services are Java methods that FS-QUO uses to perform operations on policies.

For example, when you create a new account using the FS-QUO user interface, FS-QUO invokes the create method in the Account Business Service. Similarly, to add a form, FS-QUO invokes the addForms method in the Form Business Service.

Business services aren't specific to products.

You can extend the application by adding business service classes or methods, or by adding logic to a business service method.

4.1.1 Accessing the javadoc

You can view the FS-QUO javadoc to see details about the stem classes and their functions and arguments. This javadoc contains information about the FS-QUO API.

Context

For details about the business service classes and methods, see the FS-QUO javadoc. This javadoc contains information about the FS-QUO API. For information about business services, see the following package: `com.sap.fs.quo.service.api`

Procedure

1. Contact your FS-QUO administrator to get the FS-QUO software download ZIP file.

   The ZIP file is available to customers in the SAP Software Download Center under *PRODUCT AND QUOTATION MGMT*.
2. From the software ZIP file, find the following file and unzip it: FS-QUO`-javadoc-releasenumber.zip`

   The files are extracted.
3. Open `index.html` in a browser.

Results

The javadoc is displayed.

4.1.2 Business Service Components and Methods

4.1.2.1 Package Structure for Business Services

The Business Service class uses the following package structure: `com.camilion.as.service.business.<serviceCategory>`

`<serviceCategory>` Specifies the name of a category of services, such as Account services.

The Business Service API is defined in: `com.sap.fs.quo.service.api`

`com.sap.fs.quo.service.api` Represents our publicly accessible business services.

The Business Service implementation classes are defined in: `com.sap.fs.quo.service.impl`

`com.sap.fs.quo.service.impl` Represents the functionality of the business services.

The Core Business Services are defined in: `com.sap.fs.quo.service.internal`

`com.sap.fs.quo.service.internal` Provides the core workings of the system.

4.1.3 Understanding the Core Service Components

Core services are the underlying layer for business services.

In a FS-QUO upgrade, the signatures of the core services might change. Therefore, don't create a service class that extends a core service. Instead, you can extend a business service. FS-QUO preserves the signatures of the business services.

4.1.3.1 Package Structure for Core Services

The `Core Service` class uses the following package structure: `com.sap.fs.quo.service.internal`

4.1.4 Preserving Extended Services After Upgrading

When you upgrade FS-QUO, your data is migrated to a new environment. To preserve the changes made to your extended business service beans, merge the `quoServicesV2.xml` file, and build, deploy, and restart the server.

4.2 Underwriting Application Configuration

The FS-QUO application configuration is stored in the `Underwriting Application Configuration` standalone object in the Product Modeler. You can modify the configuration using objects inherited from the `Underwriting Application Configuration` standalone object.

The application configuration is accessed through the `Underwriting Application Configuration` service bean. Any Business Service that requires configuration elements use the `Underwriting Application Configuration` bean to retrieve its configuration from the `Underwriting Application Configuration` product.

The following folders in the `Underwriting Application Configuration` object are associated with specific frameworks and are described in separate topics:

- `Validation` folder
- `Data Privacy` folder

The `Underwriting Application Configuration` object contains all the information that is used to configure the FS-QUO application. For example, these settings determine how search results are displayed and the number of days before a quote expires.

4.2.1 Locating the Standalone Underwriting Application Configuration

The `Underwriting Application Configuration` standalone object is located in the *System Repository* > *Underwriting Application Template Library* > *Configuration Objects* > *Underwriting Application Configuration* folder of the Product Modeler.

> ⚠ Caution
>
> Don't modify the `Underwriting Application Configuration` standalone object. The standalone object is replaced when you upgrade your environment. To define application changes that are specific to your organization, modify an extended (inherited) `Underwriting Application Configuration` object.

4.2.2 Extending the Underwriting Application Configuration Standalone Object

You can define application changes that are specific to your organization.

Context

We recommend that you do not modify the `Underwriting Application Configuration` standalone object because it is replaced during FS-QUO upgrades. Instead, define application changes in an object extended (inherited from) the `Underwriting Application Configuration` standalone object.

> **Note**
>
> During implementation, an `Extended Underwriting Application Configuration` object may have already been created in the *Product Repository* > *Company Library* > *Configuration Objects* folder. If that object exists, you can skip this procedure. You can modify the objects in the `Extended Underwriting Application Configuration` object as needed.

Procedure

1. Log in to the Product Modeler.
2. Expand your `<company>` *Library* folder in the *Studio Tree* and choose *Config Objects*.
   The *Object List* tab opens.
3. Choose the  icon in the *Object List* tab,.

   The *New Object* dialog appears.
4. Complete the fields as follows:
   - *Name*—Enter the name for the new object (for example, **`Extended Underwriting Application Configuration`**)
   - *Component Type*—Choose the *Application Configuration* option.
   - *Based On*—Select *Underwriting Application Configuration* from *System Repository* > *Underwriting Application Template Library* > *Configuration Objects* > *Underwriting Application Configuration*.
5. Choose *OK*.
   The new object is created and displayed in the *Object List* tab.
6. Register the new extended `Underwriting Application Configuration` object as follows:
   a. Launch Internet Explorer and log in to the Design Time Administrative Console at the following location: `<pro_designtime_app_url>/csiroot/admin/`.

      The Design Time Administrative Console will open after a short delay.
   b. Choose *System* > *Configuration Manager*.
      The *Configuration* folder tree is displayed.

c. Choose *Configuration* > *Application* > *AuthoritySuite* > *Env* > *Application Environment* .
The *Application Environment* properties are displayed.
d. Go to the *AppConfig_Product_Name row* and specify the name of the new extended `Underwriting Application Configuration` object in the *Override* field.
e. Go to the *AppConfig_Product_Version* row and specify the version number of the new extended `Underwriting Application Configuration` object in the *Override* field,.
f. Save your changes.

The extended `Underwriting Application Configuration` is registered in the Administrative Console of the FS-QUO runtime environment.

Results

The extended `Underwriting Application Configuration` object is created.

> **Note**
>
> The FS-QUO runtime environment can only point to one extended `Underwriting Application Configuration` object. You might have multiple extended `Underwriting Application Configuration` objects for different libraries in the Product Modeler. However, the FS-QUO runtime uses only the one extended object specified in the Administrative Console for its application configuration.

4.2.3 Modifying Application Configuration Settings

You can modify the FS-QUO application settings according to your organization's needs.

Procedure

1. Open your extended `Underwriting Application Configuration` object.

   Typically, the object is named `Extended Underwriting Application Configuration` and is located in the `<Company>` *Library* > *Configuration Objects* folder.

   XML representation of the data for a specific policy transaction. The model is generated by FS-QUO and contains information from the FS-QUO `Data Model` and the `Product Data Model`. The `Domain Layer` lets you access data without querying the FS-QUO database tables and views. Business services can get policy information from the Domain Layer.
2. Publish the extended `Underwriting Application Configuration` object.

4.2.4 Folders in the Underwriting Application Configuration Object

The `Underwriting Application Configuration` object consists of the following folders:

`Config API`	Contains an object that calls business services, runs rules, and specifies the return type for the FS-QUO application. Contains objects that configure the permissions on controls, mandatory fields, and business questions. The objects in this folder use parts of the permissions and validation frameworks.
`Configuration`	Contains an object that specifies a Service method and function to populate reusable dropdown lists (such as *State*).
`Business Behavior`	Contains objects that specify business behavior within the application (for example, the number of days before a quote expires or the values that appear in the *Task Type* dropdown list).
`Validation`	Contains objects that configure validation for screen controls and business services. The objects in this folder use the validation framework.
`Reference`	Contains an object that defines the insurance products that can be selected in the `Select Product` activity.
`Error Messages`	Contains a list of error messages that are returned when the type of data entered into a field isn't consistent with the type of data the `Domain Layer` column expects.

4.2.4.1 `Application Configuration API` Folder

The `Underwriting Application Configuration` folder contains the `Underwriting Application Configuration API` object, which contains the services and rules used by FS-QUO.

4.2.4.1.1 Application Configuration API object

The `Underwriting Application Configuration API` object contains rules that call methods to get information for the FS-QUO application.

The `Underwriting Application Configuration API` object has the following attributes:

SERVICE_NAME

Specifies the name of the service that's called.

SERVICE_RULE

Defines the rule that contains the XPath to get the data.

Return Type

Specifies the data return type.

The following options are available:

- Boolean
- DataTable
- String

Inputs

Specifies information to pass to the `Underwriting Application Configuration API` method.

This information comes from FS-QUO.

4.2.4.2 `Configuration` Folder

The `Configuration` folder contains the `Lookup Service Configuration` component. This component is used to access external lookup methods.

4.2.4.3 `Business Behavior` Folder

The `Business Behavior` folder contains objects that configure business behavior within the FS-QUO application.

4.2.4.3.1 `Transaction Management` Folder

The `Transaction Management` folder contains objects that configure policy transaction information.

4.2.4.3.1.1 `Business Behavior` Object

The `Business Behavior` object specifies configurable transaction business behaviors within the FS-QUO application.

The `Business Behavior` object has the following attributes:

propertyName

Specifies the property name called by the code.

propertyValue

Specifies the value FS-QUO uses to change the business behavior.

4.2.4.3.1.2 `Quote Not Taken Reason List` Object

The `Quote Not Take Reason List` object contains the content that's available in the *Reason* field in the *Quote Not Taken* window.

The `Quote Not Take Reason List` object has the following attributes:

propertyName

Specifies the property name that FS-QUO uses to process the entry.

propertyValue

Defines the description that's displayed in the dropdown list.

4.2.4.3.2 `Clearance` Folder

The `Clearance` folder contains an object that configures product clearance behavior.

4.2.4.3.2.1 `Business Behavior` Object

The `Business Behavior` object specifies configurable clearance business behaviors within the FS-QUO application.

The `Business Behavior` object has the following attributes:

propertyName

Specifies the property name called by the code.

propertyValue

Specifies the value that FS-QUO uses to change the business behavior.

4.2.4.3.3 `Business Questions Set` Component

The `Business Questions Set` component assembled in the `Business Behavior` folder define the verification checks that the FS-QUO runtime can use.

These sets are groups of related business questions. For example, when a producer is selected during a transaction, FS-QUO runs a business question to determine whether the producer is active.

The business questions in the `Business Behavior` folder aren't specific to screen controls or business services.

> **Note**
>
> The `Business Questions Set` component is reused (assembled) in multiple frameworks.

4.2.4.3.4 `Business Questions` Component

The `Business Questions` component contains all the business questions that the FS-QUO runtime can use.

For business behavior, you can't select individual questions from the `Business Questions` component. Instead, anchor the questions to the parent `Business Questions Set` component and select the applicable sets.

> ⓘ Note
>
> The `Business Questions Set` component is reused (assembled) in multiple frameworks.

4.2.4.3.5 Configuring Business Behavior Questions

You can configure business behavior questions for FS-QUO.

Context

FS-QUO runs default sets of business questions to verify requirements in the runtime environment. You can modify or add questions in the `Business Behavior` folder of the `Underwriting Application Configuration` object.

Procedure

1. Open your extended `Underwriting Application Configuration` object.

   Typically, the object is named `Extended Underwriting Application Configuration` and is located in the ▶ `<Company>` *Library* ▶ *Configuration Objects* ▶ folder.
2. Expand the *Business Behavior* folder, and choose the *Business Questions Set* tab.
3. Select the *Values* tab.
4. Select All from the *Views* dropdown list.

   All the values are displayed.
5. Perform one or both of the following actions:
   - To use existing business question sets, you need to select the checkbox next to the sets that you want to apply to the screen control.
   - To add a new business question set, you need to add a new row, complete the fields, and select the checkbox.
6. Deselect the checkbox next to the business question sets that don't apply.
7. Save your changes.
8. If you created a new business question set, modify the `Underwriting Application Configuration API` object as follows:

a. Open the `Underwriting Application Configuration API` component in the `Config API` folder.
b. Select the *Values* tab.
c. In the FS-QUO code that calls the name of the product service API (defined in the `Underwriting Application Configuration API` component), pass the name of your new business question set as an argument.

Example

Adding ClearanceRules business questions

When a producer writes a policy, FS-QUO checks whether that producer is active at the time the policy becomes effective. This validation is defined in the clearanceRules business questions set in the `Business Behavior` folder of the `Underwriting Application Configuration` object.

Suppose that you want to create additional validation checks. In the extended `Underwriting Application Configuration` object, you can add new questions to the `Business Questions` object and anchor (select) them to the `ClearanceRules` business questions set.

4.2.4.4 `Reference` Folder

The `Reference` folder contains the `Available Products and Lookup Forms Type` component.

4.2.4.4.1 `Available Products` Object

The `Available Products` object contains the products that are available in the `Select Product` activity.

The `Available Products` object has the following attributes:

Product Id
Specifies the value that FS-QUO uses to identify the product.

Product Name
Specifies the product name that appears in the `Select Product` activity.

Version No
Specifies the product version number.

Display Order
Specifies the order that the products are displayed in.

4.2.4.5 Example: Modifying the Allowable Lapse Period

Context

If you cancel a policy in FS-QUO, you can reinstate it within 10 days by default. For example, if you cancel a policy on October 1st, you can reinstate it by October 10th.

The number of days is defined using the `maximumAllowableLapsePeriod` configuration variable in the `Underwriting Application Configuration` object. You can override the default value of 10 days if needed.

Procedure

1. Log in to the Product Modeler.
2. Open your extended `Underwriting Application Configuration` object.

   Typically, the object is named `Extended Underwriting Application Configuration` and is located in the *<Company> Library* > *Configuration Objects* folder.
3. Navigate to *Business Behavior* > *Transaction Management* > *Business Behavior* .

   The `Business Behavior` component opens.
4. Select the *Values* tab.
5. Override the `maximumAllowableLapsePeriod` row, and enter a new value in the `propertyValue` attribute.
6. Save and commit the change.

   The `maximumAllowableLapsePeriod` configuration variable is modified.
7. Optionally, modify the associated screen control validation and settings.

4.2.4.6 Debug Console Folder

The `Debug Console` folder contains the configuration objects for the *Debug Console*.

4.2.4.6.1 Enabling the Debug Console

The *Debug Console* is disabled by default; you can enable it.

Procedure

1. Launch Internet Explorer and log in to the Runtime Administrative Console at the following location: `<pro_runtime_app_url>/csiroot/admin/`.
   The Runtime Administrative Console will open after a short delay.
2. Choose ▶ *System* ▶ *Edit Configuration Settings* ▶ from the menu bar.
3. Navigate to ▶ *Application* ▶ *ProductAuthority* ▶ *Env* ▶ *Application Environment* ▶
4. Find the row with the name *SuppressErrorInAS* and set the override value to No.
5. Save your changes.
6. Navigate to ▶ *Application* ▶ *AuthoritySuite* ▶ *Env* ▶ *Application Environment* ▶
7. Find the row with the name *SUPPRESS_ERROR_DETAILS* and set the override value to No.
8. Save your changes.
9. Reload the configuration settings as follows:
   a. In the Administrative Console menu bar, select ▶ *System* ▶ *Reload Configuration Settings* ▶.
   b. Select *Reload All Config*.

4.2.4.6.2 Accessing the Debug Console

The *Debug Console* is available in all areas of all apps. You can access it by pressing `Ctrl` + `Shift` + `Alt` + `D`, which opens the *Debug Console* in a new browser tab.

> ⚠ Caution
>
> The *Debug Console* is only available and supported in desktop mode.

4.2.4.6.3 `Debug Categories` Object

The `Debug Categories` object defines the sections that display in the *Debug Console* navigation bar.

The *Debug Console* displays the following sections:

- *Session*
- *State*
- *Data*
- *System*

- *Trace*
- *Util*

The `Debug Categories` object has the following attributes:

PCD

Specifies the unique identifier for the data value row in the component and across the system.

You can leave this attribute blank, and the system automatically generates this value.

name

Defines the name that's displayed in the section title.

description

This attribute isn't used.

