Developer Guide | PUBLIC
2019-09-20

Coverage-based Development Guide

SAP Quotation and Underwriting for Insurance 1.1

THE BEST RUN SAP

Content

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, Spring Web Flow 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, SWF, Business Services, and Data Model.
	Quotation Web Application	Module (Feature)	Back-end for Quotation Fiori applications. Includes implementation for OData, SWF, and Business Services

Item		Type	Description
	Underwriting Web Application	Module (Feature)	Back-end for Underwriting Fiori applications. Includes implementation for OData, SWF, 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)

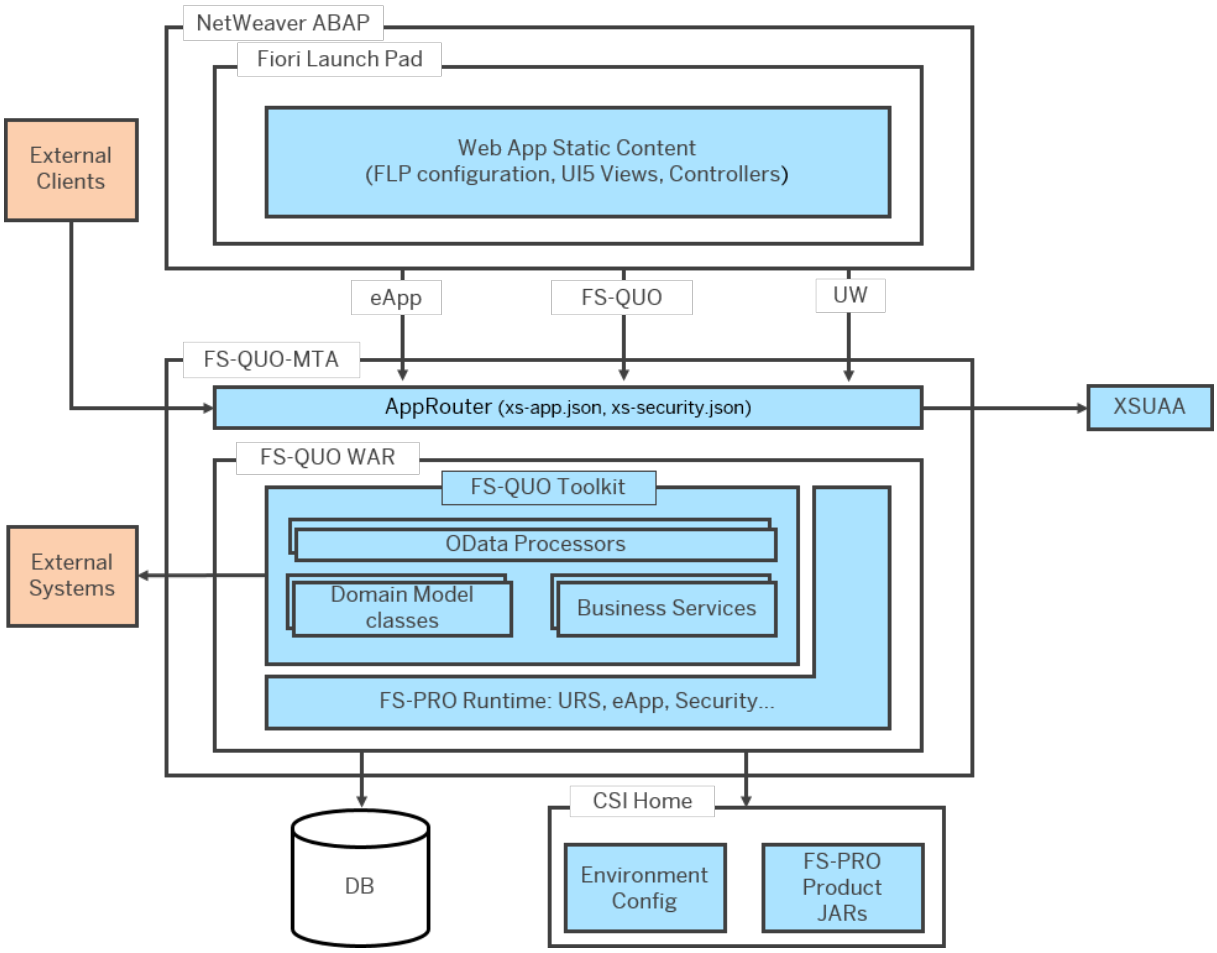
View	FS-PRO	FS-QUO	FS-IPW

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:

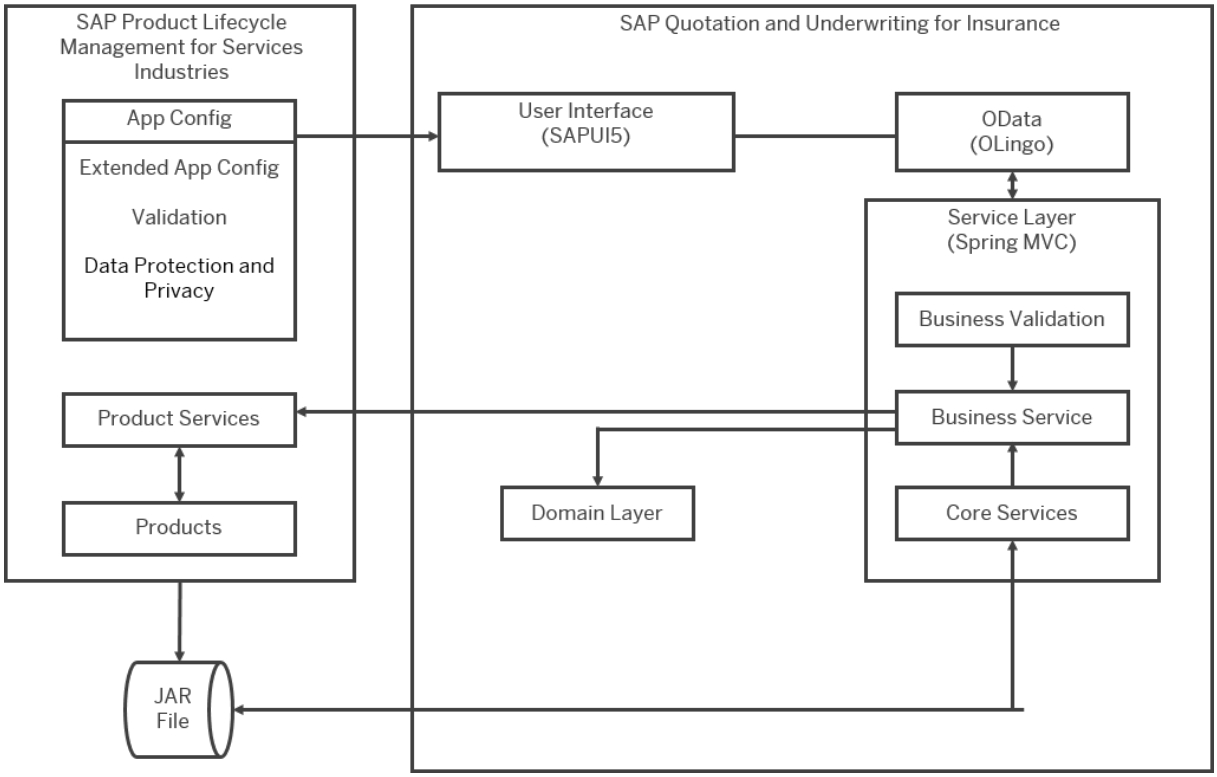
- Underwriting Application
- Permissions Framework
- Validation Framework

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.
- The process configuration XML files in the flow layer define how FS-QUO runs, the flow of the application, and how the screens transition to one another.
- 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:

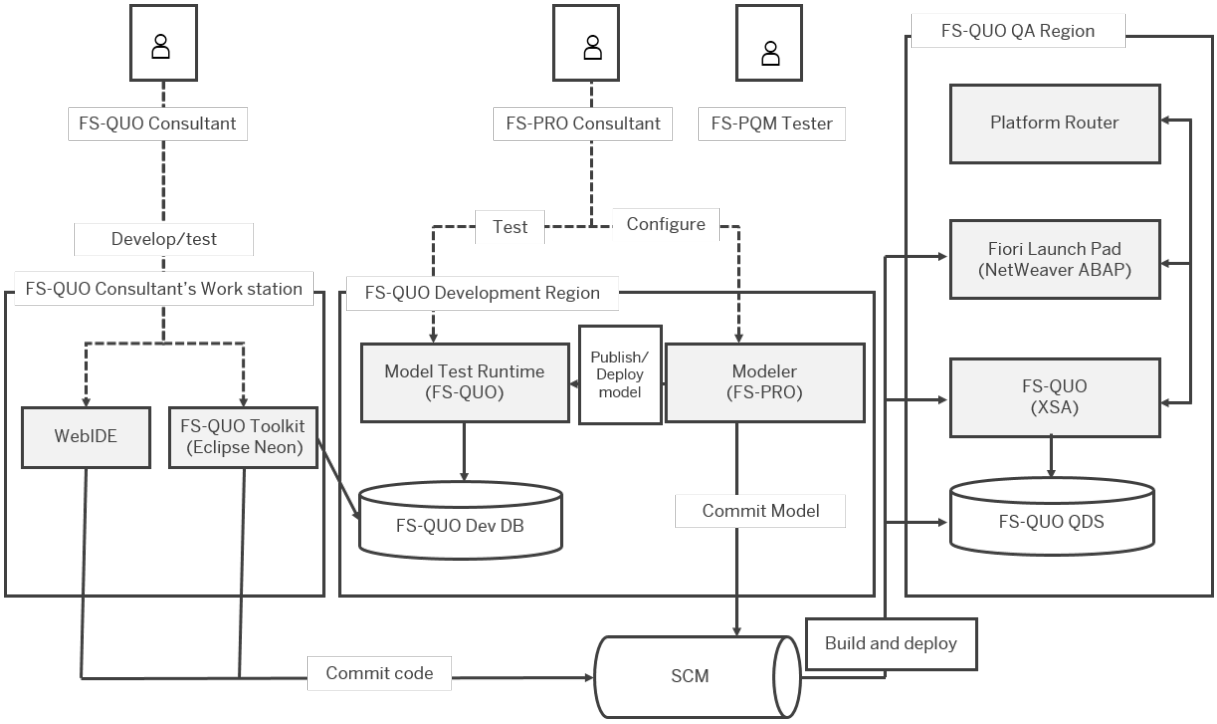
- Any configurability for a feature will be described as part of that feature's documentation (as opposed to being covered under Extensibility).
- The Toolkit includes all source code for the FS-QUO implementation, available for you to analyze or customize.
- During an upgrade of FS-QUO, you are responsible 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.

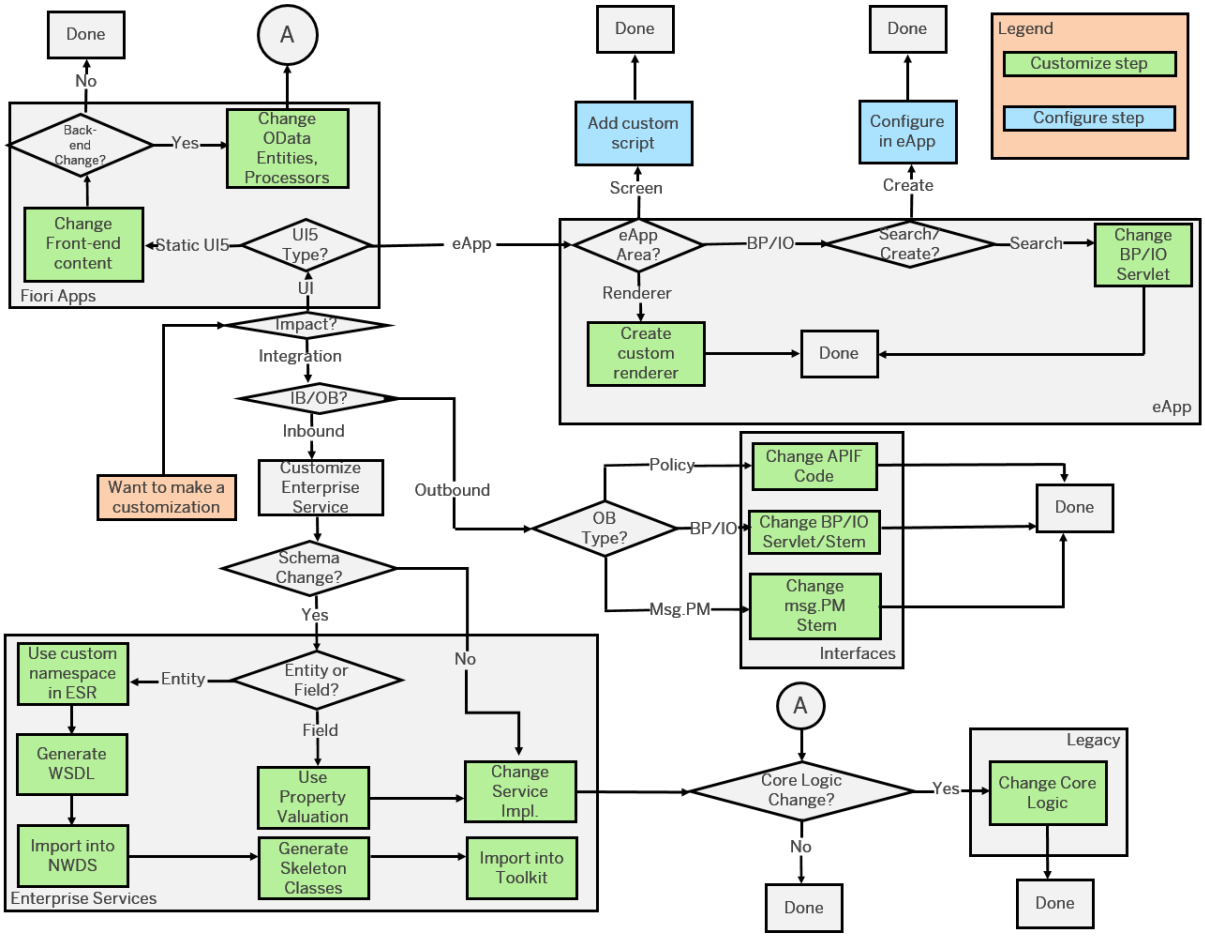
3.1 Understanding Design Time

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 `QUOODataLibrary` project, open class `com.sap.fs.quo.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.odata.model.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: Modifying the UW Case Screen in the UW Fiori App

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

Procedure

1. Create a new template:
   a. Create the viewer and controller for your new template under the path ./view/pool/common. In this case, create a new template which called SAPMAP.