4.2.4.6.4 `Debug Sections` Object

The `Debug Sections` object defines the options that display in the sections in the *Debug Console* navigation bar.

The *Debug Console* displays the following options:

- *Tracing*
- *UserInfo*

The `Debug Sections` object has the following attributes:

PCD

Specifies the unique identifier for the data value row in the component and across the system.

You can leave this attribute blank, and the system automatically generates this value.

name

Defines the name that's displayed in the section.

description

Defines the instruction text that displays in the title bar when you hover over the section.

url

Specifies the URL of the page that's called to render the debug page.

4.2.4.6.5 Modifying a Debug Console Setting

You can modify the *Debug Console* settings according to your organization's needs.

Context

For example, you can change the sections that display in the *Debug Console* navigation bar by modifying the *Debug Categories* object.

Procedure

1. Open your extended `Underwriting Application Configuration` object.

   Typically, the object is named `Extended Underwriting Application Configuration` and is located in the `<Company>` *Library* > *Configuration Objects* folder.
2. Expand the *Debug Console* folder.
3. Open the component that contains the settings that you want to modify.
4. Select the *Values* tab.

   The *Values* tab opens.
5. Right-click the row heading (the left-most cell) of the row that contains the configuration setting that you want to modify, and choose *Override*.

   The *Override* dialog opens.
6. Select *Override* association and value.

   The row is now local and can be modified.
7. Modify the attributes as needed and save your changes.
8. Right-click the row heading, choose *Locking* and then choose *Commit*.

Results

The configuration setting is modified.

5 Risk-Based Architecture

5.1 Extending Fiori Apps UI

FS-QUO is composed of UI5 web applications, and the framework is based on UI5 Fiori. The core library is "UI5 Mobile" (sap.m).

Browser and platform support aligns with the UI5 specification.

The middle tier is based mainly on OData services.

The UI is customized by configuring Components. You can extend the delivered standard application through such customization.

Prerequisites

Because the customization is implemented on Fiori web applications, the rules of Fiori must be complied with.

You should use the component.js as the entry file to define all configuration parameters of your web application, such as includes, dependencies, routing, customizing and so on.

Principle

The front end is based on the UI5. The following aspects can be customized:

- Views can be replaced entirely by custom-developed Views
- Views can be modified by customizing specific properties of controls
- Controllers can be extended with additional code (optionally overriding existing code)
- i18N resource texts can be customized
- New views can be added
- New navigation paths can be added
- Navigation routes can be customized
- Inheritance

For detailed information about extending Fiori Apps, see Extending Apps.

5.2 Working with Roles and the Permissions Framework

In FS-QUO, each public service requires permission to execute that service.

These permissions are defined in the following XML file: `<CSI_HOME>/ppms/app/pa/custom/integration/permission/permission_config.xml`

Each user must be defined in a <UserPermissions> tag. This tag will contain a child tag <UserLoginName> with the user's login name. The tag will also contain one or more child <Permission> tags. These tags will point to a PermissionSet or the name of a service that the user has access to.

A <PermissionSet> tag defines a set of permissions available to a user. This tag will contain a <ParentPermission> tag defining the name of the PermissionSet. This tag will also contain one or more <ChildPermission> tags which can refer to other PermissionSets or the name of a service that the user has access to.

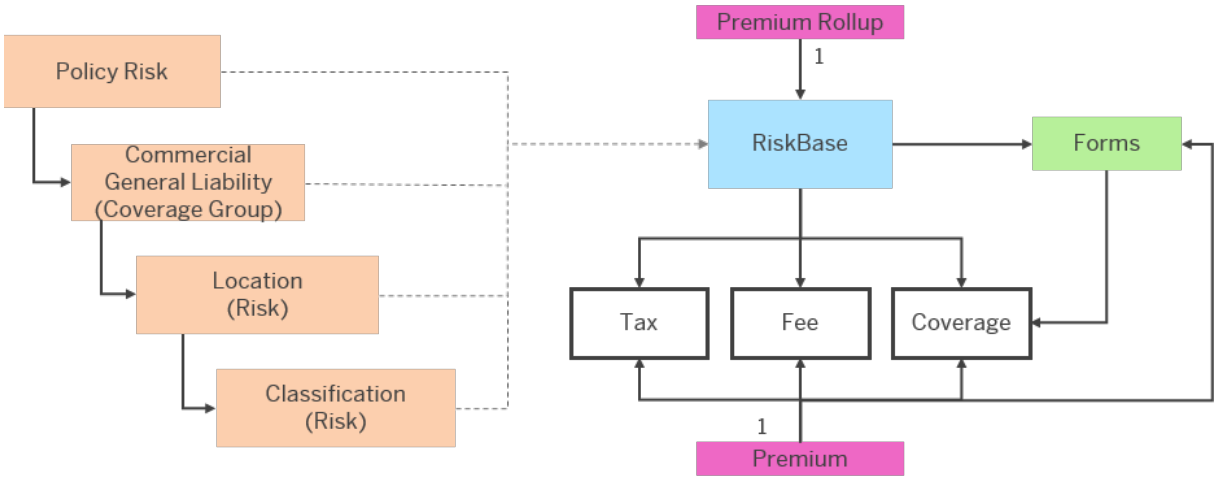
Each public service that the user needs access to must be defined by name in a <Permission> or <ChildPermission> tag.

5.3 Understanding Visitor Patterns

Visitor patterns are the mechanism that FS-QUO uses to traverse the policy tree (the structure of the FS-QUO `Domain Layer`) to return policy information.

The visitor pattern visits each node (risk) and determines the premium, premium rollup, referral, and forms for that risk.

The following diagram shows an example of how the policy risks are extensions of the RiskBase method and the relationship of the RiskBase method to premium rollup, coverage, forms, and premiums:



5.3.1 Model Visitor

The model visitor is the name of the FS-QUO visitor class. The visitor pattern abstracts the recursive traversal of a `Domain Layer` instance or a `Domain Layer` definition. This abstraction lets you apply functions to the traversal of a model instance or model definition without knowing how to traverse the object tree.

For example, coverage selection can implement a visitor extension to run rules on the coverage definitions for a given product. As another example, you can write a visitor extension to return all attached forms for a given model instance.

5.3.2 SubmissionVisitor Class

To determine the premium, premium rollup, referrals, and forms for each risk in a policy tree, the model visitor uses the following methods:

```
package com.sap.fs.quo.service.internal.impl;
public class SubmissionVisitor
{
protected void visitOption(SubmissionData policyTransaction, QuoteOptionData
quoteOption, Map<String, Object> context)
protected void visitCoverageGroup(SubmissionData policyTransaction,
QuoteOptionData quoteOption, CoverageGroupData coverageGroup, Map<String,
Object> context)
protected void visitRisk(SubmissionData policyTransaction, QuoteOptionData
quoteOption, CoverageGroupData coverageGroup, RiskData risk, Map<String, Object>
context)
protected void visitCoverage(SubmissionData policyTransaction, QuoteOptionData
quoteOption, CoverageGroupData coverageGroup, RiskData risk, RiskCoverageData
coverage, Map<String, Object> context)
protected void visitFee(SubmissionData policyTransaction, QuoteOptionData
quoteOption, CoverageGroupData coverageGroup, RiskData risk, RiskCoverageData
fee, Map<String, Object> context)
protected void visitTax(SubmissionData policyTransaction, QuoteOptionData
quoteOption, CoverageGroupData coverageGroup, RiskData risk, RiskCoverageData
tax, Map<String, Object> context)
}
```

5.3.2.1 Extending a Visitor

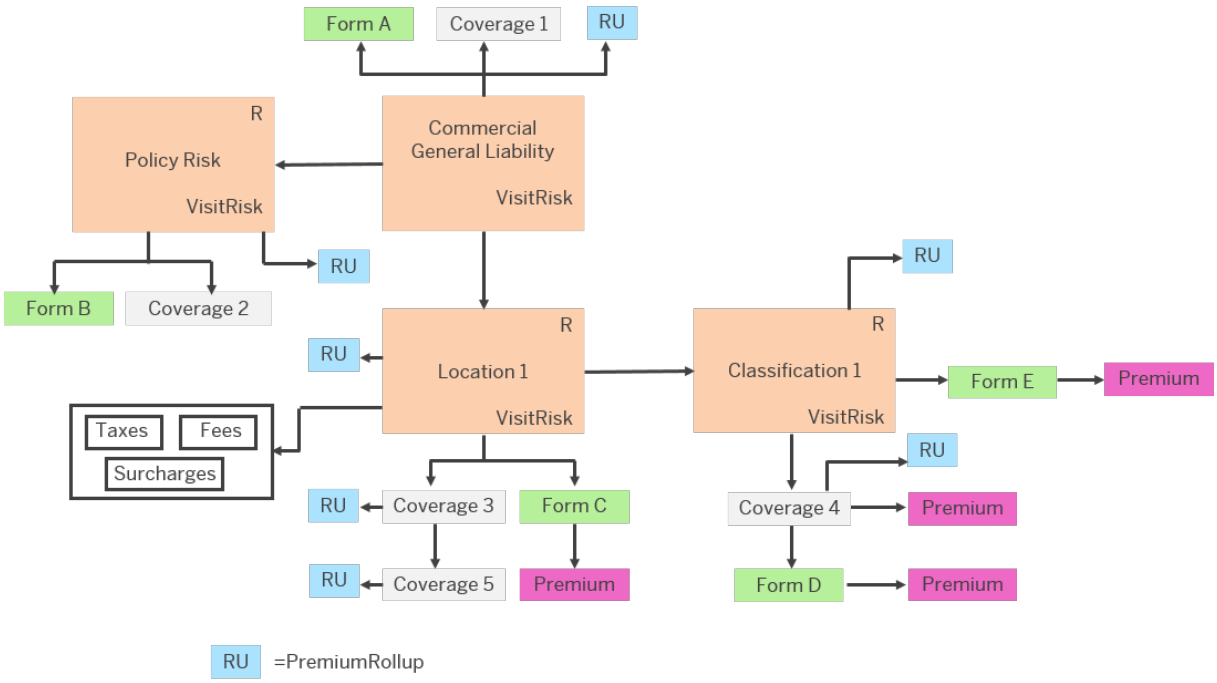
To extend a visitor, create a new class and extend from the SubmissionVisitor class.

It's safe to override any of the methods listed in the *SubmissionVisitor* topic of this guide. It is not recommended to override any of the methods listed as `Internal`.

Each of these methods has a context map parameter which can be used to pass objects and values from one method to another.

5.3.3 Policy Traversal

The following example is a visitor pattern that uses the `Commercial General Liability` line of business (coverage group):



The visitor pattern traverses the following policy parts:

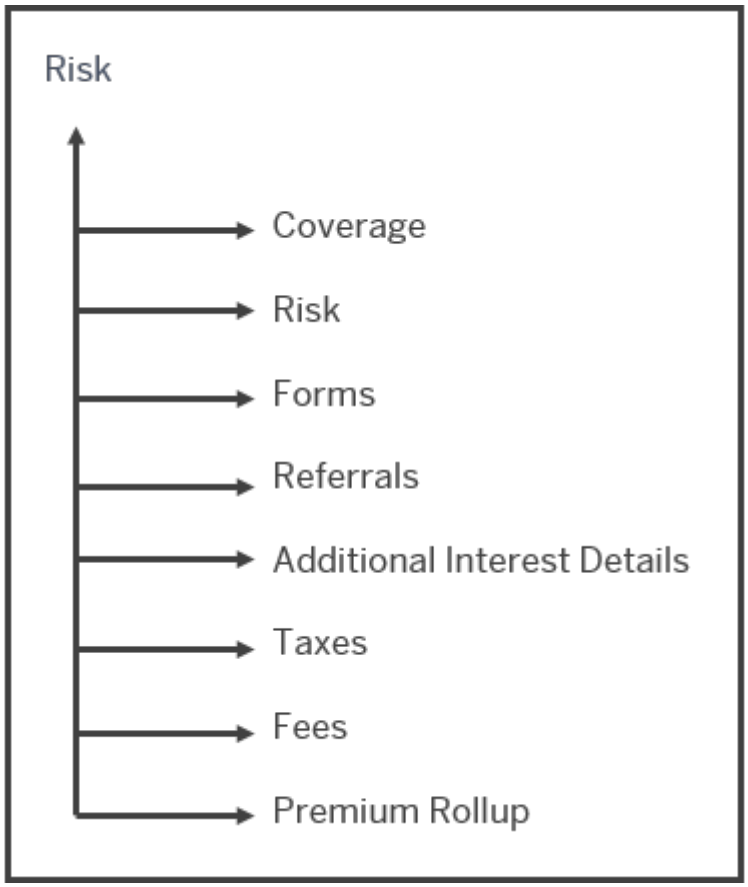
PolicyRisk	The policy risk is the product. It contains coverage groups, forms, and premium rollups.
Coverage Groups	A coverage group consists of related coverages. Coverage groups can have forms, financials, and premium rollups.
Risks	A risk is anything that has a coverage. Risks can have forms, financials, and premium rollups. Risks can also have sub-risks.
Coverages	Coverages are the rated categories of the risk. Coverages can have forms, financials, and premium rollups.
Forms	A form is associated with a risk or a coverage. Forms can have premiums associated with them.
Financial	Financials are charges that include coverages, taxes, and fees that are associated with a risk.
PremiumRollup	The premium rollup holds the sum of all the premiums that are under a level. For example, the premium rollup for a risk includes all financials associated with that risk.
StatCodes	Stat codes are associated with financials. Stat codes are markers for reporting.

5.3.4 Visitor Patterns

The visitor pattern traverses the FS-QUO `Domain Layer` to create a snapshot of the policy. The traversal pattern is different for risks, coverages, forms, and premiums.

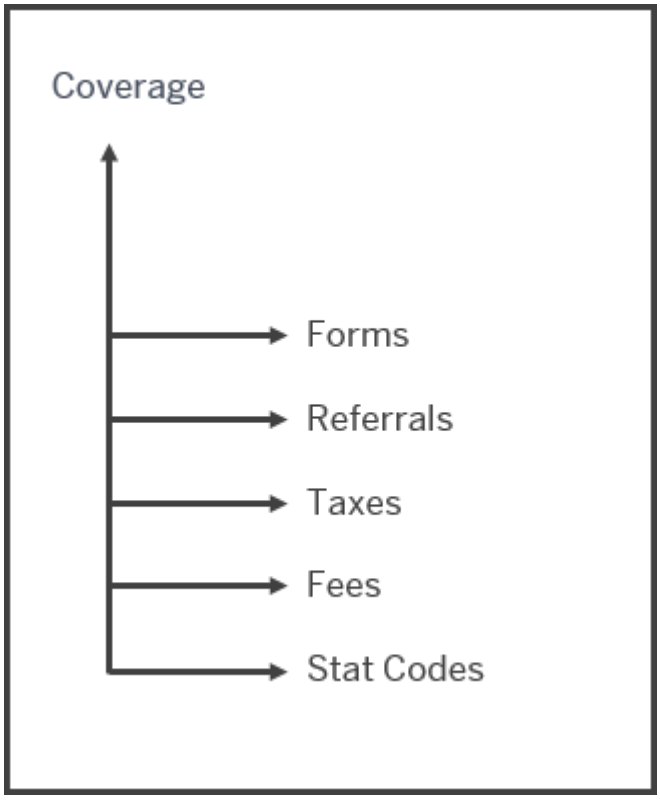
Risk Visitor Pattern

The risk visitor pattern has the following traversal pattern:



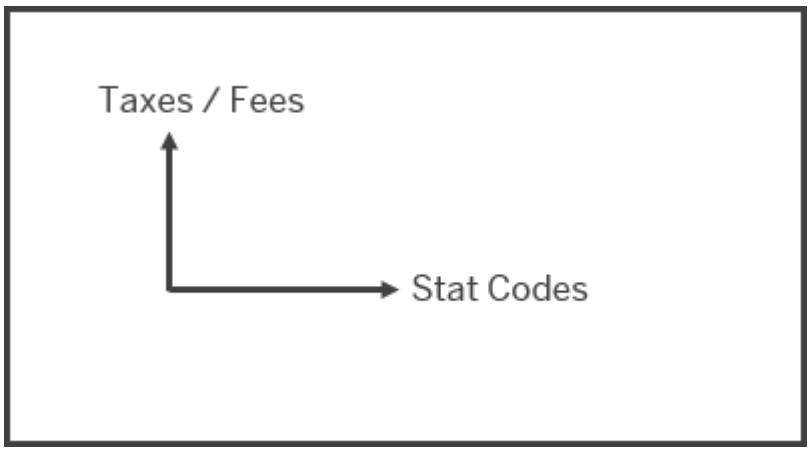
Coverage Visitor Pattern

The coverage visitor pattern has the following pattern:



Taxes and Fees Visitor Patterns

Taxes and fees have the same pattern, as follows:



5.4 Domain Layers

The `Domain Layer` is an XML representation of the data for a specific policy transaction.