      > ⓘ Note
      >
      > If you use controls included in a special library, you should specify the library in `component.js`.

   b. If you create a common template, define the validation rule for its own data binding.
2. Create a new block:
   a. Create the viewer and controller for your block.

      > ⓘ Note
      >
      > This code was copied from `violation.controller.js` and modified.

   b. Specify the templates in the block.
   c. Specify the data binding for the templates in the block.
3. Configure the block into a section:

a. Add the block as a child node under the sections list.

> **Note**
>
> If you use a router, configure the new section into your routers.

b. Your new module displays in the detail information page.

3.3.3 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`.

3.3.4 Example: Modifying the *BP Search* UI

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

Context

The purpose of this example is to change the title of the *BP Search* popup from *Business Partner Search* to *Customized Business Partner Search - this is a customization*.

> **Note**
>
> BP Search back-end logic isn't covered because it's using the same servlet as in FS-QUO.

Procedure

1. In the `FS-QUO-install` project (CSI_HOME contents), open the `BPSearchUtil.js` file and replace the following line: `this._oAccountDialog = sap.ui.xmlfragment(that.getView().getId(), "eapp.screen.controller.fragment.BPSearch", this);` with the following text: `this._oAccountDialog = sap.ui.xmlfragment(that.getView().getId(), "com.customer.fragment.CustomBPSearch", this);`
2. Corresponding to the previous step, create a folder named `/src/main/resources/ppms/app/as/custom/web/com/customer/fragment` and copy the `/src/main/resources/ppms/app/as/custom/web/eapp/screen/controller/fragment/BPSearch.fragment.xml`file and paste it as `CustomBPSearch.fragment.xml` in the new folder.
3. Open the `CustomBPSearch.fragment.xml` file and replace the `title="{i18n>BP_SEARCH}"` line with `title="{i18n>BP_SEARCH} - this is a customization"`
4. Update the `i18n_en_US.properties` file.
   a. Go to the `eapp/screen/controller/i18n` folder
   b. Open the `i18n_en_US.properties` file.
   c. Replace the following line: `BP_SEARCH=Business Partner Search` with line: `BP_SEARCH=Customized Business Partner Search`
5. Update the `i18n_en.properties` file.
   a. Go to the `eapp/screen/controller/i18n` folder
   b. Open the `i18n_en.properties` file.
   c. Replace the following line: `BP_SEARCH=Business Partner Search` with line: `BP_SEARCH=Customized Business Partner Search`

6. Build the toolkit.
7. Test the change in runtime:
    a. Start a new quote and go into the eApp.
    b. Open a BP search. For example, search for Policyholder.
    c. Confirm that the title of the popup is *Customized Business Partner Search - this is a customization*.

3.3.5 Example: Modifying the msg.PM Rating Stem

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

Procedure

1. Do a baseline rating test:
    a. Using the sample request XML, call the `laRatingWS` Product Web Services service on the `Whole Life` product.
    b. Note that the value in the `INSAMOUNTABS_AM` element (the Sum Insured amount) is 53996.99.
2. In the ASLibrary project, open the `MsgRating` class.
    a. Modify the rating.

        For example, alter `PREMAFTERTAX_AM` by changing it to a static amount in the `Rating` method.
3. Build the toolkit.
4. Update your environment with the latest `ASLibrary.jar`, which is produced when you built the Toolkit in step 3.
5. Re-run the rating test.
    a. Using the sample request XML, call the `laRatingWS` Product Web Services service on the `Whole Life` product.

Results

Even though the request is the same, the premium value has changed according to the modification in `MsgRating`.

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

Business services are defined in the `asServices.xml` file. Every service component implements a set of methods that are specific to that component. Each method has a return type of `ServiceResponse`. `ServiceResponse` is a generic type, which is typed by the service method signature to a specific type. The following types are allowed:

- Java types (List, String, Date)
- Primitives (int, boolean, long)
- Domain Layer Objects

> **Note**
>
> Business Service methods can't call other service methods within the same bean. If you call a method on the same bean, AOP (aspect-oriented programming) aspects won't be applied. If you need to call a method on the same bean, refactor that method so that the part you need to call is in a private method. Both methods on the bean can call that private method.

4.1.2.1 Extending Spring Configurations

You can extend and override the beans from the default out of the box configuration, for example, to override the `searchService` bean by a new custom service bean called `newService`.

Procedure

1. Using the Administrative Console, place a copy of `context-manifest.xml` in the `META-INF` directory under the custom class directory.

   > **Note**
   >
   > This path and file name can't be changed or the code won't be able to find it.

2. Within the `context-manifest.xml` file, list the customized XML files containing your overridden beans.

   The file should look like the following example:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
     <context-manifest>
       <context location="com/camilion/as/beans/runtime/
   mockCompanyServices.xml" />
   ```

```
  </context-manifest>
```

3. Using the Administrative Console, upload the `mockCompanyServices.xml` file to the `com/camilion/as/beans/runtime` directory in the custom class folder.
4. This path and file name can be changed, but this matches the way that AS defines these files.
5. Ensure that the customized file does not have the same file name as any of the existing files (for example, `asServices.xml`, `asHandlers.xml`). This prevents the system from finding the copy of the file.
6. In the customized file (`mockCompanyServices.xml`), define any new beans required or overwrite any beans to be changed.

   Spring only allows one bean with the same id, and will use the last instance found, which will be the one under the custom class folder.

   For example, in `mockCompanyServices.xml`:

```
<?xml version="1.0"?>
  <beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:lang="http://www.springframework.org/schema/lang"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
               http://www.springframework.org/schema/beans/spring-
beans-2.5.xsd
               http://www.springframework.org/schema/lang
               http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop-2.5.xsd"
       default-lazy-init="true">
  <bean id="searchService"
class="com.camilion.as.service.business.search.MockCompanySearchServiceImpl"
parent="businessServiceComponent">
    <constructor-arg index="0"><bean factory-bean="databaseTypeProvider"
factory-method="getDatabaseType"></bean></constructor-arg>
    <constructor-arg index="1" ref="PA-AppEnvironment"/>
    <property name="lookupService" ref="lookupService" />
  </bean>

  <bean id="newService" class="com.mockcompany.service.NewServiceImpl"
parent="businessServiceComponent">
  </bean>
</beans>
```

   You must copy the `<beans>` tag from the original XML file being overridden, as it contains information required by Spring.

4.1.2.2 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.

4.1.2.3 Business Service Class Definition

Each `Service` component is implemented as an interface and an implementation class, which implements the interface.

Interface

All `Business Service Component` interfaces extend the `BusinessService` interface. The following is an example of a `Business Service Component` interface:

```
public interface AccountService extends BusinessService
{
    public List<Account> search (AccountSearchCriteria criteria);
    public List<Account> getAccountListForPkIdList (List<Long> list);
    public ArrayList<Long> pkIdListSearch (AccountSearchCriteria c);
    public AccountTransferObject retrieve(Long accoutPkId);
}
```

Implementation Class

The implementation class for business service components extends the `BusinessComponentBase` class. The following is an example of an Implementation Class:

```
public class AccountServiceImpl extends BusinessComponentBase implements
AccountService
{
    public AccountServiceImpl()
    {
    }
    private AccountCoreService accountCore;
    public void setAccountCore (AccountCoreService coreService)
    {
        this.accountCore = coreService;
    }
    public ArrayList<Long> pkIdListSearch (AccountSearchCriteria criteria)
    {
        return ...;
    }
```

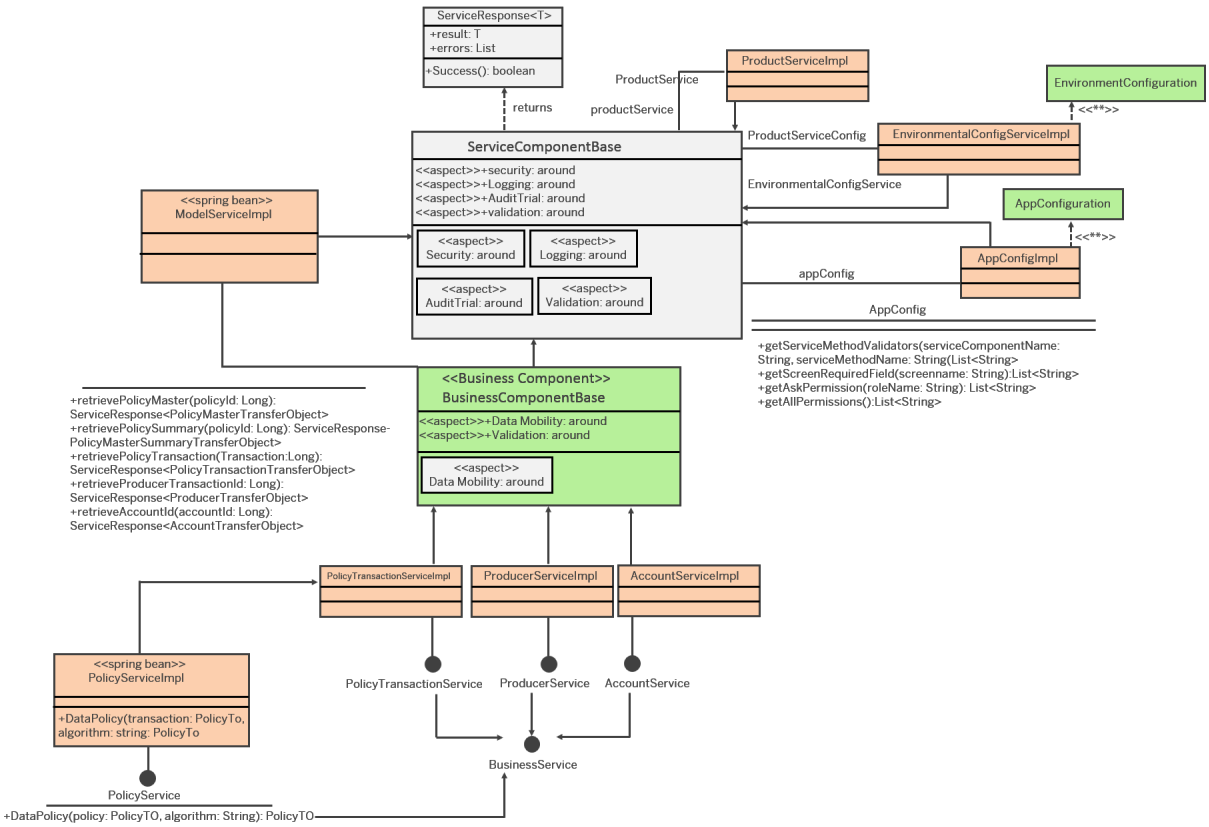
```
    public List<Account> getAccountListForPkIdList (List<Long> list)
    {
        return ...;
    }

    public List<Account> search (AccountSearchCriteria criteria)
    {
        return ...;
    }

    public AccountTransferObject retrieve(Long accoutPkId)
    {
            return ...;
    }
    public void update(AccountTransferObject ato)
    {
        return ...;
    }
}
```

4.1.2.4 Service Component Class Diagram

The following diagram shows the basic structure of the `Business Service Components`. All `Business Service Components` extend the `BusinessServiceComponent` base class. This base class provides wiring for the `AppConfig` bean and the `ModelService`.



4.1.2.5 The `asServices.xml` file

The `asServices.xml` file defines all the FS-QUO service java classes available in the Spring MVC architecture. For example, the `accountService` bean is defined as an available service in the `asServices.xml` file. If the UI needs to invoke a method within an Account Service, you can specify the `accountService` bean as a property in the corresponding handler entry in the `asHandlers.xml` file. This specification injects the `accountService` bean into the handler bean. By injecting the `accountService` bean, the handler bean can use the business service to handle the logic triggered by the UI event.

The `asServices.xml` file is located in the following directory: `/ASlibrary/src/com/camilion/as/beans/runtime/asServices.xml`

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`

4.1.3.2 Core Services Class Definition

Each `Service` component is implemented as an interface and an implementation class, which implements the interface.

Interface

All `Core Service` component interfaces extend the `CoreService` interface. The following is an example of a `Core Service` component interface:

```
public interface AccountCoreService extends CoreService
{
    public AccountTransferObject retrieve(Long accoutPkId);
public void update(AccountTransferObject ato);
}
```

Implementation class

The implementation class for core service components extends the `ServiceComponentBase` class. The following is an example of an `Implementation` Class:

```
public class AccountCoreServiceImpl extends ServiceComponentBase implements
AccountCoreService
{
    public AccountServiceImpl()
    {
    }
    public AccountTransferObject retrieve(Long accoutPkId)
    {
            return ...;
    }
    public void update(AccountTransferObject ato)
    {
    }
}
```

4.1.4 Extending a Business Service

If you want to add or modify FS-QUO functionality to work specifically with your environment, you can extend existing business services.

Context

For example, the Producer Service includes a `setDefaultCommission()` method. This method sets the commission percentage in the Policy Transaction given a commission amount. Suppose that your organization has its own algorithm of determining the default commissions, and you want to set the default percentage elsewhere in theDomain Layer. You can extend the Producer Service and override the `setDefaultCommission()` method to perform your own logic.

Procedure

1. Create a new class that extends the parent service class.
2. Override the methods as needed.
3. Optionally, you can add new methods to the extended interface and implementation.

   > **Note**
   >
   > Call the new methods from the extended classes (for example, extended UI handlers).

4. Add the following bean XML to the `asServices.xml` file:

```
<bean id="AS-<serviceComponentName>Service" parent="businessServiceComponent"
class="<fullyQualifiedClassName>" >
        <property name="<additionalServiceInjections>"
        ref="<serviceId>" />
</bean>
```

5. Optionally, you can add validation extensions using `AppConfig`.

   The validation extensions are added. These new extensions don't impact existing code.

6. Build, deploy, and restart the server.

Results

The service bean configuration in `asServices.xml` uses the new class. The additional methods don't impact how other services use this service.

4.1.5 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.

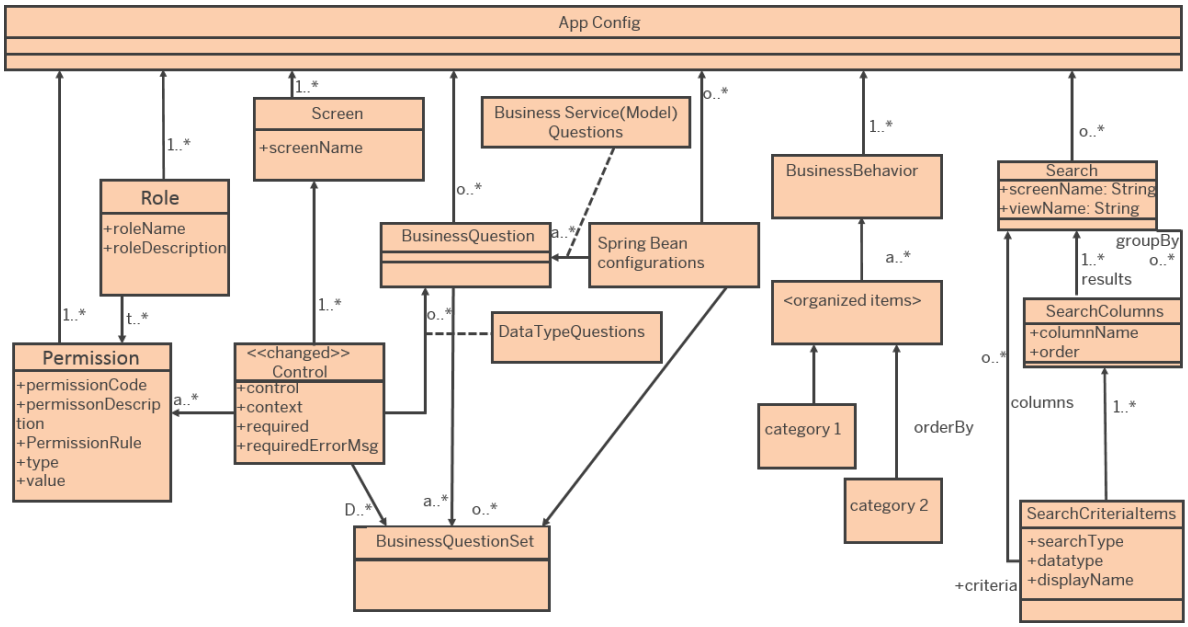
The application configuration is accessed through the `Underwriting Application Configuration` service bean. Any Business Service that requires configuration elements use the `Underwriting Application Configuration` bean to retrieve its configuration from the `Underwriting Application Configuration`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

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

- `Role Framework` folder
- `Screen Framework` folder
- `Search Framework` folder
- `Validation` folder
- `Data Privacy` folder

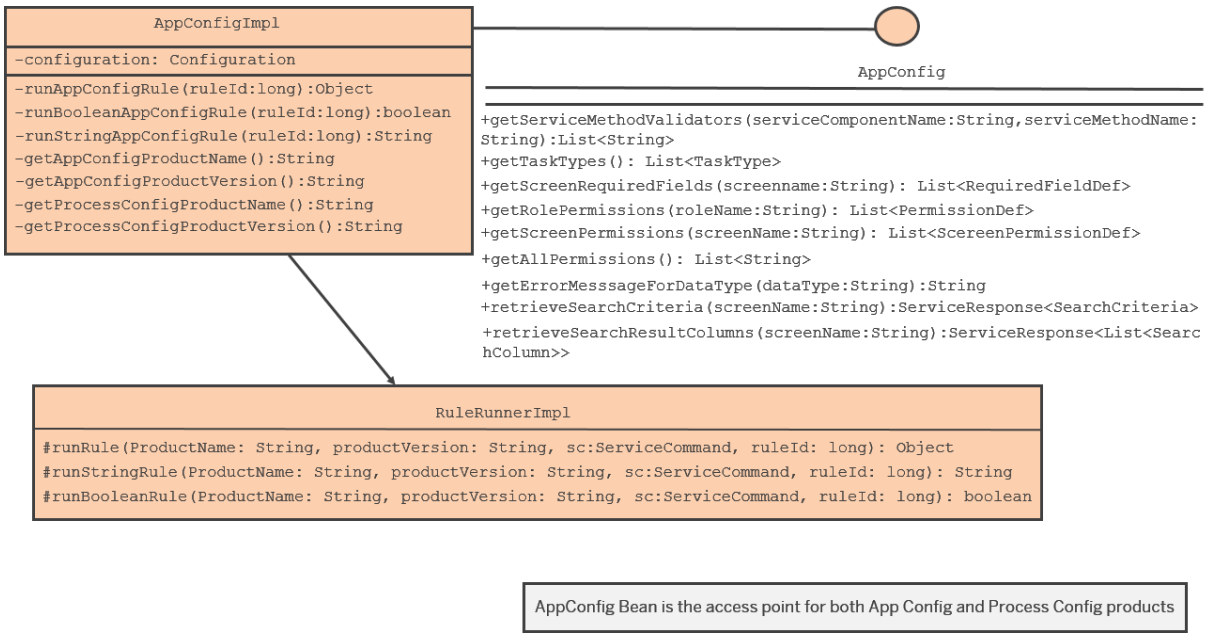
The `Underwriting Application Configuration` object contains all the information that is used to configure the FS-QUO application. For example, these settings determine how search results are displayed and the number of days before a quote expires.

The following diagram represents the logical organization of the `Underwriting Application Configuration` object:



4.2.1 AppConfig Service Bean

The following diagram shows the `AppConfig Service` bean definition. Configure every Business Service to inject the `AppConfig` bean.



AppConfig Bean is the access point for both App Config and Process Config products

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.2.5 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.
`Role Framework`	Contains objects that define roles and the permissions associated with those roles.
`Screen Framework`	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).
`Search Framework`	Contains objects that configure how searches are performed and how the results are displayed.
`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.

4.2.5.1 `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.

4.2.5.1.1 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.

4.2.5.2 `Configuration` Folder

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

4.2.5.2.1 Lookup Service Configuration object

The `Lookup Service Configuration` object is used to access external methods to populate controls (usually dropdown lists) in the UI.

The `Lookup Service Configuration` object has the following attributes:

lookupName
: Specifies the look up name in the code.

lookupServiceMethodName
: Specifies the Business Service method.

beanId
: Specifies the name of the `Service` component.

  Enter the value configured in the `beanId` attribute located in *Spring Bean Definitions* > *Spring Bean Configurations* > *Validation*.

4.2.5.3 `Business Behavior` Folder

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

4.2.5.3.1 `Task Management` Folder

The `Task Management` folder contains objects that configure task behavior.

4.2.5.3.1.1 `Task Management Config` Object

The `Task Management Config` object configures the behavior of tasks.

The `Task Management Config` object has the following attributes:

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

propertyName
: Specifies the property name called by the code.

4.2.5.3.1.2 Task Type List Object

The `Task Type List` object contains the content that's displayed in the *Task Type* dropdown list in the *Manage Task* and *Add Task* windows.

The `Task Type List` object has the following attributes:

taskName

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

This value is displayed in the *Task Type* dropdown list.

taskDesc

This attribute isn't used.

4.2.5.3.2 `Transaction Management` Folder

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

4.2.5.3.2.1 `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.

4.2.5.3.2.2 `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.

4.2.5.3.3 `Clearance` Folder

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

4.2.5.3.3.1 `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.

4.2.5.3.4 `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.

4.2.5.3.5 `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.

4.2.5.3.6 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.

4.2.5.4 `Reference` Folder

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

4.2.5.4.1 `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.

4.2.5.4.2 `Forms` Object

The *TransactionType* column in the `Forms` object specifies the types of forms to display on the screen for forms attachment.

4.2.5.5 `Error Messages` Folder

The `Error Messages` folder contains the `Error Messages` component, which contains error messages that are displayed when the data type isn't what the `Domain Layer` expects. The data type validation occurs when a screen is saved by clicking the *Save* button or clicking the next activity button.

4.2.5.5.1 `Error Messages` Object

The `Error Messages` object contains the data types that the `Domain Layer` checks for and the messages that are displayed when the data type is incorrect.

The `Error Messages` object has the following attributes:

messageKey
: Specifies the data type that's being validated.

errorMessage
: Defines the error message that's displayed when a data type is wrong.

4.2.5.6 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.

4.2.5.7 Debug Console Folder

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

4.2.5.7.1 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*.

4.2.5.7.2 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.

4.2.5.7.3 `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.

4.2.5.7.4 `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.

4.2.5.7.5 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 Coverage-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.

5.1.1 Extending Views

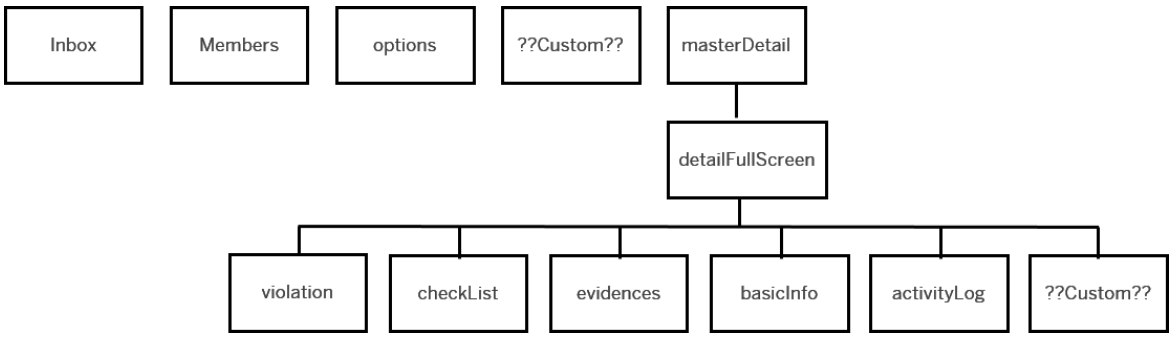
Extension points are inserted in the views of the delivered standard applications, making it much easier to insert custom content to the custom application.

5.1.1.1 Adding New Routers

Before adding any custom routers, you must clear the routing of the main application.

For example, assuming that we want to perform the following actions:

- Add a new router "custom" into the routing of Underwriting
- Add another new router "custom" as a sub-router of `underwritingDetailFullScreenPage`.



Here is how to add a router at the first level:

```
customizing: {
    "routerExtend": [
        {viewId: "customWeb",
         pattern : "custom",
         name : "custom",
         view :"fs.ipw.custom.app.custom",
         targetAggregation : "pages",
         viewLevel: 1,
         targetControl: "app"
        }
]}
```

Here is how to add a sub-router to "detailFullScreen":