The model is generated by FS-QUO and contains information from the FS-QUO `Data Model`. The `Domain Layer` lets you access data without querying the FS-QUO database tables and views. Business services can get policy information from the `Domain Layer`.

The `Domain Layer` consists of the static model and dynamic model. The static model is a representation of the data that's the same for every policy transaction. For example, the static model contains attributes for the select account and producer fields.

The dynamic model is a representation of the data for a specific product. The dynamic model can't be determined until a product is selected in a policy transaction.

5.4.1 Domain Layer and Database Tables

The `Domain Layer` attributes are mapped to physical database tables. The database tables are grouped as follows:

- Policy tables
- Supporting tables
- Task table

Related Information

Policy Database Tables [page 46]
Supporting Database Tables [page 78]

5.4.2 Modifying the Domain Layer

Prerequisites

To modify the `Domain Layer`, the following is required:

- Java version 1.8 is installed in your development environment. No other Java version is compatible with FS-QUO.
- You have access to the FS-QUO Development Toolkit.

Context

Generally, we recommend that you don't modify the `Domain Layer` because you must manually recreate the changes when you upgrade FS-QUO. However, if you want to collect or display additional information in the FS-QUO runtime, you can add new fields or sections to the user interface. To add a new field or section, you must add a field to a database to support the product (using the Product Modeler) and add an attribute or class to the `Domain Layer`.

For example, you might want to add a *County* field to the address section. To store County data, you need to create a column to the ADDRESS table and add an attribute to the `Domain Layer`. As another example, you might want to add a new section to display claims history information gathered from an external source. To store the claims history data, you need to create a new flowstore and add a new claims class to the `Domain Layer`.

> ### Note
>
> The following process describes how to modify the static model of the `Domain Layer`. The dynamic model can't be determined until a product is selected in the policy transaction.

Procedure

1. Install the FS-QUO Eclipse projects (libraries and source) from the FS-QUO Development Toolkit.
2. Update the flowstore.
3. Open the Domain Layer Packages.
4. Do one or more of the following:
   - Add a Class to the `Domain Layer`
   - Add an Attribute to a Class
   - Add a Reference to a Class
   - Delete a Class or Attribute
5. Generate the classes.

Next Steps

We recommend that you track all changes to the `Domain Layer` in a log file and version the file with the `com.sap.fs.quo.domain.data` project. During an upgrade, you can re-apply the modifications to the `Domain Layer`.

The Domain Layer classes are all located in the `com.sap.fs.quo.domain.data` project.

5.4.2.1 Updating the Flowstore

If you want to collect or display additional information in the FS-QUO runtime, you can add new fields or sections to the user interface. To store the additional data that you gather or display, you must update the flowstore (database) tables.

Procedure

1. Launch Internet Explorer and log in to the Product Modeler at the following URL: `https://<pro_designtime_app_url>/csiroot/ii/pc/`
   The Product Modeler will open after a short delay.
2. Do one of the following:
   - If you are adding a new field to an existing table, add an entry to the column component of the table's data definition.
   - If you are adding a new table to the user interface, create a data definition object, define the table component, and include entries for each field in the column component.
     Then, assemble the data definition into the `UW Application Reference Insurance Data Model`. To define the hierarchy, ensure the table you are adding is the child of the appropriate parent table.

   > **Note**
   >
   > We recommend that you prefix modified tables and columns with a value that's specific to your organization (for example, yourCompanyName_NEW_TABLE or yourCompanyName_NEW_COLUMN). By prefixing the table and column names, you can avoid name clashes when you upgrade FS-QUO.

3. Synchronize process flowstores:
   a. Launch Internet Explorer and log in to the Runtime Administrative Console at the following location: `<pro_runtime_app_url>/csiroot/admin/`.
      The Runtime Administrative Console will open after a short delay.
   b. Go to *Process* > *Synchronize Process Flowstore*.
      A log file appears. During this process, *Next* is dimmed.
   c. When the log is complete, choose *Next*.
   d. The flowstores that appear in red don't match the contents of the database. SQL statements are generated by the system to match the database to the flowstore. To view SQL statements related to those flowstores, select them. The SQL statements appear in the detail panel.
      - If the flowstore table is missing from the database, you will see statements that will create it.
      - If the table exists on the database, but the structure doesn't match that of the flowstore table, you will see SQL statements that will modify the database table accordingly.
   e. To create the database tables, select each table name that is red and then choose *Apply*. Upon successful execution of the SQL statements, the table name will turn blue.

      A confirmation message appears.

> **Note**
>
> When you choose *Apply*, you are applying all SQL statements that appear on the screen. To apply individual statements, copy them and execute them using a different SQL tool.

f. Select *Logout* at the upper-right of the page.

5.4.3 Policy Database Tables

This section lists the database tables related to policies. Attributes in the `Domain Layer` are mapped to these tables.

5.4.3.1 Account and Related Accounts

5.4.3.1.1 ACCOUNT table

The `ACCOUNT` table contains information about the accounts (customers or primary named insureds).

ACCOUNT is implemented as a flowstore master.

Class	Attribute	ColumnName	DataType	Comment
AccountData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
AccountData	isDeleted	DELETE_FLAG	VARCHAR(1)	Core Column – Soft Delete indicator.
AccountData	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
AccountData	createdBy	CREATE_USERID_TX	VARCHAR(254)	Core Column – User ID that created the record.
AccountData	createdOn	CREATE_TS	DATETIME	Core Column – Timestamp when the record was created.
AccountData	updatedBy	UPDATE_USERID_TX	VARCHAR(254)	Core Column – User ID that last updated the record.

Class	Attribute	ColumnName	DataType	Comment
AccountData	updatedOn	UPDATE_TS	DATETIME	Core Column – Time-stamp when the record was updated.
AccountData	lastName	LAST_NM	VARCHAR(30)	Last name of the ac-count – if the ac-count's an individual.
AccountData	initial	MIDDLE_IN_NM	VARCHAR(1)	Middle initial of the ac-count- if the account's an individual.
AccountData	firstName	FIRST_NM	VARCHAR2(20)	First name of the account – if the ac-count's an individual.
AccountData	title	SALUTATION_TX	VARCHAR2(4)	Salutation – if the account's an individ-ual. Customizable dropdown.
AccountData	accountName	ACCOUNT_NM	VARCHAR2(254)	Name of the account – either the commer-cial name, or 'Saluta-tion Last Name, First Name, Middle Initial'. This is set when the account is created au-tomatically.
AccountData	revenue	REVENUE_AMT	NUMBER(16)	Company annual rev-enue. Not histori-cal. Transaction data should replicate to use as a basis for rating.
AccountData	sicCode	PRIMARY_SIC_CD	VARCHAR(4)	Primary industry clas-sification of company
AccountData	emailAddress	EMAIL_TX	VARCHAR(50)	Email address of the individual.
AccountData	naicsCode	NAICS_NO	NUMBER	NAICS number. Looked up from table.
AccountData	website	WEBSITE_TX	VARCHAR(50)	Website of the organi-zation (Optional).

Class	Attribute	ColumnName	DataType	Comment
AccountData	phoneNumber	PHONE_NO	NUMBER(50)	Organization phone number.
AccountData	naicsDescription	DESC_OPER_TX	VARCHAR(300)	Description of primary operations. Populated from NAICS.
AccountData	sicDescription	PRIMARY_SIC_TX	VARCHAR(254)	Description of the SIC code.
AccountData	legalEntity	LEGAL_ENTITY_CD	VARCHAR(254)	Describes what type of legal entity owns the policy: • ANON (Anonymous) • INDIV (Individual) • CORP (Corporation) • OTHER (Other)
AccountData	businessAs	DBA_TX	VARCHAR(50)	What the policy owner prefers to be called (Doing Business As).

5.4.3.2 Policy and Policy Term

The Policy and Policy Term tables contain data about the policy and the policy term.

5.4.3.2.1 `POLICY` table

The `POLICY` table is created for each policy (the sale of a product to an account). Each policy can have one or more policy terms.

Class	Attribute	ColumnName	DataType	Comment
PolicyMasterData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
PolicyMasterData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.

Class	Attribute	ColumnName	DataType	Comment
PolicyMasterData	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
PolicyMasterData	createdBy	CREATE_USERID_TX	VARCHAR(254)	Core Column – User ID that created the record.
PolicyMasterData	createdOn	CREATE_TS	DATETIME	Core Column – Timestamp when the record was created.
PolicyMasterData	updatedBy	UPDATE_USERID_TX	VARCHAR(254)	Core Column – User ID that last updated the record.
PolicyMasterData	updatedOn	UPDATE_TS	DATETIME	Core Column – Timestamp when the record was updated.
PolicyMasterData	accountId	ACCT_ID	INTEGER	Key to the Account table REF: ACCOUNT: Read Only.
PolicyMasterData	submissionNumber	POLICY_ID	INTEGER	Same as PK_ID.
PolicyMasterData	policyActive	POLICY_ACTIVE_FG	VARCHAR(8)	Post-bind use. Set to 'N' when the policy is canceled or non-renewed.
PolicyMasterData	effectiveDate	POLICY_EFFECTIVE_DT	DATETIME	Initial effective date of the policy. Each term will have its own effective date.
PolicyMasterData	expirationDate	POLICY_EXPIRATION_DT	DATETIME	Final expiration date of the policy. Updated as each new term is created.
PolicyMasterData	producerContactId	PRDCR_REP_ID	INTEGER	Producer rep as selected in the workdesk transaction. No reference link.

Class	Attribute	ColumnName	DataType	Comment
PolicyMasterData	producerId	PRDCR_CO_ID	INTEGER	Producer as selected in the workdesk transaction. No reference link.
PolicyMasterData	renewalRequired	RENEWAL_REQ_IN	VARCHAR(1)	Indicates that a renewal is required on the policy. Typically if lead time for termination is greater than the number of days remaining on the policy.
PolicyMasterData	isFrozen	FREEZE_IN	VARCHAR(1)	Indicates that the policy has been flagged for no further transactions.
PolicyMasterData	frozenComments	FREEZE_COMMENT_TX	VARCHAR(250)	Underwriter comments.
PolicyMasterData	isAuditRequired	AUDIT_IN	VARCHAR(1)	Indicates that a policy audit is required.
PolicyMasterData	auditComments	AUDIT_COMMENTX_TX	VARCHAR(250)	Underwriter comments.
PolicyMasterData	isDoNotRenew	DNR_IN	VARCHAR(1)	Indicates that the DNR flag has been set on the policy, and the policy shouldn't be renewed.
PolicyMasterData	dnrComments	DNR_COMMENTS_TX	VARCHAR2(250)	Underwriter Comments.
PolicyMasterData	changeProducerContactId	PRDCR_REP_CBOR	NUMBER	Maps to the PK_ID of the PRODUCER_REP table. Used for change broker of record.

Class	Attribute	ColumnName	DataType	Comment
PolicyMasterData	changeProducerId	PRDCR_CBOR	NUMBER	Maps to the PK_ID of the PRODUCER table. Used for change broker of record.
PolicyMasterData	hasClaim	CLAIM_IN	VAR-CHAR(1)	Used to flag a claim against the policy.
PolicyMasterData	statusUpdatedOn	LAST_STATUS_CHG_TS	DATE	Timestamp when the status of the policy transaction was changed.
PolicyMasterData	changeProducerBy	CHANGE_PRDCR_BY	NUMBER	The ID of the user that changed the producer.

5.4.3.2.2 POLICY_PARTY_ROLE table

The POLICY_PARTY_ROLE table contains information about the relationship between a policy and parties (underwriter of record, producer, and producer contact).

Class	Attribute	ColumnName	DataType	Comment
PolicyPartyRoleData	id	PK_ID	NUMBER(16)	Core Column – Primary Identifier.
PolicyPartyRoleData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
PolicyPartyRoleData	parentId	PARENT_ID	NUMBER(16)	Core Column – Parent table Foreign Key – in this case, PK_ID of POLICY.
PolicyPartyRoleData	partyId	PARTY_ID	NUMBER(16)	A foreign key into the related party table.
PolicyPartyRoleData	effectiveDate	EFFECTIVE_DT	DATETIME	The date this party role became effective.
PolicyPartyRoleData	expirationDate	EXPIRATION_DT	DATETIME	The date this party role will expire.

Class	Attribute	ColumnName	DataType	Comment
PolicyPartyRoleData	roleType	ROLE_TYPE	VARCHAR	The role type can be one of the following: • UWOR—Underwriter of Record. • PRODUCER—Producer associated with the Policy • PCONTACT—Producer Contact associated with the Policy

5.4.3.2.3 `POLICY_TERM` table

The `POLICY_TERM` table contains details that change on a term-by-term basis (typically annually) on the policy.

Class	Attribute	ColumnName	DataType	Comment
PolicyTermData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
PolicyTermData	parentID	PARENT_ID	INTEGER	Core Column – Parent table Foreign Key – in this case, PK_ID of POLICY_TERM.
PolicyTermData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
PolicyTermData	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
PolicyTermData	createdBy	CREATE_USERID_TX	VARCHAR(254)	Core Column – User ID that created the record.
PolicyTermData	createdOn	CREATE_TS	DATETIME	Core Column – Timestamp when the record was created.

Class	Attribute	ColumnName	DataType	Comment
PolicyTermData	updatedBy	UPDATE_USERID_TX	VARCHAR(254)	Core Column – User ID that last updated the record.
PolicyTermData	updatedOn	UPDATE_TS	DATETIME	Core Column – Time-stamp when the record was updated.
PolicyTermData	effectiveDate	EFFECTIVE_DT	DATETIME	Effective date of the policy term.
PolicyTermData	expirationDate	EXPIRATION_DT	DATETIME	Expiration date of the policy term.
PolicyTermData	policyNumber	POLICY_NUMBER_TX	VARCHAR(25)	Policy number (which may change by term).
PolicyTermData	statusCode	TERM_STATUS_CODE	VARCHAR(20)	Valid status codes: • NEW (Newly created term) • INF (In force) • CAN (Cancelled)

5.4.3.2.4 `TERM_HISTORY` table

Class	Attribute	ColumnName	DataType	Comment
TermHistoryData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
TermHistoryData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
TermHistoryData	effectiveDate	EFFECTIVE_DT	DATETIME	Effective date of the policy term.
TermHistoryData	expirationDate	EXPIRATION_DT	DATETIME	Expiration date of the policy term.
TermHistoryData	policyTermId	POLICY_TERM_ID	INTEGER	ID of the current term. Will match PK_ID.

5.4.3.2.5 POLICY_ACTIVITY_LOG table

The POLICY_ACTIVITY_LOG logs all activities on a policy that aren't transactions (such as setting DNR flags, reassigning underwriters, or flagging a policy for audit).