```
customizing: {
    "routerExtend": [
        {viewId: "customTab",
         parent: "detailFullScreen",
         pattern : "{previousPage}/StatusCode/{StatusCode}/caseId/{caseId}/
uwItemId/{uwItemId}/customTab",
         name : "customTabScreen",
         view :"fs.ipw.custom.app.custom",
         targetAggregation : "content",
         viewLevel: 3,
         targetControl: "iconTabBarBlock--customTab"
        }]
```

The `routeExtend` has the following properties:

Name	Type	Note
ViewId	String	View id of router
parent	String	Parent router name
Name	String	Router name
View	String	View path
targetControl	String	Views will be put into a container Control (it may be a control, or when working with mobile, any other container). The ID of this control must be placed here.
targetAggregation	String	The name of an aggregation of the targetContro that contains views. For example, `m.app` has an aggregation "pages". Another example is that the layout has "content".
viewLevel	Int	The router level

5.1.1.2 Adding New Views and Fragments

The delivered standard `My Underwriting Worklist` application provides the following view extension points

View	Point Name	Parent	Type
fs.ipw.myuwwork-list.view.frame.inbox	inboxFilterBarExtension	smartFIl-terBar	sap.ui.comp.filterbar.FilterI-tem
	inboxTableHeaderBarExtension	tool-Bar(ta-ble)	sap.m.Button
	inboxFooterBarExtension	tool-Bar(page)	sap.m.Button
	inboxActionSheetExtension	action-Sheet	sap.m.Button

View	Point Name	Parent	Type
fs.ipw.myuwwork-list.view.frame.member	membersObjectHeaderAttrExtension	objec-tHeader	sap.m.ObjectAttribute
	membersTableHeaderBarExtension	tool-Bar(ta-ble)	sap.m.Button
	membersFooterBarExtension	tool-Bar(page)	sap.m.Button
	membersActionSheetExtension	action-Sheet	sap.m.Button
fs.ipw.myuwwork-list.view.frame.options	optionsObjectHeaderAttrExtension	objec-tHeader	sap.m.ObjectAttribute
	optionsTableHeaderBarExtension	tool-Bar(ta-ble)	sap.m.Button
	optionsFooterBarExtension	tool-bar(page)	sap.m.Button
	optionsActionSheetExtension	action-Sheet	sap.m.Button
fs.ipw.myuwwork-list.view.frame.masterDetail	masterDetailNavBelowSearchExtension	sub-Header(page)	sap.ui.core
fs.ipw.myuwwork-list.view.frame.detail-FullScreen	detailFullScreenObjectHeaderExtension	objec-tHeader	sap.m.ObjectAttribute
	masterDetailActionSheetExtension	action-Sheet	sap.m.Button
fs.ipw.myuwwork-list.view.sections.violation	masterDetailActionSheetExtension	tool-bar(ta-ble)	sap.m.Button
	violationTabContentAboveExtension	layout	sap.ui.core
	violationTabContentBelowExtension	layout	sap.ui.core
fs.ipw.myuwwork-list.view.sections.evidences	evidencesTableHeaderBarExtension	tool-bar(ta-ble)	sap.m.Button
	evidencesTabContentAboveExtension	layout	sap.ui.core
	evidencesTabContentBelowExtension	layout	sap.ui.core

View	Point Name	Parent	Type
fs.ipw.myuwwork-list.view.sections.checkList	checkListTableHeaderBarExtension	tool-bar(ta-ble)	sap.m.Button
	checkListTabContentAboveExtension	layout	sap.ui.core
	checkListTabContentBelowExtension	layout	sap.ui.core
fs.ipw.myuwwork-list.view.sections.basicInfo-Tab	basicInfoTabTabContentAboveExtension	layout	sap.ui.core
	basicInfoTabTabContentBelowExtension	layout	sap.ui.core
fs.ipw.myuwwork-list.view.sections.activityLog	activityLogTableHeaderBarExtension	tool-bar(ta-ble)	sap.m.Button
	activityLogTabTabContentAboveExtension	layout	sap.ui.core
	activityLogTabTabContentBelowExtension	layout	sap.ui.core

The delivered standard `Create Group Insurance Quote` application provides the following view extension point:

View Name	Screen Name	Point Name	Parent	Type
fs.ipw.grpinsquote.cre-ate.view.generalInfo	General Info	generalInfoGroupTea-mAndProductExten-sion	formContainer	sap.ui.layout.form.For-mContainer
		generalInfoFooterBar-Extension	toolBar(page)	sap.m.Button
fs.ipw.grpinsquote.cre-ate.view.generalInfo-Summary	General Info Summary	generalInfoSummar-yGroupTeamAndPro-ductExtension	formContainer	sap.ui.layout.form.For-mContainer
		generalInfoSummary-FooterActionSheetEx-tension	actionSheet	sap.m.Button
		generalInfoSummary-FooterBarExtension	toolBar(page)	sap.m.Button
fs.ipw.grpinsquote.cre-ate.view.managePlan	Plan Management	planManagementOb-jectHeaderExtension	objectHeader	sap.m.ObjectAttribute
		planManagementIcon-TabExtension	toolBar(table)	sap.m.Button
		planManagementFoo-terBarExtension	toolBar(page)	sap.m.Button

View Name	Screen Name	Point Name	Parent	Type
		planManagementActionSheetExtension	actionSheet	sap.m.Button
fs.ipw.grpinsquote.create.view.managePlanPlans	Plan Management	planManagementPlansTableHeaderExtension	toolBar(table)	sap.m.Button
		planManagementPlansContentAboveExtension	layout	sap.ui.core
		planManagementPlansContentBelowExtension	layout	sap.ui.core
fs.ipw.grpinsquote.create.view.memberTable	Plan Management	planManagementMembersTableHeaderBarExtension	toolBar(table)	sap.m.Button
		planManagementMembersContentAboveExtension	layout	sap.ui.core
		planManagementMembersContentBelowExtension	layout	sap.ui.core
	Single Plan	singlePlanMembersTableHeaderBarExtension	toolBar(table)	sap.m.Button
		singlePlanMembersContentAboveExtension	layout	sap.ui.core
		singlePlanMembersContentBelowExtension	layout	sap.ui.core
fs.ipw.grpinsquote.create.view.singlePlan	Single Plan	singlePlanFooterBarExtension	toolBar(page)	sap.m.Button
		singlePlanActionSheetExtension	actionSheet	sap.m.Button
		singlePlanObjectHeaderExtension	objectHeader	sap.m.ObjectAttribute

View Name	Screen Name	Point Name	Parent	Type
fs.ipw.grpinsquote.create.view.reviewQuoteSummary	Review Quote Summary	reivewQuoteSummaryObjectHeaderExtension	objectHeader	sap.m.ObjectAttribute
		reivewQuoteSummaryIconTabExtension	iconTabbar	sap.ui.m. IconTabFilter
		reivewQuoteSummaryPremiumBreakdownExtension	formContainer	sap.ui.layout.form.FormContainer
		reivewQuoteSummaryBelowContentExtension	layout	sap.ui.core
		reivewQuoteSummaryFooterBarExtension	toolBar(page)	sap.m.Button
		reivewQuoteSummaryTabQuoteContentBelowExtension	layout	sap.ui.core
		reivewQuoteSummaryTabPlansContentBelowExtension	layout	sap.ui.core
		reivewQuoteSummaryPlansTableHeaderExtension	toolBar(table)	sap.m.Button
		reivewQuoteSummaryMembersTableHeaderExtension	toolBar(table)	sap.m.Button
		reivewQuoteSummaryTabMembersContentBelowExtension	layout	sap.ui.core
		reivewQuoteSummaryFooterActionSheetExtension	actionSheet	sap.m.Button

Example: Add content to the extension point extensionLastTabIcon

```
customizing: {
"sap.ui.viewExtensions": {
            "fs.ipw.myuwworklist.view.frame.detailFullScreen": {
            "extensionLastTabIcon": {
                classname: "sap.ui.core.Fragment",
                fragmentName: "fs.ipw.custom.app.iconTab",
                type: "XML"
            }
        }
}
    }
```

⚠ Caution

- There are two types of className: `sap.ui.core.Frgment` and `sap.ui.core.mvc.View`.
- fragmentName/viewName: UI view name/path
- type: "XML"

5.1.1.3 Modifying the View

Currently you can only modify the property `visible` of UI5.

Example: Hide all the content of the Inbox screen

```
customizing: {
     "sap.ui.viewModifications": {
        "fs.ipw.myuwworklist.view.frame.inbox": {
            "mainContainer": {
                "visible": false
            }
        }
}
}
```

⚠ Caution

View modifications only apply to XML Views.

5.1.1.4 Replacing the View

You can replace the whole view without touching the routing.

Example: Replacing the view of Inbox screen with the view `custom`

```
customizing: {
"sap.ui.viewReplacements": {
        "fs.ipw.myuwworklist.view.frame.inbox": {
            viewName: "fs.ipw.custom.app.custom",
            type: "XML"
        }
}
}
```

5.1.1.5 Controller Extension

You can insert the function before or after the original handle function.

It isn't recommended to replace the whole function, because this may break the logic of the original function.

Example: Displaying a dialog box after rendering (Inbox screen)

Custom controller:

```
sap.ui.controller("fs.ipw.custom.app.custom",
    onAfterRendering : function() {
        sap.m.MessageBox.show("CUSTOM APPLICATION", {
        title: "CUSTOM"
        })
    }
)
```

Custom Component:

```
customizing: {
"sap.ui.controllerExtensions": {
    "fs.ipw.myuwworklist.view.frame.inbox": {
        controllerName: "fs.ipw.custom.app.custom"
    }
}
}
```

5.1.1.6 Extension Points

For the `Create Insurance Quote` transactional App, extension points are added before and after the forms on the general information screen, the general information review screen, and the review quote screen. You can add your own forms, tables, buttons and so on to these screens to meet your business requirements.

5.1.2 Extending Standard Mass-Uploading Data Sheet

Within the `Create Group Insurance Quote` app, you can create a group quote, create different plans, and add members to those plans. You can add members individually by selecting or creating business partners, or add them collectively by mass-uploading.

And in the `Create Insurance Quote from Master` app, you can manually create all individual quotes, or manually create one valid individual quote as a template, and then create the rest by mass-uploading.

When mass-uploading plan members or individual quotes, you need to use a standard data sheet (a CSV file).

To meet your additional requirements, you can extend the standard data sheet.

5.1.2.1 Extending Dynamic Columns

A standard data sheet has some static columns and some dynamic columns.

You can extend the standard data sheet by adding your own dynamic columns to meet additional requirements.

5.1.2.1.1 Extending the Group Plan Member Data Sheet

Context

Each data value row represents the information of the policyholder/insured person, the beneficiary and so on.

The values of the first 22 mandatory columns are the details of the policyholder/insured person.

The Service API rule `populateGroupMemberPolicyDataWithSpreadSheetRow` uses the values of the mandatory columns to do the following:

- Populates the columns of table BUSINESS_PARTNER/BUSINESS_PARTNER_ADDRESS/ BUSINESS_PARTNER_IDENTIFIER. This is the local cache for policyholder Business Partner.
- Populates the columns of table ABDAPOLHLDR (Policyholder).
- Populates the columns of table COV_ABDASUBJCT (Insured Person).

The values of the next 22 dynamic columns are the details of the beneficiary.

The service API does the following:

- Populates the columns of table BUSINESS_PARTNER/BUSINESS_PARTNER_ADDRESS/ BUSINESS_PARTNER_IDENTIFIER. This is the local cache for beneficiary Business Partner.
- Populates the columns of table POLPR_ABDABNFCRY (Beneficiary).

Procedure

1. Add the column name in the first row of the data sheet.

   The column name can't contain a space.
2. Log in to the Product Modeler.
3. Open the Life Capital 2009 92H0000S0002 product.
4. Select *Configuration* > *Product Services* > *Service API* from the *Product* tree.

5. On the right panel, choose the *Values* tab, and then edit/override the script rule `populateGroupMemberPolicyDataFromSpreadsheetRow`.

   > → Remember
   >
   > The script rule argument `groupMemberUploadRowData` contains all the column name - value mappings of the whole data sheet.

6. Inside the script rule, use `groupMemberUploadRowData:OBJECTTABLE.<Spreadsheet Column Name in upper case>` to access the data, and define your own logic and save it to the database.

   For example, if the new column name is *NewColumn*, then use `groupMemberUploadRowData:OBJECTTABLE.NEWCOLUMN` to access the data.

   Here are some other examples of accessing the currently included column:

   - `groupMemberUploadRowData:OBJECTTABLE.FIRSTNAME`
   - `groupMemberUploadRowData:OBJECTTABLE.BENEFIRSTNAME`

7. Save your changes.
8. Select *Build* > *Rules* from the top menu bar to compile all the rules.
9. Select *File* > *Publish Product* from the top menu bar to publish the product.
10. Deploy the product to runtime FS-QUO to bring the new rule into effect.

5.1.2.1.2 Extending the Multi-Individual Quote Data Sheet

Context

Within the `Create Insurance Quote from Master` app, you can select a master policy as the template of each individual quote. Each individual quote inherits the policyholder, commission, and sales product from the master policy. And then you can manually create all individual quotes, or manually create one valid individual quote as a template, and then create the rest by mass-uploading.

The Service API rule `populateInsuredObjectDataFromSpreadsheetRow` uses the values of the mandatory columns to do the following:

- Populates the columns of table COV_ABDASUBJCT (Insured Person).

Procedure

1. Add the column name in the first row of the data sheet.

   The column names can't contain space.

2. Log in to the Product Modeler.

3. Open the `Sample Household Product Extended` product.
4. Select *Configuration* > *Product Services* > *Service API* from the *Product* tree.
5. Choose the *Values* tab on the right panel, and then edit/override the script rule `populateInsuredObjectDataFromSpreadsheetRow`.

   > → Remember
   >
   > The script rule argument `insuredObjectData` contains all the column name - value mappings of the whole data sheet.

6. Inside the script rule, use `insuredObjectData:OBJECTTABLE.<spreadsheet_column_name_in_upper_case>` to access the data, and define your own logic and save it to the database.

   For example, if the new column name is *NewColumn*, then use `insuredObjectData:OBJECTTABLE.NEWCOLUMN` to access the data.
7. Save your changes.
8. Select *Build* > *Rules* to compile all the rules.
9. Select *File* > *Publish Product* from the top menu bar to publish the product.
10. Deploy the product to runtime FS-QUO to bring the new rule into effect.

5.1.2.2 Extending the File Parser

Prerequisites

You have `com.sap.fs.quo.service.jar` on your system.

Context

The default file type of the supported data sheet is CSV. But you can add new supported file type with the file parser extension.

Procedure

1. Launch Eclipse and select a workspace.
2. Select *File* > *New* > *Other* from the menu.

   The *New* dialog box appears.

3. Create a Maven project:
    a. Select *Maven* > *Maven Project* in the dialog box
    b. Choose *Next*.
    c. In the next screen, accept all default settings.
    d. Choose *Next*.
    e. In the next screen, select the Artifact ID *maven-archetype-quickstart*.
    f. Choose *Next*.
    g. In the next screen, enter **`com.customer`** in the *Group Id* field and enter **`extensibility`** in the *Artifact Id* field
    h. Choose *Finish*.
4. In the *Project Explorer* tab, expand the extensibility project, open `pom.xml`, and then add the following dependency for your project, while specifying the version to 300.2.0-SNAPSHOT or higher:

```
<dependency>
        <groupId>com.sap.fs.quo</groupId>
        <artifactId>com.sap.fs.quo.service</artifactId>
        <version>300.2.0-SNAPSHOT</version>
</dependency>
```

5. Create a new Class with the name `XXXFileParser` (for example, `XLSFileParser`) and the superclass `AbstractInsuredObjectFileParser` (for parsing insured object)/`AbstractInsuredPersonFileParser` (for parsing insured person) (package: `com.sap.fs.quo.service.helper`).
6. Implement the `loadContent` method of the superclass for loading file content.
7. Create a `META-INF` folder if there isn't one in `src/main/resources` package, and create an XML file with the name `<user-customer>` (for example, `custQuoteServices.xml`) for ready spring configuration at the same package.
8. Add the following code into the `<user-customer>` file

```
<?xml version="1.0"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:lang="http://www.springframework.org/schema/lang"
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/lang
http://www.springframework.org/schema/lang/spring-lang-2.5.xsd
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd"
default-lazy-init="true">
</beans>
```

9. Add the spring bean into the `<user-customer>` file, which then looks like:

```
<bean id="fileParserFactory"
class="com.sap.fs.quo.service.helper.FileParserFactory">
  <property name="fileType2IOParser">
         <map>
             <entry key="file_type"
value="com.customer.extend.XLSFileParser" />
         </map>
      </property>
 <property name="fileType2IPParser">
         <map>
```

```
            <entry key="file_type"
value="com.customer.extend.XLSFileParser" />
        </map>
      </property>
</bean>
```

> **⚠ Restriction**
>
> `file_type` is the extension name of the standard data sheet. For example, it is `csv` for data sheets in CSV format. You aren't allowed to modify the attribute ID `fileParserFactory` and property `fileParserMap`.

10. In the `META-INF` folder, create a new XML file with the name context-manifest and the following content:

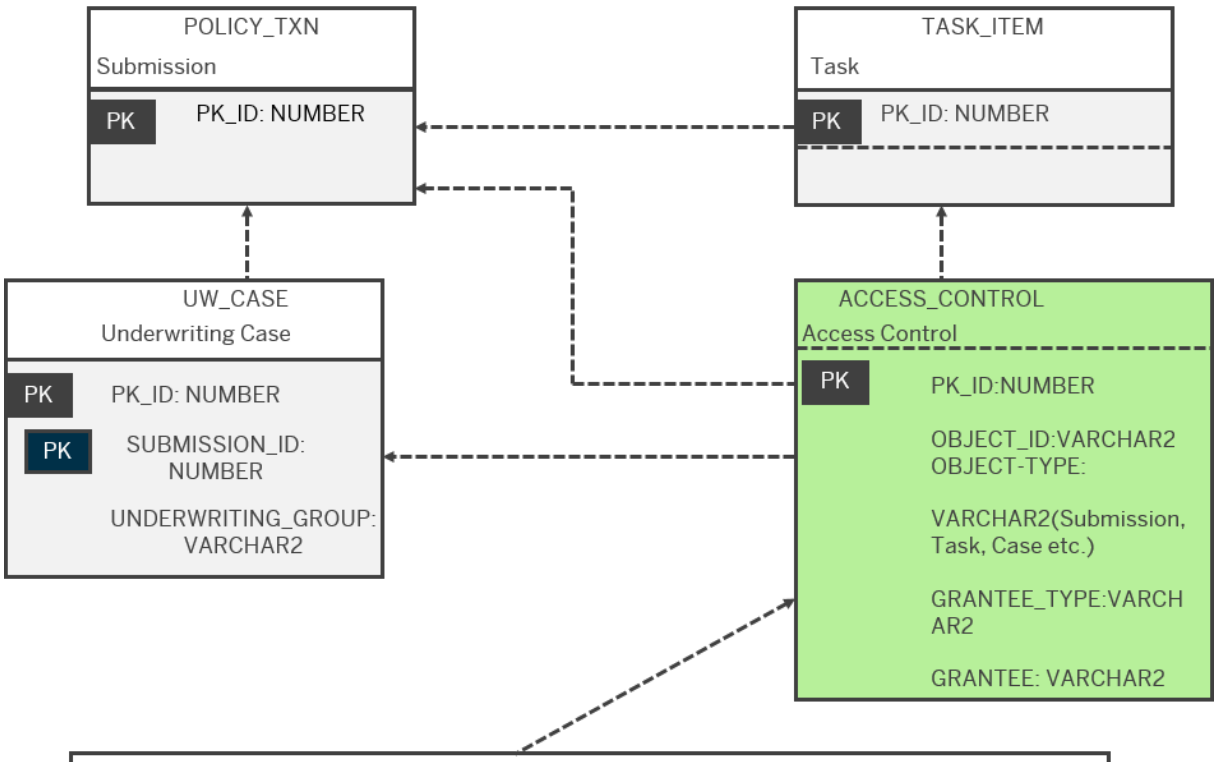
```
<?xml version="1.0" encoding="UTF-8"?>
<context-manifest>
    <context location="classpath:/user-customer.xml" />
</context-manifest>
```

11. Right-click the project name, select *Run As* > *Maven install*, expand the `target` folder, copy the JAR file and then deploy it into `<server-name>/lib`.

5.1.3 Configuring Data Visibility

5.1.3.1 Understanding the Data Visibility Framework

To implement data visibility for underwriting, quotation, and insurance task, you need to associate the group information with each underwriting case, submission, or insurance task, as described in the diagram below:

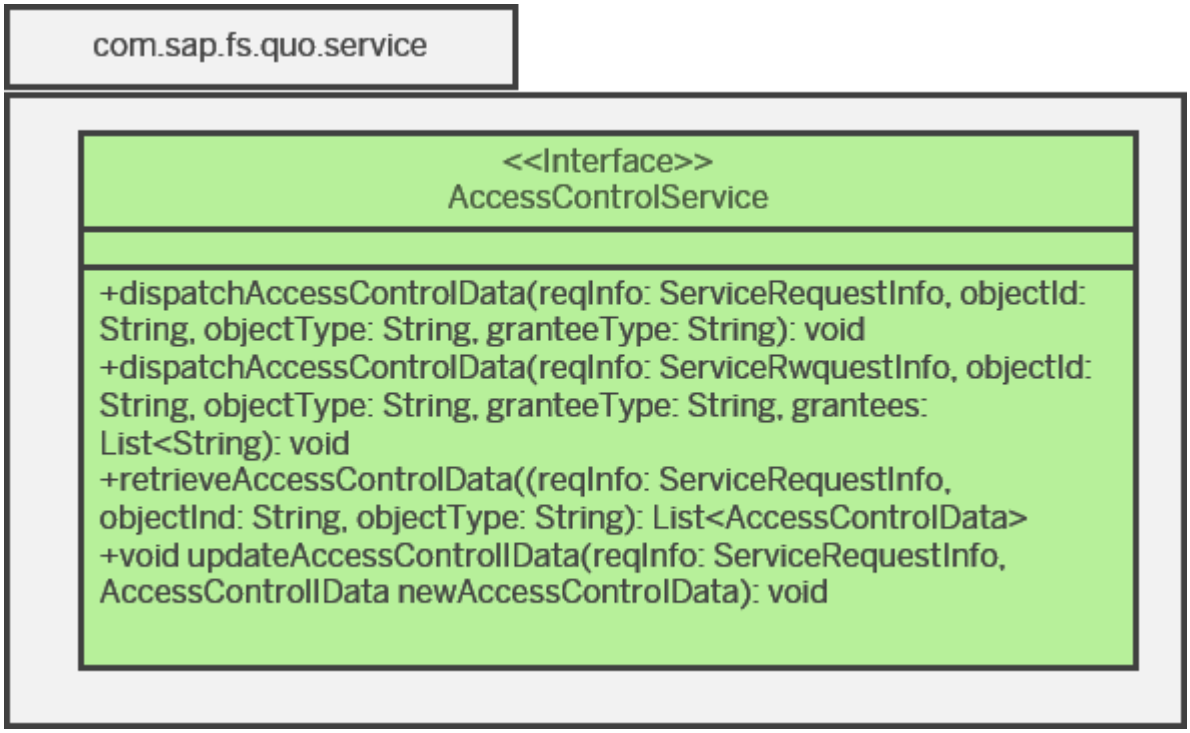


1. OBJECT_ID: the PK _ID of the object to be controlled.
2. OBJECT_TYPE: can be submission, task, case, etc.
3. GRANTEE_TYPE: the type of grantee, OOB implementation supports GROUP only.
4. GRANTEE: OOB implementation stores the user group information dispatched to specific object.

Note: Current implementation could be enhanced to grant permission to a user instead, then GRANTEE_TYPE = USER and GRANTEE = PK _ID of the user.

For Underwriting Worklist, show all the submissions associated with underwriting group of cases. The UnderwritingGroup data is initialized after the execution of system underwriting rules.

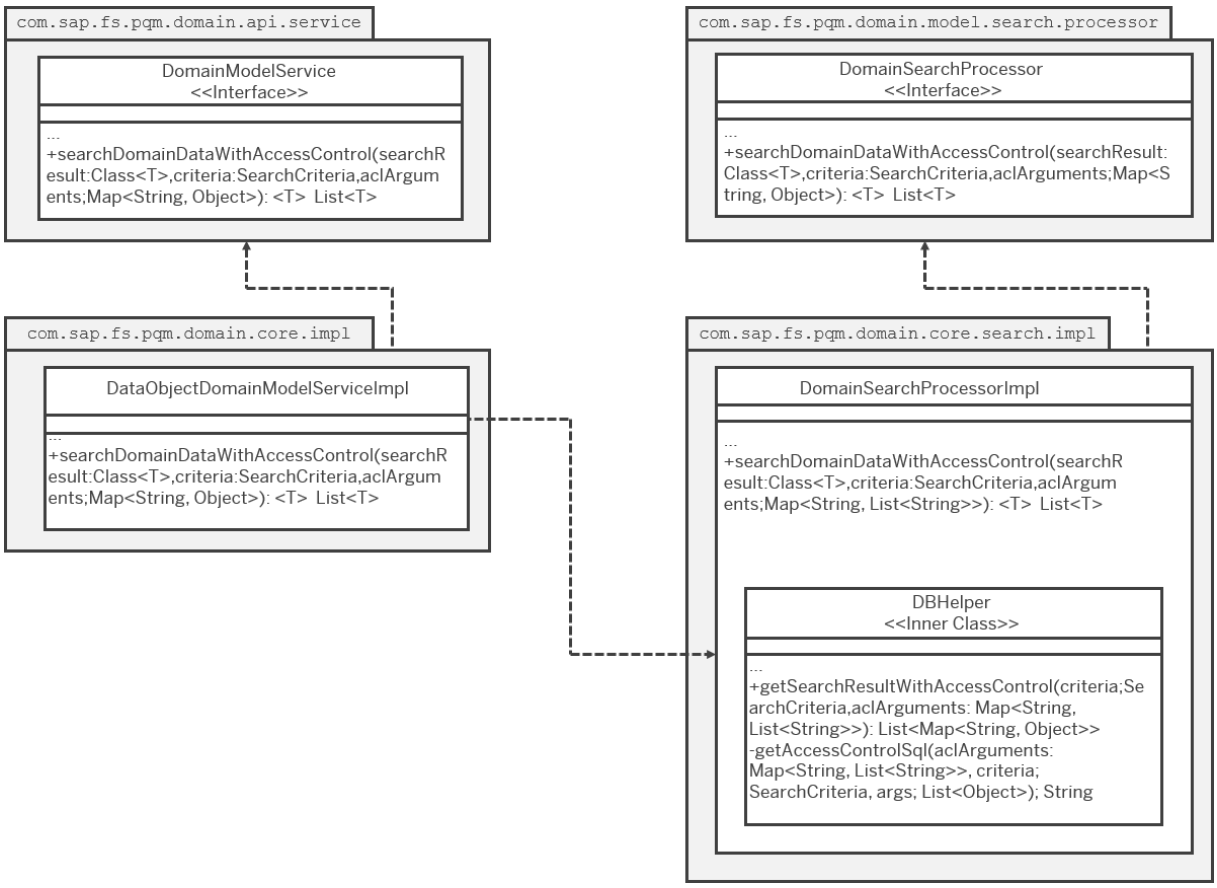
For Quotation and Task worklists, show all submissions/tasks associated with any groups that the user belongs to. User groups are defined by role collections.



To enable access control for the submission worklist, insurance task worklist, and underwriting worklist, you should follow the principles below:

- For the underwriting worklist:
  - Use the existing Dao query to add `ACCESS_CONTROL` join.
  - Don't introduce any security vulnerability issues.
- For the insurance task worklist and submission worklist:
  - Extend `DomainModelService.searchDomainData` to optionally add `ACCESS_CONTROL` query.
  - Don't introduce any security vulnerability issues.

To add the access control support for the `searchDomainData` interface of `DomainModelService`, we introduced a new interface `searchDomainDataWithAccessControl`. The main class diagram for this change is as follows:



Sample of the generated SQL Scripts (prepared statement) for ACCESS_CONTROL:

```
select SUBJECT, PK_ID, POLICY_TXN_ID, DESCRIPTION, DUE_DATE, PRIORITY,
LAST_UPDATE_DATE, STATUS, OWNER, ASSIGNEE, CREATED_DATE, MANDATORY, DELETE_FLAG
from TASK_ITEM
where ( ASSIGNEE = ? AND DELETE_FLAG = ? ) and
PK_ID in (select OBJECT_ID from ACCESS_CONTROL ac where 1=1
and ac.OBJECT_TYPE in('task')
and ac.GRANTEE_TYPE in('group' ,'user')
and ac.GRANTEE in('SAP_UW_HOUSEHOLD' ,'SAP_UW_LIFE' ,'SAP_UW_SYSTEM') );
```

5.1.3.2 Data Visibility Configuration

You can define the relationship between a user and the groups in XSA using role collections.

> **Note**
>
> For more information about role collections in XSA, see the *SAP HANA Administration Guide for SAP HANA Platform*.

In the details panel of a user, on the *Assigned Groups* tab page, you can maintain the user group information of the user.

Submissions and Tasks

In the current implementation of data visibility, submissions and tasks are visible to their creator, as well as creator's peers. That is, anyone who is part of any of the user groups that the creator belongs to will see creator's submissions and tasks.

Example:

There are two teams reporting to a manager. The agents within each team should see each other's submissions and tasks but should not see submissions and tasks of the other team. A manager of the two teams wants to see everyone's submissions. To achieve this data visibility the following user setup should be implemented:

	Role Collection	Users
Team A	TEAM_A_RC	Uw_A1, Uw_A2, Uw_A3,...
Team B	TEAM_B_RC	Uw_B1, Uw_B2, Uw_B3,...
Manager	TEAM_A_RC, TEAM_B_RC	Uw_Mgr1

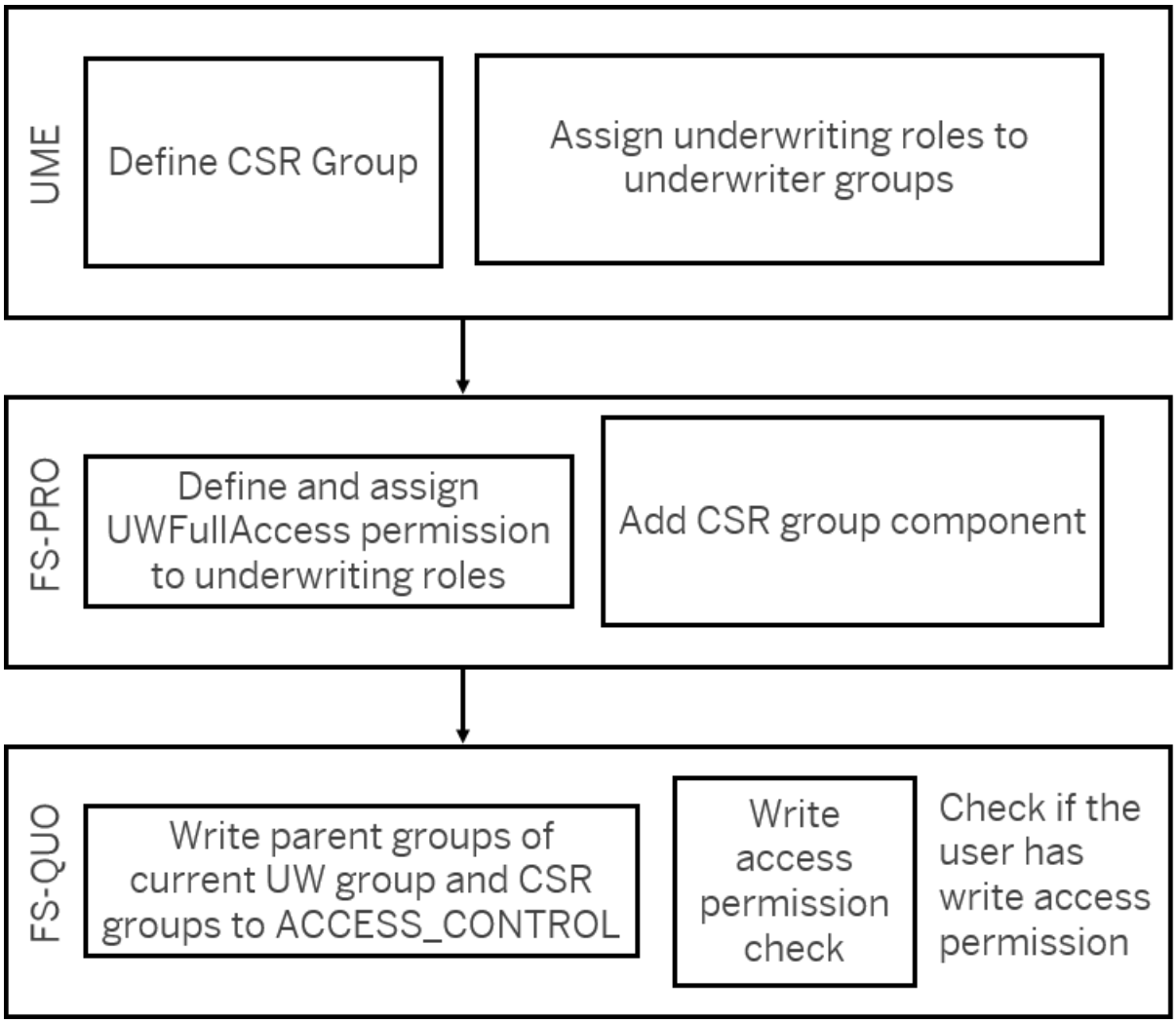
> ⓘ Note
>
> Be aware that if there is a role collection which is assigned to all users then everyone will see everyone's submissions and tasks.

> → Remember
>
> The user group information that is defined in a product should be provided to your FS-QUO system administrator for role collection and user group mapping.

5.1.4 Configuring Read-only Access to Underwriting Cases

To allow customer support representatives (CSR) to access the underwriting worklist in read-only mode, you need to configure the read-only access to underwriting cases for them.

The main activity diagram is as follows:



The read-only access to underwriting cases is fulfilled at two dimensions:

- Data visibility of submission worklist: this is controlled with groups.
- Data operations on underwriting cases: this is controlled with permissions and roles.

At design time, you need to define a coarse-grained write permission (for example, `UWFullAccess`) to related underwriting roles.

At Runtime, the resources should be protected with permission. The read-only access to underwriting is the default behavior, unless the user has other write permission.

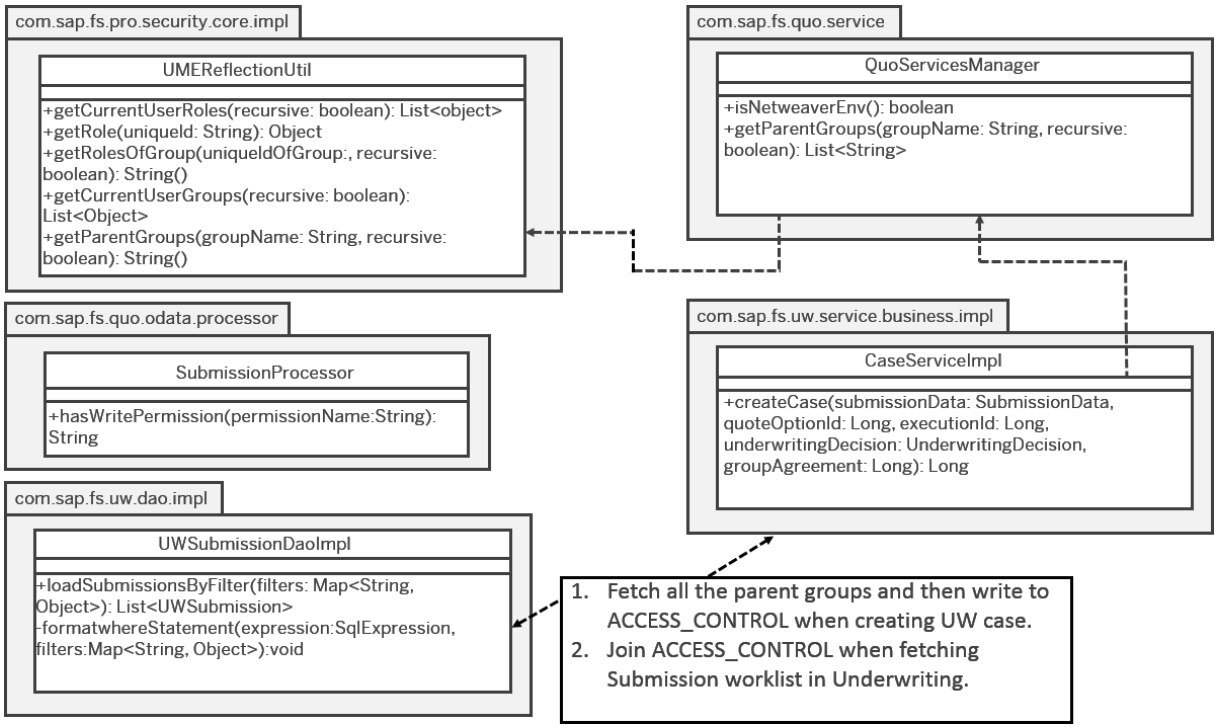
The data visibility part can be implemented based on the framework as explained in the *Configuring Data Visibility* topic. To allow CSR users view the underwriting worklist, simply write the CSR groups to ACCESS_CONTROL for underwriting cases. And the same mechanism can be applied to quotation worklist and insurance task worklist.

The write access permission check is performed with the request `/hasWritePermission?PermissionName=UWFullAccess`, which loads the roles assigned to the current user and groups and their related permissions for checking. If the user has `UWFullAccess` permission, it returns true, otherwise it returns false. The front-end will enable or disable the action buttons based on the check result.

> ⓘ Note
>
> If a particular resource needs to be protected, you should define fine-grained permission and enable the permission filter.

> ⓘ Note
>
> The roles configured in UME should be consistent with FS-PRO configuration in SAP `Role` components. The CSR groups defined in UME should be consistent with FS-PRO configuration in `CSR Group` component under `Extended Underwriting Application Configuration`.

The main class diagram is as follows:

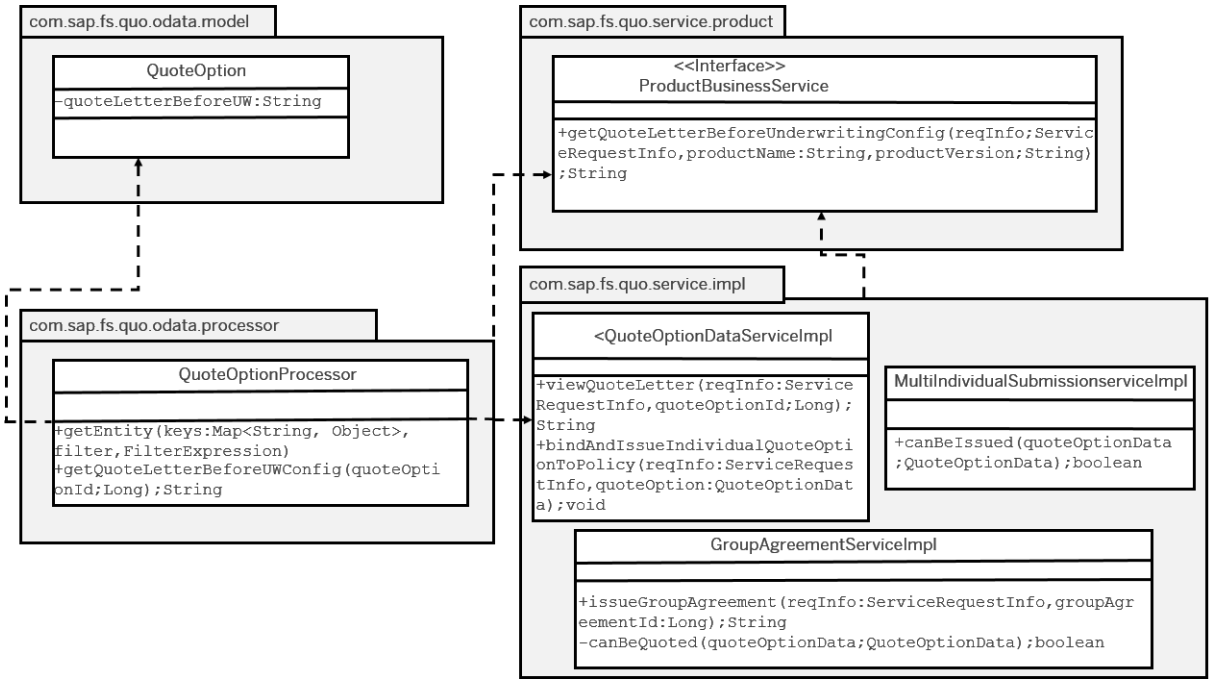


> → Tip
>
> See the `Sample Household Product` for an example of how underwriting case permissions can be configured.

5.1.5 Configuring Quote Letter Generation

The quote letter generation step can be configured to be before or after submitting for underwriting for a specific product from different lines of business.

The main activity diagram is as follows:

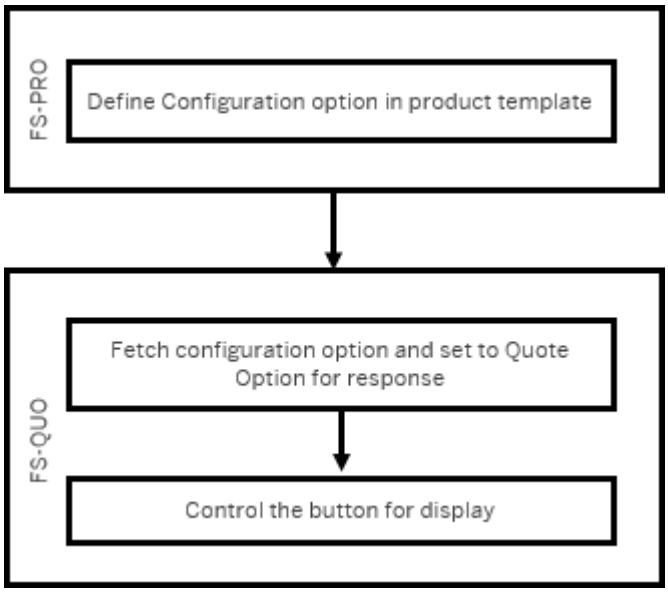


At design time, you need to add a configuration option `generateQuoteLetterBeforeUW` to `SAP Insurance UWA Product Template` under *Configuration* > *Config*. The standard products inherit this option, and you can override it for specific products.

If `generateQuoteLetterBeforeUW = true`, the *Generate Quote Letter* button appears first. When the user clicks it, the quote option status changes to `Quoted`, and then the *Submit for Underwriting* button is visible.

If `generateQuoteLetterBeforeUW = false`, the *Submit for Underwriting* button appears first. When the user clicks it, the quote option status changes to one of the underwriting statuses according to the underwriting decision. And when it changes to `Approved`, the *Generate Quote Letter* button is visible.

The main class diagram is as follows:



A quote letter should be generated when the quote option status is `Approved` or `Working`, and the submission status is `Working`.

If the quote option status is `Quoted`, it means a quote letter has been generated. If the user clicks the *Generate Quote Letter* button again, it won't generate a quote letter again, but shows the generated one.

If `generateQuoteLetterBeforeUW = true`, the user can issue a policy when the quote option status changes to `Approved`.

If `generateQuoteLetterBeforeUW = false`, the user can issue a policy when the quote option status changes to `Quoted`.

5.1.6 Using Sequential Numbers for Submission ID, Application Number and Policy Number

You can use sequential numbers for the submission ID, application number, and policy number.

The generation of sequential submission number is enabled by default in the standard product. When not enabled, submission numbers are created randomly, which can cause usability issues and confuse users.

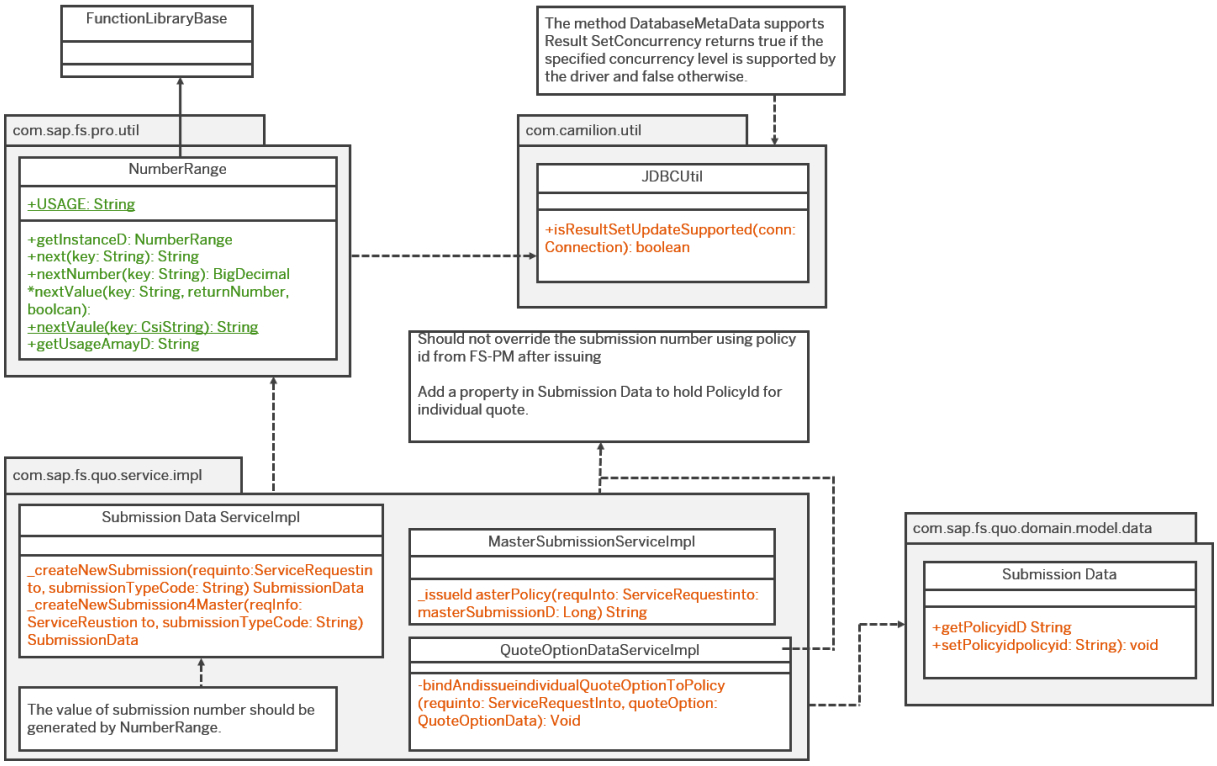
The generation of sequential application number and policy number is supported, but it's disabled by default. To use this feature, you need to enable it in the Administrative Console.

5.1.6.1 Generating Sequential Submission ID

To generate sequential submission ID, follow this procedure:

- Define a system configuration item in the Administrative Console.
- Create a table to hold the policy ID from FS-PM for individual quotes.
- Initialize the table.

The main class diagram is as follows:



5.1.6.1.1 Defining a System Configuration Item in the Administrative Console

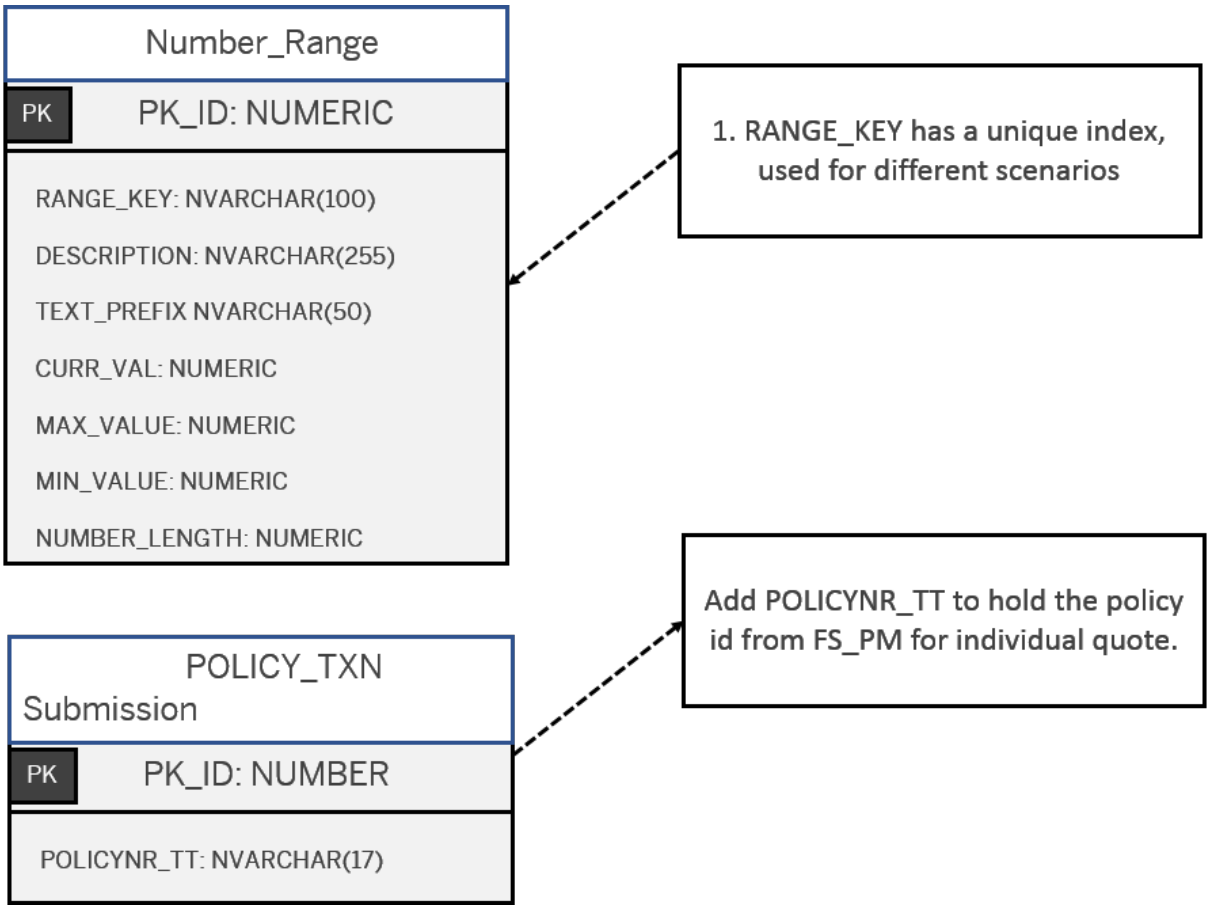
To enable the sequential number generation when creating submissions, define the following system configuration item in the Administrative Console:

```
Key: submission.number.range
```

The default value is 1, which means it's enabled by default.

5.1.6.1.2 Creating a Table to Hold the Policy ID from FS-PM for Individual Quotes

The main data model is as follows:



The table needs to be created during the clean install process, and you need to put the creation script in `com.sap.fs.pro\FS-PRO-db\src\main\resources\${DB_TYPE}\sql\tables\sys\create_sys_tables.sql`.

Example: Sample Script for Creating the Table

For SAP HANA:

```
CREATE COLUMN TABLE NUMBoutputclass="lang-sql"ER_RANGE (
PK_ID DECIMAL(16) PRIMARY KEY NOT NULL,
RANGE_KEY VARCHAR(100) NOT NULL UNIQUE,
DESCRIPTION VARCHAR(255),
MIN_VALUE DECIMAL(20),
MAX_VALUE DECIMAL(20),
NUMBER_LENGTH DECIMAL(2),
TEXT_PREFIX VARCHAR(50),
CURR_VAL DECIMAL(20)
```

```
);
```

5.1.6.1.3 Initializing the Table

For clean install:

- The CURR_VALUE should be empty (default)
- The MIN_VALUE should be 1 (default) or any other specified number
- The initialization data should be put in `com.sap.fs.pro`\FS-PRO-`db\src\main\resources\common\data\sys\NUMBER_RANGE.txt`.

For upgrade install (only need to execute once):

- The CURR_VALUE should be empty (default)
- The MIN_VALUE should be `max(POLICY_TXN.POLICY_SUBMISSION_NUMBER) + 1`
- And you need to write Java interfaces in `com.sap.fs.pro`\FS-PRO-`db-migration` to implement it

Example: Sample Script for Initializing the Table

```
INSERT INTO NUMBER_RANGE (PK_ID, RANGE_KEY, DESCRIPTION, TEXT_PREFIX,
MIN_VALUE, MAX_VALUE, NUMBER_LENGTH )
VALUES (1, 'POLICY_TXN.SUBMISSION_NUMBER', 'Submission Number', '',
(select MAX(POLICY_SUBMISSION_NUMBER) from POLICY_TXN ) + 1,
9999999999999999, 16);
```

5.1.6.1.4 Data in NUMBER_RANGE.txt

```
POLICY_TXN.SUBMISSION_NUMBER~Submission
Number~1~9999999999999999~16~~~
```

5.1.6.2 Generating Sequential Application Number and Policy Number

To enable the sequential number generation when creating temporary applications, also define the following system configuration item in Administrative Console:

```
Key: model.calc.number.range
```

The default value is 0, which means it's disabled by default.

Data in NUMBER_RANGE.txt

```
MODEL_CALC_ABDAPOLICY.POLICYNR_TT~Number range used for creating
temporary applications in FSPM for Model
Calculation~19900000000000001~19999999999999999~17~~~
MODEL_CALC_ABDAPOLICY.APPLNR_CD~Number range used for creating
temporary applications in FSPM for Model
Calculation~00000000000000001~99999999999999999~17~999~~
```

The above number range must be consistent with the one defined in FS-PM, which can be found with the following steps:

1. Login to the FS-PM system.
2. Enter transaction code **SPRO**.
3. Select *SAP Reference IMG*.
4. Expand *Policy Management* > *General Settings* > *Number Ranges* > *In-Force Business Management* .

The number range defined for application and policy will be displayed.

5.2 OData Extensions

5.2.1 DataTable-related Annotations

DataTableMapper annotations has been introduced to indicate an entity that can be mapped by the `DataTable` object. The `postPropertiesSet` parameter provides the capability to run customized mapping after DataTable to Entity mapping completes. By default, if the parameter isn't specified, the `afterPropertiesSet` API will be invoked if it exists in the Entity bean.

DataTableMapper annotations has been introduced to indicate an entity that can be mapped by the `DataTable` object. The `postPropertiesSet` parameter provides the capability to run customized mapping after DataTable to Entity mapping completes. By default, if the parameter isn't specified, the `afterPropertiesSet` API will be invoked if it exists in the Entity bean.

5.2.2 ProductRuleDataSource

The `ProductRuleDataSource` provides OData data source connection to the rules engine. The following API is provided, implementing `PqmODataSingleDataSource` as an abstract class. The `read()` method should

be implemented by an inherited data source that injects into a processor the process requested by OData for linked entities. The following are the provided APIs by `ProductRuleDataSource`.

```
/**
     * Executes a rule with the specified information.
     *
     * @param productName
     *            Name of the product where the rule is located, as a String.
     * @param productVersion
     *            Version of the product where the rule is located, as a String.
     * @param contextDate
     *            Date when the rule is to be executed, as a Timestamp object.
     * @param serviceName
     *            The name of the service where the rule is located, as a
     *            String.
     * @param args
     *            List of input arguments for the rule to be executed, as
     *            Map<String, Object>.
     * @return The result of the rule execution, as a RuleResult object.
     */
    protected RuleResult runProductRule(String productName, String
productVersion, Timestamp contextDate,
            String serviceName, Map<String, Object> args);
/**
     * Executes a rule with the specified information.
     *
     * @param productName
     *            Name of the product where the rule is located, as a String.
     * @param productVersion
     *            Version of the product where the rule is located, as a String.
     * @param contextDate
     *            Date when the rule is to be executed, as a Timestamp object.
     * @param ruleId
     *            The rule identify which is primary key of rule in general.
     * @param args
     *            List of input arguments for the rule to be executed, as
     *            Map<String, Object>.
     * @return The result of the rule execution, as a RuleResult object.
     */
     protected RuleResult runRule(String productName, String productVersion,
             Timestamp contextDate, long ruleId, Map<String, Object> args);
    /**
     * Checks if an entity matches the primary key. The matching function has
been implemented in this DataSource. It can be directly called by sub-data
source.
     *
     * @param object the entity to check
     * @param keys the key-value pairs
     *
     * @return true if matches, false otherwise
     */
@Override
    public boolean matches(T object, Map<String, Object> keys);
```

5.2.3 Processor

`Processor` processes OData requests that declare to the supported `entityType`. In the OData data source methodology, `Processor` links entity type to a specific data source to retrieve data or perform any other functions.

5.2.4 ModelValueMapper

Currently default the `ModelValueMapper` implementation is `DataTableValueMapper`. The purpose of `ModelValueMapper` is to create generic/reusable mapping mechanisms between entities and back-end service objects. Customers can create their own value mapper specific to an entity and service.

```
/**
     * Map valueObjects (service objects) to List of OData entity.
     * @param valueObject
     * @param entity
     * @return entitySet
     */
    public <T> List<T> mapEntities(Object valueObjects, T entity);

    /**
     * Map a single object value to entity
     * @param value
     * @param entity
     * @return
     */
    public <T> T mapEntity(Object value, T entity);


    /**
     * Map entity value to valueObject (service object), mainly for create/
update/delete mapping.
     * @param valueObject
     * @param entity
     * @return
     */
    public <T> T mapObject(Object valueObject, T entity);
```

5.2.5 DataTableValueMapper

`DataTableValueMapper` maps the general DataTable with a desired entity object. This mapper converts all data table rows to an entity list. In the above example, it converts DataTable to Product entity.

The following APIs have been following implemented by this mapper:

```
/**
     * Map entire data table (all rows) to an entity list.
     *
     * @param value        The DataTable object.
     * @param entity    Target entity.
     *
     * @return             Entity list with mapped value.
     */
    @Override
    public <T> List<T> mapEntities(Object valueObjects, T entity);
/**
     * Map one single data table row to an entity. It will load current row of
passed DataTable object.
     *
     * @param value    The single row of data table.
     * @param entity    Target entity.
     *
     * @return             Entity with mapped value.
```

```
     */
    @Override
    public <T> T mapEntity(Object value, T entity);
```

5.3 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.

5.3.1 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
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
CurrencyExchRateLookup	com.sap.fs.pro.currency.stems.CurrencyExchangeRateLookupStem
DataObjectAccess	com.sap.fs.pro.stem.DataObjectAccessStem
IFBCScriptUtils	com.camilion.pa.integration.fspm.ifbc.stems.IFBCScriptUtils
MsgRating	com.camilion.custom.stems.MsgRating
Product	com.camilion.util.functions.Product

Stem Name	Source
ProductService	com.camilion.ruleengine.ruleserver.services.iips.ProductServices
Pws	com.camilion.custom.integration.fspm.pws.stems.PwsDataObjectCopyStem
ScriptUtil	com.camilion.util.functions.ScriptUtil
TargetDataRow	com.camilion.util.functions.TargetDataRow
EAppAccess	com.sap.fs.pro.stem.EAppAccessStem
ASDataTypeStem	com.camilion.as.stem.ASDataTypeStem
ASFlowStem	com.camlion.as.stem.ASFlowStem
ASPolicyModelStem	com.camilion.as.stem.ASPolicyModelStem
ASPremiumAuditStem	com.camilion.as.stem.ASPremiumAuditStem
ASServiceMethodStem	com.camilion.as.stem.ASServiceMethodStem
asUserStem	com.camilion.as.stem.ASUserStem
bpLookup	com.sap.fs.quo.core.stem.FSPMIntegrationBPLookupStem
CalculatePremiumStem	com.sap.fs.quo.core.stem.CalculatePremiumStem
DataTableStem	com.sap.fs.quo.ibu.stem.DataTableStem
DisplayIllustration	com.sap.fs.quo.core.stem.DisplayIllustrationServiceStem
DomainBasedQuoteOptionStem	com.sap.fs.quo.service.impl.stem.DomainBasedQuoteOptionStem
DPPLoggingStem	com.sap.fs.pro.dpp.stem.DPPLogginStem
FormFN	com.camilion.custom.stems.FormLibrary
FSPMInsurObjStem	com.sap.fs.quo.core.stem.FSPMInsurableObjectStem
GroupPolicy	com.sap.fs.quo.core.stem.GroupPolicyStem
InsuObjLookup	com.sap.fs.quo.core.stem.FSPMInsurableObjectLookupStem
QuotationStem	com.sap.fs.quo.ibu.stem.QuotationStem
Reference	com.camilion.custom.stems.ReferenceUtil
ResetRating	com.camilion.custom.stems.ResetRating

Stem Name	Source
SubmissionStem	com.sap.fs.quo.core.stem.SubmissionStem
TransactionStem	com.camilion.as.stem.TransactionStem
URSCreation	com.sap.fs.quo.core.stem.UtilABDAPolicyStem
URSStem	com.sap.fs.quo.ibu.stem.URSStem
UsersList	com.camilion.custom.stems.UsersList
userStem	com.camilion.as.stem.User
ValidationErrorDetailStem	com.sap.fs.quo.service.impl.stem.ValidationErrorDetailStem

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.

5.3.2 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.

5.3.3 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.

5.3.4 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.

5.3.5 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.

5.3.6 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.

5.4 Troubleshooting Tips

The following table outlines some troubleshooting tips with suggested fixes

Issue	Fixes
The FS-QUO back-end is down/inaccessible as you are getting HTTP response codes of 404 or 503.	Restart the server.
The FS-QUO back-end is up and running but Single Sign On (SSO) isn't going through. For requests to URL `/ipw/*`, you see HTTP response code 200 but the response body is in the HTML format of the login page.	In this case, check that you are logging in using a user that exists in both the ABAP server and in the FS-QUO Java server in UME. There could also be an SSO configuration error.

5.5 Cross-Site Request Forgery (XSRF/CSRF) Protection

Cross-Site Request Forgery (CXRF/ XSRF) is a cyber attack that tricks a victim's browser into sending a request to a vulnerable Web application, which then performs an undesired action on behalf of the victim.

If you are writing your own application that call FS-QUO services including eApps, you will need to account for CSRF/XSRF security. This requires the use of a CSRFToken header, and a specific request and response sequence.

5.5.1 CSRF Protection Within FS-QUO Applications

The SAP `com.sap.xsrf.rest.filter.XsrfRestFilter` is leveraged to implement a token-based solution to protect FS-QUO apps against attacks of this type. The filter provides access to the resource when a modifying HTTP method is used and a CSRF token is received with the same value as the token stored in the current session. To obtain the CSRF token, the client has to use a non-modifying HTTP method containing the string x-csrf-token with the value Fetch. The token is issued only if the user has already been authenticated. If the user hasn't been authenticated, any request with a modifying method is rejected by this filter. In addition, the client also has to include all previously received cookies in the modifying requests.

CSRF token support was added to eApp HTTP request APIs.

The OData requests provided by the UI5 framework support this token-based solution out of the box so no additional implementation is required to ensure CSRF protection. However, other requests, such as jQuery AJAX requests, will need to be secured in the following way:

1. Retrieve the `x-csrf-token` for the current authenticated session. The following JavaScript utility function is an example taken from the `Create Insurance Quote` application to perform this task:

```
fs.ipw.insquote.create.util.SecurityHelper = {
```

```
    /**
     * Retrieves the XSRF token for an authenticated session and performs a
callback on the controller function
     * with the token ID in the function parameter.
     *
     * @returns {string|number|null} XSRF token of authenticated session.
     */
    getXsrfToken: function(){
      var oTokenId;
      jQuery.ajax({
          async: false,
          url: "/ipw/quo/getUserInfo.do",
          type: "GET",
          beforeSend: function(xhr){
              xhr.setRequestHeader("x-csrf-token", "Fetch");
              xhr.setRequestHeader("Accept-Language",
sap.ui.getCore().getConfiguration().getLanguage());
          },
          contentType: "application/json;charset=utf-8",
          dataType: "json",
          cache: false,
          success: function(data, status, xhr) {
              oTokenId = xhr.getResponseHeader("x-csrf-token");
          },
          error: function(err) {
jQuery.sap.log.error(this.oBundle.getText("COULD_NOT_GET_XSRF_TOKEN") + err);
          }
      });
      return oTokenId;
    }
  };
```

The request header is populated with the key `x-csrf-token` and it has the value Fetch. This server side call will then create the necessary token and return it upon success.

2. Populate the `x-csrf-token` within the request header for any subsequent requests.
   The following is an example of usage in JavaScript:

```
createNewFlow: function() {
        var oTokenId =
fs.ipw.insquote.create.util.SecurityHelper.getXsrfToken();
        var oTransactionId = "";
        jQuery.ajax({
          type: "POST",
          beforeSend: function(xhr){
            xhr.setRequestHeader("Cache-Control", "no-cache, must-
revalidate");
            xhr.setRequestHeader("Pragma", "no-cache");
            xhr.setRequestHeader("x-csrf-token", oTokenId);
          },
          context: this,
          async: false,
          url:
fs.ipw.insquote.create.Configuration.prototype.getSetupParameter("WebflowUrl")
,
          dataType: "json",
          success: function(result) {
            oTransactionId = result.response.transactionId;
          }
        });
        return oTransactionId;
      },
```

In the above code example, an initial request is made to the `fs.ipw.insquote.create.util.SecurityHelper` JavaScript class to obtain the `x-csrf-token`. The request header is then populated with the key `x-csrf-token` and it has the actual value of the token. This server side call then validates this token and either allows or rejects the request.

5.5.1.1 XSRF/CSRF Protection and UI5 eApps

For eApps provided out of the box, when a web client calls the `com.sap.fs.ps.eapp.renderer.controller.JSONLayoutController` with the action `GetNavigationModel`, a random string of 32 alphanumeric characters is generated as a CSRF token. This string token is set into the HTTP session and to the NavigationModel response with the name CSRFToken.

With each `POST`, `PUT` or `DELETE` request, the CSRFToken is passed in the request header and compared against the one in the session. If there isn't a match, the request is rejected with an error.

The following JavaScript example code snippet shows an example of an AJAX request to the `com.sap.fs.ps.eapp.renderer.controller.JSONLayoutController` with the `GetNavigationModel` action set as well as obtaining the CSRFToken from the response:

```
$.ajax({
      type: 'GET',
      async: false,
      data: jQuery.param({"Action":"GetNavigationModel"}),
      url: "/csiroot/eapp-runtime/JSONLayoutController",
      dataType: 'json',
      beforeSend: function(){
      },
      success: function(result) {
           var eAppNavigationModel = new sap.ui.model.json.JSONModel();
           eAppNavigationModel.setData(result);
           var csrfToken = eAppNavigationModel.getProperty("/CSRFToken");
      },
      error: function(e){
        jQuery.sap.log.error(e.message);
      },
      complete: function(){
      }
});
```

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