Class	Attribute	ColumnName	DataType	Comment
PolicyActivityData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
PolicyActivityData	activityTypeCode	ACT_TYPE_CD	VARCHAR(10)	The activity being logged. Current values: • BOR – Broker (Agent) of Record change • UOR – Underwriter of record change • DNR – Policy flagged for 'do not renew' • FREEZE – Policy has been frozen from further processing • AUDIT – Policy has been flagged for audit
PolicyActivityData	activityValue	ACT_VALUE_TX	VARCHAR(50)	Indicates the new value (if applicable) set by the policy activity.
PolicyActivityData	comment	COMMENT_TX	VARCHAR(250)	User entered.
PolicyActivityData	userId	USER_ID	NUMBER(16,0)	ID of the user that set the flag.
PolicyActivityData	effectiveDate	EFFECTIVE_DT	DATE	Date the flag was set.

5.4.3.3 Policy Transaction

5.4.3.3.1 `POLICY_TXN` table

The `POLICY_TXN` table represent the transactions that can occur on a policy. As a new transaction is started, the previous transaction is copied as a starting point.

Class	ColumnName	DataType	Comment
SubmissionData	PK_ID	NUMBER(16)	Core Column – Primary Identifier.
SubmissionData	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
SubmissionData	PREV_PK_ID	NUMBER(16)	Core Column – Used for record cloning – ID of previous record.
SubmissionData	CREATE_USERID_TX	VARCHAR2(254)	Core Column – User ID that created the record.
SubmissionData	CREATE_TS	DATE	Core Column – Timestamp when the record was created.
SubmissionData	UPDATE_USERID_TX	VARCHAR2(254)	Core Column – User ID that last updated the record.
SubmissionData	UPDATE_TS	DATE	Core Column – Timestamp when the record was updated.
SubmissionData	POLICY_ID	NUMBER(16)	Identifier to the Policy this transaction is attached to.
SubmissionData	ACCT_ID	NUMBER(16)	Identifier to the ACCOUNT this transaction is attached to.
SubmissionData	PRDCR_CO_ID	NUMBER(16)	Agency/Brokerage – Foreign key to Producer table.
SubmissionData	PRDCR_REP_ID	NUMBER(16)	Agent/Broker/Individual producer – foreign key to the producer contact table.
SubmissionData	ADMITTED_IN	VARCHAR2(1)	Indicated admitted or non-admitted business. Optional Y/N.

Class	ColumnName	DataType	Comment
SubmissionData	BOUND_OPTION_ID	NUMBER(16)	Quote option selected for issuance. PK_ID of the child record.
SubmissionData	CNCL_EFFECTIVE_DT	DATE	Effective date of the cancellation – essentially the transaction effective date.
SubmissionData	CNCL_NOTICE_DAYS_NO	NUMBER(3)	Days notice required, as defined by business rules, or ODEN.
SubmissionData	CNCL_NOTICE_REQ_FG	VARCHAR2(1)	Notice required indicator. '1' or blank .
SubmissionData	CNCL_OVR_PREM_AMT	NUMBER(16)	Return premium – UW override amount, if the calculated premium isn't acceptable.
SubmissionData	CNCL_REASON_CD	VARCHAR2(15)	Reason for cancellation. The list is provided by the product and may include such values as 'non pay' and 'insured request'.
SubmissionData	CNCL_TYPE_CD	VARCHAR2(5)	Type of cancel. The values may include F – Flat, SR – Short Rate, and PR – Prorate.
SubmissionData	COMMISSION_PCT	NUMBER(4,1)	Agent commission.
SubmissionData	ENV_CD	VARCHAR2(5)	Interface processing – DEV or PROD. Can be looked up by interfaces to use stub, test, or production interfaces.
SubmissionData	INTERFACE_STATUS_CD	VARCHAR2(5)	Interface processing – status of the interface. Set by interface. Typically Good, Warning, Error.
SubmissionData	MARKET_SEG_CD	VARCHAR2(4)	Optional – can be used for market segmentation logic.
SubmissionData	MARKET_SEG_TX	VARCHAR2(250)	Optional – can be used for market segmentation logic.

Class	ColumnName	DataType	Comment
SubmissionData	NEW_ASSIGNED_TO_ID	NUMBER(16)	UW reassignment – UW about to receive the policy.
SubmissionData	PAPER_CO_TX	VARCHAR2(50)	Optional – Paper on which to write the policy.
SubmissionData	PATH_CD	VARCHAR2(10)	Used for workflow navigation.
SubmissionData	PAY_PLAN_CD	VARCHAR2(20)	Billing pay plan – Codes as defined in billing system.
SubmissionData	PAY_PLAN_DESC_TX	VARCHAR2(255)	Billing pay plan – description as defined in billing system.
SubmissionData	PAY_PLAN_PCD	VARCHAR2(40)	Billing pay plan – Internal part code.
SubmissionData	POLICY_DELIVERY_OPTION_IN	VARCHAR2(1)	Extendable – O – Online, U – US Mail. Can extend to include multiple copies, direct to customer, etc.
SubmissionData	POLICY_TERM_FACTOR	NUMBER(18,4)	The ratio of the term length to one year. This factor is used to calculate the PREM_FULL_TERM_AMT value.
SubmissionData	PREV_NAV_QUOTE_CD	VARCHAR2(20)	Used for workflow navigation.
SubmissionData	PREV_OPTION_ID	NUMBER(16)	Post-bind transactions – identifies the record that was a copy of the previous transaction for delta calculations.
SubmissionData	PREV_TXN_STATUS_CD	VARCHAR2(20)	Used for workflow navigation.
SubmissionData	PREV_TXN_TYPE_CD	VARCHAR2(30)	Used for workflow navigation.
SubmissionData	PRINT_DOC_TYPE	VARCHAR2(50)	Print interface – document type requested.
SubmissionData	PRINT_REQ_TYPE_CD	VARCHAR2(40)	Print interface – print request type.
SubmissionData	PRINT_RESPONSE_ERROR	VARCHAR2(254)	Print interface – if error, message received.
SubmissionData	PRINT_RESPONSE_URL	VARCHAR2(254)	Print interface – URL to PDF to display.

Class	ColumnName	DataType	Comment
SubmissionData	PRINT_TEMPLATE_ID	VARCHAR2(254)	Print interface – optional – if multiple templates can be used.
SubmissionData	PRINT_TEMPLATE_TYPE	VARCHAR2(254)	Print interface – optional – if multiple templates can be used.
SubmissionData	PRIORITY_CD	VARCHAR2(1)	Can be used for worklist sorting.
SubmissionData	PRODUCT_CD	VARCHAR2(250)	Product selected.
SubmissionData	PRODUCT_NAME_TX	VARCHAR2(254)	Product selected.
SubmissionData	PRODUCT_VERSION_NO	NUMBER(3)	Version of the product selected (automatically based on date).
SubmissionData	PRODUCT_XPATH_TX	VARCHAR2(254)	Path to the product.
SubmissionData	QUOTE_EXPIRATION_DT	DATE	Set by application rules – the date when the quote is no longer valid.
SubmissionData	QUOTE_ID	NUMBER(16)	Used for workflow navigation.
SubmissionData	REAPPLY_FG	VARCHAR2(1)	Used for OOS processing.
SubmissionData	REFER_UW_ID	NUMBER(16)	Management referral –the UW the policy is being reassigned to.
SubmissionData	REFERRAL_IN	VARCHAR2(1)	Indicates that a referral rule has triggered.
SubmissionData	REV_TXN_TYPE_CD	VARCHAR2(20)	Used for OOS processing.
SubmissionData	REVERSED_IN	VARCHAR2(1)	Used for OOS processing.
SubmissionData	REVERSIBLE_IN	VARCHAR2(1)	Used for OOS processing.
SubmissionData	REW_CNCL_TXN_ID	NUMBER(16)	For cancel/rewrite – during the rewrite transaction, tracks the cancellation transaction ID for finalization.
SubmissionData	REW_NEW_POL_NO_IN	VARCHAR2(1)	Indicates if the rewrite keeps the same policy number or generates a new one.
SubmissionData	SIGN_ON_USER_ID	NUMBER(16)	Foreign key to the employee table – logged on user.
SubmissionData	SIGNED_APP_IN	VARCHAR2(1)	Indicates if a signed application was received. Captured at policy issuance.
SubmissionData	SIGNED_APP_RCVD_DT	DATE	Date the signed application was received.
SubmissionData	SOURCE_CD	VARCHAR2(20)	Future use. App, electronic.

Class	ColumnName	DataType	Comment
SubmissionData	TERM_POLICY_PREM_AMT	NUMBER(16,2)	Not used.
SubmissionData	TXN_COMMENTS_TX	VARCHAR2(100)	Description of the endorsement. Set to 'New Business' automatically for NB.
SubmissionData	TXN_DECLINE_IN_CD	VARCHAR2(20)	Not used.
SubmissionData	TXN_DECLINE_REASON	VARCHAR2(255)	Declination reason, as entered by user.
SubmissionData	TXN_EFFECTIVE_DT	DATE	Effective date of the transaction defaults to policy effective date OR most recent transaction date, if later.
SubmissionData	TXN_EXPIRATION_DT	DATE	Expiration date of the transaction, defaults to term expiration date.
SubmissionData	TXN_PREM_AMT	NUMBER(16,2)	Delta premium for the transaction. Denormalized from selected quote.
SubmissionData	TXN_PROCESS_CD	VARCHAR2(20)	Used for workflow navigation.
SubmissionData	TXN_SEQ_NO	NUMBER(5)	Sequence number of the transaction.
SubmissionData	TXN_STATUS_CD	VARCHAR2(20)	State of the transaction: • CRE – Created • SUB - Submitted • DSC - Discarded • REF – Referred • QUO – Quoted • BOU – Bound • REV – Reversed • ISS – Issued • PEN – Pending Renewal • DEC – Declined
SubmissionData	TXN_TYPE_CD	VARCHAR2(30)	Type of transaction: • NEWBUS • RENEWAL • REWRITE • CANCEL • REINSTATE • RIX – Reissue • END – Endorsement
SubmissionData	UW_OF_RECORD_ID	NUMBER(16)	Underwriter of record for current and future transactions.

Class	ColumnName	DataType	Comment
SubmissionData	WORKING_USER_ID	NUMBER(16)	Current working user – broker, UW, manager – the user currently executing the transaction.
SubmissionData	LAST_STATUS_CHG_TS	DATE	Timestamp when the status of the policy transaction was changed.
SubmissionData	OOSE_IN	VARCHAR(1)	Indicates if the transaction is involved in an OOS set of transactions. Used for OOS transaction ordering.
SubmissionData	TARGET_TXN_IN	NUMBER(16)	Used for OOS transaction handling.
SubmissionData	PRINT_PREVIEW_IN	VARCHAR(1)	Selected by a user to determine if the document generation engine should flag a generated print request as draft.
SubmissionData	BATCH_PRINT_IN	VARCHAR(1)	Indicates to the document generation engine to batch print the policy.
SubmissionData	DEFAULT_COMMISSION_PCT	NUMBER(4,1)	Default commission amount, as configured for the broker or agent.
SubmissionData	PREV_TXN_EFFECTIVE_DT	DATETIME	Effective date of the previous transaction.
SubmissionData	QUOTE_DT	DATE	Stores the initial date the transaction was quoted.
SubmissionData	COUNTERSIG_NO_TX	VARCHAR2(20)	Used for the Counter Signature field on the Issue screen.
SubmissionData	PRINT_WAIT_SEC	NUMBER(5)	Customer specific column.
SubmissionData	PREV_OOSE_IN	VARCHAR2(1)	Used in OOS processing.
SubmissionData	TS_TASK_STATUS	NUMBER(3)	Customer specific column.
SubmissionData	COUNTERSIG_REQ_IN	VARCHAR2(1)	Flag on whether or not the counter signature is required.
SubmissionData	RENEWAL_LAUNCH_TYPE	VARCHAR2(50)	Used for renewal processing.
SubmissionData	TERM_HISTORY_ID	NUMBER	Used for workflow navigation.

5.4.3.3.2 TRANSACTION_PARTY_ROLE table

The `TRANSACTION_PARTY_ROLE` table contains information about the relationship between a transaction and parties (working user or previous working user).

Class	Attribute	ColumnName	DataType	Comment
TransactionPartyRole-Data	id	PK_ID	NUMBER(16)	Core Column – Primary Identifier.
TransactionPartyRole-Data	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
TransactionPartyRole-Data	parentId	PARENT_ID	NUMBER(16)	Core Column – Parent table Foreign Key – in this case, PK_ID of TRANSACTION.
TransactionPartyRole-Data	partyId	PARTY_ID	NUMBER(16)	A foreign key into the related party table.
TransactionPartyRole-Data	effectiveDate	EFFECTIVE_DT	DATETIME	The date this party role became effective.
TransactionPartyRole-Data	expirationDate	EXPIRATION_DT	DATETIME	The date this party role will expire.
TransactionPartyRole-Data	roleType	ROLE_TYPE	VARCHAR	The role type must be the following: • WU—Working User of the associated Transaction

5.4.3.4 Policy Quote

Policy Quote contains the quote-specific data for any quotes or quote options. The quote set of tables stores all the policy details, including all risks, selected coverage options, forms, and premium.

5.4.3.4.1 `POLICY_QUOTE` table

The `POLICY_QUOTE` table contains the premium and identification details for all quotes. It is also the table that the main product tables extend from to capture product-specific data.

Class	Attribute	ColumnName	Data Type	Comment
QuoteOptionData	id	PK_ID	NUMBER(16)	Core Column – Primary Identifier.

Class	Attribute	ColumnName	Data Type	Comment
QuoteOptionData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
QuoteOptionData	previousId	PREV_PK_ID	NUMBER(16)	Core Column – Used for record cloning – ID of previous record.
QuoteOptionData	createdBy	CREATE_USERID_TX	VARCHAR2(254)	Core Column – User ID that created the record.
QuoteOptionData	createdOn	CREATE_TS	DATE	Core Column – Timestamp when the record was created.
QuoteOptionData	updatedBy	UPDATE_USERID_TX	VARCHAR2(254)	Core Column – User ID that last updated the record.
QuoteOptionData	updatedOn	UPDATE_TS	DATE	Core Column – Timestamp when the record was updated.
QuoteOptionData	optionSequenceNo	OPTION_SEQ_NO	NUMBER(3)	Unique identifier of option, within quote.
QuoteOptionData	prevOptionStatus	PREV_QUOTE_STATUS	VARCHAR	Option status prior to the last change.
QuoteOptionData	effectiveDate	EFFECTIVE_DT	DATE	Transaction effective date.
QuoteOptionData	expirationDate	EXPIRATION_DT	DATE	Transaction expiration date .
QuoteOptionData	isActive	ACTIVE_IN	VARCHAR2(1)	'N' if the quote is inactive.
QuoteOptionData	annualizedCommissionAmt	ANN_COMMISSION_AMT	NUMBER(16,2)	Annualized commission, calculated as a percent of premium.
QuoteOptionData	contextDate	APP_CONTEXT_DATE	DATE	Used to look up against the product to pull the proper rates and values.

Class	Attribute	ColumnName	Data Type	Comment
QuoteOptionData	overriddenCancellationPremiumAmt	CNCL_OVR_PREM_AMT	NUMBER(16)	When cancelling, the return premium, if overridden by the underwriter.
QuoteOptionData	declineReason	DECLINE_REASON	VARCHAR2(255)	As entered by underwriter.
QuoteOptionData	changedPremiumAmt	PREM_ADD_RET_AMT	NUMBER(16)	Delta premium.
QuoteOptionData	annualizedPremiumAmt	PREM_ANNUALIZED_AMT	NUMBER(16)	Annualized premium.
QuoteOptionData	earnedPremiumAmt	PREM_EARNED_AMT	NUMBER(16,2)	Earned premium.
QuoteOptionData	previousAnnualizedPremiumAmt	PREM_PREV_ANNUALIZED_AMT	NUMBER(16)	Previous annualized premium (for delta calculation).
QuoteOptionData	termPremiumAmt	PREM_TERM_AMT	NUMBER(16)	Term premium.
QuoteOptionData	previousEarnedPremiumAmt	PREV_PREM__EARNED_AMT	NUMBER(16,2)	Previous earned premium (for earned premium calc).
QuoteOptionData	proRatedCommissionAmt	PRORATE_COMMISSION_AMT	NUMBER(16,2)	Prorated commission, for endorsement.
QuoteOptionData	prorateRatio	PRORATE_RATIO_RT	NUMBER(16,14)	Ratio of number of days of coverage for transaction to days in 1 year
QuoteOptionData	QuoteNo	QUOTE_NUMBER_TX	VARCHAR2(25)	Unique number assigned to a quote, prior to a policy being generated.
QuoteOptionData	referralAcceptedNo	REF_ACCEPTED_NO	NUMBER(3)	Referral processing – number accepted.
QuoteOptionData	referralDeclinedNo	REF_DECLINED_NO	NUMBER(3)	Referral processing – number declined.
QuoteOptionData	referralEscalatedNo	REF_ESCALATED_NO	NUMBER(3)	Referral processing – number escalated.

Class	Attribute	ColumnName	Data Type	Comment
QuoteOptionData	referralNotHandled	REF_NOT_HAN-DLED_NO	NUMBER(3)	Referral schedule – number yet to be handled.
QuoteOptionData	termRatio	TERM_RATIO_RT	NUMBER(16,14)	Ratio of number of days of coverage for full term to days in 1 year.
QuoteOptionData	sequenceId	SEQ_NO_ID	NUMBER(16,0)	Sequence number of the quote.
QuoteOptionData	quoteDate	QUOTE_DT	DATE	Date that the quote was generated.
QuoteOptionData	isTerrorismCoverage-Selected	TERRORISM_IN	VARCHAR(1)	Indicates if terrorism was globally selected for the quote.
QuoteOptionData	printRequestTypeCode	PRINT_REQ_TYPE_CD	VARCHAR(50)	Used for document generation – the type of print request.
QuoteOptionData	isShowPrintPreview	PRINT_PREVIEW_IN	VARCHAR(1)	Indicates to the document generation services if a printed document is to be marked as draft.
QuoteOptionData	terrorismCoverage-TierCode	TERRORISM_TIER_CD	VARCHAR(1)	Rating field – the tier for terrorism, if selected globally.
QuoteOptionData	terrorismCoverageTer-ritoryCode	TERRORISM_TERRI-TORY_CD	VARCHAR(5)	Rating field – the territory for terrorism.
QuoteOptionData	specialtyBrokerCode	SPE-CIALTY_BROKER_CD	VARCHAR(20)	For E&S Lines – the broker responsible for recording the specialty business.
QuoteOptionData	specialtyBrokerNum-ber	SPE-CIALTY_BROKER_NM	VARCHAR(50)	For E&S Lines – the broker responsible for recording the specialty business.

Class	Attribute	ColumnName	Data Type	Comment
QuoteOptionData	terrorismFactor	TERROR-ISM_TIER_FACTOR	NUMBER(10,3)	Rating value - tier factor for terrorism, if selected globally.
QuoteOptionData	commissionAmount	COMMISSION_AMT	NUMBER(16,2)	The amount of commission for the transaction.
QuoteOptionData	remarks	REMARKS_TX	VARCHAR(500)	Underwriter remarks, to be printed on the quote letter.
QuoteOptionData	cancellationMinimum-PremiumPercentage	CAN-CEL_MIN_PREM_PCT	NUMBER(6,3)	Minimum premium to be retained during cancellation.
QuoteOptionData	minimumPremiumPer-centage	MIN_PREM_PCT	NUMBER(6,3)	Minimum Premium.
QuoteOptionData	statusUpdatedOn	LAST_STA-TUS_CHG_TS	DATE	Timestamp when the status of the policy transaction was changed.
QuoteOptionData	primaryRiskState	TEMP_RISK_STATE_C D	VARCHAR2(100)	The default state for ratebook lookups.

5.4.3.4.2 REFERRAL table

The REFERRAL table contains all the triggered referrals and the actions taken by underwriters on those referrals.

Class	Attribute	ColumnName	DataType	Comment
ReferralData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
ReferralData	parentId	PARENT_ID	INTEGER	Core Column – Parent table Foreign Key – in this case, PK_ID of POLICY_QUOTE.
ReferralData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.

Class	Attribute	ColumnName	DataType	Comment
ReferralData	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
ReferralData	coverageCode	CVG_CD	VARCHAR(100)	Coverage for which the referral applies.
ReferralData	referralCode	REFERRAL_CD	VARCHAR(100)	Referral code, as configured in the product.
ReferralData	overrideCode	OVERRIDE_CD	VARCHAR(20)	Minimum authority which can override the referral.
ReferralData	triggerColumn	COLUMN_NAME_TX	VARCHAR(100)	Product attribute which triggered the referral.
ReferralData	isAccepted	ACCEPT_IN	VARCHAR(1)	Indicates if the referral was accepted.
ReferralData	isDeclined	DECLINE_IN	VARCHAR(1)	Indicates if the referral was declined (causing a declination of the quote).
ReferralData	isReferred	ESCALATE_IN	VARCHAR2(1)	Indicates if the referral has been referred to a higher level.
ReferralData	createdOn	REFERRAL_DT	DATE	The date the referral is triggered.
ReferralData	value	VALUE_TX	VARCHAR2(255)	The value that triggered the referral (number, which can be used in a calculation).
ReferralData	policyTxnRecordId	RECORD_ID	NUMBER	Reference to the `POLICY_TXN` table.
ReferralData	reason	REASON_TX	VARCHAR2(500)	The description for why the referral happened.
ReferralData	triggeredRiskId	RISK_ID	NUMBER	PK_ID of the risk that triggered the referral.
ReferralData	sequenceId	SEQ_NO_ID	NUMBER	Unique identifier.

Class	Attribute	ColumnName	DataType	Comment
ReferralData	triggerRiskSequenceId	RISK_OBJ_ID	NUMBER	The sequence number ID of the risk that triggered the referral.
ReferralData	triggerRiskParentId	PARENT_RISK_PK_ID	NUMBER	Reference to the parent table of the risk that triggered the referral.

5.4.3.4.3 FORM table

The `FORM` table contains all of the policy forms, notices, endorsements, and other documents that attach to the policy.

Class	Attribute	ColumnName	DataType	Comment
FormData	id	PK_ID	NUMBER(16)	Core Column – Primary Identifier
FormData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator
FormData	previousId	PREV_PK_ID	NUMBER(16)	Core Column – Used for record cloning – ID of previous record
FormData	createdBy	CREATE_USERID_TX	VARCHAR2(254)	Core Column – User ID that created the record.
FormData	createdOn	CREATE_TS	DATE	Core Column – Timestamp when the record was created
FormData	updatedBy	UPDATE_USERID_TX	VARCHAR2(254)	Core Column – User ID that last updated the record
FormData	updatedOn	UPDATE_TS	DATE	Core Column – Timestamp when the record was updated
FormData	isDeselectionAllowed	ALLOW_DESELECTION_IN	VARCHAR2(1)	Is the form mandatory, or can it be deselected

Class	Attribute	ColumnName	DataType	Comment
FormData	isMultipleAllowed	ALLOW_MULTIPLE_IN	VARCHAR2(1)	Can multiples of the form be attached?
FormData	isSelectionAllowed	ALLOW_SELEC-TION_IN	VARCHAR2(1)	Can the form be manually selected
FormData	attachType	ATTACH_SOURCE_TX	VARCHAR2(254)	How the form was attached (auto or manual)
FormData	clausName	CLAUSE_NAME	VARCHAR2(250)	Used to manage form clauses – name of clause
FormData	clausNumber	CLAUSE_NO	VARCHAR2(10)	Used to manage form clauses- number
FormData	hasContinuationPage	CONTINUA-TION_PAGE_IN	VARCHAR2(1)	Indicates if the form has a continuation page (used for some doc generation engines)
FormData	coverageCode	COVERAGE_CD	VARCHAR2(254)	Coverage the form applies to
FormData	isDeselected	DESELECTED_IN	VARCHAR2(1)	If form was deselected by the user
FormData	isDisplayOnEndorse-ment	DISPLAY_INDICATOR	CHAR(1)	'N' indicates the form doesn't display in the endorsement list on the screen
FormData	docManagementId	DOC_MGMT_ID	VARCHAR2(20)	ID of the form in the document generation engine
FormData	editionDate	EDITION_DT1	VARCHAR2(10)	Edition date of the form
FormData	effectiveDate	EFFECTIVE_DT	DATE	Effective date of the form
FormData	endSeqNumber	END_SEQ_NO	NUMBER(3)	Unique sequence number for the endorsement. Displayed on the printed endorsement

Class	Attribute	ColumnName	DataType	Comment
FormData	expirationDate	EXPIRATION_DT	DATE	Expiration date of the form (won't attach after that date)
FormData	isAutoAttached	FORM_ATTACHMENT_IN	VARCHAR2(1)	Is the form auto-attached
FormData	description	FORM_DESC	VARCHAR2(255)	Description of the form
FormData	formName	FORM_NAME	VARCHAR2(255)	Name of the form
FormData	formNumber	FORM_NUMBER	VARCHAR2(20)	Internal form number, as printed on the form
FormData	formType	FORM_TYPE	VARCHAR2(50)	Type of form (types custom defined)
FormData	formTypeCode	FORM_TYPE_CD	VARCHAR2(10)	Type of form (types custom defined)
FormData	instanceNumber	ITERATION_NO	NUMBER(16)	Iteration of the form (if multiples are allowed)
FormData	legalFormNumber	LEGAL_FORM_NUMBER	VARCHAR2(100)	Legal form number (may be different than form number)
FormData	lobCode	LOB_CD	VARCHAR2(255)	Line of business that the form applies to
FormData	minimumAuthority	MINIMUM_AUTHORITY	VARCHAR2(10)	Minimum authority level to select the form
FormData	modifiedDate	MODIFIED_DT	DATE	Date the form was modified
FormData	overriddenPremiumAmount	OVERRIDDEN_PREM_MOD_AMT	NUMBER(16,2)	Overridden premium
FormData	pcd	PCD	VARCHAR2(40)	Part code for the form
FormData	isPremiumOverriden	PREM_OVR_IN	VARCHAR2(1)	Indicates if calculated premium can be overridden
FormData	premiumAmt	PREMIUM_MOD_AMT	NUMBER(16,2)	Calculated premium amount

Class	Attribute	ColumnName	DataType	Comment
FormData	isPremiumAttached	PREMIUM_MOD_IN	VARCHAR2(1)	Indicates if form has a premium attached
FormData	isToBePrinted	RISK_ID	NUMBER(16)	PK_ID of the risk the form attached to
FormData	riskTable	RISK_TABLE_NAME_TX	VARCHAR2(50)	Type of risk the form attaches to
FormData	isScheduleRequired	SCHEDULE_INDICATOR	CHAR(1)	Does the form require a schedule to be attached
FormData	sequenceId	SEQ_NO_ID	NUMBER(16)	Unique, cross transaction sequence number
FormData	stateApplicable	STATE_APPLICABLE_TX	VARCHAR2(254)	List of states that the form applies to
FormData	hasSysVariableData	SYS_VARIABLE_DATA_IN	VARCHAR2(1)	Indicates if the form has variable data defaulted by system calc fields
FormData	hasVariableData	VARIABLE_DATA_IN	VARCHAR2(1)	Indicates if the form has variable data
FormData	comments	COMMENTS_TX	VARCHAR(250)	Optional – if any comments were attached to the form
FormData	stateCode	STATE_CD	VARCHAR(5)	Indicates if the form applies to only one state
FormData	riskObjId	RISK_OBJ_ID	NUMBER(16,0)	The unique ID of the risk to which the form applies. Only populated if not a policy level form
FormData	txnIdWhenAdded	TXN_WHEN_ADDED_PK_ID	NUMBER(16,0)	Used for post-bind. Indicates what transaction the form was added on so that an endorsement print request can print just the forms added on that transaction
FormData	manuscriptPremiunAmt	MANU_PREM_AMT	NUMBER(16,2)	Manuscript premium amount.

Class	Attribute	ColumnName	DataType	Comment
FormData	manuscriptMajorPeril-Code	MANU_MA-JOR_PERIL_CD	VARCHAR2(20)	Manuscript major peril code.
FormData	manuscriptMinor-ClassCode	MANU_MI-NOR_CLASS_CD	VARCHAR2(20)	Manuscript minor class code.
FormData	manuscriptText	MANU_TX	VARCHAR2(4000)	Stores the text of the Manuscript.
FormData	manuscriptSubline-Code	MANU_SUBLINE_CD	VARCHAR2(10)	Manuscript subline code.
FormData	manuscriptDescription	MANU_DESC_TX	VARCHAR2(100)	Manuscript description text.
FormData	manuscriptId	MANU_ID_TX	VARCHAR2(30)	Manuscript ID text.
FormData	manuscriptRiskState-Code	MANU_RISK_STATE_C D	VARCHAR2(5)	Manuscript risk state code.
FormData	manuscriptStatCode	MANU_STAT_CD	VARCHAR2(20)	Manuscript state code.
FormData	manuscriptHasFlatCh-arge	MANU_FLATCHRG_IN	VARCHAR2(5)	Manuscript flat rate charge flag.
FormData	isManuscriptValidFor-Loss	MANU_VALID_FOR_LO SS_IN	VARCHAR2(1)	Manuscript valid for loss flag.

5.4.3.4.4 FORM_VARIABLE_DATA table

The `FORM_VARIABLE_DATA` contains data that's specific to a form that isn't captured through the application process.

Class	Attribute	ColumnName	DataType	Comment
FormVariableData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator
FormVariableData	createdBy	CREATE_USERID_TX	VARCHAR(254)	Core Column – User ID that created the record.
FormVariableData	createdOn	CREATE_TS	DATETIME	Core Column – Time-stamp when the record was created
FormVariableData	updatedBy	UPDATE_USERID_TX	VARCHAR(254)	Core Column – User ID that last updated the record

Class	Attribute	ColumnName	DataType	Comment
FormVariableData	updatedOn	UPDATE_TS	DATETIME	Core Column – Time-stamp when the record was updated
FormVariableData	displayStateCode	DISPLAY_STATE_CD	VARCHAR(10)	Used to filter the ques-tion based on domicile state
FormVariableData	isDataRequired	REQUIRED_IN	VARCHAR(1)	Indicates if the data is required or optional for the form
FormVariableData	value	VALUE_TX	VARCHAR(512)	Value entered in the variable data screen
FormVariableData	isActiveStatus	ACTIVE_STATUS_IN	VARCHAR(1)	Defines if the ques-tion's active (should be shown on the screen)
FormVariableData	questionId	QUES_ID	VARCHAR2(50)	Unique key for the question
FormVariableData	dataType	QUES_DATA_TYPE	VARCHAR(20)	Text, Number, Date
FormVariableData	mandatoryQuestion	IS_QUES_MANDA-TORY	VARCHAR(5)	Indicates if the data's required or optional for the form
FormVariableData	sequenceId	SEQ_NO_ID	NUMBER(16,0)	Sequences the ques-tions on the screen
FormVariableData	questionExtraProperty	QUES_EXTRA_PROP-ERTY	VARCHAR2(250)	Used to control screen display
FormVariableData	childQuestionTrigger-Value	CHILD_QUES_TRIG-GER_VALUE	VARCHAR(5)	Used to control screen display
FormVariableData	questionOtherIn	IS_QUES_OTHER	VARCHAR2(10)	Used to control screen display
FormVariableData	questionQuesTrigger-Value	PARENT_QUES_TRIG-GER_VALUE	VARCHAR(5)	Used to control screen display
FormVariableData	questionControlStyle	QUES_CON-TROL_STYLE	VARCHAR(50)	Used to control screen display
FormVariableData	questionDisplayType	QUES_DISPLAY_TYPE	VARCHAR(20)	Used to control screen display

Class	Attribute	ColumnName	DataType	Comment
FormVariableData	questionType	QUES_TYPE	VARCHAR(10)	Used to control screen display

5.4.3.4.5 REASON_COMMENTS table

The `REASON_COMMENTS` table contains data that corresponds to the Quote Not Taken feature.

Class	Attribute	ColumnName	DataType	Comment
QuoteNotTakenData	date	QNT_DATE	DATETIME	Core Column – Primary Identifier.
QuoteNotTakenData	comment	COMMENT_TX	VARCHAR(256)	Core Column – Parent table Foreign Key – in this case, PK_ID of FORM.
QuoteNotTakenData	reason	VALUE_TX	VARCHAR(256)	Core Column – Not used in FS-QUO.

5.4.3.4.6 REPEATING_DATA_TABLE table

The `REPEATING_DATA_TABLE` contains information about the repeating data group, row, and variable data.

Class	Attribute	ColumnName	DataType	Comment
ReapeatingDataTable	id	PK_ID	INTEGER	Core Column – Primary Identifier.
ReapeatingDataTable	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
ReapeatingDataTable	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
ReapeatingDataTable	repeatingGroupName	REPEATING_GROUP_NAME	VARCHAR(40)	Name of the schedule.
ReapeatingDataTable	sequenceId	SEQ_NO_ID	NUMBER(3)	Unique identifier.

5.4.3.4.7 REPEATING_DATA_ROW table

Class	Attribute	ColumnName	DataType	Comment
RepeatingDataRow	id	PK_ID	INTEGER	Core Column – Primary Identifier
RepeatingDataRow	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator
RepeatingDataRow	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record
RepeatingDataRow	rowNumber	ROW_NUMBER	INTEGER	Sequence number of the record
RepeatingDataRow	sequenceId	SEQ_NO_ID	NUMBER(3)	Sequence number of the record

5.4.3.4.8 REPEATING_VARIABLE_DATA table

Class	Attribute	ColumnName	DataType	Comment
RepeatingVariableData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
RepeatingVariableData	parentId	PARENT_ID	INTEGER	Core Column – Parent table Foreign Key – in this case, PK_ID of REPEATING_VARIABLE_DATA.
RepeatingVariableData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
RepeatingVariableData	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
RepeatingVariableData	isRequired	REQUIRED	VARCHAR(1)	Indicates if the row is required or not.
RepeatingVariableData	questionId	QUESTION_ID	VARCHAR(255)	Unique identifier for the question.

Class	Attribute	ColumnName	DataType	Comment
RepeatingVariableData	domainCode	DOMAIN_CD	VARCHAR(10)	Text, Number, Date.
RepeatingVariableData	sequenceId	SEQ_NO_ID	NUMBER(3)	Unique identifier.
RepeatingVariableData	questionDataType	QUES_DATA_TYPE	VARCHAR(20)	Used to control display.

5.4.3.4.9 QUOTE_COVERAGE table

The QUOTE_COVERAGE table contains information about the lines of business and coverages on the quote. The data is used to calculate all premiums and do all statistical coding. Will include coverages at the lowest statistically reported level, as well as any additional fees and taxes.

Class	Attribute	ColumnName	DataType	Comment
RiskCoverageData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
RiskCoverageData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
RiskCoverageData	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
RiskCoverageData	coverageTX	COVERAGE_TX	VARCHAR(100)	Description of the coverage.
RiskCoverageData	CoverageType	COVERAGETYPE	VARCHAR(256)	Indicates if this row denotes a tax or a fee.
RiskCoverageData	CoverageCode	COVERAGE_CD	VARCHAR(100)	Coverage code.
RiskCoverageData	isDeselected	COVERAGE_QUOTE_FG	VARCHAR(1)	Indicates if the coverage was deselected for a particular quote.
RiskCoverageData	isSelected	COVERAGE_SEL_FG	VARCHAR(1)	Y or N – coverage can be deselected
RiskCoverageData	deductibleAmount	DEDUCTIBLE_AMT	NUMBER(16)	Deductible, if applies for the coverage.
RiskCoverageData	limitAmount	LIMIT_AMT	NUMBER(16)	Limit, if applies for the coverage.

Class	Attribute	ColumnName	DataType	Comment
RiskCoverageData	lobCode	LOB_CD	VARCHAR2(255)	The line of business the coverage is attached to.
RiskCoverageData	transactionId	POLICY_TXN_ID	NUMBER(16)	Same as the PK_ID – used for denormalization on related records.
RiskCoverageData	returnPremium	PREM_ADD_RET_AMT	NUMBER(16,2)	Delta premium (onset/offset).
RiskCoverageData	annualizedPremium	PREM_ANNUALIZED_AMT	NUMBER(16,2)	Annualized Premium.
RiskCoverageData	cancelReturnPremium	PREM_CANCEL_RETURN_AMT	NUMBER(16,2)	Return premium on a cancel transaction.
RiskCoverageData	earnedPremium	PREM_EARNED_AMT	NUMBER(16,2)	Earned Premium.
RiskCoverageData	offsetAmount	PREM_OFFSET_AMT	NUMBER(16,2)	Premium accounting – offset premium.
RiskCoverageData	onsetAmount	PREM_ONSET_AMT	NUMBER(16,2)	Premium accounting – onset premium.
RiskCoverageData	previousAnnualizedPremium	PREM_PREV_ANNUALIZED_AMT	NUMBER(16,2)	Annualized premium for the previous transaction.
RiskCoverageData	termPremium	PREM_TERM_AMT	NUMBER(16,2)	Term premium, accounting for all transactions.
RiskCoverageData	riskSequenceId	RISK_OBJ_ID	NUMBER(16)	Object ID of the risk that the coverage applies to.
RiskCoverageData	riskName	RISK_TABLE_TX	VARCHAR(50)	The type of risk that the coverage applies to. Physical name of the table.
RiskCoverageData	isTax	TAX_IN	VARCHAR2(1)	Indicates if the entry is a tax, surcharge or fee. Used for rollup logic. '1' indicates it is. Blank indicates the item is a true coverage.

Class	Attribute	ColumnName	DataType	Comment
RiskCoverageData	isRemoved	REMOVE_IN	VARCHAR2(1)	Indicates if the coverage has been removed (in a post-bind transaction).
RiskCoverageData	isRollup	ROLLUP_IN	VARCHAR2(1)	Indicates if the coverage is a lowest level (statistically reported) coverage, or has rolled up premium for display purposes.

5.4.3.4.10 QUOTE_ROLLUP table

The `QUOTE_ROLLUP` table contains information about the rolled up premiums for the various risks. Coverages are rolled up on the `QUOTE_COVERAGE` table.

Class	Attribute	ColumnName	DataType	Comment
PremiumRollupData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
PremiumRollupData	parentId	PARENT_ID	INTEGER	Core Column – Parent table Foreign Key – in this case, PK_ID of `POLICY_QUOTE`.
PremiumRollupData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
PremiumRollupData	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
PremiumRollupData	returnPremium	PREM_ADD_RET_AMT	NUMBER(16,2)	Delta premium (onset/offset).
PremiumRollupData	annualizedPremium	PREM_ANNUALIZED_AMT	NUMBER(16,0)	Annualized Premium.
PremiumRollupData	cancelReturnPremium	PREM_CANCEL_RETURN_AMT	NUMBER(16,2)	Return premium on a cancel transaction.
PremiumRollupData	earnedPremium	PREM_EARNED_AMT	NUMBER(16,2)	Earned Premium.

Class	Attribute	ColumnName	DataType	Comment
PremiumRollupData	offsetPremium	PREM_OFFSET_AMT	NUMBER(16,2)	Premium accounting - offset premium.
PremiumRollupData	onsetPremium	PREM_ONSET_AMT	NUMBER(16,2)	Premium accounting - onset premium.
PremiumRollupData	previousAnnualized-Premium	PREM_PREV_AN-NUALIZED_AMT	NUMBER(16,2)	Annualized premium for the previous transaction.
PremiumRollupData	termPremium	PREM_TERM_AMT	NUMBER(16,2)	Term premium, accounting for all transactions.
PremiumRollupData	riskObjectId	RISK_OBJ_ID	NUMBER(16)	Object ID of the risk that the coverage applies to.
PremiumRollupData	riskTable	RISK_TABLE_TX	VARCHAR(50)	The type of risk that the coverage applies to. Physical name of the table.
PremiumRollupData	lobCode	LOB_CD	VARCHAR(255)	The coverage group code (LOB code) that this premium rollup applies to.
PremiumRollupData	rollupType	ROLLUP_TYPE	VARCHAR(15)	The rollup type determines the categories of premiums that are being rolled up: • FULL_ROLLUP (Sum of the coverages and non premiums) • COVERAGE (Sum of the coverage premiums) • NONPREMIUM (Sum of the taxes and fees premiums)

5.4.4 Supporting Database Tables

This section lists the supporting database tables. Attributes in the Domain Layer are mapped to these tables.

5.4.4.1 ADDRESS table

The ADDRESS table is linked to any entity requiring address information.

Class	Attribute	ColumnName	DataType	Comment
AddressData	address1	ADDR1_TX	VARCHAR(64)	Mandatory – line 1 of address.
AddressData	address2	ADDR2_TX	VARCHAR(64)	Optional.
AddressData	address3	ADDR3_TX	VARCHAR(64)	Optional, unused on screens.
AddressData	address4	ADDR4_TX	VARCHAR(64)	Optional, unused on screens.
AddressData	city	CITY_TX	VARCHAR(32)	Mandatory – city.
AddressData	state	STATE_PROV_CD	VARCHAR(2)	State or province – matches to reference data. Uses North American postal standard.
AddressData	zip	POSTAL_CD	VARCHAR(20)	Zip Code (US), Postal code (Canada), as entered by the user.
AddressData	country	COUNTRY_CD	VARCHAR(5)	Country code – name stored in reference data.
AddressData	createdOn	CREATE_TS	DATETIME	Core Column – Timestamp when the record was created.
AddressData	updatedOn	UPDATE_TS	DATETIME	Core Column – Timestamp when the record was updated.

5.4.4.2 PRODUCER table

The PRODUCER table represents the brokerage or agency on the policy.

Class	Attribute Name	ColumnName	DataType	Comment
ProducerData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
ProducerData	iSDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
ProducerData	previousId	PREV_PK_ID	INTEGER	Core Column – Used for record cloning – ID of previous record.
ProducerData	createdBy	CREATE_USERID_TX	VARCHAR(254)	Core Column – User ID that created the record.
ProducerData	createdOn	CREATE_TS	DATETIME	Core Column – Timestamp when the record was created.
ProducerData	updatedBy	UPDATE_USERID_TX	VARCHAR(254)	Core Column – User ID that last updated the record.
ProducerData	updatedOn	UPDATE_TS	DATETIME	Core Column – Timestamp when the record was updated.
ProducerData	agencyId	PRDCR_CO_ID	INTEGER	Same as PK_ID.
ProducerData	agencyName	PRDCR_CO_NM	VARCHAR(254)	Brokerage or Agency name.
ProducerData	regionCode	PRDCR_REGION_CD	VARCHAR(5)	Region in which the producer works.
ProducerData	regionName	PRDCR_REGION_NM	VARCHAR(254)	Region in which the producer works.
ProducerData	agencyCode	PRDCR_CD	VARCHAR(10)	Business key - provided by the licensing system.
ProducerData	distributionOffice	PRDCR_DIST_CD	VARCHAR(3)	Distribution Office.
ProducerData	branchId	BRANCH_ID_TX	VARCHAR(10)	Branch in which the producer works.
ProducerData	agencyWebsite	WEBSITE_URL_TX	VARCHAR(1024)	Brokerage or Agency Website.

Class	Attribute Name	ColumnName	DataType	Comment
ProducerData	agencySubId	PRDCR_SUB_CODE_TX	VARCHAR(8)	Sub-producer id.
ProducerData	businessAddressId	PHYS_ADDR_ID	INTEGER	Foreign key to the address table – for the physical address.
ProducerData	mailingAddressId	MAIL_ADDR_ID	INTEGER	Foreign key to the address table- for the mailing address.
ProducerData	isMailAddressDifferent	IS_MAIL_ADDR_DIFF	CHAR(1)	'Y' indicates that a different mailing address is used.
ProducerData	billType	BILL_TYPE	VARCHAR(1)	Agency or client bill.
ProducerData	commissionPercentage	COMMISSION_PCT	NUMBER(4,1)	The agent/broker commission percentage.
ProducerData	terminationDate	EXPIRATION_DT	DATE	Expiration date of the agency.
ProducerData	terminationReason	TERMINATE_REASON	VARCHAR(1000)	User entered text to describe why an agent/broker was terminated.

5.4.4.3 PRODUCER_REP table

The PRODUCER_REP table defines the contact, agent, or broker responsible for the policy.

Class	Attribute	ColumnName	DataType	Comment
ProducerContactData	id	PK_ID	INTEGER	Core Column – Primary Identifier.
ProducerContactData	producerId	PRDCR_CO_ID	INTEGER	Foreign key to the producer table.
ProducerContactData	isDeleted	DELETE_FLAG	CHAR(1)	Core Column – Soft Delete indicator.
Individual	firstName	FIRST_NAME_TX	VARCHAR(40)	Contact First Name.

Class	Attribute	ColumnName	DataType	Comment
Individual	initial	MIDDLE_INITIAL_TX	VARCHAR(5)	Contact middle initial.
Individual	lastName	SURNAME_TX	VARCHAR(40)	Contact Last Name.
Individual	otherGivenName	OTHER_GIVEN_NM	VARCHAR(40)	Contact Other given name (if applicable).
Individual	nickName	NICK_NM	VARCHAR(40)	Contact nickname.
Individual	title	TITLE_PREFIX_TX	VARCHAR(254)	Contact title.
ProducerContactData	phoneType	PHONE_TYPE_CD	VARCHAR(10)	Office, Cell.
ProducerContactData	phoneNumber	PHONE_NUMBER_TX	VARCHAR(32)	Contact Phone number.
ProducerContactData	faxNumber	FAX_NUMBER_TX	VARCHAR(32)	Contact Fax Number.
ProducerContactData	emailAddress	EMAIL_ADDR_TX	VARCHAR(128)	Contact email address.
ProducerContactData	branchId	BRANCH_ID_TX	VARCHAR(10)	Branch at which the contact works.
ProducerContactData	isNewContact	ISNEWCONTACT	VARCHAR(1)	For navigation - indicates a new contact being created.
ProducerContactData	isCanceled	CANCEL_FG	VARCHAR(1)	Used for workflow navigation.

5.5 Product and Reference Services

Services are functions that FS-QUO applications and external application systems can call to access marketable product and reference data.

Services are defined in the Product Modeler. Each service contains a rule expression that returns a string, looks up data, or performs an operation on data. For example, an FS-QUO business service can call a product service to get product-specific data for a particular policy.

Services comply with key technical and industry standards that let you integrate product data defined in FS-PRO with any technology (for example, SOA, XML, ACORD XML, and IBM IAA).

FS-PRO provides the following two types of services:

Product Services	These services access marketable product definitions and invoke product rule and rating execution.

	For example, you call a product service in an eApp questionnaire field that returns all referrals for a marketable product.
Reference Services	These services access reference object data (static information used to populate dropdown lists and common data that's reused across marketable products). For example, you can call a reference service that returns all forms for a specified risk level.

FS-PRO provides a default set of product services and reference services.

5.5.1 User-Defined Services

You can define your own product and reference services. Typically, user-defined product services are used for implementing forms attachments, referral rules, and rating rules.

When you define a rule expression as part of a service, you need to handle or convert the data that the rule is calling for the rule to function properly. This data conversion is required for rating rules. The data handling functions in *Rule Painter* are called using stems.

For more information about creating user-defined services, see Managing your User-Defined Product Services.

5.5.2 Location of the Product Services

The product services are defined in the following location in the Product Modeler: *Product Repository* > `<company_name>` *Library* > *Product Templates* > `<company_name>` *Commercial P and C Product Template* > *Product Services* > *Service API*.

For example, the Mock Company Library provided with FS-QUO includes a a product template. The default product services for the `Mock Commercial P and C Product Template` are defined in the Service API object.

If you need to add, modify, or delete product services, update the Service API object in your company library's product template. These product services apply to all marketable products that inherit from that template. Alternatively, you can update the Service API object for a specific marketable product. Those changes apply only to that product.

> ⚠ Caution
>
> Your company library's Service API object inherits from the product Service API object in the `Commercial P and C Template Library`. Don't modify this base object because it can apply to multiple company libraries.

5.5.3 Location of the Reference Services

The reference services are defined in the following location in the Product Modeler: *Product Repository* > `<company_name>` *Library* > *Reference Objects* > `<company_name>` *Reference Object Template* > *Reference Services* > *Service API*.

For example, the Mock Company Library provided with FS-QUO includes a reference object template. The default reference services for the `Mock Reference Object Template` are defined in the Service API object.

If you need to add, modify, or delete reference services, update the Service API object in your company library's reference object template. These services apply to all reference objects that inherit from that template. Alternatively, you can update the Service API object for the countrywide reference object or a specific state reference object. Those changes apply only to that countrywide or state level.

> ⚠ Caution
>
> Your company library's Service API object inherits from the reference Service API object in the `Commercial P and C Template Library`. Don't modify this base object because it can apply to multiple company libraries.

5.5.4 Service API for the Commercial P and C Product Template

The product services for the `Commercial P and C Product Template` are defined in a Service API object in the Product Modeler. This section describes some of the frequently-used product services provided with FS-PRO.

Related Information

Location of the Product Services [page 83]

5.5.4.1 `getCoverageForms`

This topic describes the `getCoverageForms` product service.

Description

Gets all forms in a coverage for a particular risk. These are Coverage Level forms.

Return Type

DataTable

Inputs

CoverageCode

RiskName

Expected Data

Forms Library row

Sample XPath

```
//*[@modified_type="COV" and .//
*[@modified_type="IDENTIFICATION"]/Code="<Premises Operations>"]/Forms//
*[@modified_type="FORMS_LIBRARY" and ./Associated_Risk=""]
```

5.5.4.2 `getFormUsageForCoverage`

This topic describes the `getFormUsageForCoverage` product service.

Description

Retrieves a list of forms to attach to a particular coverage

Return Type

DataTable

Inputs

CoverageGroupCode

CoverageCode

Expected Data

A list of form numbers

5.5.4.3 `getFormUsageForPolicyRisk`

This topic describes the `getFormUsageForPolicyRisk` product service.

Description

Retrieves a list of forms to attach to the policy risk

Return Type

DataTable

Inputs

None

Expected Data

A list of form numbers

5.5.4.4 `getFormUsageForRisk`

This topic describes the `getFormUsageForRisk` product service.

Description

Retrieves a list of forms to attach to a particular risk

Return Type

DataTable

Inputs

CoverageGroupCode

Expected Data

A list of form numbers

5.5.4.5 `runCoverageGroupRating`

This topic describes the `runCoverageGroupRating` product service.

Description

Invokes the rating calculation for a specific coverage group.

The expected rule name is `runCoverageGroupRating`.

Return Type

String

Inputs

CoverageGroupCode

5.5.5 Service API for the `Commercial P and C Reference Object Template`

The reference services for the `Commercial P and C Reference Object Template` are defined in a Service API object in the Product Modeler. This section describes the reference services provided with FS-PRO.

5.5.5.1 `getFormEditions`

This topic describes the `getFormEditions` reference service.

Description

Retrieves the form editions available for a particular form

Return Type

DataTable

Inputs

Form PCD

Expected Data

Edition date

Effective date

Expiry date

5.5.5.2 `getForms`

This topic describes the `getFormEditions` reference service.

Description

Retrieves the Form details for a particular risk level (POLICY_QUOTE, LOCATION, and so on)

Return Type

DataTable

Inputs

AssociatedRisk

Expected Data

Form data (form name, form number, attachment indicator and rule, selection indicator and rule, and so on)

5.5.5.3 `getRepeatingData`

This topic describes the `getRepeatingData` reference service.

Description

Retrieves the repeating data for a form, if configured

Return Type

DataTable

Inputs

Question PK_ID

Expected Data

Column headers and types for a repeating grid

5.5.5.4 `getVariableData`

This topic describes the `getVariableData` reference service.

Description

Retrieves the variable data for a form, if configured

Return Type

DataTable

Inputs

Edition PCD

Expected Data

Variable data information (question ID, question type, question text, and so on)

5.6 Using Stems

In FS-QUO, a stem is a short name that represents a Java class. These classes contain related functions that you can use in rule expressions in the Product Modeler.

Context

For example, the `com.sap.fs.quo.service.impl.stem.DomainBasedQuoteOptionPremiumStem` class (`quoteOption` stem) contains an `updateCoverageAnnualizedPremium` function. You can define a rule that calls that function to update the Domain Layer with the annualized premium amount for a specified coverage.

> **Note**
>
> With any rule, but particularly with rating rules, you must handle or convert the data that the rule returns for the rule to function properly. The data handling functions in *Rule Painter* are called using stems.

You can use stems to specify a function in a rule expression.

Procedure

1. Define a rule in *Rule Painter*.
2. Perform one of the following actions in the *Expression Editor*:
   - Enter the name of the stem that contains the function you want to use.
   - Choose  and select the stem from the *Stem List* dialog.

   The stem name is added as the first element (the rule object) in the expression.
3. Enter a period (.) at then end of the stem name.
   The *Functions List* dialog opens and displays the list of functions for the specified stem.
4. Select the function that you want use.
   The function signature is added to the expression.
5. Complete the rule expression as appropriate.

Next Steps

Whenever a stem is specified, the `ServiceCommand` is passed to the rule. For the rule to be executable, you must specify a reference to a valid URS (Updatable Result Set) for the key MasterResultSet and ResultSet.

5.6.1 Stem Classes and Functions

This topic lists the stems provided in FS-QUO.

Stem Name	Source
AccountStem	com.camilion.as.stem.AccountStem
DomainBasedQuoteOptionStem	com.sap.fs.quo.service.impl.stem.DomainBasedQuoteOptionStem
Flowstore	com.camilion.custom.stems.FlowStore
FormsValidation	com.sap.fs.quo.service.impl.stem.DomainBasedFormsValidation
QUERYTOTABLE	com.camilion.custom.stems.QueryToDataTable
quoteOption	com.sap.fs.quo.service.impl.stem.DomainBasedQuoteOptionPremiumStem
rating	com.sap.fs.quo.service.impl.stem.DomainBasedRatingStem
ratingrollup	com.sap.fs.quo.service.impl.stem.DomainBasedRollupStem
ASReference	com.sap.fs.quo.service.impl.stem.DomainBasedReferenceUtilStem
Risk	com.camilion.as.stem.ASRiskStem
userStemNew	com.sap.fs.quo.service.impl.stem.UserStem
util	com.camilion.custom.util.UtilLibrary

Notes:

- Any stem that isn't listed in this topic is reserved for internal use by SAP or is deprecated.
- For more information about the stem classes and their functions and arguments, see the FS-QUO javadoc.
- In addition to the FS-QUO stems, FS-PRO provides system stems that you can use (for example, `form.`, `data.`, `step.`, `PRODUCT.`, and `ProductService`). For more information about system stems, see Rule Objects.

5.6.2 Accessing the javadoc

You can view the FS-QUO javadoc to see details about the stem classes and their functions and arguments. This javadoc contains information about the FS-QUO API.

Context

For details about the business service classes and methods, see the FS-QUO javadoc. This javadoc contains information about the FS-QUO API. For information about business services, see the following package: `com.sap.fs.quo.service.api`

Procedure

1. Contact your FS-QUO administrator to get the FS-QUO software download ZIP file.

   The ZIP file is available to customers in the SAP Software Download Center under *PRODUCT AND QUOTATION MGMT*.
2. From the software ZIP file, find the following file and unzip it: FS-QUO`-javadoc-releasenumber.zip`

   The files are extracted.
3. Open `index.html` in a browser.

Results

The javadoc is displayed.

Related Information

Stem Classes and Functions [page 92]

5.6.3 Extending Stem Implementation Classes and Stem Classes

This topic details how to extend a stem implementation class and stem class to create a stem that performs tasks specific to your organization.

Context

> ⚠ Caution
>
> For extensibility, don't modify an existing stem class to add methods. Instead, to add methods, add a stem class.

Stems consist of two types of classes:

Stem implementation class	These implementation classes extend the `StemProxy` class. They contain the logic that performs tasks. These classes reside in FS-QUO. Each implementation class corresponds to a stem class.
Stem class	These classes extend the `FunctionLibraryBase` class. Each class acts as an interface to a corresponding stem implementation class. For *Rule Painter* to use these class signatures, you must upload and register these classes in FS-PRO.

Procedure

1. Find the stem implementation class that you want to extend from the following location:
   `com.sap.fs.quo.service.impl/com/sap/fs/quo/service/impl/stem`
   For example, `DomainBasedQuoteOptionStem.java`.
2. Extend the stem implementation class and add the functions you need.
3. Save the file.
4. Find the stem class that corresponds to the implementation class from the following location:
   `com.sap.fs.quo.service.impl/com/sap/fs/quo/service/impl/stem`
   For example, `DomainBasedQuoteOptionStemImpl.java`.
5. Extend the stem class and add the signatures that correspond to the extended implementation class.
6. Save the file.
7. Package the extended classes in a JAR file (for example, `custom_stems.jar`) and upload it to the server.
8. Modify the Java classpath to load your custom JAR file before the FS-QUO JAR file.

Results

The stem implementation class and stem class are extended.

5.6.4 Testing the Extended Stem Classes in FS-QUO

You can test your extended stem classes in FS-QUO during development.

Context

> **Note**
>
> After testing, remove these extended stem classes from the FS-QUO folder. For production, you will package the extension files and upload them to the FS-PRO Administrative Console.

Procedure

1. Log in to the FS-QUO Administrative Console.
2. Go to *System* > *Edit File System*.
   The *PA - Custom Web Root (Write)* tree appears.
3. Select *AuthoritySuite - Custom Class Folder (Write)* from the *Path* dropdown list in the navigation pane.
   The *AuthoritySuite - Custom Class Folder (Write)* tree appears.
4. Create folders that match the package name of your stem classes:
   a. Right-click in the main pane and select *New Folder*.
      The *Folder Name* dialog opens.
   b. Enter `com`, and choose *OK*.
      The `com` library is created.
   c. Repeat steps a-b to create folders required to establish the following path:
      `com.sap.fs.quo.service.impl/com/sap/fs/quo/service/impl`
5. Upload your new stem class:
   a. Right-click in the main pane and select *Upload*.
      The *Upload Files* dialog opens.
   b. Browse for the stem definition and stem class that you created.
   c. Choose *Upload*.
      The stem definition and class are uploaded and displayed.
6. Navigate to the `impl` folder and upload your implementation file:
   a. Right-click in the main pane and select *Upload*

The *Upload Files* dialog opens.

b. Browse for the implementation file that you created.

c. Choose *Upload*.

The implementation file is uploaded and displayed.

7. Restart the FS-QUO server.

Results

The stems are uploaded to FS-QUO and can be used by rules.

Related Information

Uploading New Stem Classes to FS-PRO [page 96]

5.6.5 Uploading New Stem Classes to FS-PRO

For FS-PRO to access the new stem class that you extended, you must upload it to the FS-PRO Administrative Console.

Context

> → Remember
>
> Don't upload the stem implementation class.

Procedure

1. Launch Internet Explorer and log in to the Design Time Administrative Console at the following location: `<pro_designtime_app_url>/csiroot/admin/`.

   The Design Time Administrative Console will open after a short delay.

2. Go to *System* > *Edit File System*.

   The *PA - Custom Web Root (Write)* tree appears.

3. Select *PS - Custom Class* from the *Path* dropdown list in the navigation pane.

   The *PS - Custom Class* tree appears.

4. Expand the tree and navigate to the `com.sap.fs.quo.service.impl/com/sap/fs/quo/service/impl` folder where the stem classes were created.

   If this location doesn't exist, create the appropriate folders by performing the following steps:

   a. Right-click in the main pane and select *New Folder*.

      The *Folder Name* dialog opens.

   b. Enter **`com`** as the folder name.

   c. Choose *OK*.

      The `com` library is created.

   d. Repeat steps a-b to create folders for the following path:
      `com.sap.fs.quo.service.impl/com/sap/fs/quo/service/impl`

5. Upload your new stem class, as follows:

   a. Right-click in the main pane and select *Upload*.

      The *Upload Files* dialog opens.

   b. Browse for the stem definition and stem class that you created.

      Don't select the implementation class.

   c. Choose *Upload*.

Results

The new stem class is uploaded and displayed.

Next Steps

If you make changes to the extended stem classes later, you must re-package the extended classes in a JAR file (for example, `custom_stems.jar`), upload the JAR file to the server, and upload the stem class to FS-PRO again.

5.6.6 Registering Stems in the FS-PRO Function Library

The Function Library is a registry of stems in FS-PRO. You must register the new stem class in the Function Library so that the *Rule Painter* can access it.

Procedure

1. Launch Internet Explorer and log in to the Design Time Administrative Console at the following location: `<pro_designtime_app_url>/csiroot/admin/`.

   The Design Time Administrative Console will open after a short delay.

2. Choose *System* > *Register Function Library* > .
   The *Library Manager* appears.
3. Complete the following fields in the *Register a New Function Library* section:

Class Name	Specifies the fully qualified name of the stem class that you uploaded to the *PS - Custom Class*. Example: `com.sap.fs.quo.service.impl.ExtendedStemName`
Library Name	Specifies the name of the library.
Library Description	Defines a short description of the stem.
Stem Name	Defines the name of the stem. Use a naming convention that distinguishes your stem name. This name is used in rules. We recommend that you specify a name that corresponds to the Java class name (for example, `<yourProjectName>`_factorLookup). This helps with maintaining consistency and locating the code. Don't use synonyms because they might cause name classes.
Stem Synonyms	Defines synonyms for the stem name. You can enter a synonym in *Rule Painter* to access the same stem. This field is optional.

4. Choose *Register*.
   The stem class is registered in the Function Library. A new row appears in the Current Function Libraries table containing the stem information.
5. In the row for the new stem class, choose *Validate*.
   The validation result indicates that the stem appears to be valid.

5.7 Selecting a Ratebook

The ratebook selection feature lets you define a particular set of rates that apply to policies written on or after a certain date using the appropriate rates, rules, and forms (for a specific state) for a defined effective period.

When upgrading or installing FS-QUO, manual configurations related to ratebook selection are required.

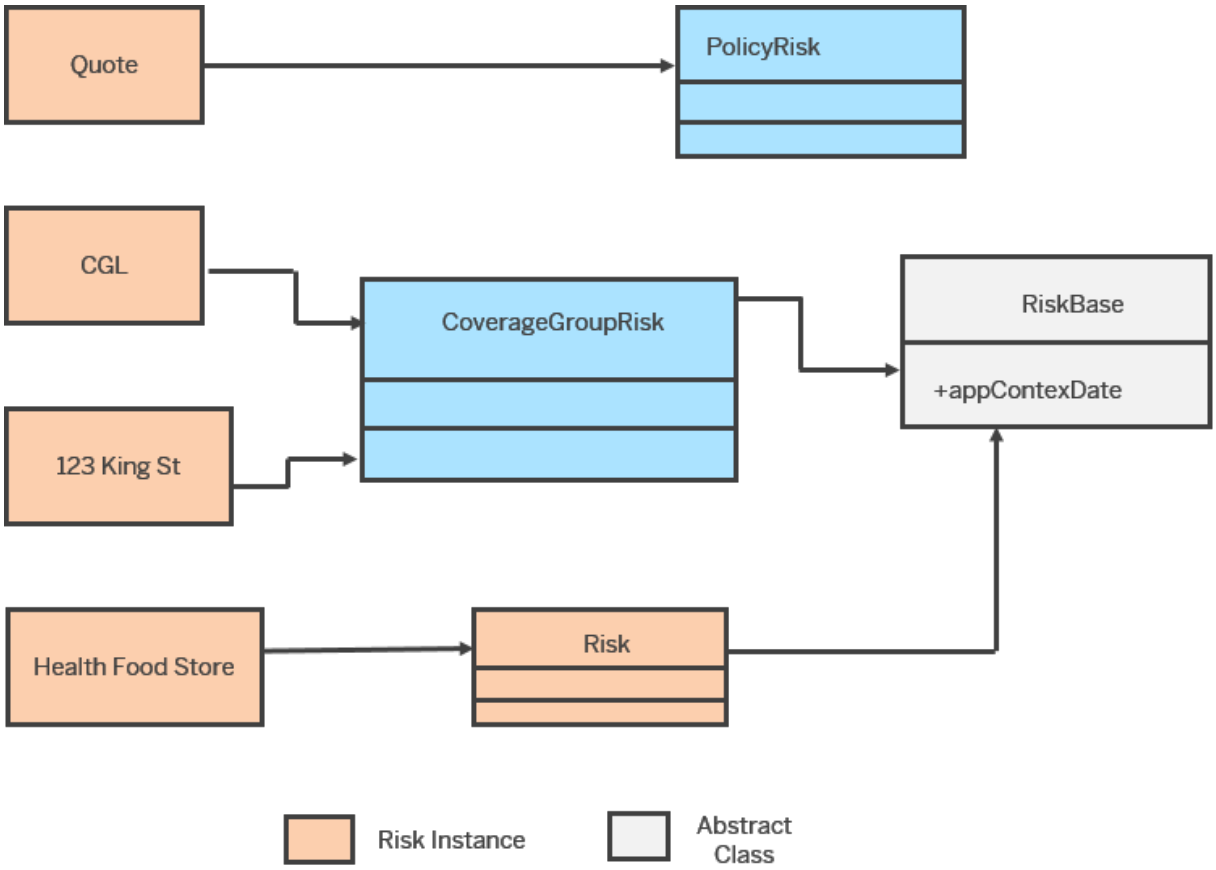
During an upgrade, the following updates are required, whether you use ratebook selection or not:

- Updates to your custom flowstore risk tables. These changes are performed as part of the process described in the *SAP Underwriting for Insurance Support Package Upgrade Guide*.
- Updates to the `Risk Hierarchy` component, main eApp, and Ratebook eApp for each marketable product. If you are using multiple ratebooks, you must also configure the `Reference Object Link` component.

Class diagram example

The illustration provided are based on the ISO GL product and a risk structure of: Policy_Quote: Location and Classification. The structure of your product determines the number of rows required when configuring the `Risk Hierarchy` component.

There is only one physical data model for a given line of business and it's used across multiple versions. The application context date is an attribute of all risks. The following diagram provides an illustration:



> **Note**
>
> The `currentRefVersion` and `latestRefVersion` attributes are part of all the risks. The system populates the `currentRefVersion` and `latestRefVersion` attributes (by running the reference rule) for the risk after the context date is populated.

5.7.1 Configuring Ratebook Selection

You must make configuration changes in the Product Modeler to support ratebook selection.

Context

> **Note**
>
> Verify that the corresponding flowstore risk tables were updated for the risks you are working with.

Procedure

1. Create ratebooks by adding rows to a reference object in the Product Modeler. For each ratebook, specify the *Effective Date* and *Expiration Date*.

   For more information about configuring the reference object, see Understanding Reference Objects and Reference Config Objects.

   This step is only required if you are using multiple ratebooks.
2. Configure `Application_Context_Date_Rule` for each risk.
3. Configure `Reference_Object_Version_Rule` for each risk.
4. Configure `Inherit_Parent_Reference_Indicator` for each risk.
5. Configure the `Reference Object Link` component

   This step is only required if you are using multiple ratebooks.
6. Configure the main eApp.
7. Configure the Ratebook eApp.
8. Assign permissions related to Ratebooks.
9. Publish the product.
10. Restart the Product Modeler server.

5.7.1.1 Configuring `Application_Context_Date_Rule` for Each Risk

For each coverage group in a marketable product, define an application context date rule for each risk.

Context

This rule returns a date representing the context date for the associated risks. For example, in the Class diagram example, 123 King Street is an associated risk for the risk type, LOCATION.

Procedure

1. Log in to the Product Modeler.
2. Open the following object:  `<product_name>` > *Coverage Groups* > `<coverage_group_name>` > *Identification* > *Risk Hierarchy*  .

   This component defines the risk structure of the coverage group in your product.
3. In the *Values* tab, add rows based on the risk structure of the coverage group in your product.
4. For each row (risk), perform the following steps:
   a. In the `Application_Context_Date_Rule` attribute, add a context rule.

      Create the rule to meet your company requirements. This rule represents the context date that's associated with the risk.
   b. Complete additional attributes as required.
   c. Select the checkbox to select the row.
5. Save your changes.
6. Publish the product.
7. Restart the server.

Example: Context rule

The following is an example of a context rule (in pseudo code):

```
If transaction type = new business || cancel/rewrite || endorsement
    Returns effective date of the term
Else if transaction type = renewal
        Returns effective date of the renewal term
End
```

5.7.1.2 Configuring `Reference_Object_Version_Rule` for Each Risk

For each coverage group in a marketable product, define a rule for each risk that returns a data table representing the latest version of the state reference object available for the associated risks.

Context

> **Note**
>
> The following procedure must be completed for all risks in your structure.

Procedure

1. Log in to the Product Modeler.
2. Open the following object: ▶ `<product_name>` ▶ *Coverage Groups* ▶ `<coverage_group_name>` ▶ *Identification* ▶ *Risk Hierarchy* ▶.

   This component defines the risk structure of the coverage group in your product.
3. In the *Values* tab, for each row (risk), do the following:
   a. In the `Reference_Object_Version_Rule` attribute, add a rule that requests the latest version of the state reference object.

      Create the rule to meet your company requirements.
   b. Complete additional attributes as required.
   c. Verify that the checkbox for the row is selected.
4. Save and commit the changes.

Example: Reference rule

The following is an example of a reference rule (in pseudo code):

```
If context date for the associated risk is not null
    If state code for the associated risk (by running the state code rule) is
not null
    Returns latest version with context date, state code, LOB and transaction
type as inputs to the Reference Object Link component
    End
End
```

5.7.1.3 Configuring the `Reference Object Link` Component

Context

If you are using multiple ratebooks, create entries for new ratebook dates by setting the transaction type in the `Reference Object Link` component. Add an individual row in the `Reference Object` Link component for each transaction type and version number. If you are using a single ratebook, you can skip this task.

Notes:

- If you are adding a new row to the `Reference Object Link` component, you must attach (anchor) the row to the `Reference Object Line` component.

- If you are only modifying an existing row, you don't need to reattach it to the `Reference Object Line` component.
- Before you delete a row from the `Reference Object Link` component, you must first detach (deselect the checkbox) the row from the `Reference Object Line` component.
- In your marketable product, check that `AppConfigProduct` is set to `<reference_config>` in the following component: `<product_name>` > *Configuration* > *Config*.

Procedure

1. Log in to the Product Modeler.
2. Open your `Reference Config` object.

   This object is usually assembled in the following location: `<company library>` > *Configuration Objects* > `<reference config>`.
3. Open the following object: `<reference config>` > *Reference Config* > *Reference Object Line* > *Reference Object Link*.
4. In the *Values* tab, add a row for each *State Code*, *Transaction Type*, and *Version Number*, and complete the values.

   You might need multiple *Transaction Types* such as New Business, Endorsement, Renewal, and so on. In addition, some transaction types might have more than one row with different versions as shown in example in this topic.
5. Select the checkbox for the added rows.
6. Save and commit the changes.
7. Anchor the new rows to the `Reference Object Line` component.

Example: Reference Object Link configuration

An example of configuring the `Reference Object Link` component is illustrated in the following table:

PCD	State Code	Transaction Type	Reference Object Name	Version Number	Effective Date	Expiration Date
1699554	OH	NEWBUS	GL REFERENCE OH	1	01-01-2011	01-01-2012
16995545	OH	RENEWAL	GL REFERENCE OH	1	03-01-2011	03-01-2012
1699556	OH	NEWBUS	GL REFERENCE OH	2	01-01-2012	01-01-2013

5.7.1.4 Configuring the Main eApp

Context

Depending on your company implementation, it is the product analyst's responsibility to add or change any validation rules for a field or question in the eApp to enforce any required restrictions.

For example, a field or question may not contain a validation rule because its input was previously restricted by the UI or a dropdown list. As another example, a field or question may need a validation rule to display or hide depending on the version number.

Notes

- For more information on configuring objects, see Defining Objects.
- For more information on adding rules, see Working with Scripting Rules

5.7.1.5 Configuring the Ratebook eApp

Context

You can configure the provided Ratebook eApp as part of your main eApp. The Ratebook eApp shows a tab that lists all the risks in the policy transaction that have multiple ratebooks. Roles that have the `updateAndRetrieveReferenceInfo` permission can select which version of the ratebook to use.

For more information on configuring objects, see Defining Objects

Procedure

1. Log in to the Product Modeler.
2. Open the following object: `<product_name>` > *Data Capture* > *eApp Layout* > *eApp Layout Section*.
3. Add a row to the object, complete the following required fields, and save your changes:

   Section Name

   Defines the name to appear in the eApp (for example, Ratebook).

   Target eApp

   Specifies Ratebook eApp.

   This value matches the name of the eApp defined in the following object: `<product_name>` > *Data Capture* > *eApplication* > *Ratebook eApp*.

Section Order

Specifies the order to display the section or tab.

4. If you only want roles that have the `updateAndRetrieveReferenceInfo` permission to see ratebook versions in the *Ratebook* tab, add a rule in the *Section Display Rule* field to only show ratebook versions to these roles.
5. Open the following object: `<product_name>` > *Data Capture* > *eApp Layout* .
6. Expand the row where *eApp Name* is Application (your main application).

   The `eApp Layout Section` component appears.
7. Select the row you created for the Ratebook eApp.
8. Save and commit the changes.

5.7.2 Example of Ratebook Selection Configuration

To help you with configuring ratebook selection, see the `Mock Commercial General Liability Monoline` example product provided out of the box in FS-PRO.

> **Note**
>
> To see `Mock Commercial General Liability Monoline`, you must have imported the `Mock GL Underwriting Bootstrap` during the installation process.

5.7.2.1 Viewing the Risk Hierarchy Rules in the Mock Product Example

To help you see how risk hierarchy rules are configured, you can view the sample rules in the `Mock Commercial General Liability Monoline` product.

Context

The ratebook selection configuration involves defining rules for the following attributes in FS-PRO:

- `Application_Context_Date_Rule`
- `Reference_Object_Version_Rule`
- `Inherit_Parent_Reference_Indicator`

Procedure

1. Log in to the Product Modeler.
2. Open the following object: *Mock Company Library* > *Marketable Products* > *General Liability*.
3. Open the following object: *Mock Commercial General Liability Monoline* > *Coverage Groups* > *Mock Commercial General Liability* > *Identification* > *Risk Hierarchy*.

   The *Values* tab shows three rows based on the risk structure for the `Mock Commercial General Liability` coverage group.
4. For a row (risk) in the *Values* tab, find the following attributes, right-click the  Rule icon and select *View*:
   - Application_Context_Date_Rule
   - Reference_Object_Version_Rule
   - Inherit_Parent_Reference_Indicator

   The *Rule Painter* dialog appears, showing the rule.

5.7.2.2 Viewing the Reference Object Link in the Mock Product Example

To help you see how the `Reference Object Link` object is configured for multiple ratebooks, you can view the example component in the `Mock Commercial General Liability Monoline` product.

Procedure

1. Log in to the Product Modeler.
2. Enter **`Mock Reference Config`** in the *Search* panel.
3. Choose *Search*.
4. Double-click *Mock Reference Config*.
5. Expand the *Reference Config* folder.
6. Expand the *Reference Object* Line.
7. Select *Reference Object Link*.
8. Select the *Values* tab.

Results

The configuration for `Mock Commercial General Liability Monoline` appears. In this example a separate row has been created for each state. If there is more than one version, a separate row is created for each version number for each state. In this example, the *Transaction Type* is shown as ALL. If separate

transaction types are required, then a separate row is added for each transaction type and version for each state.

5.8 Additional Configuration Procedures

5.8.1 Configuring Forms Attachment

When forms are defined for a line of business in a Form Library, FS-QUO looks for these forms when the *TransactionType* column in the `Form` object is populated with specified values.

Depending on the configuration, the forms are automatically attached to the policy, or the forms are displayed so that you can attach them to or detach them from the policy.

For more information, see Mapping and Publishing your Forms.

5.8.2 Configuring Tax Rules

To calculate tax premiums, FS-QUO runs rules that you define in FS-PRO. The `CalcTaxPremiumService` then calculates the annual transaction and term premium.

Context

> → Remember
>
> In your marketable products, if the tax premium calculation rules are already defined in ▶ `<product_name>` ▶ *Coverage Groups* ▶ `<coverage_group_name>` ▶ *Rating* ▶ *Rating Rule* ◀, you must move the rules to the `Premium Handling` component indicated above. If you don't move the rules, the tax premiums will be calculated incorrectly.

Procedure

1. Log in to the Product Modeler.
2. Open the following component: ▶ `<product_name>` ▶ *Coverage Groups* ▶ `<coverage_group_name>` ▶ *Taxes* ▶ `<tax_type>` ▶ *Rating* ▶ *Premium Handling* ◀
3. Override the Calculation Rule, and define the following in the rule:
   a. Calculate the tax premiums.
   b. Call the `DomainBasedQuoteOptionStem..updateCoverageAnnualizedPremium` stem.

c. Call the `DomainBasedQuoteOptionStem..updateTransactionPremium` stem.

4. Save your changes.
5. Publish the product.

Results

The tax rule is configured for the product.

5.8.3 Configuring Validation for Flat Cancellation Type

Context

By default, FS-QUO is configured to prevent a policy that's in `In Force` status from being canceled in the following situation:

- The specified *Transaction Effective Date* for cancellation is the same as the policy effective date.
- The specified *Cancellation Type* isn't Flat.

A message appears indicating that you must select the Flat cancellation type when the dates are the same. You can change this configuration to let users specify any cancellation type when the dates are the same.

Procedure

1. Log in to the Product Modeler.
2. Open your extended `Underwriting Application Configuration` object.

   Typically, the object is named `Extended Underwriting Application Configuration` and is located in the `<Company Library>` > *Configuration Objects* folder.
3. Expand the *Screen Framework* folder, and choose *Screen*.
4. In the *Values* tab, expand the *Screens_Transaction_Cancel_EnterDate_CancelTransactionScreenContent* row.
5. In the anchored `Screen Control` component, expand the *cancelTransactionInfoForm.cancelEffectiveDate* row.
6. In the anchored *Business Questions* tab, deselect the *isCancelEffectiveDateWithinPolicyTermForFlatCancel* row.
7. Save and commit the changes.
8. Publish the extended `Underwriting Application Configuration` object.

Results

The business question isn't used for validation any longer. When the cancellation and policy effective dates are the same, you can now select any cancellation type.

5.8.4 Referrals Configuration Note

When referrals are configured, they are required to have a table and column name that contains the value that triggered the referral. This is used for display purposes and to determine if the value of a referral has changed from when it was initially accepted.

The column that's used for the table name was changed from *SYSATTR_TABLE_NAME* to *Process Table Id*. Previously, the rule had to work on the same table that contained the value column. This is no longer required. An exception will be thrown if the table.column (Process Table Id.Field Name) doesn't exist in the flowstore.

6 Unsupported Features

The following features are not supported in SAP Quotation and Underwriting for Insurance, even if they are visible in the user interface or the directory structures.

APIs:

- `importFlowletGroup`
- `importFlowletGroup`
- `buildAndSyncProcessFlowlets`
- `exportFlowletGroup`
- `buildProcessFlowlets`

Administrative Console

- Uploading custom stems to the Runtime Administrative Console

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP