User Guide | PUBLIC
2023-11-17

Product Modeler User Guide for Coverage-based Products

SAP Product, Quotation, and Underwriting Management 2022 SP01

THE BEST RUN SAP

Content

1 Introduction

1.1 About this Document

This document is a reference for the various features of the Product Modeler and describes how to create and define coverage-based products for your business model.

1.2 Audience

This document is for insurance product managers, actuaries, underwriters, and any other stakeholder in the product development process.

It is assumed that you have been introduced to the fundamentals of SAP Product, Quotation, and Underwriting Management through classroom training.

1.3 Unsupported Features

The following features are not supported in SAP Product, Quotation, and Underwriting Management, 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

2 Getting Started

2.1 User Authorization System

To complement the User Directory, FS-PRO provides the User Authorization System, which enables rules in products and applications to query user authorization profiles at Runtime.

Through the User Directory, a user's logon ID links them to a specific profile. Rules make decisions and apply business logic based on the information in the profiles. Profiles are totally customized. A profile can be assigned to groups of people or to a single user.

Administrators and other users create authorization profiles using FS-PRO, and then publish them to the User Authorization Service.

Version management refers to the process for organizing the life cycle of a product, from development through to testing and into production. Because products require maintenance and enhancements, the process is ongoing. In FS-PRO, change management centers on the objects, and occurs in Product Studio with its object administration functions.

Lock and Unlock

FS-PRO includes a reservation system to enable multiple product developers to work together safely on the same environment. If you have the necessary permissions, you can lock an object, reserving it for your exclusive use. All other users with access privileges for that object are restricted to read-only access. After you release the object to the repository by unlocking it, the object becomes available for use by other users who have the proper permission.

Publishing

Publishing is a critical element in change management, providing a mechanism for the orderly transfer of a product and its dependent objects between servers in the FS-PRO environment. When a product is published in FS-PRO, a JAR file is placed on the production or test server and is then ready to be deployed.

Normally, you only need to publish marketable products—the products that you sell in the marketplace. The lower-level products (for example, base products) aren't published, as they would serve no purpose in the Product Services repository.

> **i Note**
>
> If you make changes to base products or other low-level objects, these changes are automatically inherited by the high-level products. However, you must republish them if you want the changes to appear in the Product Services repository.

2.2 Product Modeler Concepts and Workflow

There are several major concepts used in the Product Modeler which you should understand before defining products:

Products Versus Product Objects

In the Product Modeler environment, a product is an object that represents a real world product, with all its features, data storage, and business logic. You create a product in the Product Modeler by assembling component, questionnaire, coverage, and other types of objects, into a product object. The coverages and components store values, rating formulas, and rules for business logic.

A product can be created from scratch, but more typically, to save time and effort, it uses existing product objects, inheriting their features. When the final details that make the product unique are added, you have a marketable product, which you publish to the product repository in Product Services, so that it can be available at Runtime.

Reference Products

Reference products provide a central repository for enterprise-wide data. Products access this data by linking to these reference products. By maintaining this shared data in reference products, only the reference products need to be republished.

To use reference products in base or marketable products, the product's system component `Config` must contain the details of the reference products.

Product Manuals

In FS-PRO, a completed product can essentially serve as its own product manual, containing marketing material and detailed information on product options and availability. You can choose to define this information and add it to a product, if you want to generate a product manual. The manual information can be exported in different formats and made available both for access by customer service representatives, agents, and brokers, and for display on intranets and web sites.

Predefined Components

You create a product by assembling and customizing various components. FS-PRO comes with a library of existing components that you can use as-is or modify to suit your company's needs.

Product and Coverage Inheritance

You develop a line of products in FS-PRO by using its inheritance feature, which allows you to create a new product or coverage based on an existing one. You then extend and customize the new product or coverage, building on previous work rather than starting from scratch. Inheritance provides the ability to easily maintain and implement changes to products, coverages, and families of products.

Product Rules

Product rules can be used to apply business logic, perform database lookups, or run data functions that provide default values, validations, and calculations.

Using *Rule Painter*, you create a rule by setting an attribute as a rule attribute. You then create a rule object and define the rule content.

Product rules are applied when an end user invokes them in either a component or questionnaire. These rules operate in the Runtime client. For example, you could create a "quick quote" product that contains a product rule that calculates a preliminary quote for a policy application.

eApps

eApps are a FS-PRO feature that enables you to encapsulate all the elements of a product, which can then be used to create an electronic version of the application that can be added to a Runtime solution and work immediately, without customization. eApps thus make it possible to reuse the same application for multiple products, thereby significantly reducing development time.

An eApp is essentially a structure that ties together a set of related objects, which consist of four basic parts: data definitions, questionnaires, screens, and rules. The eApp structure links and organizes these elements into a single entity that you can "drop in" to a Runtime solution.

User Defined Product Services

A user-defined product service enables you to encapsulate business logic in a rule that becomes a product service, callable from any enterprise application. All user-defined product services for a product are contained in a special purpose component named Services API.

After the product is published, the rule becomes accessible from enterprise applications.

User-defined product services are commonly used in products for implementing forms attachments, rating rules, and referral rules.

Design Time

A server where you use the Product Modeler to design an create products.

Runtime

A server where you operationalize a product.

Related Information

The Product Studio [page 30]
Working with Product Rules [page 318]
Using the Sample Content [page 26]

2.3 Understanding Roles and Responsibilities

Creating and maintaining products in FS-PRO mainly requires knowledge of your industry and familiarity with the Product Modeler interface. However, some technical support is essential in configuring and customizing the system, and in performing certain advanced activities.

This section describes the skills needed.

Product Management Group

FS-PRO was designed to allow any stakeholder in the product development process to create products. A product modeler, with FS-PRO training, should be able to:

- Design and create components, questionnaires, coverages, and products
- Design and create data definitions, questionnaires, and screens for eApp
- Add rules to products, coverages, and components
- Publish products
- Set access permissions for the objects the developer creates

For advanced activities, an analyst typically will need IT support, as follows:

Database queries To create database queries, knowledge of SQL (Structured Query Language) is required. However, a product developer with SQL skills can work with minimal support.

Data integration	Linking the products created in FS-PRO to other corporate systems. Data integration requires knowledge of the local IT infrastructure and legacy systems.

Administrator

The FS-PRO administrator is responsible for installing, configuring, and maintaining the environment. Generally, this person is a senior developer or systems administrator, someone who has the necessary permissions to set up and connect servers and databases, and to grant network resource privileges. See the Installation Guide and the Administration Guide.

2.4 Understanding the FS-PRO Sub-modules

FS-PRO is comprised of different function sub-modules.

The definition aspect of FS-PRO exclusively contains the following sub-modules:

Product Studio	Enables you to create applications and their objects (products, components, and questionnaires) and to organize them into a hierarchy.
Product Painter	Enables you to combine products, components, and questionnaires, to define their details, including attributes, values, and availability. Within a product, *Product Painter* also allows you to create eApps for use in applications.

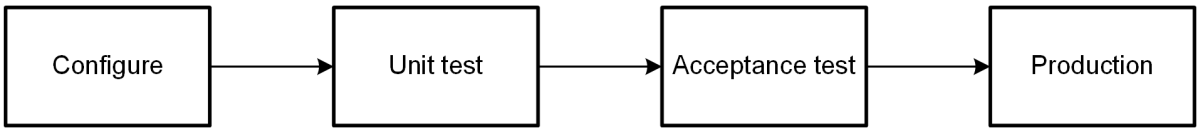
The design aspect of FS-PRO contains the following sub-modules:

Attributes tab	Allows you to create new attributes (data elements) for components and data definition objects.
Rule Painter	Enables you to define rules that provide default values, validate entries, and perform calculations, for values entered in Product Modeler or at Runtime.
Screen Painter	Enables you to design data-entry screens for adding values to components.

2.5 Product Content Lifecycle

In the SAP Product, Quotation, and Underwriting Management solution, insurance products are defined by a set of configuration elements which are in general called product content. Through the overall SAP Product, Quotation, and Underwriting Management landscape, product content follows a well-defined lifecycle.

Conceptually, the lifecycle of product content looks like this:



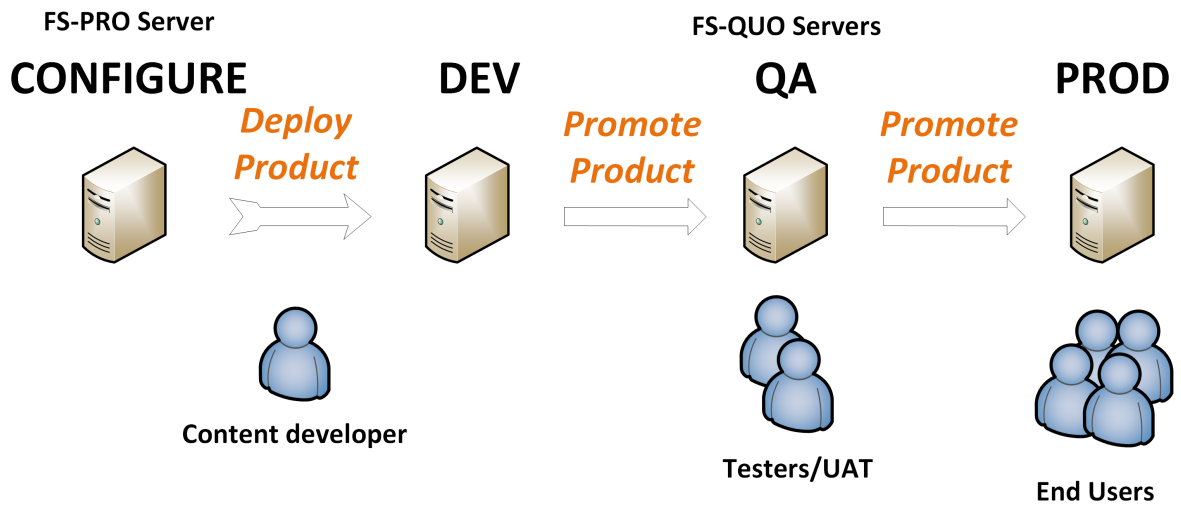
Configure	In this stage, a user (content developer, product manager) configures a product by defining its structure, business rules, eApps and data rows
Unit test	In this stage, a user (content developer, product manager) verifies the configuration of the product in a test environment.
Acceptance test	In this stage, multiple users (QA, acceptance testers) test the product configuration to ensure that it meets acceptance criteria.
Production	In this stage, end users (agents and brokers) use the product configuration in the quotation and underwriting process.

Each stage in the product lifecycle occurs on a different server. There are specific processes which are executed to move product content from one stage to the next.

To ensure that product content moves through the lifecycle stages reliably and predictably, automation tools can be used to execute and monitor the processes which move product content. Any automation tools that can execute HTTP requests can be used for this purpose, such as SAP CTS+, or open source applications such as Jenkins or Ansible.

Technical Perspective

The following diagram provides a more detailed view of the product content lifecycle:



Stage: CONFIGURE

In this stage, a content developer uses the Product Modeler web application, which runs on FS-PRO, to define product content

Stage: DEV

In this stage, a content developer verifies product configuration changes that they have made, using Fiori web applications deployed to FS-QUO and ABAP front-end server dedicated to development unit testing.

Process: Deploy Product

In this process, the content developer publishes and deploys the product to the DEV instance of FS-QUO. This process either be automated or performed manually.

For more information on the manual process, see Publishing a Product [page 147].

To automate deployment, two APIs are used:

1. Call the `Download Published Product` Operations API on the FS-PRO server to export the Product JAR.

   > **i Note**
   >
   > For additional control and roll-back, you can check in the downloaded Product JAR file to a version control system.

2. Call the `Deploy Product` Operations API on the FS-QUO server to import the product.

For more information, see Operations APIs

Optional: Synchronize the Database

If the product being deployed is an Application Data Model Product, call the `/service/syncProcessFlowstore.do` Operations API to synchronize database tables and columns with the data model configured in this product. This will create new tables and columns that are missing from the database. This step is not required if the required database changes are made by a DBA through a separate process.

If the product being deployed is a product with data model extensions, call the `/service/syncProductFlowstore.do` Operations API to synchronize database tables and columns with the data model configured in this product. This will create new tables and columns that are missing from the database. This step is not required if the required database changes are made by a DBA through a separate process.

For more information on synchronizing the database for the ADMP product, see Synchronizing the Database to all Flowstores. For more information on synchronizing the database for other products, see Synchronizing the Database to Flowstores for a Product.

Process: Promote Product

In this process, after the content has tested the product configuration in the DEV environment, the product is exported from DEV and deployed to the next stage QA environment. This process can be executed using an automation tool.

Stage: QA

In this stage, multiple testers ensure that the product configuration meets acceptance criteria by testing it in the QA environment, consisting of a FS-QUO server and an ABAP front-end server.

Process: Promote Product

In this process, the product is exported from QA and deployed to the final stage PROD environment. This process can be executed using an automation tool.

Stage: PROD

In this stage, the product configuration is applied in Fiori web applications accessed by end-users. It's 'in production'.

Additional Environments

In addition to the servers described above, additional environments can be set up for:

- User Acceptance Testing (UAT)
- Performance

Similar automated process can be set up to promote product JAR files from QA to these environments.

Related Information

Publishing a Product [page 147]

2.6 Understanding the Product Modeler User Interface

FS-PRO uses a single window to host the two main work areas: the Repository (Product Studio) and the modeling tool. These work areas exist side by side and you move between them by selecting tabs.

The Repository is a catalog of products and the various objects that comprise them. Inside the Repository, you create, organize, edit, and delete objects, as well as reporting on them and exporting/importing them. You can also search for objects and bookmark the ones that you use frequently, for quicker access.

When you open an object (for example, a product, a component, or a questionnaire) it appears inside a modeling tool tab, where you can specify the details of the object. If the object is a product or coverage, you can also assemble in other objects, in order to create a structure.

Logging in puts you in the Product Modeler window. By default, only the Repository (Product Studio) appears after you log in. The main elements of the Product Modeler window are described in the following list:

- The title bar at the top contains the global menus; these menus provide access to tools and are always available regardless of whether you are in the Repository or the modeling tool.
- Immediately below is the work area tab bar and below this, the work area menu. Only the work area tab visible is the *Studio* tab; the modeling tool tabs don't appear until you open repository objects. For each object that you access, a separate work area tab appears. These modeling tool tabs give you access to the internal structure of products and other objects, and allow you to assemble and edit them.
  The work area menu is context sensitive, changing according to whether the current work tab is for the Repository or the modeling tool.
- In the Repository, the navigation panel consists of two items:

Studio Tree panel	Located in the top left, it provides a structured view of all the libraries and objects in your FS-PRO environment
Search panel	Located in the bottom left, it is search tool for finding objects in the Repository. When a modeling tool tab is selected, the navigation panel changes.

- The work area on the right is determined by the work tab selected and consists of the detail area, detail tabs, and a context-sensitive toolbar.

If the Repository is selected, the detail panel tabs *Object List* and *Bookmarks* appear, and if you perform a search, the *Search Result* tab also appears. You use these tabs to access the objects that you want to work on.

- The status bar (at the bottom) shows the server name and your user name.

Applying the `Open` command to any object in the Repository causes a configurator work area tab to appear. These tabs show the name of the selected object and always appear to the right of the Product Studio tab. Up to five product tabs can be displayed. Selecting one of these tabs takes you into the configurator work area.

Selecting a Product Modeler tab causes the following changes in the work area:

- The Product Modeler menus appear.
- The navigation panel contains a tree (top left) that shows the structure of the object and that allows you to navigate within the object; the navigation panel also shows the *Inheritance List* (bottom left), which displays the object's ancestors (choosing an object in this list opens it in its own tab).
- Each modeling tool tab contains its own work area. The tabs in the detail area reflect various aspects of the object (such as the underwriter's manual, attributes, or values). Besides showing the name and version information for the object it contains, each modeling tool tab also displays an icon that indicates your permission level for the object.

A red flag icon appears in the tabs when you make changes in the *Values* tab, to remind you to save your changes. In all changed cells within the *Values* tab, a red triangle appears in the top-left corner.

In certain areas, such as the *Attributes* or *Values* tabs, popup menus are accessed by right-clicking the row heading or the column header.

The *Tools* menu is located in the top-right corner of the Product Modeler screen. This menu is available at all times. From it you can access several supporting tools, such as the *Object Modifier Type Manager*.

Related Information

The Tools Menu [page 25]
Row Headings and Column Headers [page 26]
The Product Studio [page 30]
Defining Objects [page 88]
Understanding Permissions and FS-PRO [page 52]

2.6.1 The Tools Menu

The *Tools* menu is located in the top-right corner of the Product Modeler screen.

This menu is available at all times. From it you can access several supporting tools, such as the *Object Modifier Type Manager*. All the items in the *Tools* menu are discussed in their own chapters in this guide.

2.6.2 Row Headings and Column Headers

In certain areas, such as the *Attributes* or *Values* tabs, popup menus are accessed by right-clicking the row heading or the column header.

2.7 Using the Sample Content

A library of sample content, including sample products, is provided out-of-the-box with the Product Modeler to help you quickly build your own inventory of customized marketable products.

The following sample libraries are located in the `System Repository` and are available for use to build your products:

SAP Auto Insurance Template Library	Contains the IFBC-related templates that are integrated with the Auto add-on. Includes the sample product for auto insurance, which you can copy into your company library and extend.
SAP Insurance Base Library	Contains the standards templates for insurance products.
SAP LnA Insurance Template Library	Contains the IFBC-related templates for Life line of business. Includes the sample product for Life insurance that uses MSG-PM as a rating engine, which you can copy into your company library and extend.
SAP Longterm Insurance Template Library	Contains the IFBC-related templates for Longterm line of business.
SAP Master Policy Library	Contains the IFBC-related templates for the Master Policy line of business.
SAP OData Library	Contains the eApp OData model.
SAP PnC Insurance Template Library	Contains the IFBC-related templates for Personal and Commercial (PnC) line of business.
SAP S4 Insurance Base	Contains the set of templates that are integrated to work with the S4 suite of products.
SAP Sample Content Library	Contains the IFBC-integrated objects that can be assembled into a product.
P and C Insurance Template Library	Contains the standard templates for risk-based products.

> ⚠ Caution
>
> Do not customize or extend any of the objects in the `System Repository`. Instead, duplicate them to the `Content Repository`.

For your convenience, the `Extended Underwriting Application Configuration` component and extended Data Definitions components (for example, `Extended DD <Object>`, `Extended DD Policy` and `Extended DD Contract`) are included during installation. They are already extended and can be used as is.

> → Tip
>
> Any objects in the `Content Repository` that are prefixed with `Extended` can be used as is, without further extension.

Sample products are placed in the Product Modeler `System Repository` during the FS-PRO installation process. The contents of the *System Repository* is read-only, which means that you will need to make copies of the sample products and then customize them. Use the *Duplicate* feature to quickly make copies and save them to the `Content Repository`, which is the home of all of your user-created Product Modeler objects.

The following sample products are available in the `System Repository`:

Auto	SAP S4 PR Motor Vehicle Insurance
LnA (Life)	SAP S4 PR Life Capital Life Capital 2009 92H0000S0002
Personal PnC	Sample Household Product Sample Household Product Traditional Chinese Sample Household Product Korean Sample Household Product Extended SAP S4 PR Best Protect Insurance
	> i Note > In the `SAP S4 PR Best Protect Insurance` product, the `SAP S4 HC Postal Code Domain` was replaced with the `SAP S4 HC Tariff Zone Domain` and was moved to the template level.
Master Policy	SAP Master Policy SAP Master Policy Korean SAP Master Policy Traditional Chinese
Commercial PnC	Mock Commercial General Liability Monoline
Homeowners	SAP S4 PR Homeowners Insurance (for internal use only)
Health	SAP S4 PR Health Wellness (for internal use only)

2.8 Browser Requirements for FS-PRO

The FS-PRO client is browser-based. The following browsers are supported for used with the Product Modeler:

- Microsoft Internet Explorer
- Microsoft Edge

> → Remember
>
> The URL for the Product Modeler instance must be added to the *Compatibility View Settings* in Internet Explorer, even if Microsoft Edge will be used.

Enabling Internet Explorer Mode in Microsoft Edge in Enterprise Mode

If your System Administrator has enabled an Enterprise Mode Site List for Internet Explorer in Microsoft Edge, you will need to add the site domains in Internet Explorer.

In Internet Explorer, go to *Internet Options* > *Security* > *Local Intranet* > *Sites* > *Advanced* and add the domain names provided to you by your FS-PRO System Administrator.

Manually Enabling Internet Explorer Mode in Microsoft Edge

If your System Administrator has not enabled an Enterprise Mode Site List for Microsoft Internet Explorer in Microsoft Edge, you will need to manually add the URL for your Product Modeler instance via the following steps:

1. Launch Microsoft Internet Explorer and ensure that the URL for your FS-PRO instance has been added to the list in the *Compatibility View Settings*.
2. Launch Microsoft Edge.
3. Go to *Settings* > *Default Browser*
4. Set *Allow sites to be reloaded in Internet Explorer mode* to *Allow*.
5. Choose *Restart*.
6. Add the URL for your FS-PRO instance to the *Internet Explorer mode pages* list.
7. Login to your FS-PRO instance.

> **i Note**
>
> The URL will be available in your *Allow sites* list for 30 days.

2.9 Logging in to the Product Modeler

The first step in using the Product Modeler is to log in to the web application.

Prerequisites

You will need a user name and password. These are issued by the FS-PRO system administrator.

If you require a password change, contact your system administrator.

Procedure

1. Open your web browser and go to `<pro_designtime_app_url>/csiroot/ii/pc/`.

   > **i Note**
   >
   > Microsoft Edge and Internet Explorer are the only web browsers that are compatible with the Product Modeler. The URL for the Product Modeler instance must be added to the *Compatibility View Settings* in Internet Explorer, even if Microsoft Edge will be used.

   The *Logon* screen appears.
2. Enter your user name in the *User Name* field .
3. Enter your password In the *Password* field.
4. Select *Log In*.
   The Product Modeler will open after a short delay.

3 The Product Studio

Logging in to the Product Modeler puts you in the Product Studio, which is the place where you create and access the objects of the Product Modeler environment.

Product Studio consists of the following objects:

- Folders
- Products
- Coverages
- Components
- Questionnaires
- Data definitions

You organize these objects hierarchically in the Product Studio using a tree view. You group objects logically in the tree using folders. The items in a given branch of the tree typically share similar characteristics, or are closely related in terms of function.

The products that you create in the Product Modeler are composed of interrelated objects: products, coverages, components, questionnaires, and data definitions. A marketable product is represented by a single product object, which may contain coverages, components, questionnaires, data definitions, and possibly other product objects.

Components

You use a component to store a group of related values that product or coverage objects use. You add or edit the values that the component contains.

For example, if you are assembling a policy product and it needs to record billing details, you create a component called `Billing`. Later in the modeling tool, you create the `Billing Method` attribute, from which the values "Cash," "Check," "Credit Card," and "EFT" derive as a result of querying the Billing component.

Questionnaires

A questionnaire is a specialized component. Its attributes are predefined, and its values consist of questions and their properties. At Runtime, the questions appear on the screen for the end user, whose answers are collected as data.

Data Definitions

A data definition object defines the metadata for a single entity that stores transaction data at Runtime. For example, a data definition named Customer could specify the details of a customer record. This data definition could in turn link to a child data definition named Address, specifying the various addresses the customer uses.

To function as part of a Runtime application, data definitions are published along with their product, and become extension tables in a flowstore.

While working in the Product Studio, you should also consider the following concepts:

Inheritance

A key concept to understand in the Product Modeler is inheritance, a feature that enables you to build on previous work by re-using and extending objects instead of recreating them from scratch. Inheritance is often described as a parent-child relationship, because the child inherits the features of the parent.

You assemble a product object from other objects, which you add directly to the product. Inheritance enables you to build on this previous work: when you create a new product, you can select an existing product on which to base the new product. The result is that the new product automatically uses all the objects contained in the existing product object—all its coverages, components, questionnaires, and products. You can then customize the new product by overriding, extending, or disabling inherited objects, while creating or adding further components, values, questionnaires, and products.

Inheritance reduces your maintenance effort: when you update an object contained in a product, your changes are automatically available to all the products that inherit that object.

Inheritance is most effective when you start by carefully planning the levels of your inheritance tree, defining what you want to achieve at each level, and keeping in mind how it benefits the next level. FS-PRO has identified an effective four-level inheritance model:

1. `Base Definitions` – A simple product object that contains features that analysis identifies as common to all objects in the environment. It is likely that none of these features are even industry-specific. For example, the base product might contain a component called Object ID, used for storing information required by the IT environment.
2. `Industry Definitions` – Inherits the features of the base product and adds generic industry-specific features. For example, basic underwriting rules for the insurance industry. At this level there can be multiple products, each laying the foundation for a specific business stream.
3. `Company Deviations` – Inherits the features of the industry definitions and adds product features specific to your company.
4. `Marketable Products` – The final, specific product that is published and used in your company's marketing channels for sale to potential customers.

> **i Note**
>
> The *Inheritance List* always shows a single branch in the tree; it doesn't show other branches. The closer an object is to the root of the inheritance tree, the more general its features; the farther out, the more specific the object's features.

Security

Product Studio is where you, as the owner of products and their objects, view and set their security properties. Permissions determine who can access or edit these objects in the Product Modeler. User authentication and authorization is managed at the Identity Provider (IdP) level. For information, see the Administration Guide.

Version Management

Version management is a process for organizing the life cycle of a product, from development to testing and into production, and then creating new versions, which restart the cycle. In the Product Modeler, this centers on the Product Studio with its versioning, history and audit trail functions.

Related Information

Product Studio Objects [page 32]
Managing Studio Tree folders [page 32]
Changing Inheritance for a Product [page 36]
Understanding Permissions and FS-PRO [page 52]

3.1 Product Studio Objects

The objects that appear in the Product Studio are represented by icons.

The icons are are grouped as follows:

Product Definition Objects	The "working" objects that you create, inherit, or assemble-in; different branches are available to different users, based on such factors as the place of the objects in the inheritance hierarchy, and the user's role.
System Objects	Predefined objects that come with the system and are required for it to function properly.

3.2 Managing *Studio Tree* folders

Studio Tree provides a hierarchy of folder types that enables you to organize your objects in a logical structure. In most cases, a given object can only be created within a specific folder type.

The most commonly used folder types are those that belong to the branch that stems from the Product Modeler folder:

- `Configuration Group` folder
- `Product` folder
- `Coverage` folder
- `Component` folder
- `Data Definition` folder
- `Questionnaire` folder

The `Configuration Group` folder is the most general. You use folders of this type to create a logical structure that reflects your product architecture, such as the various companies or divisions and the product lines within them. At lower levels in the structure where you want to locate products and their related objects, you must first add the specialized folders, such as product folders to contain product objects or coverage folders for coverage objects.

> → Tip
>
> For an example of how folders can be used to provide structure, explore the installed folder hierarchy.

Folders are important for controlling security. When you set the permissions for a folder, the settings cascade to all subfolders and objects contained in the parent folder.

Only folder owners and users that have been granted permission by the folder owner can view and access a folder.

Studio Tree folders have their own set of properties, which you can view. In the *Object List* tab, right-click the folder and choose *Properties*.

Related Information

Creating Folders in the Studio Tree [page 33]
Renaming Folders in the Studio Tree [page 34]
Deleting Folders in the Studio Tree [page 34]
Moving a Folder in the Studio Tree [page 35]
Viewing Folder Properties [page 35]
Product Studio Objects [page 32]
Understanding Permissions and FS-PRO [page 52]
Managing Permissions [page 56]

3.2.1 Creating Folders in the *Studio Tree*

All folder types are created and managed in the same way. However, a folder's type restricts where it can be created.

Procedure

1. Navigate in the *Studio Tree* to the appropriate folder .
2. Select the *New Folder* icon from the toolbar.
   The *New Folder* dialog appears.
3. Enter the name you want to give the new folder in the *Name* field.
4. Select the kind of folder you want to create from the *Folder Type* dropdown list.

   There are restrictions on the type of folder you can create. The choice of folder type may result in restrictions in the type of objects that can be created in that folder.
5. If you want to give the new folder the same permissions as its containing folder, leave the *Inherit Folder Permission* checkbox selected.

6. Choose *OK*.

Related Information

Product Studio Objects [page 32]

3.2.2 Renaming Folders in the *Studio Tree*

You can change the name of a folder in the *Studio Tree* at any time.

Procedure

1. In the *Object List* tab, right-click the folder that you want to rename and select *Properties*.
2. Enter the new name for the folder.
3. Choose *OK*.

3.2.3 Deleting Folders in the *Studio Tree*

You can delete a folder in the *Studio Tree* if it doesn't contain any folders or objects.

Context

> **!** Restriction
>
> Some FS-PRO system folders can't be deleted.

Procedure

1. Select the folder in the *Object List* tab.
2. Right-click the folder and select *Delete*

3.2.4 Moving a Folder in the *Studio Tree*

You can move a folder within the *Studio Tree*. You can't move a folder that contains subfolders, and a folder's type restricts where it can be relocated.

Prerequisites

Note the following restrictions about moving a folder:

- You can't move a folder that contains subfolders.
- A folder's type restricts where it can be relocated.

Procedure

1. Right-click the folder in the *Object List* tab and choose *Cut*.

   The folder still appears in its current location until you paste it.
2. Navigate the *Studio Tree* to the destination folder.
3. Select the *Paste Cut Objects* icon in the *Object List* tab.

   The *Clipboard* dialog appears, displaying all currently cut objects.

   > → Remember
   >
   > Items that are added to your clipboard during a session will remain in the clipboard until you log out and end your current session.
4. Select the checkbox beside the folder you want to move.
5. Choose *Paste*.

3.2.5 Viewing Folder Properties

Studio Tree folders have their own set of properties, which you can view.

Procedure

1. Select a folder in the *Object List* tab.
2. Right-click the folder and choose *Properties*.

Results

The following information is displayed:

Language	The language the folder appears in.
Name	The name of the folder. This field is editable.
Folder Type	The type of object.
Created By	The name of the user who created the folder.
Created On	The date and time the folder was created.
Last Modified By	The name of the user who last edited the folder.
Last Modified On	The date and time of the last modification to the folder.
My Permissions	The permissions for the item that apply to the user who is currently logged in.

3.3 Changing Inheritance for a Product

You can change the parent product that an existing product inherits from. The change inheritance function has been expanded for objects with product base type that has a parent product. That is, there is a value in the *Based On* field located in the *General* tab of a Product base type object *Properties* dialog.

Prerequisites

Typically, this procedure is used to insert extra levels in the product hierarchy or to update to a newer parent version. The new parent must meet the following criteria:

- You require Full Control permission to use this feature.

Context

> **i Note**
>
> Changing a product's inheritance can cause data loss and may affect existing applications that use the product.

> **⚠ Caution**
>
> You aren't able to undo any change inheritance action.

You can avoid the need for this procedure by carefully planning your object hierarchy before creating products.

An eligible parent product meets the following two conditions:

1. The product is a version of the current parent product, a peer product of the current product, or an inheritance of a peer of the current product.
2. The resulting tree doesn't have more than one node with *Parent Can Be Substituted* selected.

Inheritance change is disallowed when there are locked data value rows. When the current product or any of its inheritance has locked data value rows, an error message is displayed on attempting to change parent product, asking you to handle the locked data value rows before doing so. You are unable to change inheritance until all locked data value rows are committed or reverted.

Procedure

1. Access the product in the *Studio Tree*.
2. Right-click the product and choose *Properties*.
   The *Properties* dialog appears.
3. Beside the *Based On* field, choose the *Change Inheritance* icon.
   The *Change Inheritance* dialog opens, with the *Product Tree* displayed starting from the parent level. A checkmark is displayed in front of an eligible parent product. An X mark is displayed in front of an ineligible parent product. If there is no eligible parent, a message is displayed alerting you there is no eligible parent.
4. In the left hand panel, choose the eligible product to select it as the new parent product.
   After selecting a new parent product, if the current IPS value doesn't exist on the new parent tree, the current IPS value will be cleared. A warning message is displayed to alert you about the blank IPS value. You can select a new value. The *Change Inheritance* dialog is closed.
5. If there is orphaned data as a result of the change inheritance action, the system displays a confirmation prompt: `Change in inheritance/inheritance path will cause local data to be deleted. See log file for details. Do you want to continue?`
   a. If you want to open the log file and view the path to orphaned data, select the link.
   b. If you want to abort the task, select *No*.
      The warning message is discarded and you are returned to the *Component Properties* dialog where you are able to select another parent product.
   c. To continue with the task, select *Yes*.

   The *Properties* dialog is closed and the parent product of the current product is changed to the selected product.
6. Choose *OK* or *Apply* on the *Properties* dialog.
   Change Inheritance and IPS action will then be performed together.

Results

The *Properties* dialog refreshes and reflects the changes in the *Based On* field, the *Inherited From* field and *Inheritance List* tab the when reopened. A warning message is displayed as notification that all rules in the inheritance tree have to be rebuilt.

> **i Note**
>
> If change inheritance fails, a rollback of the product to its original state is performed and no data is lost.

3.4 Working with Objects

3.4.1 Accessing Objects

Procedure

1. Select the *Studio* tab.
2. Navigate in the *Studio Tree* to the folder that contains the object; it appears in the *Object List* tab
3. Alternately, if the object is bookmarked, select the *Bookmarks* tab.

   Every time the *Bookmarks* tab is selected it automatically refreshes its contents; only choose the *Refresh* icon if the tab has been open for some time.

3.4.2 Opening Objects

Procedure

1. Select the *Studio* tab.
2. Access the object by navigating to the folder that contains the object.

   There are two ways to access an object: standalone or, if it is assembled in a product or coverage, by opening the host object. The method you choose affects the appearance of the object and what you can do with it.
3. Open the object using one of the following methods:
   - Right-click the object and select *Open*.
   - Double-click the object.

3.4.3 Searching for Objects

Procedure

1. Enter all or part of the object's name in the *Name* field on the *Search* panel.

   The search isn't case sensitive. You can enter multiple search parameters by separating the items with a semi-colon (;). For example, `Company A;Company B;CompanyC`
2. To narrow the search to a specific object type, select a value from the *Type* dropdown list. To search on all objects, select All.
3. Choose *Search*.

Results

If successful, the objects appear in the *Search Result* tab. Searches return all objects that meet your query parameters, regardless of your permissions for those objects. A maximum of 250 results will be displayed. This behavior differs from the *Object List* tab, where objects for which you don't have Read permission are hidden from you.

3.4.4 Searching for Assembled Objects

Context

To search for objects assembled within another object, perform the following steps:

Procedure

1. Open the object where the search will be performed.
2. Select *File* > *Search for Assembled Object*.

   The *Search for Assembled Objects* dialog opens.
3. Enter a valid search term and select *Search*.

Results

All objects that match the search criteria are displayed.

The list of results can be downloaded by selecting *Download*.

3.4.5 Using Bookmarks

A bookmark is a shortcut to a *Studio Tree* object.

Context

You can create your own unique list of bookmarks, which appear in the *Bookmarks* tab and allow you to quickly access objects that you work on frequently. The *Bookmarks* tab can't be closed, and as long as it contains at least one bookmark becomes the default tab in the Product Studio.

You can bookmark any object, even those for which you lack `Read` permission, in which case the system grants you Read permission for the object.

Procedure

1. Select the *Object List* tab.
2. Navigate to the object.
3. Right-click and select *Add to Bookmarks*.

Results

A shortcut to the object is added to the *Bookmarks* tab.

3.4.6 Removing Bookmarks

When you no longer need to access an object regularly you may want to remove its bookmark in order to keep your *Bookmarks* tab uncluttered.

Procedure

1. Select the *Bookmarks* tab.
2. Select the bookmark that you want to remove.
3. Right-click and select *Delete Bookmark*.

3.4.7 Creating Objects

All Product Studio objects are created in much the same way. For every object type, you are restricted to creating the object in its own type of folder.

Context

For example, coverages can only be created in coverage folders, components can only be created in component folders .

For certain object types, you have the choice of creating them in two ways: "from scratch", or based on an existing parent (inheritance). This applies to products, coverages, and data definitions. Objects that can't be created from parents include components and questionnaires.

If you are utilizing Inheritance Path Substitution the following rules apply for a Reference Object:

- You will be prompted to confirm a change in the *Based On* field if it creates floating data in the node, child nodes or products composed of the node.
- Nodes with `Parent Can be Substituted = Yes` (and their children) aren't displayed.
- You must rebuild all rules to get the latest rule inheritance metadata before publishing.

Procedure

1. Navigate in the *Studio Tree* to the appropriate folder.
2. Select the *New Object* icon .
3. If you are creating a product-type object (from the *Modifier Type* dropdown list), select the *Primary Language* from the dropdown list.

4. Enter the name for the new object.

   When naming data definitions, you must not use spaces or start the name with a numeric character. Also, the only allowed special characters are _, #, and $.
5. If you want to create an object without a parent, perform the following steps:
   a. Enter a new object name in the *New Object* dialog.
   b. Select the object type you want to create from the *Modifier Type* dropdown list.

      It can be any type of object. Note that products, components and questionnaires offer multiple types.
6. If you want to create an object from a parent, perform the following steps:

   > **! Restriction**
   >
   > The parent object can only be a product base object.

   a. Enter a name for the new object in the *New Object* dialog.
   b. Select the object type you want to create from the *Modifier Type* dropdown list.
   c. Choose the *Inherit From* icon, which is located next to the *Based On* field.
      The *Object Search* dialog appears.
   d. If the object you want appears, select it; otherwise, search or browse to find the object and select it.
      The dialog closes and the object name appears in the *Based On* field (this only applies to objects with base type = product).
7. If you want to have the object use the security permissions of the folder in which you are creating it, accept the default setting of *Inherit Folder Permissions*.
8. Choose *OK*.

Results

The new object appears in the *Object List* tab.

3.4.8 Editing an Object's General Properties

You can alter an object's properties after it has been created.

Context

Because you can open and edit multiple objects at the same time, it's possible for you to be editing an object that is contained in another object that you have opened (for example, you have opened a coverage and a product, and the product contains that coverage); in this case, if the changes you make to one instance of the object aren't immediately reflected in the other, select the second object and choose the *Refresh* icon to update the view.

Procedure

1. Access the object in the *Studio Tree*.
2. Right-click it, and choose *Properties*.
3. Edit the information, as required:

Primary Language

Specifies the object's primary language.

The dropdown list contains all languages defined in the Administrative Console `AvailableLanguages` setting. The *Primary Language* displayed is the primary language at the time the object was created.

This value is only applicable to product base type objects.

For details, see System > Env > Globalization Setting.

> **i Note**
>
> When the *Primary Language* setting is changed for an object, the change propagates across all versions of an object.

Name

Specifies the name of the object.

If you have `Full Control` permission for the containing product, this field is editable.

Can have values?

This attribute is applicable to components only.

Controls whether the component uses values.

If you have `Full Control` permission for the component, these radio buttons are selectable.

> **i Note**
>
> If a component has values and you change this field to *No*, the values are hidden. Later, selecting *Yes* reveals the values.

Modifier Type

Identifies the object type.

Include custom classes

This attribute is applicable to products only.

Choose this option if you want the environment's custom classes to be included in the JAR file when the product is published

Based On

This attribute is applicable only if the object was created through inheritance.

Specifies the name of the parent object.

The *Change Inheritance* option is enabled for objects with `Base Type = Product` that has a parent product (i.e. with value in the *Based On* field).

Definition ID

Specifies the definition ID of the top-level object in a *Product Tree*. The top-level object can only be a marketable product or a reference object.

If the *Based On* field is specified, this field is automatically set to the value assigned to the parent object, but you can change the value.

Parent Can Be Substituted

Radio button option of *Yes* or *No*

- *Yes* - This triggers the ability for you to amend the Inheritance Path Substitution.

	• *No* - You can't amend the Inheritance Path Substitution.
Created By	Specifies the name of the user who created the object.
Created On	Specifies the date and time the object was created.
Last Modified By	Specifies the name of the user who last edited the object. Owners can change permissions and validate objects.
Last Modified On	Specifies the date and time of the last modification to the object.
Tree Path	Specifies the path to the object in the *Studio Tree*.
Inherited From	This attribute is applicable to products and coverages only. Specifies the immediate ancestor of the object.
My Permissions	Specifies the permissions for the object that apply to the user who is currently logged in.

4. Choose *OK*.

Related Information

Changing Inheritance for a Product [page 36]
Managing Permissions [page 56]

3.4.9 Viewing the Inheritance of an Object

From the *Studio Tree* you can view the inheritance of any product, coverage, or data definition.

Procedure

1. Access the product, coverage, or data definition in the *Studio Tree*.
2. Right-click and choose *Properties*.
   The object's *Properties* dialog appears.
3. Select the *Inheritance List* tab.
   At the top of the inheritance tree, the object appears; its parent objects appear below it.
4. Choose *OK*.

3.4.10 Deleting an Object

You can delete an object from a product, coverage or contract if there are no local data value rows defined and there is no selection within the product, coverage or contract.

Context

> → Tip
>
> If you want to delete a component from a product, coverage or contract, you first need to remove all local data value rows and all local selection that was specified within the component. Otherwise, attempting a deletion will result in an error message indicating the local data that is preventing the deletion.

A Reference Object with a *Parent Can Be Substituted* option can be deleted only if the conditions are met.

A Reference Object with the *Inheritance Path Substitution* option can be deleted only if the conditions are met.

Procedure

1. Navigate to the appropriate folder in the *Studio Tree*.
2. Right-click the object in the *Object List* tab and choose *Delete*.
3. Choose *Yes* to confirm your choice.
4. If you are deleting a product and if servers are configured to receive the product's published JAR files, the *Delete Products* dialog appears, showing a list of the servers. To delete the product and its JAR file from the current server and all the configured remote servers, choose *Delete*.
5. When the process is complete, choose *Close*.

Results

The object is deleted and all of the object's association are severed.

If the deletion isn't immediately reflected in the tree, choose the *Refresh* icon to update the view.

3.4.11 Moving an Object

You can use the cut and paste feature to relocate an object.

Procedure

1. Navigate to the appropriate folder in the *Studio Tree*.
2. Right-click the object in the *Object List* tab and choose *Cut*.

   The object still appears in its current location and will do so until you complete the procedure.
3. Navigate to the destination folder, which must be of the appropriate type.
4. Choose the *Paste Cut Objects* icon in the *Object List* tab.

   The *Clipboard* dialog appears, displaying all currently cut objects.

   > → Remember
   >
   > Items that are added to your clipboard during a session will remain in the clipboard until you log out and end your current session.
5. Select the checkbox beside the object that you want to move.
6. Choose *Paste*.

3.4.12 Locking an Object

You can lock an object for which you have Write or Full Control permission. When you lock a standalone object, it is reserved for your use.

Context

All other users are restricted to view-only access, regardless of their permissions for the object. The exception to this is if the object is contained within a product. In this case, users with `Full Control` permission for the product who access the object from within the product can make changes; these changes remain local to the product.

For example, component A is part of product X; you access component A standalone and lock it; when `Full Control` users of product X open component A from within the product, they can edit or add attributes and values. However, these changes only appear within product X. The standalone version of component A remains locked for your use, and is unchanged.

Objects available for locking have a blank *Locked By* column. When you lock an object, your user name appears in the *Locked By* column. An object that you lock stays locked between sessions.

> **! Restriction**
>
> You can't lock folders.

Procedure

1. Ensure that the object is unlocked.
2. Right-click it and choose *Admin* > *Lock*.

Results

The object is locked and your user name appears in the *Locked By* column.

3.4.13 Unlocking Objects

You can unlock single objects or multiple objects.

Context

Administrators (those with the role `SE_Admin_Role`) can unlock any object at any time.

Procedure

1. If you want to unlock a single object, right-click it and choose *Admin* > *Unlock*.
   Your user name is removed from the *Locked By* column.
2. If you want to unlock multiple objects, perform the following steps:
   a. Go to *File* > *Unlock Objects* in the Product Studio..
      The *Unlock All* dialog appears. By default, all your locked objects are selected.
   b. Select the checkboxes of the objects that you want to keep locked.
      The dialog indicates the total number of objects checked in.
   c. Choose *Unlock*.

3.4.14 Overriding the Reserved Status of an Object

Procedure

1. Go to *File* > *Unlock All* in the Product Studio.
   The administrative version of the *Unlock All* dialog opens.
2. From the *Locked by* dropdown list, select the user ID that the locked objects belong to.
   By default, all the reserved objects for that user ID are selected.
3. Select the checkboxes of the objects that you want to keep locked.
4. Choose *Unlock*.
   The dialog indicates the total number of objects unlocked.

3.4.15 Changing the Ownership of an Object

Administrators (those with the role `SE_Admin_Role`) can re-assign an object to a new owner.

Context

If the object is locked, re-assigning ownership doesn't affect the lock or the permissions of the user identified in the *Locked By* column.

Procedure

1. Right-click the object in the *Studio Tree* and select *Admin* > *Change Owner*.
   The *Change Owner* dialog appears.
2. Select the new owner from the dropdown list.
3. Save your changes.

Results

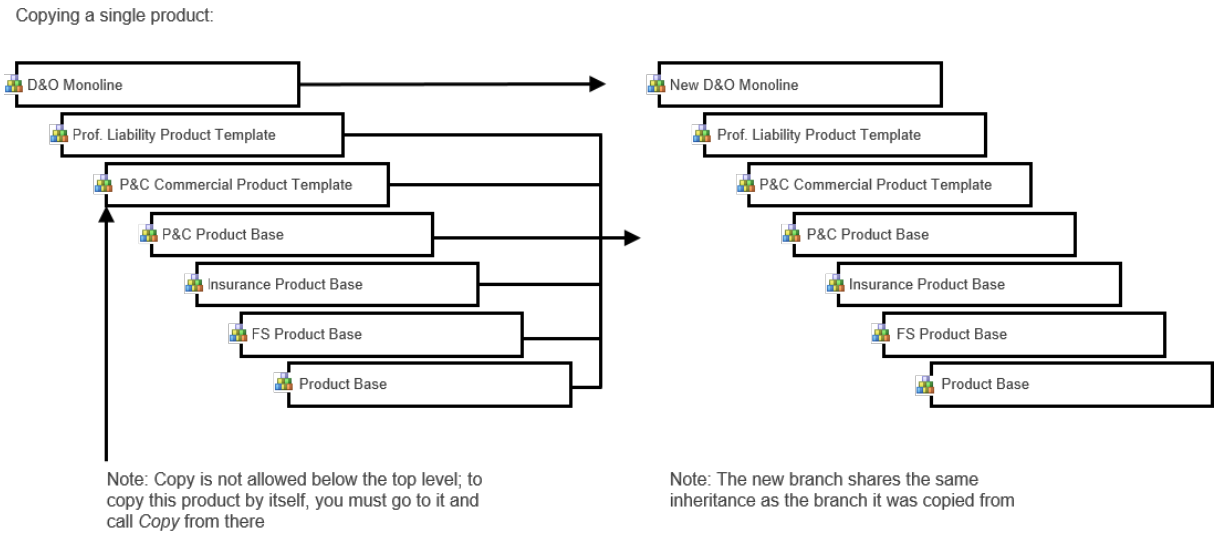
The new user name appears in the *Owner* column in the detail panel.

3.4.16 Copying Products or Coverages

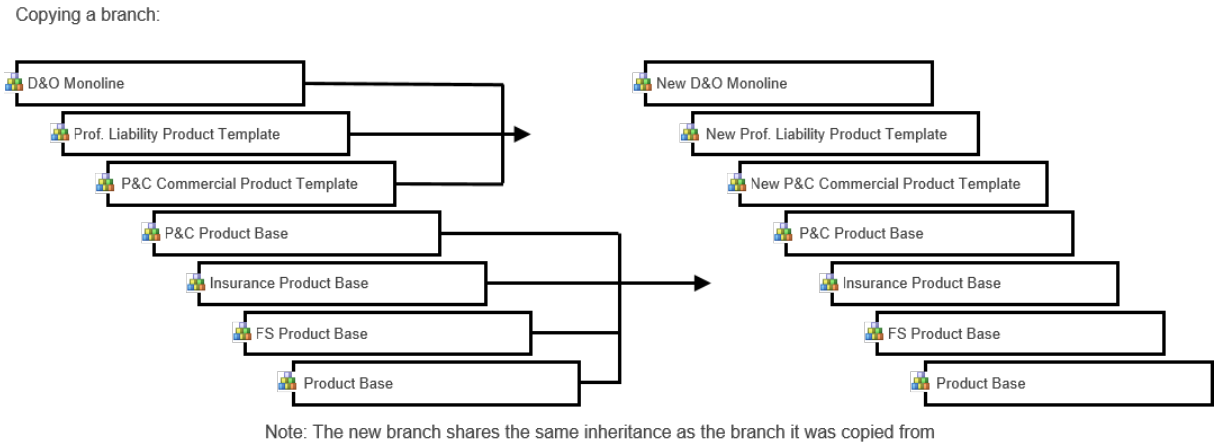
Typically, you create a marketable product by using inheritance. However, there may be cases where you want to create a new marketable product that is similar to an existing one, but without having to inherit the future changes that will be made to that instance of the original product. In this case, you can `Copy` the product, and all or part of its inheritance hierarchy (the entire "family" of products, or just a branch of that family).

Context

In the following diagram, a top-level product is copied:

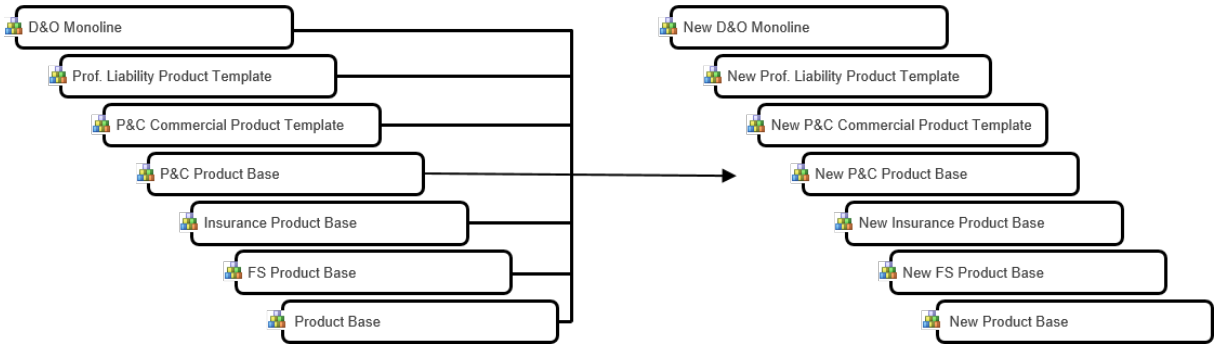


The next diagram shows copying a branch of the hierarchy, resulting in a new, separate branch (note that the copy must start at the top-level product):



Finally, you can copy the entire inheritance hierarchy, making the products in it completely free of the original family's inheritance:

Copying an entire product family:



With `Copy`, aside from the name and version information, the resulting products is initially identical to the original, having the same structure and referencing the same objects as the original. You can also copy coverages in this way.

The new products and the originals are separate: structural changes in one doesn't affect the other. However, changes to the inherited or referenced objects affect both products. For example, if Product A inherits Template A, and you copy Product A to create Product B, any changes to Template A affect both Product A and Product B.

> **i Note**
>
> If the product or coverage contains locked data value rows, when you attempt to `Paste`, a list of the rows and the users who locked them appears. Locked rows don't stop you from copying. However, only the committed versions of the rows are copied. The new product or coverage starts off with all data value rows unlocked.

Before using the `Copy` command for a product or coverage, you should understand its effect in order to determine whether it suits the needs of your project. Note that `Copy` always uses the selected version of the product or coverage.

Copying a product copies the following information:

- Content from the *Manual* tab
- Committed local attributes, data value rows, screens, and rules
- Local questionnaire objects
- Locally assembled objects in the *Product Tree*

In the copied products, the following attributes are preserved from the original product:

- XML tag names
- The history and audit trail of the source product
- The locked versions of data value rows
- PCDs, both system-generated and user-created, are replaced in the target with newly generated values

You must manually change the following attributes in the copied product:

- You must update the name in any XPath that references the product name.
- You must update the PCD in any XPath that references a PCD.

For a copied product, the new product starts at version 1.0

Use the following procedure to copy a single product or coverage, or a "family". You can copy any version of the source object. The `Copy` command requires Read permission or better for the source object. In all cases, the newly created objects has a version of 1.0 D.

Procedure

1. Navigate In the *Studio Tree* to the appropriate product folder or coverage folder .
2. Right-click the product or coverage in the *Object List* tab and select *Copy*.
   The *Copy Object* dialog appears, displaying the selected object, as well as its ancestors.
3. Select the checkbox for each ancestor that you want to include in the Copy operation.

   Gaps aren't allowed in the chain of objects that you select. For example, if the hierarchy shows five products, you can't select products 1, 2, 4, and 5, leaving out product 3; doing so would break the inheritance structure.
4. Choose *Copy*.
   The *Copy Object* dialog closes.
5. If necessary, navigate to the target product folder or coverage folder.
6. Choose the *Paste Copied Objects* icon in the *Object List* tab toolbar.

   The *New Object* dialog appears.
7. For each object in the dialog, enter a unique name.
8. Select the *Inherit Folder Permission* checkbox if you want to give the object's folders the same permission settings as in the original object.

   Starting from the bottom of the hierarchy, you can still deselect checkboxes to exclude objects from the paste operation.
9. Choose *Paste*.

Results

A progress bar appears while the `Paste` command runs. The copied objects appears in the *Object List* tab.

> **i Note**
>
> If the new product is created by copy and pasting from another product, the source product's primary language setting is copied over.

3.5 Understanding Permissions and FS-PRO

In FS-PRO the relationship between a user and any *Studio Tree* folder or object is governed by a permission assignment. The assigned setting determines what actions the user is allowed to perform on the folder or object.

These actions range from viewing the object without being able to make changes of any kind, all the way up to deletion. The permission categories are as follows: `Read`, `Write`, and `Full Control`. If a user doesn't have a minimum of `Read` permission for a folder or object, that object is hidden from the user.

Permissions enhance the product development process. By assigning permissions thoughtfully, the project leader can ensure that all team members can access the product simultaneously for development, yet when necessary a senior member can lock the entire product in order to make structural changes, such as importing objects, or to perform global functions.

> **i Note**
>
> Product permissions don't inherit. When you create a product through inheritance, the permission settings in the child product are empty.

The user who creates an object is its owner. A user's permissions for an object are assigned by its owner. However, the owner doesn't assign permissions directly to users, but rather to roles (users are assigned one or more roles in FS-PRO, based on job function). When the owner assigns a permission to a role, all users belonging to that role are affected. If the owner of an object assigns `Full Control` to a role, users belonging to the role have the same level of authority as the owner.

> **i Note**
>
> Certain administrative roles in the system have the authority to override the owner and other `Full Control` users. This is useful, for example, when an object is locked and needs to be released but the owner is unavailable.

Permissions are assigned from the *Studio Tree* by accessing the *Properties* dialog of the folder or object. Setting permissions from a folder's *Properties* dialog offers the option of assigning the settings to all objects and subfolders contained therein, thus saving time and reducing error.

> **i Note**
>
> The context in which you access an object can affect your permissions, in effect overriding the original permissions you were assigned for it.

Permissions take effect in three different contexts: in the *Studio Tree*, when defining a standalone object, and when defining a product.

Permissions at the *Studio Tree* Level

In the *Studio Tree*, permissions have the following effect:

Read	Navigate to and open any folder

	Open any object for viewing, including products View the properties of a folder or object
`Write`	All actions allowed by `Read` permission Lock or unlock any object, including products Use `Copy` for a product object
`Full Control`	All actions allowed by `Write` permission Create, rename, or delete folder Version a product object Copy a product object Reassign ownership of an object Edit the values in an object's *Properties* dialog Change a product's inheritance (assigning a different parent)

Permissions at the Object Level

In the modeling tool, permissions for an object opened standalone have the following effect:

`Read`	View all attributes, values, and other properties of the object
`Write`	All actions allowed by `Read` permission Add, import, edit or delete values Add content to the *Manual* tab Lock the object
`Full Control`	All actions allowed by `Write` permission Add, edit, or delete attributes Add sub-components

Permissions at the Product Level

The list below describes the permissions for a product object opened in the Product Modeler.

`Read`	Navigate to and open any folder View any object Test rules Generate product specification documents
`Write`	All actions allowed by `Read` permission Add values Override data value rows Edit the *Manual* tab

	Define and test rules
	Publish and deploy the product
`Full Control`	All actions allowed by `Write` permission
	Version a product object
	Copy a product object
	Anchor data value rows
	Assemble objects to the product
	Move objects up or down in the product structure
	Delete or undelete objects from the product
	Add folder to the product structure
	Add local attributes to components
	Delete inherited or common attributes

> **i Note**
>
> When you open a product, the permissions for that product override the permissions of the objects that it contains.

Related Information

Overriding Permissions [page 54]
Managing Permissions [page 56]
Defining Permissions [page 57]
Removing Permissions [page 58]
Creating a New Version of an Object as a Clone [page 58]
Creating a Version of an Object as a Clean Inheritance of its Parent Product [page 60]
Opening Previous Versions of Objects [page 61]
Viewing Object Histories and Audit Trails [page 61]

3.5.1 Overriding Permissions

In certain contexts permissions can be overridden, in some cases permanently, in others only locally.

Product Level Permissions Override Standalone Permissions

Permissions assigned to a non-containing object (for example, components, questionnaires, data definitions) affect only that object. Permissions assigned to folders, or to objects that can contain other objects (products or coverages) affect its contained objects as well, overriding locally the permission settings assigned to those objects.

Consider this example: you have Write permission for component A, but you have `Full Control` permission for Product X; if you assemble component A to Product X, and then access it in the product, you have `Full Control` permission for component A, but only within the product. All changes that you make to the component are local to the product; they don't appear in the standalone version of the component. However, any product that inherits from product X receives this local version of component A.

Folder Permissions can Overwrite Child Permissions

When you edit the permissions for a *Studio Tree* folder, the changes affect only the folder, unless you select one of the options in the *Properties* dialog. The effect is as follows:

Reapply all permissions to children	The permissions in the current folder, both existing ones and changes, overwrite the permission settings in all subfolders and objects currently in the folder.
Apply incremental permission changes to children	Only changes to the permissions in the current folder are applied to subfolders or objects in the current folder.

> ⚠ Caution
>
> After applying your changes with *Reapply all permissions to children*, you can't undo the operation: you should be certain that you want to overwrite the permission settings of all contained folders and objects.

When you cut-and-paste an object into a folder, the object retains its permission settings.

Example: How Folder Permission Options Affect Subfolders/Objects

The following example illustrates the effect of the two options available in the *Folder Properties* dialog.

A folder contains an object and a subfolder, with permissions assigned as indicated:

Object	Role A	Role B
Folder1	`Full Control`	`Write`, `Read`
Subfolder1	`Write`	
Object1	`Write`	

A user opens the *Folder Properties* dialog for Folder1, adds Role C to it, and assigns Write permission.

If the user selects *Reapply all permissions to children* and chooses *Apply*, the result is as follows:

Object	Role A	Role B	Role C
Folder1	`Full Control`	`Write`, `Read`	`Write`

Object	Role A	Role B	Role C
Subfolder1	`Full Control`	`Write`, `Read`	`Write`
Object1	`Full Control`	`Write`, `Read`	`Write`

If the user selects *Apply incremental permission changes to children* and chooses *Apply*, the existing settings are unchanged and only the new settings are added:

Object	Role A	Role B	Role C
Folder1	`Full Control`	`Write`, `Read`	`Write`
Subfolder1	`Write`		`Write`
Object1	`Write`		`Write`

> **i Note**
>
> If the user selects both options, the effect is the same as selecting only *Reapply all permissions to children*.

3.5.2 Managing Permissions

If you are the owner or a user with `Full Control` permission for the object or folder, you can define or change its permissions; in the case of a folder, you can optionally change those for all the objects and subfolders contained by the folder.

Context

> **⚠ Caution**
>
> The permissions that you define for a folder can change the permissions for all folders and objects within the folder. Use extra care when the folder already contains objects.

Any user can view the permission settings of any folder or object at any time.

Procedure

1. Select a folder in the *Studio Tree*.
2. Go to the *Object List* tab, right click the object and choose *Properties*.
3. Select the *Permission* tab.

Roles that have permissions for the selected item appear in the *Selected* list.

4. Select a role from the *Selected* list.

   The permissions assigned to that role are indicated by check marks in the *Permissions* section.

3.5.3 Defining Permissions

If you are the owner or a user with `Full Control` permission for the object or folder, you can define its permissions; in the case of a folder, you can optionally change those for all the objects and subfolders contained by the folder.

Context

> ⚠ Caution
>
> The permissions that you define for a folder can change the permissions for all folders and objects within the folder. Use extra care when the folder already contains objects.

Any user can view the permission settings of any folder or object at any time.

Only new permissions (that is, those that you add) apply to contained folders and objects; permissions that you remove from the folder don't affect its contained folders and objects.

Procedure

1. Select the appropriate roles from the list.

   You can use `Ctrl` + `click` or `Shift` + `click`.
2. Choose *Add* to copy the chosen roles to the *Selected* list.
3. Choose one or more of the roles in the *Selected* list.
4. Select the permissions for the selected roles from the *Permissions* section.
5. Choose the following options to define permissions for a folder:
   - To have the new permissions apply to all subfolders and objects currently in the folder, select *Reapply all permissions to children*. Note that selecting this option overwrites the permission settings of all contained folders and objects.
   - To have only the new permissions apply to the folder and all subfolders and objects in the folder, select *Apply incremental permission changes to children*.

   If you select both options, the effect is the same as selecting only *Reapply all permissions to children*.
6. Choose *Apply*.

3.5.4 Removing Permissions

You can remove permissions from folders or objects.

Context

In a folder, the *Reapply all permissions to children* option only affects new permissions that you add; permissions that you remove from the folder don't affect its contained folders and objects.

> ⚠ Caution
>
> If you aren't the owner of the object, take care not to remove your own `Full Control` permission for it.

Procedure

1. Select the role description in the *Selected* section.
2. Choose *Remove*.

Results

The permission is removed.

3.6 Versioning Objects

3.6.1 Creating a New Version of an Object as a Clone

You are able to create a new version of an object as a clone of the selected object.

Context

Versioning an object makes a copy of that object and increments the version number. For example, an object that already had two versions would, after versioning, be labeled 3.0. A previous version can still be accessed and worked on from the *Studio Painter*.

When a new version of an object as a clone is created the following criteria apply:

- Version number of the object is incremented by 1.
- Each committed local data value row is cloned to the new version, as committed local data value row, with the same PCD as in source (UUID will be different).
- Each committed locally overridden data value row is cloned to the new version, as committed locally overridden data value row, with the same PCD as in the source.
- Each locally locked new data value row isn't cloned to the new version.
- Each locally locked existing data value row is cloned to the new version with the latest committed changes, as committed local data value row, with the same PCD as the source.
- Each local rule in the data value row gets a new rule ID.
- Rule inheritance:
  - Each overridden data value row except rules is cloned to the new version with the same PCD as in source, with the rules remain inherited.
  - Each deviated rule in the data value row is cloned to the new version.
- Each screen in the data value row gets a new screen ID.

Procedure

1. Navigate the *Studio* tab to access the object in the *Object List* tab.

   > → Tip
   >
   > You don't need to lock an object to version it.

2. Right-click the object and go to *Admin* > *Create Version*.
   The *Create Version* dialog opens.
3. If you want to create a clone with a clean inheritance of the selected version, select the *Do not clone local data* checkbox . This means that the local data belonging to the object will not be copied.
4. At *Please choose a version* field, select a version from the dropdown list.
5. Select *Create*.

   A *Create Version* task is sent to the *Scheduled Activity Manager*.

   The *Create Version* dialog closes and the version number is incremented by 1.

   The *Create Version* task is sent to the *Scheduled Activity Manager*.

   The *Create Version* notification is displayed to user.

Results

A confirmation prompt is presented if there are locally locked data value rows. A list of objects containing the locked data value rows is displayed.

Next Steps

Before publishing a versioned (cloned) product, you must build its eApps, screens, and rules.

Related Information

What Happens During a Product Import? [page 74]
Locking Data Value Rows [page 118]

3.6.2 Creating a Version of an Object as a Clean Inheritance of its Parent Product

Procedure

1. Navigate the Product Studio to find the object.
2. From the *Studio* tab, access the object in the *Object List* tab.

   > → Tip
   >
   > You don't need to lock an object to version it.

3. Right-click the object and go to *Admin* > *Create Version*.
4. Select the checkbox for *Do not clone local data* to create a clone with a clean inheritance of the selected version.
5. Select *Create*.
6. The *Create Version* task is sent to the *Scheduled Activity Manager*.

   While the *Create Version* task is Pending or In Progress you are able to continue working. Until the *Create Version* task is complete you won't see the new version in the Product Modeler. However, you are still able to access and edit any version of the object.

   If you modify the selected version after creating the versioning request but before the scheduler finishes the task, the changes will be included in the new version that is being created.

   When the *Create Version* task completed with errors, the following actions take place:

   - A status notification is displayed. No new version for the selected object is created
   - A *Create Version* task will complete with errors if there are errors with the scheduling queue or if the object gets deleted before the *Create Version* task starts.

   The *Create Version* notification is displayed to user.

Results

The newly created version is an empty instance of the object with only inherited data.

3.6.3 Opening Previous Versions of Objects

In the *Studio Tree*, you can open all previous versions of an object (that is, 1.0, 2.0, or 3.0).

Procedure

1. Perform one of the following actions:
   - Navigate to the object in the *Studio Tree*; the object appears in the *Object List* tab
   - Search for the object; if found it appears in the *Search* tab

   In either tab, if the object has previous major versions, a white triangle appears beside its icon.
2. Select the white triangle.

Results

The previous versions appear beneath the current version. You can open one or more of the displayed objects; each will appear in its own tab.

3.6.4 Viewing Object Histories and Audit Trails

From the moment an object version is created, all the changes made in each version are recorded in an audit trail.

Context

The audit trail records all actions users perform on a product, with the exception of those performed in the *Manual* tab or the following:

- Changes caused by the `Copy Product` or `Create Version` commands
- Changes caused by the `Build` command
- Changes caused by inheritance
- Changes to the order of objects in the *Product Tree*

- Changes to the attributes of individual data value rows (are tagged with the generic message "updated value row")
- Changes to the `Default Value`, `Tag Name`, `Node Type`, `Extension`, and `Lookup Path` attributes
- Changes to the order of attributes in a component
- Changes to `Questionnaire View` objects
- Changes to question mandatory rules in `Questionnaire Model` objects
- Changes to the properties of individual questions
- Changes to the order of questions in a questionnaire
- Changes to `XML Definition` objects
- Publishing related actions (such as: publish, deploy, index, and document generation)

> **i Note**
>
> Locking a data value row and then editing an object that is embedded in the row (for example, a screen object) causes two entries to appear in the audit log, as in the following example: The first entry reflects that the data value row was edited. The second, where *Action* column contains a reference to the change made, for example: `Add Screen`. This occurs because the original screen object was cloned in order to provide a copy to edit safely.

From the *History* dialog, you can also create a detailed product report for any version of a product. The features of this report are identical to a standard product report.

The *Audit Trail* dialog has two separate modes, depending on where you launch it from. If from the *Tools* menu, you get the audit trail for the entire system, and all the tabs show audit data from across the system; if from within a product or component, you get the audit trail for that object alone.

The following table outlines which tab on the *Audit Trail* dialog contains information for a given object type :

Configurator Object Type	Studio tab	Object tab	Attributes tab	Values tab	Questionnaires tab	Schema tab
deleted from Studio	X					
product root		X				
component root		X	X	X		
questionnaire root		X				
data definition root		X				
component		X	X	X		
folder		X				
data definition		X				
table		X	X	X		
column		X	X	X		

Configurator Object Type	Studio tab	Object tab	Attributes tab	Values tab	Questionnaires tab	Schema tab
eApp questionnaire		X			X	
questionnaire model		X			X	
questionnaire base		X			X	
eApp objects		X	X	X		
XML Definition						X

> **→ Tip**
>
> To see a complete list of all the actions audited for a given functional area in the Product Modeler, such as data value rows, select the appropriate tab in the *Audit Trail* dialog and open its *Action* dropdown list.

Product import operations, as well as data value row imports, are audited for the same set of actions as your manual operations: the actions caused by imports appear in the audit trail, with additional text to indicate that they resulted from an import; for example: `Added value row by import product "ABC GL."`

> **i Note**
>
> The addition of question text as part of a product import is not included in the audit trail, but is included if the question text is added manually using the *Product Studio*.

When you are auditing XML Definition objects,

- The *Studio* tab records all `XML Definition` object deletions from the Product Studio.
- The *Object* tab records both the addition and deletion of an `XML Definition` object into a *Product Tree* structure.
- The *Schema* tab displays the following `XML Definition` object actions:
  - *Add Element* – Inserting a new element in the Schema of an `XML Definition` object.
    - Message: Added element `<element name>`.
    - Includes elements added using the `Import XSD` function.
    - Includes copied and pasted elements.
  - *Delete Element* – Deleting an element in the Schema of an `XML Definition` object.
    - Message: Deleted element `<element name>`.
    - Includes elements deleted when using the `Import XSD` function (the `Import XSD` function deletes all elements in the current schema and replaces with the elements in the imported file).
  - *Edit Element* – Editing any element property field in the Schema of an `XML Definition` object (not including cutting or pasting).
    - Message: Updated element `<element name>`.
    - Logging is at the element level only. The system doesn't track old and new values of individual field properties.

Procedure

1. In the *Object List* tab or the *Bookmarks* tab, right-click the object and select *Admin* > *History*.
2. To view or edit a version, choose *Open*.
3. Perform one of the following actions:
   - Go to *Tools* > *Audit Trail*. This option gives you access to the audit trail for all changes in the system.
   - In the *Bookmarks* tab or *Object List* tab, right-click an object and go to *Admin* > *Audit Trail*.
   - In the *Bookmarks* tab or *Object List* tab, right-click an object and go to *Admin* > *History*. In the *History* dialog that opens, choose *View Audit Trail*. This option displays the changes for a specific version of an object.
   - In a product, in the *Product Tree*, right-click an object, and choose *Audit Trail*.
   - In a component, in the *Values* tab, right-click the row header, and choose *Audit Trail*.

   The *Audit Trail* dialog opens. The dialog is divided into several tabs, each of which pertains to changes made in a specific part of FS-PRO, such as questionnaires, component values, or component attributes. The tabs are enabled or disabled based on the type of the currently selected object. The audit table lists the individual audited actions for the currently selected tab.
4. To filter the audit trail, you change one or more of the parameters in the Common area, and/or the parameters unique to the current tab, and choose *Filter*. The Common parameters are as follows:

User	Select name of the user who performed the action
Object	Enter the name of the object the action was performed on
From/To Date/ Time	Use these parameters to construct a time frame within which the action was performed
Full Path	By default displays the full path to the object within the *Product Tree*; select the checkbox to include audit records for the object's children

The parameters exclusive to the individual tabs are as follows:

Tab	Parameter	Description
Studio	`Action`	The only option here is *Delete Object*; it allows you to filter for objects that have been deleted at the Studio level.
Object	`Action`	A list of actions that can be applied to the selected object type.
Schema	`Add Element`	Inserts a new element in the Schema of an `XML Definition` object.
	`Delete Element`	Deletes an element in the Schema of an `XML Definition` object.

Tab	Parameter	Description
	`Edit Element`	Edits any element property field in the Schema of an `XML Definition` object (not including cutting or pasting).
Attributes	`Action`	A list of actions that can be applied to the attributes.
	`Attribute Name`	The name of the attribute. The name isn't case sensitive.
Values	`Action`	A list of actions that can be applied to data value rows.
	`PCD`	The PCD of the data value row.
	`PCD Version`	Enter a number greater than zero and less than or equal to ninety-nine.
Questionnaires	`Action`	A list of actions that can be applied to questionnaires.
	`Question Text`	All or part of the actual question.
	`Column ID`	The name of the column, in the format `<table>.<column>`. The name isn't case sensitive.
	`PCD`	The PCD of the question data value row.

Results

For more information on an audit item, select the + in its row header.

To export the audit data (using the current filter) from the selected tab into a CSV file, select *Filter* to show the data and then select *Export*. If you don't select *Filter* first, no data will be selected for export.

3.7 Generating Product Reports

You can generate detailed reports on any published product object. The reports are known as `Product Specification Documents`, and are derived from views, which are sets of product characteristics.

Context

Several standard views for the generated reports are provided. You can also create custom views, if you have the necessary permission.

The standard views are as follows:

`Business View`	Includes all the information contained in the *Manual* tab of each object in the product. Excludes information from the *Attribute* and *Values* tabs.
`Technical View`	Includes *Attribute* tab information and excludes *Values* tab information. Excludes *Manual* tab information.
`Data View`	Excludes *Manual* and *Attributes* tabs information, but includes data.
`Rules View`	Includes all rules in the product.
`Forms View`	Includes all data and rules for the forms in the product.

By default, all users can generate `Product Specification Documents` using the standard views. An Administrative Console setting controls this feature. If you can't generate reports, your administrator can add/edit the necessary setting: in the Administrative Console, open the *Edit Configuration Settings*, go to *Configuration* > *PC* > *Env* > *PC Environment*, and add or edit the parameter `AllowGenerationFromOutOfTheBoxViews`. The parameter takes the inputs true and false.

If you want to create custom views, you need to be assigned the role `CUSTOM_VIEW_CREATOR_ROLE`. If you have this role, the *Create* button appears enabled in the *Create Product Specification Document* dialog.

The following limitations and cautions should be kept in mind when working with product reports:

- Any formatting applied to rule and rule step comments aren't displayed in the Rules Report.
- Any formatting applied to content in a TEXT or CLOB field within the *Values* tab doesn't appear in the Data Report. If you want the text formatting to be retained in the `Product Specification Document` you should define the field with the Rich Text data type.
- The Business Report and Data Report may not display all fonts used when defining *Manual* tab content and may default to Times New Roman font if the font that was used can't be found. The fonts that are recognized can be defined.
- You must publish a product before you can generate reports for it.

Use the following procedure to generate a product report from an existing view. You can also generate after you create a view.

Procedure

1. Go to the *Bookmarks* or *Object List* tab in the *Product Studio*.

   You can also generate product reports from an item in a *Search* result.

2. Right-click the product and select *Create Product Specification Document*.
   The *Create Product Specification Document* dialog appears; it shows the standard views and any custom views that have been created for this product.
3. Right-click the view you want to use and select *Generate*.

   A message appears stating that the report request has been added to the activity queue.
4. Choose *OK*.
5. You can repeat the above process to add further reports to the queue; otherwise close the dialog.
6. If you want to see the report in the queue, go to *Tools* > *Scheduled Activity Manager*. If the report is queued or in progress, look in the *Active* tab. If the report has completed, select the *Completed* tab. Note that for *Request Type*, reports show *DOC GEN*.
7. To view the report or to save the file, select its *Log/Output* icon and then choose *Download*.
   The *File Download* dialog appears.
8. Choose *Close*.
9. To view the report, open it in a word processing application.

> → Tip
>
> If the report's table of contents appear blank, press `Ctrl` + `A` and then `F9` to trigger a refresh and rebuild the table of contents.

Related Information

Creating Views [page 67]
Viewing Object Histories and Audit Trails [page 61]
Managing Scheduled Activities [page 70]

3.7.1 Creating Views

The following procedure describes how to create a product view, both from scratch or by basing it on an existing view.

Prerequisites

Your user profile requires the `CUSTOM_VIEW_CREATOR_ROLE` role in order to create views.

Context

> **! Restriction**
>
> The standard views that come with the Product Modeler can't be deleted.

> **→ Remember**
>
> You can reuse a view in the inheritance of a product. Views are automatically passed to the child products from the parent.

Procedure

1. Right-click the product in the *Product Studio*'s *Bookmarks* or *Object List* tab and select *Create Product Specification Documentation*.

   You can also select an item in a *Search* result.
2. You can create a new view in the following ways:
   - To re-use an existing view, right-click the view and select *Copy*. Choose *Paste* and provide a name and description for the new view. The view appears in the list. Right-click the view and select *Open*.

     > **i Note**
     >
     > You can copy a view from the view list within one product and paste it into that of another
   - To create a view from scratch, choose *Create* and provide an appropriate name and description.

   You can define a custom template for Product Specification Documentation. Templates are loaded into the system using the Administrative Console's *System* > > *Edit Configuration Settings* menu. The path is *Configuration* > *PC* > *Env* > *PC Environment*.

   In either case, the dialog changes to show the details of the view, starting with the *General Product Information* tab.
3. Use the *General Product Information* tab to define the content that you want included in most of the objects in the view. Keep in mind that these are only general settings and that you can override them later for individual objects.
   - If you know that you want to include all the objects in the product, leave *Select entire product tree* selected. If you want to pick the objects that belong to the view, deselect this box—at a later stage you will be presented with an empty tree to fill in.
   - In the *Folder Information* section, set the values for the folder name, folder label, empty sections and inherited content.
   - In the *Object Information*, *Attributes Tab*, and *Values Tab* sections, specify whether you want information from these areas to appear for each object in the report, and if so, the specific details.
   - If you want to include inherited *Manual* tab content on a per component basis, select the *Display inherited content* checkbox in the *Object Information* section of the *General Product Information* tab. Select the *Component-Specific Information* tab and select the desired folders and objects on the left side of the screen, then choose the *Manual* tab on the right side of the screen and select or deselect the checkboxes accordingly.

- If you wish to include product objects (components, coverages, and so on) in products linked to the selected product in the generated report, select the *Include linked objects (non-published linked objects will not be included in the generated report)* checkbox in the *Entire Product* section of the *General Product Information* tab.
- To optionally select if *Values Tab* information (on a per component basis) is included inline within the MS Word document or included as an embedded MS Excel file, select the *Include Values Tab Information* checkbox in the *Values Tab* section.
  This selects the *Include Data* and *Include Rules* checkboxes. Selecting *Include Data* selects you to specify the report layout as *Inline* or *As Attachment*.
  Select the *Component-Specific Information* tab. Select the desired components on the left side of the screen, then specify the report layout on the right side of the screen in the *Values Tab*.
- Similarly, select the *Include Attributes Tab Information* checkbox in the *Attributes Tab* section to optionally select if *Attributes Tab* information (on a per-component basis) is included inline within the word processing document or included as an embedded spreadsheet file.
- If you want to provide a business user-friendly label in the generated report instead of the object name as configured in the repository, from the *Component-Specific Information* tab select the *Object Information* or *Folder information* tab for a selected object or folder, respectively. Perform one of the following actions:
  - Select a *Default name*
  - Select and specify a *Custom name*
  - Display no name at all by deselecting either the *Display object name* (for objects), or the *Display folder name* (for folders) checkboxes.

4. To define the look of the report, choose the *General Document Information* tab.
   - If you want this view to use the default view settings defined for the environment, leave *Use Defaults* selected.
   - If you want to override the default view settings, make changes in the other sections (*Use Defaults* is then automatically deselected). You can specify whether the report uses a cover page, headers and footers, and an introduction, as well as various details pertaining to each.

   > **i Note**
   >
   > To specify the default view settings for all views in all products, when you are in the list mode of the *Create Product Specification Document* dialog, choose *Default Document Options*; you are presented with these same settings.

5. Use the *Component-Specific Information* tab to narrow down the product objects included in the view, and their individual content:
   - If you deselected *Select entire product tree*, you now find the tree in the *Selected Components* panel empty except for the top node, which is the product itself. To define the objects that you want in the report, drag them from the tree in the *Source Components* panel to the tree in *Selected Components*. Note that you can arrange objects in any order you want, although anchored relationships must be respected (for example, you can't reverse a parent-child relationship by placing the child above the parent, although in an anchored set of four components, you could choose to include only the first two components in your view).
   - To remove objects from the view, in the *Selected Components* panel, right-click the object and choose *Remove from selected*.
   - To override the default information settings for individual objects, as defined in the *General Product Information* tab, use the tabs in the *Settings* panel.

- To automatically include all the objects of a given type, right-click the product node, any coverage or folder, and select *Component of Type...*. Select the modifier code for the object type and choose *OK*. A placeholder appears in the tree. When you generate the report, all the objects of the chosen type appear at this point in the *Product Tree*. Note that you can use the *Settings* panel to specify the settings for these objects.
- ⚠ Caution

  In the *Selected Components* panel, if you drag a sub-component away from its parent, the anchoring is lost and you can't "re-anchor" by dragging the sub-component back. To recreate the relationship, you must remove both components from the tree, and then drag them back in from the *Source Components* panel.

6. Save your changes.
7. To delete a view, right-click it in the list and select *Delete*. The view is removed.

3.8 Managing Scheduled Activities

Scheduled activities are termed asynchronous: you launch them and then continue using the system for other work.

Context

Examples of such activities are requests for product publishing and product specification reports. In order that the different activity types don't affect one another, each type has its own queues in the system, depending on how the administrators configure the environment.

The Product Modeler provides a tool for viewing and managing all the requests in the environment. You can access this tool from the *Studio* tab or from within a product. Activity requests can have any of the following statuses: PENDING, IN PROGRESS, SCHEDULED, COMPLETED or COMPLETED WITH ERRORS. You can view the requests by any of these statuses. You can delete pending and scheduled requests, as well as view the logs of in-progress or completed requests (including those with errors).

Procedure

1. Go to *Tools* > *Scheduled Activity Manager*.

   The dialog appears with the *Active* tab selected by default, listing all the activity requests either currently running or pending.
2. To change the date or time for a scheduled request, select its *Date/Time Request* field, choose the calendar control, and make your changes. In the dialogs' toolbar, choose the *Save* icon.
3. To see all completed requests, select the *Completed* tab; to view the log for a completed request, select the icon in its row; the log appears in-line.

4. In either tab, you can do the following:
   - Sort the rows by selecting a given column heading; the triangle in the heading indicates whether the sort is ascending or descending.
   - Filter the listing by selecting Filter, selecting one or more columns, selecting operators, entering values, and choosing *Apply*.
   - View the details of a request by selecting its plus sign icon; for example, for a publishing request you can see which users' uncommitted changes are to be included, the deployment servers, and any selected pre-publishing or post-publishing services.
   - Metrics are displayed for completed task, indicating the length of time each building and publishing took to be executed.
5. If you want to delete scheduled or completed requests that you created, select the checkbox for the row and choose the *Delete* icon. Note that you can't delete running requests or those created by other users.
6. You can copy the Activity Log for your reference. Select the *Copy log* link in the Activity Log of the action. This will copy the log to your clipboard. Note that, if the log is large, this action may take a few moments to display..

3.9 Exporting and Importing Products

The product export and import feature provides a convenient way to copy a product from one *Studio Tree* to another.

To export, you only need to have `Full Control` permission for the product object, not for the standalone objects it contains. Importing a product isn't permission-based. The product information is exported to multiple XML files and stored in a JAR file.

Limitations and Cautions

The following constraints and behaviors apply when exporting or importing a product:

- You can only export and import products, not standalone objects.
- You are able to export and import products between minor (maintenance) releases of FS-PRO as the system only checks for major releases (e.g. 1.0) and not minor releases (e.g. 1.0.0.1).
- The source and target server names must be unique between regions.
- Only an exported JAR file can be imported. The exported readable XML files can't be used for an import.
- The export operation excludes configuration information for PPM server remote publishing. You must configure the target server manually. For more information, see your system administrator.
- Custom stems are excluded from export operations.
- Any name conflicts that arise in rule XPaths must be corrected manually after import.
- The export operation excludes reference products. (If your product needs a particular reference product, you can export/import it separately.)

- If you select the *Overwrite* option, all changes made to rules and screens in the target product are lost after an import. The import operation doesn't update rules and screens in the target product; instead it deletes them and then recreates them from the imported product.
- The import operation ignores locked data value rows on the target product. The import log file notes all such rows.
- The import feature provides no clean-up or synchronization process: if data value rows existing in the first imported version are later deleted in the source product, re-importing the product doesn't delete those rows on the target product. To remove them you must delete them manually.
- You can build imported validation rules but when you attempt to publish the product you will receive an error message. The workaround for this behavior is to open each imported validation rule, assign its error message from the error code picker, save the rule (which builds the error code XML). The product is now publishable.
- If you perform a re-import, a data value row that was previously deleted in the source and that subsequently has been undeleted, after import remains locally deleted in the target.
- Never attach the same object at the same location within a target product. That is, if you create an object locally and add it to the target product, and the object has the same type, name, and location as one in the source product, a subsequent import operation fails to overwrite the new object, instead creating a new version of the object (for example, if 2.0 D is present, 3.0 D is created). Furthermore, the object's data isn't imported; instead, the data in the locally attached object is copied into the imported object. This behavior results because the locally created object has a different UUID than the object in the source product.

> **i Note**
>
> When you export or import a product, ensure that no users are currently editing any of the objects that the export/import will include. All objects should be closed.

Exporting and Importing with Multi-language Support

The following points apply to product export and import with multi-language support:

- Language information of the products is imported.
- Primary language content of the product is imported.
- Primary language content of the product's parents is import.
- Available language content of the components that are assembled in the product is imported.
- Available language content of the components that are assembled in the product's parents is imported.
- *Enable multi-language* selection of attributes are imported.
- If the product doesn't exist in the target instance, the product is imported with the primary language as specified in the import file.
  - If the importing products (both the product and its parents) primary languages aren't within the list of available languages in the target instance, an error message is displayed.
  - If the available languages in the importing components aren't available in the target instance, a warning message is displayed.
- If the product exists in the target instance:
  - If the primary language is the same, the product is imported with the primary language.
  - If the primary language of the source product and target products are different, a warning message is displayed.

- If the available languages in the importing components aren't available in the target instance, a warning message is displayed.
- If attributes are multi-language disabled in the source but enabled in the target:
  - Values are imported into the multi-language enabled attributes in the importing product's language.
  - A warning message is displayed to inform the user.
- If attributes are multi-language enabled in source but disabled in target:
  - Target attributes are converted to multi-language enabled.
  - Values are imported into the multi-language enabled attributes in the languages specified in the source.
  - A warning message is displayed to inform the user.

Related Information

What a Product Export Includes [page 73]
What Happens During a Product Import? [page 74]
Exporting Products [page 81]
Importing Products [page 82]
Reviewing Product Import Operations [page 84]

3.9.1 What a Product Export Includes

When you export a product, the operation includes the complete product, as well as the product's entire inheritance hierarchy—all parent products are included in the JAR file or the ZIP file of XML files.

Depending on the options that you choose during import of the JAR file, the result is an identical product in the target *Studio Tree*.

> **i Note**
>
> Product data that was exported in XML format can't be imported. Only the JAR file can be used to import product data.

The following list indicates the items that are exported to the JAR file:

- Environment details: server name, FS-PRO version, and the date of export
- The product object and its structure, including all:
  - Folders
  - Components, with current versions
  - Committed data value rows
  - Contained products (for example, coverages, LOBs, or data definitions)
  - Questionnaires
  - Locally deleted components

    - Questionnaire models
- Other elements of the product:
    - Rules, both their steps and inputs
    - Screens and views, including layout information
    - Product details from the *Manual* tab: *Name*, *Detail Description*, *Features*, and *Notes*
    - Custom objects
- The following information is exported for the product's components:
    - *Manual* tab content
    - All attributes, including local
    - Global data value rows
    - Local data value rows
    - All child data value rows
    - Overridden information for data value rows in a component
    - Data selection at individual component and its child (anchored) components
    - Locally deleted selection information
- The following information is exported for standalone components:
    - Attributes
    - Child components
    - Subassembly data selection
- All Product Studio folders and objects, in their relative locations in the tree structure
- All parent products and the objects that they contain

The following items aren't included when a product is exported to an XML ZIP file:

- User information
- Inheritance path substitution
- Custom classes used by the product
- Servers for deployment
- Cardinality of each assembled object on the *Product Tree*
- Local attributes
- Association only overridden data value rows
- Content of customized dropdown lists

3.9.2 What Happens During a Product Import?

The product import feature requires you to import the entire product: you can't choose to leave out some objects while including others.

If you are importing the product for the first time, and select to import both structure and data, the product in the target *Studio Tree* will be identical to that in the source.

The object name and version is maintained after the import process. If the Object UUID, object name and object version doesn't exist in the system, the system creates the product using the same name and version of the exported product. This is illustrated in the following example: in a clean Product Studio environment you

import ISO GL Countrywide (v3.0). The system imports the product ISO GL Countrywide as version 3.0 (**not** version 1.0).

An object isn't imported if the Name and Version of the object exists in the target environment if the UUID is different. This includes all components, questionnaires and product assembled under the imported product.

Product Import Error Messages

If the product can't be imported an error message displays: `The following objects you are trying to import already exist locally in the system`. A List of local object names and versions that need to be removed is displayed.

The solution is to remove the local version and try again.

For all subsequent imports, the system matches UUIDs and updates old content with new content. Any objects that aren't present in the target will be recreated. For example, if after the last import you move component X from folder A to folder B in the target product, the next time you run import for the product a new component X will be added to folder A, while the one in folder B is ignored.

For all objects, if the name already exists in the target but with a different UUID, modifier code, and object type, a suffix is added to the name of the imported object. For example, after import the component ABC is renamed ABC_1.

Similarly for custom objects, if the modifier code already exists in the target and the UUIDs differ between the source and the target, a suffix is added to the code for the imported custom object. For example, if the modifier code EAPP already exists on the target, the imported custom object is assigned the modifier code EAPP_1.

The entire import process is recorded in a log, which is stored in the database. You can review the log through the *Import Export Archive Manager*, and optionally save the log to a file.

Impact on Create Version Functionality

The system creates the new version by finding the greatest version of the object and uses the next sequential number. This is illustrated in the following example:

Your environment already contains products: ISO GL Countrywide (v1.0) and ISO GL Countrywide (v3.0). When the *Create Version* option is selected, the system creates ISO GL Countrywide version 4.0 (NOT version 2.0).

Product Import Options

During a product import you must select from two sets of options that determine how the import is run and what's imported. The "how" options are Retain and Overwrite. Overwrite is the default. The options for what to import are: Structure Only, Data Only, or Both Structure and Data. The last is the default. (During a first-time import the Data Only option is unavailable.)

When you choose the Structure Only option, *Manual* tab data and data value rows aren't imported. The following items are imported:

- All new objects (that is, weren't present in previous imports) as well as any folders created to contain them
- Standalone components used by the product are imported without their data
- Local and global attributes
- Object modifier codes

> **i Note**
>
> Under the Structure Only option, even if the product was previously imported with data, new objects are imported without data value rows.

The following table summarizes the relationship of the Structure Only option to components during a re-import.

The relationship of the Structure Only option to components during a re-import

Action on the Source	Action on the Target
A component was added on the source	The component is added, along with its folder, if any, on the target
A component was deleted on the source	The component isn't deleted on the target
A component had a new attribute added on the source	The component is updated on the target
A component had an attribute changed on the source	The component is updated on the target
A component had an attribute deleted on the source	The component isn't updated on the target

When you choose the Data Only option, structural changes to the source product aren't imported. The following items are imported:

- The contents of the *Manual* tab
- All data value rows, including the inherited, the overridden, and child rows
- Data value rows added to the source after the first import are appended to their respective components

The following table summarizes what happens to data value rows under the Data Only option.

The relationship of the Structure Only option to data value rows under the Data Only option

Action on the Source	Action on the Target
A data value row was added	The data value row is appended to its component on the target
A data value row was changed	The data value row is updated, based on matching UUIDs and version numbers on the target
A data value row was deleted	The data value row isn't deleted on the target
A data value row was attached to its product	The data value row is attached on the target
A data value row was unattached from its product	The data value row remains attached on the target

Action on the Source	Action on the Target
A data value row was overridden	The data value row is overridden on the target
A data value row had a child row added	The child row is added on the target
A data value row had a child row changed	The child row is updated on the target
A data value row had a child row deleted	The child row isn't deleted on the target

After the initial import, data value rows are updated by comparing their hidden identifiers:

- If the UUID, version, and server name are the same between the source and the target, no update is required
- If the UUID matches but the server names are different, an update is required
- If the row is locked, no update is allowed

> **i Note**
>
> Updates are performed only if you have selected the overwrite option.

The third option, Structure and Data, combines the features of the Structure Only and Data Only options, as described in the preceding paragraphs.

The Retain Changes and Overwrite Changes Options

The import options to retain changes or overwrite changes don't affect a first-time import. However, whenever you perform a re-import of a product, it is important to understand their differences and to choose carefully.

> **i Note**
>
> These options only apply to existing objects—during a re-import, objects that aren't present on the target are treated as new and are added to the target, regardless of the option that you chose.

The following table compares the retain and overwrite options.

Comparing the Retain and Overwrite Options

Action on the Source	Action on the Target, if the Retain Option was Used	Action on the Target, if the Overrride Option was Used
The name of a component, questionnaire, rule, or screen was changed on the source	The name isn't updated	
The name of an attribute was changed on the source	The attribute name isn't updated	The attribute name is updated

Action on the Source	Action on the Target, if the Retain Option was Used	Action on the Target, if the Overrride Option was Used
The contents of the *Description*, *Features*, or *Notes* fields in the *Manual* tab were changed on the source	The fields aren't updated	The fields are updated
Data in the *Values* tab were changed on the source	The data isn't updated	The data is updated (except in data value rows that are locked)
An attribute and data was added in the *Values* tab on the source	The attribute and its data are added	
A questionnaire was changed on the source	The questionnaire isn't updated	The questionnaire is updated
A rule step or a rule step name was changed on the source	There isn't an update	An update occurs
A screen was changed on the source	The screen isn't updated	The screen is updated

When you choose the retain changes option, the import ignores all data value rows that have been changed on the target (they are flagged by their hidden server name being changed from that of the source server to that of the target server). All data value rows that were excluded from the import because of the changes on the target are noted in the log file.

When you choose the overwrite changes option, the import overrides all data value rows whose UUID matches that of a data value row in the source. All data value rows that were updated should be noted in the log file. For rules, questionnaires, and screens:

1. The existing data value row and all its child rows are deleted in target
2. The data value rows from the JAR file are imported again into target
3. The association value for each data value row/rule is updated with a new key

Comparing Product Import Options

An import has six possible combinations of "how" and "what" options. The following table compares the options.

Comparing the Import Options

Option	Retain changes made in target	Overwrite changes made in target
Structure Only	When you select the Structure Only option, the Retain and Overwrite options have no effect.	When you select the Structure Only option, the Retain and Overwrite options have no effect.

Option	Retain changes made in target	Overwrite changes made in target
Data Only	• No update to the structure of target-modified objects • Updates imported data (that is, in *Manual* and *Value* tabs) based on match of server name and UUID and a lower version number than on the source; after update, server name and version number are same as in source and `MasterFlag = 0`	• No update to the structure target-modified objects with matching UUID • Updates imported data (that is, in *Manual* and *Value* tabs) based on match of server name and UUID and a lower version number than on the source; after update, server name and version number are same as in source and `MasterFlag = 0` • Overrides any target-modified data; after update, server name and version number are same as in source and `MasterFlag = 0`
Structure and Data	Combines the two preceding rows.	• Updates data but not structure of target-modified objects with matching UUID • Updates objects tagged with the same server name as the source and with a lower version number than in the source • Overrides any target-modified data **i Note** In all cases, after update, server name and version number are same as in source and `MasterFlag = 0`

Product Import and Versioning

Version management information always belongs to the *Studio Tree*. When you export a product, no version management information from the source *Studio Tree* is included in the JAR file. With the initial import to the target *Studio Tree*, that environment starts to track its own version information for the product. The product export feature allows you to export any version of a product.

The following table outlines typical product import cases and how they affect product versioning.

How an Import Affects Versioning

Version of the Product Exported from the Source	Product Version on the Target	Product Version on the Target After an Import	Comment
Product 1.0 D	none	Product 1.0 D	After initial import the two products are identical. The target *Studio Tree* regards the imported product as a newly created product and labels it accordingly.
Product 4.0 D	Product 1.0 D	Product 2.0 D	Importing causes the target product's version to increment by 1.
Product 2.0 D	Product 4.0 D	Product 5.0 D	The target was developed and versioned independent of the source product. Importing causes the target product's version to increment by 1.

The following table describes typical product import cases and how they affect object versioning within products.

How an Import Affects Object Versioning Within Products

Version of the Component in the Product Exported from the Source	Product on the Target	Version of the Component in the Product on the Target After an Import	Comment
Component 1.0 D	Product doesn't contain this component	Component 1.0 D	After initial import the two components are identical.

UUID and Product Export or Import

The Universal Unique Identifier (UUID) is a hidden code assigned at creation to every object and value in a FS-PRO *Studio Tree*. The UUID remains the same regardless of where the object or value is moved or what's done to it. Because UUIDs stay the same across repositories, the import/export feature uses them to identify and match components during import.

For the purposes of import/export, the UUID is supplemented by a further hidden value that stores the name of the server on which the object or value was created. If the object or value is imported to another server and is changed in any way, this value changes to that of the current server.

Another hidden value, `MasterFlag`, is assigned to all components and identifies whether a data value row was created on the current server or was imported to it.

All the following elements in the configurator have their own UUID:

- Folders
- Products
- *Manual* tab contents (*Detail Description*, *Features*, *Notes*)
- Components
- Questionnaires, along with questions, model content, views
- Rules
- Screens
- Custom objects
- Object assemblies (the usage links in a product)
- Values rows and data value row selections
- Component attributes

3.9.3 Exporting Products

The product export feature allows you to export any version of a product.

Context

There are two formats for exporting a product from FS-PRO:

a JAR file	Exports the product information into a JAR file, which can be transferred to another FS-PRO environment for import.
a zip file of XML files	Exports the product information into a zip file of structured, readable XML files. You can use any XML comparison tool to evaluate content changes in a product between different environments or from different periods of time. For example, you could perform an export before changes are made and again after changes are made. Or you could export a product from a test environment and a production environment. When the exports are complete, use a comparison tool to view the delta between the two zip files. Exporting to XML files is optional and isn't required in order to successfully import a JAR file.

The product export feature allows you to export any version of a product. If the product or coverage contains locked data value rows, a list of the rows and the users who locked them appears. Locked rows don't stop you from exporting. However, only the committed versions of the rows are exported. In the new product or coverage, all data value rows are unlocked.

Procedure

1. Access the product in the *Studio Tree*.
2. Right-click the product and select *Admin* > *Export Product*.

3. If you want to export the product to XML files, perform the following steps:
   a. Select the *Generate Readable Product XML* checkbox.
   b. Confirm your decision to export the product.
      A progress dialog appears, listing the objects exported.
   c. Choose *Download XML*.
4. If you want to export the product to a JAR file, perform the following steps:
   a. Confirm your decision to export the product.
      A progress dialog appears, listing the objects exported.
   b. Choose *Download Jar*.
5. To save the log of the export operation to a file, choose *Download Log*.
6. Choose *Done*.

Related Information

Exporting and Importing Products [page 71]
What a Product Export Includes [page 73]
Accessing Objects [page 38]

3.9.4 Importing Products

After a product has been exported to a JAR file using the `Export Product` command, the resulting JAR file can be transferred to your FS-PRO environment and you can use the following procedure to import the product.

Prerequisites

When importing, the target repository must be the same version of FS-PRO as the export repository that created the JAR file.

If you rename a component attribute in the source product and re-import to a target product, the renamed attribute overwrites the existing attribute and all of its data is lost; however, the renamed attribute is imported with all its data.

Context

> **i Note**
>
> Product data that was exported in XML format can't be imported.

You can import a product back into the *Studio Tree* that it was exported from. In this case the same guidelines apply as for any other import.

If the target product or coverage contains locked data value rows, a warning message appears. Locked rows don't stop you from importing; however, the import operation skips locked rows (these rows are recorded in the import log). This restriction applies even when an administrator performs the import.

Procedure

1. Go to *File* > *Import Product Wizard* in the *Product Studio*.
   The *Object Import* dialog appears.
2. Choose *Browse*.
3. Select the product export JAR file.
4. The procedure will create an audit record for the operation; add appropriate information in the *Description* and *Comments* fields.
5. Choose *Next>>*.

   The *Import Options* tab appears and the system begins comparing the JAR file to the target, looking for name conflicts. No changes are made to the target. If any problems are found, the specified objects are listed in the *Validation Issues* panel.

   > **i Note**
   >
   > Before selecting from the options in the *Import Options* tab, you should fully understand their effect on an existing product in the target *Studio Tree*.

6. The *Input Options* require you to choose between importing while keeping any changes you have made to the target product, or overwriting them with the product version or revision that you want to import.

   > **i Note**
   >
   > If you are importing the product for the first time, you can select either option and the effect will be the same.

7. The *Input Type* options allow you to import data alone, objects alone, or both together. For a first-time import you typically choose *Both*. If you are importing to an existing product, you must decide based on development needs.
8. By default, the key column settings are lost for any components that you are importing and that already exist in the target. Select the *Key Column* checkbox if you want the imported key column settings to overwrite all those existing in the target.
9. Choose *Next>>*.

   The *Import* tab appears and displays the progress of the import operation.

   An import operation automatically recreates the necessary pathing in the *Studio Tree* for all the exported products and their objects.
10. To save the log file, choose *Download Log*.
11. Choose *Done*.

Next Steps

All imported data value rows are locked. You must commit the rows in order to unlock them.

Related Information

Exporting and Importing Products [page 71]
What Happens During a Product Import? [page 74]

3.9.5 Reviewing Product Import Operations

The *Import/Export Archive Manager* allows you to examine the *Audit Trail* details of any previous import or export, including the log file.

Procedure

1. Go to *File* > *Import Export Archive Manager* in the *Studio Tree*.
2. Perform one of the following actions:
   - Scroll through the *Search Results* panel to locate the product import record
   - Complete one or more of the fields in the *Search Import Archives* panel with information about the operation and what type of operation and choose *Search*; any records found appear in the *Search Results* panel
3. In the *Search Results* panel, select the record you want to examine; the details appear in the *Attributes* panel, grouped among several tabs. The *Details* tab is the default. It shows the ID and summary information for the import operation.
4. Select the *Import Options* tab.
   The *Import Options* section shows how the product was imported. The *Validation Issues* section lists any objects that caused naming conflicts with existing objects in the environment, and how the imported object was renamed to solve the problem.
5. Select the *Imported Object List* tab. The tab lists all the objects that were imported.
6. Select the *Execution Log* tab.
   The tab shows the body of the log for the import operation, minus the header, which appears in the *Detail* tab; the columns in the log file are as follows:

Message Type	Contains `Info`, `Warning`, or `Message`.
Operation Type	Contains `Insert`, `Update`, or `File`.
Message	Describes the operation.
Remarks	If there was an error, describes the nature of the problem.

7. The execution log is stored in the configurator database. To copy the log to a file on your system, choose *Download Log*.

3.10 Duplicating a Product or Contract

The *Duplicate* feature creates a comprehensive copy of a source product or contract. As well as copying the basic data of the source item, it also copies the internal objects/entities and locally assembled objects within.

Prerequisites

This feature is only available for Product Base and Contract type objects.

Only users with Write or Full permissions can duplicate a product or contract.

Procedure

1. In the *Product Studio*, navigate to the product or contract that you want to duplicate.
2. Right-click on the name of the item and choose *Duplicate* from the popup menu.
   The *Duplicate* dialog launches, with the *Settings* popup window open on top of it. Before a duplicate can be created, you are prompted to assign a folder in the *Content Repository* where all of the new objects will be saved. Also, you must assign a suffix to the objects that will be created by the duplication.
3. If you are duplicating a product or contract that is located in the *Content Repository* and you want the target root folder of the duplicated objects to be the same as the source folder, you can select the *Same Folder as Source* checkbox.

   > **i Note**
   >
   > All of the internal objects/entities and locally assembled object belonging to the source product or contract must also be located in the *Content Repository* for this checkbox to be an available option.

4. If you are duplicating a sample product or contract that is located in the *System Repository*, you must choose a folder in the *Content Repository* as the target folder:
   a. Select the *Target Root Folder* field.

      The *Select Target Root Folder* popup opens, displaying the folder structure of the *Content Repository*. Only folders of the type `Configuration Group` will be available for selection.
   b. Choose a folder in the *Product Repository* as the target folder.
   c. Choose *Select*.

      The *Select Target Root Folder* popup window closes and you return to the *Settings* popup window. The full path of the target folder is displayed in the *Target Root Folder* field.
5. Enter a value in the *Object Name Suffix* field.

   The suffix will be assigned to all of the objects that will be created by the duplication.

6. Select *Update*.

   The *Duplicate* dialog opens, with the suffix appended to the original name of the product or contract and the objects. The new names are validated. You will be notified if any of the names of the newly created objects are not unique, as they will be outlined in red on the dialog.

   To change target folder or the object suffix, select *Update Setting* on the top left of the dialog and edit the information in the *Settings* popup window.

   The following fields are displayed on the dialog:

Source Object	Displays the expanded product tree and its assembled IFBC entities, with their names and version number. For example, SAP Policyholder (1.0). This data is read-only.
Copy Option	A dropdown list that allows you to determine whether a specific object will be copied or if the existing object will be reused in the new product or contract.
New Object Name	Specifies the object name that the new copy will be assigned, in the format of `<source object name> <suffix>`, in editable input fields.
Target Folder	Indicates the name of the sub-folders that will be created under the target root folder to hold the duplicated objects. The type of object being duplicated will determine the sub-folder name. Possible folder names include: Products, Contracts, Coverages, Deductibles, Surcharges and Limits..

   If there are multiple instances of an entity on the product tree, a warning icon is displayed beside each instance of the entity name. When you move your mouse over the warning icon, a tooltip is displayed with information regarding where the entity is assembled.

   If an entity is assembled in a parent product (i.e. not locally assembled), the entity name is displayed in italics on the *Source Object* column and a warning icon is displayed beside the entity name. When you move your mouse over the warning icon, a tooltip is displayed explaining why it will not be copied.

   If there are entities (including the product itself) that do not have the ID component assembled, a warning icon is displayed beside the entity name and the name appears with strikethrough text in the *Source Object* column. When you move your mouse over the warning icon, a tooltip is displayed explaining why it will not be copied.

7. The system creates a new name for all objects by default. If you want to change the system default name in the *New Object Name*, simply edit the enabled field.
8. If you want to search for a particular entity on the product tree, select the *Source Object* column header.

   A filter popup is displayed, where you can enter a search string. The list of entities is shortened to display just the entities with names that include the search string.
9. If you don't want to copy an object, but instead reuse the existing object, select *Reuse* from the *Copy Option* dropdown for that object.

   If you choose *Reuse* for an object that has child objects, all of the child objects will also be reused.

   Note that when you select the *Reuse* option, the data is removed from the *New Object Name* field.
10. Select *Execute*.

    A message is displayed: `Product Duplication request successfully placed on queue. You will be notified once it is processed`

11. Choose *OK*.

Results

The Duplicate request is sent to *Scheduled Activity Manager*. A copy of the product or contract and each of its entities that are not assembled in a parent product or contract is created in the system in the specified target folder. The current user has full permission to the created objects.

> → Tip
>
> You can copy the Activity log of the Duplicate action for reference. Select the *Copy log* link in the *Scheduled Activity Manager*, which will copy the log to your clipboard. Note that, due to the size of the log, this action may take a few moments to display.

Next Steps

For this product to be useable in Runtime, you will need to perform the following actions:

- Register the new product in the `Available Products` component in the `Extended Underwriting Application Configuration`, so that it will be included in the *Sales Product* list on the FS-QUO Fiori Apps.
- Register the new product in the Administrative Console for deployment.
- Perform an IFBC push, if your system is integrated with FS-PM.

4 Defining Objects

You create objects in the Product Studio, and work on them by opening them, which causes them to appear in their own tabs, where you can view them, add other objects to them, and edit their structure and data.

Depending on the type of object, the activities that you can perform can consist of the following:

- Defining component attributes (metadata)
- Inputting component values (data)
- Creating questions for questionnaires
- Creating rules
- Exporting and importing component values

Product objects, and to an extent coverage objects, have further specific activities, such as the following:

- Assembling in objects to the product or coverage
- Defining product features
- Overriding inherited data value rows
- Creating eApps
- Publishing the product

How the User Interface is Organized

When you open an object in the Product Modeler, the user interface consists of a tree view (the *Product Tree*) and a main panel containing tabbed pages. You use the *Product Tree* to view and navigate the structure of the product or coverage, which consists of objects. Any object that you select in the *Product Tree* is defined by the tabbed pages.

You work with the object through these tabs, which consist of the following items:

Manual tab	Summarizes the object. If the object is a product, the tab serves as a product manual: you can enter marketing material that explains the product's features and options. Team members can also add project notes here.
Attributes tab	Allows you to define the metadata for a component, naming its data columns and their data types.
Values tab	Allows to add and edit component data and questions in questionnaires. You can also add rules to a component, and directly accessing the *Rule Editor*.

> **i Note**
>
> Questionnaire objects have predefined attributes that you can't override.

Attributes and Values

Attributes are the metadata that is stored in an object. Typically, you only need to define attributes for a component object. Defining an attribute means giving it a name, a data type (for example, text, date, or number), and possibly a default value.

Values are the data elements contained in the attributes of a component or questionnaire. A component object can be defined as either of the "with values" type or the "without values" type. In the modeling tool, you can only enter values to components of the "with values" type. You use this modifier type to contain values that you already know at Design Time; for example, billing options or a rate table.

Attribute Extensions

The attribute extension feature enables you to assemble the same component to multiple locations inside a product or coverage, while adding extra attributes that are unique to each instance. Attribute extensions enable you to realize the full value of potentially high re-use components and at the same time to decrease maintenance.

Data Value Row Statuses

A component data value row always has one of two states: public or private. Its state determines the value's visibility and where the value can be modified. Public values are those created in a standalone component. When you add the component to any product or coverage, these public values are visible. Private values are those added to a component from within a product or coverage. Private values are only visible within their product or coverage, or within any inheriting child product or coverage.

By default, private values rows are local, meaning that they are only visible within the instance of the component that they were created in. To make them visible in all other instances of the component in the same product, you must apply the `Share` command.

Rules

Rules enable you to add business logic to objects in FS-PRO. You create rules using the *Rule Editor*.

You can use rules as part of a component to determine, for example, how a product is to be priced. You can also use rules when a Runtime user enters values to components.

In a questionnaire, you can attach a rule to any question. For example, the rule uses values from a component to populate the question's screen control.

Related Information

Understanding the Product Tree [page 90]
Working with the Manual Tab [page 97]
Working with Component Attributes [page 99]
Understanding Attribute Extensions [page 108]
Working with Component Values [page 111]
Understanding the Value Components Import and Export Features [page 132]
Publishing [page 143]
Downloading Product JAR Files [page 154]
Coverage-Based Business Terms [page 154]
Understanding the Product Modeler User Interface [page 24]
Working with Scripting Rules [page 369]
Data Value Row Statuses [page 114]

4.1 Understanding the *Product Tree*

The *Product Tree* is your tool for assembling, organizing, and viewing all the parts that comprise a product. In effect, the *Product Tree* shows the structure of the product.

The host object always appears at the top of the *Product Tree*. Objects that appear below the host have been added to it or have been inherited by it. The objects that appear in the *Product Tree* are as follows:

Objects in the Product Tree

Icon	Object	Description
	Product	A product object can be added to, or inherited by, the host product, and usually contains other objects.
	Folder	A folder at the second-or-lower level in the *Product Tree*. Used to organize objects, such as components and questionnaires, within a product. A folder can contain other folders as well.
	Coverage	A coverage object contains other objects and is usually added to a marketable product.
	Component	A component stores values used by product and coverage objects.
	Questionnaire Model	A questionnaire is a set of questions and the screen controls for recording a customer's answers to the questions.
	Data Definition	A data definition contains the metadata for a transactional entity.

Icon	Object	Description
	Table	A table object is automatically generated for every data definition object, and contains the name of the table that the object defines.
	Column	A column object is automatically generated for every data definition object, and contains the data definition's metadata.
	eApplication	The root component in an eApp.
	eApp Section	A specialized component that specifies the sections within an eApp.
	eApp Screen	A specialized component that specifies the screens within an eApp.
	Deleted object	A deleted object. The *Show Deleted* setting must be active for deleted objects to appear in the *Product Tree*.

> **i Note**
>
> In the *Product Tree*, an italicized name for an object indicates that the object is external rather than local. That is, the object wasn't added in but was inherited, or is part of an object that was added in, such as a coverage.

Related Information

Adding Folders to the Product Tree [page 92]
Renaming Folders in the Product Tree [page 92]
Adding Objects to the Product Tree [page 92]
Moving Objects in the Product Tree [page 94]
Creating Links to Reference Products in the Product Tree [page 94]
Deleting Objects from the Product Tree [page 95]
Undeleting a Deleted External Object [page 96]
Viewing Object Properties in the Product Tree [page 96]
Opening a Standalone Version of an Object [page 97]

4.1.1 Adding Folders to the *Product Tree*

Folders are useful for organizing the objects in a product or coverage.

Procedure

1. Open the product  or  coverage .
2. Right-click an item in the *Product Tree* and select *Add Folder*.

   Only product_base type objects and folders can have folders added to them. The menu item will be greyed out if the selected object is not a viable option.
3. Enter a name for the folder.
4. Save your changes.

4.1.2 Renaming Folders in the *Product Tree*

You can rename folders, but only those that are local. You can't rename folders that are inherited.

Procedure

1. Right-click the folder that you want to rename and select *Properties*.
2. Enter the new name in the *Folder Name* field,
3. Save your changes.

4.1.3 Adding Objects to the *Product Tree*

To assemble objects into a product, you add them to its *Product Tree*.

Prerequisites

You must have `Full Control` permission for the product, not for the object.

Context

When you add an object to the *Product Tree* two results are allowed: the object is placed inside a folder, or it's placed as a child object that depends directly on a parent component. The placement type depends on whether you right-click a folder or a component. A child object is normally referred to as a subobject. A component, a questionnaire, or even another product can be a subobject.

> → Tip
>
> Subobjects aren't restricted to products: you can add a subobject to a standalone component. When you subsequently add the parent component to a product, all its subobjects are automatically added in with it.

Procedure

1. Right-click a folder or an object and select *Add*.

   The *Object Picker* dialog displays.

   All objects that you, the current user, have created will be displayed in the *Object Search Result* list.
2. Perform one of the following actions:
   - To search for the object, go to step 3
   - If the object that you want already appears in the *Object Search Result* list, select it and go to step 8

   > → Tip
   >
   > If you want to return the list to displaying only the objects of which you are the owner, clear the *Object Name* field and then choose *My Objects*.
3. Enter the name of the object in the *Object Name* field.
4. Optionally, select the type of object to search for from the *Object Type* dropdown list.
5. Choose *Search*.
   The objects appear in the *Object Search Results* list.
6. Select the required object in the list.
7. If you are using the IBM Insurance Application Architecture, perform the following actions to set the number of instances of the object that can appear in the product at Runtime:
   - In the *Min Cardinality* field, specify the number of instances of this object that the product must have at Runtime.
   - In the *Max Cardinality* field, specify the upper limit of the number of instances of this object that the product is allowed to have at Runtime.
8. Choose *Select*.

4.1.4 Moving Objects in the *Product Tree*

You can reposition objects within the *Product Tree*.

Context

> **! Restriction**
>
> You can't move inherited objects.

Procedure

1. Navigate through the tree structure to the object of interest.
2. Right-click the object and choose one of the following options from the resulting menu:
   - *Cut*
   - *Paste As Child*
   - *Paste Above*
   - *Paste Below*

   These cut and paste options apply only within the same product structure, and can't break any inherited sibling structures, can't break any inherited anchor relationship, and can't break any local anchor relationship.

4.1.5 Creating Links to Reference Products in the *Product Tree*

Links to reference products can be designed as part of the product definition process, permitting the opening of a linked object in a new tab from within the source product.

Procedure

1. Add an `OBJECT_PICKER` attribute in any component.
2. Create data value rows under the *Values* tab.
3. On each data value row, choose the down arrow of the *OBJECT_PICKER* attribute field.

   The *Object Picker* dialog displays.
4. Search for and select the object to be linked, and choose *Add Link*.
   The *Object Picker* dialog closes and the selected object name and version appears in the field.

5. Save your changes.
   The selected object appears as a link.
6. To modify the field, select the field's down arrow.

   The *Object Picker* dialog displays.
7. Select the new object.

   The new selection appears in the field.

   > **i Note**
   >
   > The *OBJECT_PICKER* attribute field isn't editable by typing. You can only change the selected object by selecting the field's down arrow.
8. If you want to clear the field, select the down arrow and select *Clear Link*.

4.1.6 Deleting Objects from the *Product Tree*

You can delete contained objects from their host; depending on whether the object is local or external.

Context

Local objects are those that you assemble in to the product using the `Add Object` command in the *Product Tree*; their names appear in standard format. External objects are those that you didn't add in directly. They may have been inherited by the product, or may be part of another object that you added in, such as a coverage. In the *Product Tree*, the names of external objects appear italicized.

When you delete a local object from the *Product Tree*, its link to the product is broken, and can't be restored. When you delete an external object, its link to the product isn't broken, but is only hidden. This is because the object has a relationship to other objects outside the current product; for example, the coverage that contains it.

> **i Note**
>
> Deleting an object from the *Product Tree* doesn't delete the object from FS-PRO, but merely detaches the object from its host. Deleting a folder permanently removes the folder. Before you can delete a folder, you must delete the objects within it. Before you can delete an object that contains other objects, you must delete the contained objects from within it, including sub-objects from parent components.

> **→ Tip**
>
> If you want to delete a component from a product, you first need to remove all local data that was specified within the component. Otherwise, attempting a deletion will result in an error message indicating the local data that is preventing the deletion.

Procedure

1. Right-click the object in the *Product Tree* and select *Delete*.

   A confirmation message appears.
2. Choose *Yes* to confirm your decision.
   A progress dialog appears.
3. Choose *Close* when the operation completes.

Results

The object is removed from the *Product Tree*.

4.1.7 Undeleting a Deleted External Object

You can reverse the deletion of an external object.

Procedure

1. Choose the *Show Deleted* icon in the toolbar.
2. Right-click the deleted object and select *Undelete*.

4.1.8 Viewing Object Properties in the *Product Tree*

You can view the properties values for any object in the *Product Tree*.

Procedure

1. Right-click the object and select *Properties*.
   The following information is displayed:

This object was initially assembled in	Lists the products and coverages into which the object has been added.
Minimum Cardinality	The number of instances of this object that the product must have at Runtime.

Maximum Cardinality	The upper limit of the number of instances of this object that the product is allowed to have at Runtime.
Owner	The owner of the object.
Locked By	If the object is locked, the ID of the user who has locked it.
Version History	The version number of the object.
Create Date	The date and time the object was created.
Effective Date	If version management is in use, the date and time that this version of the object takes effect in the system.

2. Choose *Close* to exit the dialog.

4.1.9 Opening a Standalone Version of an Object

You can open a standalone version of an object.

Procedure

1. Right-click the object in the *Product Tree*.
2. Choose *Open* from the resulting menu.

Results

The selected node opens in a new tab.

The object opens in a manner consistent with the viewer's permissions. For example, if the viewer has read-only permissions for the object, it will be opened with read-only permissions.

4.2 Working with the *Manual* Tab

The *Manual* tab summarizes an object's features, and in the case of a product or coverage, serves as a product manual.

The *Manual* tab is divided into sections. The appearance of the *Manual* tab depends on the object you are viewing. The following list indicates the fields or sections that appear for the major object types (products, coverages, components, questionnaires and data definitions):

Name	Displays the name of the object, and is limited to 250 characters.

	You can't edit the *Name* field. Instead, you must access the object's properties in the Product Studio, where you can change the name.
Detail Description	Use this field to create product information suitable for various purposes, such as a product manual, an intranet site, or a website accessed by agents, brokers, or customers. This information is stored as part of the product's definition in the product catalog. A child product inherits the initial content of these fields from its parent product. This field provides a Rich Text Editor, which allows you to create Web-enabled content, including Web links (URLs) and tables. This field supports multiple languages.
Features	Use this field to to create product information suitable for various purposes, such as a product manual, an intranet site, or a website accessed by agents, brokers, or customers. This information is stored as part of the product's definition in the product catalog. A child product inherits the initial content of these fields from its parent product. This field provides a Rich Text Editor, which allows you to create Web-enabled content, including Web links (URLs) and tables. This field supports multiple languages.
Notes	Use the *Notes* field to track additional information on the product, component, or questionnaire. For example, the field is convenient for logging version changes to the object. A child object inherits the initial content of the *Notes* field from its parent product. This field provides a Rich Text Editor, which allows you to create Web-enabled content, including Web links (URLs) and tables. This field supports multiple languages.

Related Information

Adding Text Comments to the Manual Tab [page 98]
Editing an Object's General Properties [page 42]

4.2.1 Adding Text Comments to the *Manual* Tab

You can add text comments to the *Manual* tab. You can also edit existing comments.

Procedure

1. Select the  edit icon.
   The *Rich Text Editor* is displayed.
2. Enter comments as required.

   The *Detail Description*, *Feature* and *Notes* fields accept and display Rich Text.

You can paste content from other applications.

3. Save your changes.
4. Choose the *Close* icon when the comments are complete.

4.3 Working with Component Attributes

Attributes define the data that can be stored in an object.

Typically, you only need to define attributes for a component object (regardless of whether it has values)—Questionnaire objects use a set of standard attributes that you can't edit.

The *Attributes* tab contains a table listing the attributes. Normally you only define attributes for components: questionnaires use predefined attributes that you can't edit, and although you can define attributes for a product object, you can't add values. Click the row heading to access attribute functions and properties

The *Attributes* tab contains its own toolbar. The following table describes each tool or option in the toolbar.

Icons on the Attributes tab toolbar

Icon	Name	Description
	Save	Saves additions and edits that you make to the attributes.
	Move Up	Moves the selected attribute up one position in the table.
	Move Down	Moves the selected attribute down one position in the table.
	Delete	Removes the selected attribute from the table.
	Undo	Undoes the last action that you performed in the table.
	Add Column Before	Inserts a blank attribute above the selected attribute.
	Add Column After	Inserts a blank attribute beneath the selected attribute.

The *Attributes* tab organizes component attributes into groups, depending on the context in which you view the component. When you access a component in standalone mode, you don't see groups, because all the

attributes belong to the component. However, when you access the component within a product or coverage, you see some or all of the groups described in the following table:

Attributes Groups

Group	Content
Common Attributes	Contains attributes that were added to the component in standalone mode.
Inherited Attributes	Contains attributes that were added to the component in a parent product and that the current product inherits.
Local Attributes	Contains attributes that were added to the component in the current product (these attributes are known as attribute extensions).

Grouping happens automatically, and both helps you identify where an attribute originates and what actions you are allowed to perform on the attribute in the current context.

Related Information

Adding Attributes [page 100]
Searching For an Attribute [page 103]
Moving an Attribute [page 103]
Editing an Attribute [page 104]
Deleting an Attribute [page 105]
Defining Data List Sources [page 106]
Understanding Attribute Extensions [page 108]
Adding Attribute Extensions [page 109]

4.3.1 Adding Attributes

The attributes you create using this procedure are Common attributes: they are always visible, whether the component is accessed standalone or from within a product or coverage.

Context

When you access a component from within another object, you use almost the same steps to create Local attributes, which are also known as attribute extensions.

Procedure

1. Open the component .
2. Select the *Attributes* tab.
3. Right-click the row heading of an attribute and choose one of the following options:
   - *Insert Column > As child* - Adds a first attribute to a group
   - *Insert Column > Before* - Adds the new attribute before the selected attribute
   - *Insert Column > After* - Adds the new attribute after the selected attribute

   A blank row appears.
4. Enter a label for the attribute in the *Attribute Name* field.
   - The label can't be blank
   - It can be up to 100 characters long
   - Spaces are allowed
   - The underscore (_) is the only special character allowed in the label
   - The hash (#) and dollar sign ($) aren't allowed
   - The name must not start with a numeric character

   > **i Note**
   >
   > After you save the attribute, *Attribute Name* becomes the attribute's Internal Name, with any spaces removed. Thus spaces, as well as character case, has no effect on uniqueness. For example, AgeCalc, Age Calc, and AGE CALC are equivalent in the *Internal Name*.
5. Enter a value in the *Internal Name* field to define the name of the attribute as stored in the database.

   If this column is hidden, right-click a column header and select *Internal Name* to reveal it. Mixed case is permitted.
6. Choose a value from the *Data Type* dropdown list to assign a data type to the attribute:

BOOLEAN	For checkboxes (true/false) and Yes/No fields.
COLUMN_NAME	Define a column name. This Data Type is generally used by the Column object of a Data Definition.
CLOB	For text fields greater than 2,000 characters.
COLUMN_PICKER	Select a column in the Data Definition that is associated with the product.
CURRENCY	For currency.
DATA_LIST	For fields that have a restricted set of allowable values, the user is presented with a dropdown list. Requires that you specify the *Data List Source* attribute.
DATA_TYPE	Dropdown list allowing for data type selection (for example, TEXT or NUMBER.). **i Note** The DATA_TYPE property can't be changed after the field is saved. If you want to change the type, you must delete the attribute and create a new version.

DATE	For dates.
DateTime	Select a date using the Date Picker.
NUMBER	For numerical fields.
OBJECT_PICKER	Link to an object.
Rich Text	For text fields that use Rich Text Format (RTF) You can cut and paste content from applications such as word processors into this field type and keep the formatting of the original.
RULE	For standard and script rules.
RULE_EXEC_MODE	Dropdown list allowing the selection of the execution mode of a rule (such as Pre&Form or Pre). This Data Type is generally used by the eApp Column Info object.
Screen	For screens.
TABLE_NAME	Define a table name. This data type is generally used by the Table object of a data definition.
TABLE_PICKER	Select a table in the data definition that is associated with the product.
TEXT	For text fields; Can store up to 2,000 characters; If you need to store more than 2,000 characters, use the CLOB data type.
URL	Stores a user-entered web address (full absolute path, including the domain name) For example: `http://www.example.com` Selecting the cell in the *Values* tab launches a browser window.
VALIDATION_RULE	For a validation rule.

7. Define the maximum number of characters permitted for the attribute in the *Length* field.
   Users exceeding the maximum length receive an error message.
8. Define the number of decimal places in the *Precision* field.
9. To specify a data list for the attribute, select the name of a Data List Source service API in the *Data List Source* field.
10. Select the *Required* checkbox to indicate that the user must enter a value for the attribute.
11. Optionally, enter content in the *Default Value* field to define the attribute's default value.

    You can override this value.
12. Select the *Enable Multi-Language* checkbox to define the attribute for translation.

    This option is only available for CLOB, Text, or Rich Text data types.

    > **i Note**
    >
    > The checkbox can only be deselected when there are no values entered in all instances of the component. For attributes where the *Enable Multi-Language* checkbox isn't selected, any values entered will be be assigned the default language of English.

13. Save your changes.

Related Information

Understanding Attribute Extensions [page 108]
Adding Attribute Extensions [page 109]
Defining Data List Sources [page 106]

4.3.2 Searching For an Attribute

You can search for a specific attribute.

Procedure

1. Open the component .
2. Select the *Attributes* tab.
3. Press `Ctrl` + `F`.
   The *Find* dialog appears.
4. Enter the value to search for in the *Find what* field.
   You can search any of the fields in the table, including *Type* and *Length*.
5. Choose *Find Next*.

Results

If the search is successful, the matching text appears highlighted.

4.3.3 Moving an Attribute

You can move an attribute to a new position.

Context

Procedure

1. Right-click the row heading of the attribute that you want to move.
2. Choose *Move up* or *Move down*.

Results

The attribute moves one row at a time.

4.3.4 Editing an Attribute

You can change most properties in a component attribute to suit your requirements.

Context

Note that certain properties can't be changed after the attribute is created; for example, the *Internal Name* and the *Data Type*.

Procedure

1. Open the attribute.
2. Select the field in the attribute that you want to edit and change the properties.
3. Save your changes.

4.3.5 Deleting an Attribute

The procedure for deleting an attribute differs depending on whether you open the component standalone or within a containing object.

Context

> ⚠ Caution
>
> Deleting an attribute from a standalone component removes the attribute in all the objects that use the component.

If you open the component from within a containing object, you can perform a local deletion. The change only affects the current instance of the component and any child products that inherit it. After local deletion, the deleted attribute is struck through with a red line to indicate deletion, and is hidden on the *Values* tab. The attribute is unaffected in the parent product and the standalone component.

> → Remember
>
> In a child product, you can't undelete an inherited attribute that was deleted in an ancestor product.

Procedure

1. If you want to delete an attribute from a standalone component, perform the following actions:
   a. Right-click the row heading of the attribute and choose *Delete*.
      A warning message appears.
   b. Choose *Yes*.
   c. Save your changes.
      The attribute is permanently removed from the component.
2. If you want to delete an attribute from within a containing object,
   a. Select the product or coverage .
   b. Navigate to the component and select it.
   c. Select the *Attributes* tab.
   d. Navigate to the appropriate *Inherited Attributes* group.
   e. Right-click the row heading of the attribute that you want to delete and choose *Delete*.
      A red line appears through the data in the row.
   f. Save your changes.

4.3.6 Defining Data List Sources

If you set an attribute's data type to Data List, you must also identify a source for the list, by selecting one in the `Data List API` attribute.

Context

The `Data List API` attribute is populated from the system component `Data List Source`, located in the data list reference product.

> **i Note**
>
> Typically, you will work with your own data list reference product, which extends the one provided by SAP. In this case, you must also update the relevant configuration parameter in the Administrative Console.

Each row of the `Data List API Library` component contains a data list source rule, which accesses data from a component contained either in the product or in the reference product. The rule in a data list API must return only two columns. Typically, they are a code and a description.

Procedure

1. Open your environment's data list reference product (it may be named Data List Base).
2. Select the `Data List API Library` component .
3. Select the *Values* tab.
   Any existing data list APIs appear as data value rows.
4. Perform one of the following actions:
   - Select inside the *DATA_LIST_SOURCE_NAME* cell of the empty row (the last row) and enter the name of the new data list
   - Right-click the row heading of the row beneath which you want to add a new row and select *Insert*; in the *DATA_LIST_SOURCE_NAME* cell of the new row, enter the name of the new data list.

   The data list source name should be unique within the product, and service names should follow the naming standard for Java methods: the names should begin with a lowercase letter; second and subsequent words in the name should begin with a capital letter but not be separated with underscores. For example: getCountryCode.

   > **⚠ Caution**
   >
   > If multiple versions of the same product service name exist, the system only uses the first one and ignores all others. The system doesn't warn of duplicate names.

5. Right-click in the `DATA_LIST_SOURCE_NAME` attribute and select *Create*.
   The *Capture Rule Info* dialog appears.

6. Enter the name of the rule that the new product service contains in the the *Rule Name* field.
7. Enter the name for the original step in the rule in the *First Step Name* field.
8. Select the type of step you want to use from the *Step Type* dropdown list.
9. Select Object Table from the *Rule Return Type* field.

> **ℹ Note**
>
> Object Table is the only return type allowed, otherwise in the *Values* tab an exception will be thrown by the `Data List` attribute.

10. Choose *OK*.
    If a data definition is present, the dialog changes to show the structure of the product's data definitions.
11. In the *Rule Painter*, create a product query rule against the source of the data list. This is a component, either in the product or in the reference product. The rule must return two columns, typically a code and a description.

> **→ Tip**
>
> The `TargetDataRow` stem provides functions that access information related to the calling data value row. This stem is documented in the *Rule Painter*.

12. Save your changes.
13. Build the rule.
14. Close the *Rule Painter*.
    In the *Values* tab, the icon appears in the *DATA_LIST_SOURCE_RULE* cell.
15. Save your changes.
16. Select the `Data List Group` component.

> **ℹ Note**
>
> Data list groups enable users to search when selecting a data list source.

17. Select the *Values* tab and anchor the new data list API that you created to at least one category. If there is no appropriate category, consider creating one.

> **⚠ Caution**
>
> Failure to anchor a data list API to a group means that the data list source won't be available to the user.

18. Save your changes.
19. You must republish the data list reference product to make the data list API available.

Related Information

Working with Subobject Data Value Rows [page 128]

4.4 Understanding Attribute Extensions

The attribute extension feature enables you to assemble a single component to multiple locations, but to add attributes that are unique to each instance.

In some situations you may find that you want to use the same component in multiple locations within a product, with the majority of the component's attributes the same, but with some unique to each instance. You could create a single component with all the attributes, but this can cause confusion for end-users, who then must distinguish which attributes to use in a given context. Another option is to create a separate component for each location, but that requires re-creating the shared attributes each time, and then having to maintain them separately, increasing maintenance.

1. Create the standalone version of the component, adding the common attributes that are required in all the places that the component will be used.
2. Assemble the component to each required place in the product.
3. Access each instance of the component and add/delete attributes as the context requires.

For example, consider a component named `Investment Details`, that you want to use in a variable annuity product, in the coverages `Maturing Interest Based Account` and `Variable Investment Account`. Your analysis of the required attributes and their contexts is as follows:

Attribute in `Investment Details`	`Maturing Interest Based Account` **Coverage**	`Variable Investment Account` **Coverage**
Investment ID	4	4
Type of Account	4	4
Investment Amount	4	4
Market Value	4	4
Interest Rate	4	
Compounding Period	4	
Fund Unit Price		4
Number of Units		4

To implement the `Investment Details` component, using the attribute extension feature, you create the standalone component, adding the first four attributes listed in the table, because they are common to both contexts. You then assemble in the component to each coverage. If you access the *Attributes* tab, the attributes you created appear in the *Common Attributes* group.

Next, from within the product, you access each instance of the component and add the attributes unique to that instance. For example, in the instance in the `Maturing Interest Based Account` coverage, you add the attribute extensions `Interest Rate` and `Compounding Period`. You must add these in the `Local Attributes` group. The variable annuity product now contains two instances of the component, with both shared and unique attributes.

How Attribute Extensions Behave

If you perform a `Copy Product`, the local attributes are copied to the new product, but are then independent of the source product.

In a child product, if you delete an inherited attribute, you won't be able to undelete it in a grandchild product.

If you delete an inherited attribute, you won't be allowed to undelete the attribute. The workaround is to recreate the attribute as a local attribute.

If, after adding an attribute extension to a component, you add the same attribute in the parent product, you will get the following result:

- The data type, length, precision, and other properties default to those in the child product, and you aren't allowed to change them
- In the child product the inherited attribute is grayed to indicate the presence of the local attribute with the same name and data type
- The data entered in the parent product appears as inherited data value rows in the child product
- If you subsequently delete the local attribute in the child product, the inherited attribute no longer appears gray

Related Information

Adding Attribute Extensions [page 109]
Deleting Attribute Exensions [page 110]
Promoting Attribute Extensions to Standalone Components [page 111]

4.4.1 Adding Attribute Extensions

You can add an attribute extension to a specific instance of a component within a product. Once created, the new attribute is only visible in this instance of the component.

Context

You can add as many attributes as you need. All attributes that you add to the Local Attributes group are only available in this instance of the component; however, you can add the same attributes to other instances of the component, both in the current product and in other products. When repeating an attribute, you use the same name, and the same data type is enforced automatically.

Procedure

1. Navigate in the *Product Tree* to the instance of the component that you want to add the attribute extension to and select it.
2. Select the *Attributes* tab.
3. Right-click the row heading for the `Local Attributes` group and select *Insert Column > As Child*.
4. Complete the attribute in the standard way and save it.

4.4.2 Deleting Attribute Exensions

You can delete an attribute extension from a given instance of the component.

Procedure

1. Navigate in the *Product Tree* to the instance of the component that you want to add the attribute extension to and select it.
2. Select the *Attributes* tab.
3. Select the `Local Attributes` group.
4. Right-click the row heading for the attribute that you want to delete and select *Delete*.
   A warning message appears.
5. Choose *Yes*.

Results

The attribute is permanently removed from this instance of the component.

4.4.3 Promoting Attribute Extensions to Standalone Components

You may find that an attribute extension is useful beyond its local context and therefore want to relocate the attribute to its standalone component, thereby changing the status of the attribute to Common.

Context

> ⚠ Caution
>
> This operation can't be undone.

Procedure

1. Navigate in the *Product Tree* to the instance of the component and select it.
2. Select the *Attributes* tab.
3. Select the `Local Attributes` group.
4. Right-click the row heading for the attribute that you want to promote and select *Move to Common*.
   A message appears indicating that the attribute is being moved.

Results

When the operation is complete, the attribute appears in the `Common Attributes` group.

4.5 Working with Component Values

Values are the data elements described by the attributes of a component or questionnaire, and are contained in the *Values* tab.

> ℹ Note
>
> In a questionnaire, the questions themselves are the values. The answers to the questions are captured at Runtime, and are also stored.

The *Values* tab displays an object's data in a table. Use this tab to add, edit, and delete object data, as well as to attach or unattach data value rows to or from a product or coverage.

> **i Note**
>
> Product objects don't have a language dropdown list because you select the language in the *Properties* dialog.

When working in the *Values* tab, keep in mind that any attribute with the data type TEXT or CLOB supports multiple languages.

The tools or options in the *Values* tab toolbar are as follows:

Icon	Name	Description
	Save	Saves current changes made to the data value rows.
	Filter	Opens a dialog where you can create a set of filter statements that control the data value rows that appear.
	Multiple Sort	Opens a dialog where you can create a temporary sort setting based on multiple attributes.
	Refresh	Forces the *Values* tab to reload its data from the server.
	Delete Multiple Rows	Deletes all currently selected rows.

The toolbar of the *Values* tab also contains the following controls:

- The *Menu* – A set of less-frequently used functions:

Set Key Column	For standalone components only; opens a dialog where you can designate one or more columns as a key column. Key columns serve as a persistent sort setting for the component's data value rows. An icon in the column header indicates a key column.
Import	Opens a dialog where you can select data value rows in a JAR file or a CSV (comma-separated values) file from which to import data into the *Values* tabs.
Export	Opens a dialog where you can choose the destination of the export.
Search Data	Opens the *Find* dialog, enabling you to search the object for a specific value.
Locked Data Row Manager	Shows a list of data value rows that are locked within this *Values* tab.

- The *Views* dropdown list – When viewing the component inside a product or coverage, use to select the values displayed in the page based on attachment: All, Selected, or Unselected.

The *Values* Tab and Multi-language Support

When entering values for different languages for standalone components, you should be aware of the following information:

- The *Language* dropdown list contains all available languages as configured in the Administrative Console `AvailableLanguages` setting.
- For the local data value rows, any fields with *Enable Multi-Language* selected are editable for entering values for the selected language.
- After selecting a language from the *Language* dropdown list, the *Values* tab will refresh, and attributes specified with *Enable Multi-Language*, the values in the selected language, if available, will be displayed. Otherwise the fields will be blank.
- Values for the selected language can be input or imported.

When entering values for different languages at the product level, you should be aware of the following information:

- For product-type objects, the *Language* dropdown list isn't available in the *Values* tab of the assembled components.
- For attributes with *Enable Multi-Language* selected, the fields will either be displayed in the primary language selected for the current product, or will be blank if there are no values entered.
- For non-product type objects, users can switch to another primary language using the dropdown list. Attribute fields with the *Enable Multi-Language* checkbox selected will be refreshed to display in the selected language (if available).
  For product-type objects, the primary language is changed on the properties page.
  - Initially, attribute fields will be blank to either enter or import values in the selected primary language.
  - For attribute fields with primary language values previously entered, the previously-entered values in the selected primary language will be displayed.
- Unicode values on the attribute fields can be saved if the *Enable Multi-Language* checkbox is selected. This applies to both product and non-product objects.

> **i Note**
>
> You must save the *Values* tab content in the current selected language before switching to another language

Related Information

Data Value Row Statuses [page 114]
Adding Data Value Rows [page 115]
Editing Data Value Rows for Components [page 116]
Deleting Data Value Rows [page 117]
Locking Data Value Rows [page 118]
Viewing Usage Information for a Data Value Row [page 122]
Sorting Data Value Rows [page 123]
Setting Key Columns in the Values Tab [page 124]

4.5.1 Data Value Row Statuses

A data value row's visibility and whether the row can be modified, is determined by its state.

The major states are public and private. Where you create the data value row sets the state:

- Public values are those created in a standalone component
- Private data value rows are only visible within their product or coverage, or within any inheriting child product or coverage

Private data value rows can also be local. Private values are those added to a component from within a product or coverage.

One other state, which wraps all the preceding ones, is inherited, meaning that you are viewing the data value row in a component that originates from another product or coverage in the inheritance tree.

The following table summarizes the data value row statuses:

Public	Definition: Added to component in standalone mode. Uses: A data value row that you want available everywhere. Appearance in standalone component: regular text Appearance in a a product or a coverage: italic/bold Visibility: Available in all products and coverages that use the component, except questionnaires. Can only be edited in standalone mode. Deletion from a product: Can't be deleted. Notes: A public data value row can't be made local.
Local	Definition: Added to the component from within a product or coverage. Uses: You want the data value row to be seen/used in only one instance of the component. Appearance: regular text Visibility: Available only within this instance of the component and only within this product or coverage. Can be modified in its product. Deletion from a product: Can be deleted. Notes: local is the only status permitted for Data Definition data value rows.
Inherited	Definition: The data value row belongs to another product or coverage, of which the current product or coverage is a child. Or, a public data value row. Uses: Functions as in parent, but can't be modified. Appearance: italic bold text

Visibility: Depends on status in parent:

- public data value row is visible everywhere
- local is visible only in its component

Deletion from a product: Can't be deleted.

Notes: To modify you must Override.

4.5.2 Adding Data Value Rows

You can add a data value row to a component.

Procedure

1. Open the component .
2. Select the *Values* tab.
3. At the row above where you want to insert the new row, right-click the row heading and select *Add*; an empty row appears, with a green lock icon to indicate that the row has never been committed.
4. Select a field and enter a value.

   If you leave the `PCD` attribute blank, after you save your changes the system automatically adds a Part Code to this attribute, a unique identifier for the data value row both in the component and across the entire system.
5. Enter values for the remaining attributes in the row.
6. When you finish adding data value rows, save your changes.

   If you add a rule to a data value row, that rule is saved internally in the system, even if you don't choose *Save* in the *Values* tab. However, if you don't save the rule to the row, the rule isn't linked to the row and in the future you won't be able to open the rule.
7. If you want to unlock the row, perform one of the following actions:
   - For individual rows, right-click the row header and select *Locking* > *Commit*.
   - For all the locked rows, on the *Values* tab, go to *Menu* > *Locked Data Row Manager*. Select the rows you want to unlock, and choose *Commit*.

4.5.3 Editing Data Value Rows for Components

Procedure

1. Open the component .
2. Select the *Values* tab.
3. Locate the data value row.

   To search for a specific value in the current page, choose *Menu* in the toolbar and select the *Search Data* icon to use the *Find* dialog.
4. If you want to edit a local row and want to ensure that your changes don't affect other users, right-click the row header and select *Locking* > *Lock* .
5. If you want to edit an inherited data value row, you must make it into a local data value row: right-click the row heading and select *Override*.

   You can always override a data value row that is locked in the parent product. Overriding a data value row automatically locks it locally and it remains locked until you `Commit` it.

   > **i Note**
   >
   > With inherited data value rows, you always see the last committed version of the row (only if the row is locked and you aren't the owner). After you `Override` an inherited row and it becomes local, to see the version in the parent product, you must delete the local row.
6. If the component is contained in a product or coverage and the row is attached, a dialog appears. Perform one of the following actions:

   - To enable editing of the data value row's `Expiration Date` and `Effective Date` attributes only, select *Override association only* and choose *Select*.
   - To enable editing of all fields in the data value row, select *Override association and values* and choose *Select*.

     > **i Note**
     >
     > The Rule id changes if the option *Override association and values* is selected.
   - To enable editing of all fields in the data value row except rules, select *Override association and values except rules* and choose *Select*.

   The appearance of the selected data value row changes from bold/italic text to plain text. Also, embedded objects (rules, screens) receive a new ID. This applies to *Override association and values* but not *Override association and values except rules*. In this case rule id doesn't change and remains inherited.

   Inherited rules are impacted when a data value row changes. When overriding a data value row with values, new rules are created, losing rule inheritance. This means all child products inherit a new rule.

> **i Note**
>
> You aren't able to delete a data value row that has been overridden using the *Override association and values except rules* option in other inherited or assembled objects. A message displays advising you to delete the overridden rows and then try again.

7. Edit the cells as required.

   Note that as soon as the focus leaves a changed cell, a red triangle appears in the top-left corner of the cell and a red flag appears in the *Values* tab, to remind you that unsaved changes are present. This behavior applies to all data types except `Screen` and `Rule`, for which no indicators appear.
8. A product or coverage only uses the component values that you link to it. This happens automatically if you create the value while in the product or coverage. If you are in a product or coverage and want to attach or unattach the row, select or deselect the checkbox adjacent to the row.
9. Save your changes.

4.5.4 Deleting Data Value Rows

If you want to delete a data value row, don't lock it. Instead, apply the `Delete` command directly to the row.

Context

Deleting the data value row removes all embedded objects such as rules and screens. In addition, if a rule is deviated in the product below, the data value row will be deleted and all deviations will also be removed.

If any data value row couldn't be deleted owing to restrictions, an icon appears in the row header; hovering the pointer over the icon causes an explanatory message to appear.

> **i Note**
>
> You can unselect an inherited row locally, even if it's locked in the parent product.

Procedure

1. Open the component .
2. Select the *Values* tab.
3. Locate the data value row.

   To search for a specific value in the current page, choose *Menu* in the toolbar and select the *Search Data* icon to use the *Find* dialog.
4. If you want to delete a single data value row, perform the following steps:
   a. Right-click the row header and select *Delete*.

A confirmation message appears.

b. Choose *Yes* to confirm your decision.

5. If you want to delete multiple data value rows, perform the follow steps:

   a. For each row that you want to delete, select the cell in the row's first column; a check mark appears there.

      If you want to delete all the rows on the page, select the header of the first column; a check mark appears in each row.

   b. In the toolbar, select the *Delete Marked Data Row(s)* icon and confirm your decision.
      The rows are removed.

   c. Save your changes.

4.5.5 Locking Data Value Rows

Data value row locking is version management at the data value row level.

If you have `Write` or `Full Control` permission for an object, row locking allows a new, temporary version of a data value row without affecting the work of other users.

Locking is optional—you can edit a row without locking it. If you lock the row, you're editing a new version and your changes are hidden and can't affect the work of other users until you `Commit` the new version. Note that once you commit your changes, you can't undo them.

> ⚠ Caution
>
> If two or more users open a component at the same time and one of the users locks a data value row, the lock isn't visible to the other users. If the other users try to lock and edit the row, they can't save their changes.

When working on a project team it is important that you understand the effects of locking data value rows:

- Product locking is separate from row locking; unlocking a product doesn't unlock its data value rows.
- Locked rows affect product locking as follows: you can only lock a product if all the row locks are yours. If another user has locked rows, neither of you can lock the product. If you attempt to do so, a message informs you that there are rows locked by other users.
- Only local data value rows are lockable.
- Attributes aren't affected by locking. While you have a row locked, another user can still edit or even delete its attributes.
- Locking a row makes the pre-lock version of its values, as well as its associated rules and screens, visible to other users in read-only mode.
- Changes made to a locked row are visible only to the locking user; other users don't see the changes until the row is committed. This includes changes to values, rules, and screens.
- Locking a row locks its selection status: other users can't unselect the row. However, in child products users can select/unselect the row.
- When a user locks a parent row, all its child rows are automatically locked; child rows can't be locked individually.
- After a user locks a parent row, other users can't anchor or unanchor its child rows.

- An inherited data value row can be locked if its selection and data are overridden; however, if only the selection is overridden, the data value row can't be locked.
- The `Lock` command, and its related commands (`Commit`, `Revert`, `Reassign`) aren't recorded in the audit trail.

Administrative Accounts and Locked Data Value Rows

An administrative account uses the same tools and commands to manage data value row locking as a standard user account does, but can also:

- See all locked rows for all users
- Revert, Commit, or Reassign all changes for a given account

> **i Note**
>
> The administrative account can't manage data value row locking for an individual user, but can manage data value row locking for the *Values* tab, within a product or within the system.

Finding Out Who Owns a Data Value Row Lock

To learn who has locked a data value row, hover your pointer over the row's lock icon ; a tooltip names the owner of the lock and when the lock was made.

Viewing the Committed Versions of Your Locked Rows

While you have a data value row locked, you can optionally open the committed version by selecting the row's lock icon . The committed version opens in a separate section immediately below the locked row, in read-only mode. You can also open the row's rules and screen in read-only mode. Note that this feature only applies to the owner of the lock; all other users already view the committed version of the row. To hide the committed version, select the lock icon .

Related Information

Managing Single Data Value Row Locks [page 120]
Managing Multiple Locked Data Value Rows [page 121]

4.5.5.1 Managing Single Data Value Row Locks

The following table summarizes the commands related to data value row locking for a single row at a time:

Command	To call the command...	Notes
`Lock`	In the *Values* tab, right-click the row header of the data value row and select *Locking* > *Lock*.	For the locking user, an icon of an open lock appears beside the row. For all other users, a closed lock appears beside the row. All other users see the previous version of the row. The row can only be edited by the owner of the lock. Locking isn't recorded in the audit trail.
`Commit`	In the *Values* tab, right-click the row header of the locked data value row and select *Locking* > *Commit*.	Unlocks the data value row, commits all saved changes and replaces the previous version of the row. All changes (such as `Save` or `Delete`) are recorded in the audit trail, but prior to `Commit` they're visible only to the locking user; after `Commit` they're visible to all users.
`Revert`	In the *Values* tab, right-click the row header of the locked data value row and choose *Locking* > *Revert*.	Unlocks the data value row and all saved changes made to it are lost. **i Note** You can't `Revert` a committed row.
`Reassign`	*File* > *Locked Data Row Manager*	From the *Locked Data Row Manager* dialog, you access a list of users with `Write` or `Full Control` permission for the object, and all data value rows currently locked by you are assigned to the user you select. This user receives control of the latest saved version of the locked row. If the locked row contains unsaved changes, the changes are lost when you reassign the row. You can't `Reassign` locked rows if you've locked the product. You must first unlock the product.

4.5.5.2 Managing Multiple Locked Data Value Rows

Product Modeler provides a tool for managing many locked data value rows at once, whether at the component, product, or *Product Studio* level.

Procedure

1. Access the *Locked Data Row Manager* dialog using one of the following options:

Where	How	Result
In the *Product Studio*	*File* > *Locked Data Row Manager*	All locked rows in all objects in the environment
In a product or coverage	*File* > *Locked Data Row Manager*	All locked rows in all objects in the product or coverage
In a standalone object	*File* > *Locked Data Row Manager*	All locked rows in the object
In a component, whether standalone or within a product or coverage	In the *Values* tab toolbar, choose *Menu* > *Locked Data Row Manager*	All locked rows in this path (if the component is in a product or coverage.)

   The dialog displays the names of objects with locked rows, the number of locked rows in each object, the path within the product or coverage to the rows, and who has locked them.

2. If you want to change your view in this dialog, choose from the following options:
   - Sort rows by selecting any column heading; the triangle in the heading indicates whether the sort is ascending or descending.
   - Group the locked rows by user or object, by selecting from the *Group By* dropdown list.
   - Filter the rows based on who locked them and/or the object to which they belong, by selecting from the *Locked By* and *Locked In* dropdown lists.
3. Select the rows: :
   - Select or deselect rows by selecting their adjacent checkbox.
   - Select or deselect all rows by selecting the checkbox in the heading row.
4. Perform one of the following actions:
   - Choose `Commit` or `Revert` and confirm your decision; the command is applied to the selected rows in the same way as to a single row.
   - Choose `Reassign`, select the user's name from the dropdown list, choose *Select*, and confirm your decision. The data value rows are no longer yours and therefore are removed from your view of the *Locked Data Row Manager*.

> **i Note**
>
> If you `Reassign` locked rows that have unsaved changes, the changes are lost. You should `Save` changes before reassigning the rows.

5. When the operation is complete, continue managing rows or choose *Close*.

4.5.6 Viewing Usage Information for a Data Value Row

You can generate a Usage Report to view usage information for a data value row.

Procedure

1. Open the component .
2. Select the *Values* tab.
3. Locate the data value row.

   To search for a specific value in the current page, choose *Menu* in the toolbar and select the *Search Data* icon to use the *Find* dialog.
4. Right-click the row heading of the data value row and select *Usage Report*.
   The *Usage Report* dialog appears. By default, the *Data row information* tab appears, showing whether the value is local, and its original product.
5. If you want to see all the products that contain the component and use this data value row, choose *Where used list*.

   For each product, the list indicates its version number and status. The *Scope* column shows how the component and data value row are associated to the product, and contains one of the following values:

internal	The value was attached from within the product.
by association	The value was attached from within another product or coverage, which in turn was assembled in to the product named.
inherited	The value was attached from within another product, which the named product is based on.

   The *Path* column indicates where the object and record is assembled within the product.

Results

The report is generated, and includes the following information:

- The product where the local value is defined
- The products where the local value is assembled or inherited

- Where the data value row is located within the product
- The version and state of the data value row's product

4.5.7 Sorting Data Value Rows

You can sort the display of data value rows.

Context

The default sort order is based on the key column setting; the default key column is PCD. If you remove all key columns, the system automatically resets the key column to PCD.

The default sort order in the *Values* tab is as follows:

Sort Order

Views **Dropdown List Value**	**Sort Order**
Selected	All selected data value rows, sorted on the key. Public, inherited, and local rows are treated the same.
Unselected	All unselected data value rows, sorted on the key. Public, inherited, and local rows are treated the same.
All	All data value rows are sorted on the key.

Some points to keep in mind when using the *Values* tab sort feature:

- A sort setting is temporary. When you exit the *Values* tab the setting is lost.
- A sort setting temporarily overrides the key column setting.
- A sort setting is always localized. If the product contains multiple instances of the component, the sort affects only the component in which it is created. Likewise, a sort in a parent product doesn't affect inheriting products.
- A sort applies to all the data value rows in the component, regardless of how many pages appear in the *Values* tab.

> ⚠ Caution
>
> In copied and cloned products, the default sort order may not function correctly.

Procedure

1. If you want to sort the rows on a single attribute, select the attribute's column header. The rows are re-sorted based on the attribute and an up arrow appears indicating an ascending sort. To reverse the sort order, select the header again. A down arrow appears, indicating a descending sort.

2. If you want to sort the rows on two or more attributes, perform the following steps:
   a. Select the *Multiple Sort* icon.
      A dialog appears showing all the attributes in the component. If a sort is in effect, the sort attributes appear in the *Selected* list.
   b. Select and move attributes between the *Available* list and the *Selected* list
   c. Prioritize the sort attributes in the *Selected* list by using the up and down arrow icons
   d. Choose *Apply*.

Results

The rows are sorted.

You can choose *Clear* to remove the sort.

4.5.8 Setting Key Columns in the *Values* Tab

Key columns are attributes on which the data value rows in a component are sorted. You can create your own sort setting for a component, or accept the default setting, which is a sort on the `PCD` attribute.

Context

Key column settings are persistent: they stay in effect until someone changes them. A key column is indicated by an icon in the column header. You can only set key columns in a standalone component. If you access the component in a product or coverage, the `Set Key Column` command is disabled.

> **i Note**
>
> The system doesn't validate that the values entered to a key column are unique, or even that they aren't blank. Therefore a key column can contain duplicate or empty values. When defining a product, you must decide whether this possibility is important to your design.

Procedure

1. Open the component .
2. Select the *Values* tab.
3. Go to *Menu* > *Set Key Column* in the toolbar.
4. Select and move attributes between the *Available* list and the *Selected* list using the left and right arrow icons

5. Prioritize the sort order in the *Selected* list by using the up and down arrow icons
6. Choose *Apply*.

4.5.9 Filtering Data Value Rows

You use a data value row filter to see a subset of the values in a component.

Context

A filter affects all the pages in the *Values* tab and remains in effect until you remove it or exit the *Values* tab. A filter consists of a set of attribute selection statements and/or one data value row filter property. You can sort filtered data value rows. Each filter statement consists of three elements: an attribute, an operator, and a filter value, and takes the form: `<attribute> + <operator> + <filter value>`.

Example filter statements:

```
Gender = F
PolicyDate < 09-20-2009
Monthly_Rate != 0.4
State = CA OR NY
```

> **i Note**
>
> You can't filter on attributes that have the data types CLOB or Rich Text.

You can also filter on one of several basic data value row properties, selecting to have only rows that are one of the following types: `Public`, `Local`, or `Inherited`. If you combine this value with an attribute filter statement, the relationship is an `AND` condition. If you don't want to use any of these properties, accept the default setting of *All*.

Procedure

1. Select the *Values* tab.
2. Choose the  *Filter* icon from the toolbar.
3. If you want to include one of the basic data value row properties in the filter, select a radio button. The choices are as follows:

All	Includes all data value rows. This is the default value.
Public	Includes only data value rows added to the component in standalone mode.
Local	Includes only data value rows added to the component in the product or coverage.

Inherited Includes only data value rows that come from the parent product or coverage or a product or coverage in a family (but not child products or coverages).

4. If you want to include an attribute statement in the filter, select the attribute from the dropdown list.
5. Select one of the following operators from the *Operator* dropdown list:

   = Show only component values that exactly match the filter value.

   != Hide all component values that exactly match the filter value.

   > Show only component values greater than the filter value.
   Numbers and Dates only.

   < Show only component values lesser than the filter value.
   Numbers and Dates only.

   >= Show only component values greater than or equal to the filter value.
   Numbers and Dates only.

   <= Show only component values lesser than or equal to the filter value.
   Numbers and Dates only.

   like Show only component values that partially match the filter value.
   Text only.

6. Enter the filter value in the *Value* field.

   > **i Note**
   >
   > You can add `OR` to the statement; for example: `Annual OR SemiAnnual OR Monthly`.

7. Repeat step 4 to step 6 for each attribute statement that you want to add to the filter.
   You can add as many statements as you require; they are linked as `AND` conditions to create a single filter statement.
8. Choose *Apply*.

Results

The filter is enabled, and the filter icon changes to indicate that a filter is enabled. To view the filter expression that you created, hover the pointer over the filter icon.

To clear a filter from a component, in the filter control, choose the  *Filter* icon. The filter dialog appears. Choose *Clear* and then choose *Apply*.

4.5.10 Versioning Data Value Rows

Just as you can version products, you can also version local data value rows.

Context

Versioning a data value row has two effects:

- the parent row is expired (if an expiration date has been set)
- a new version of the data value row, using the same `PCD`, is created.

If the data value row is external, you must localize it using the `Override` command before you can version the row. You can't localize and version data value rows in questionnaires or data definitions.

Procedure

1. Open the component .

   You can open the component either standalone or within its product or coverage
2. Right-click the row heading of the local data value row and select *Insert Child*.
   A new data value row appears directly beneath the target data value row, and contains the same `PCD` as its parent.
3. If the data value row is in a product or coverage, and is attached, do the following steps:
   a. Select the *Expiration Date* field in the parent row.
   b. Set the last effective date for the data value row.

      Applications won't be allowed to use the data value row beyond this date.
   c. Select the *Effective Date* field in the cloned data value row.
   d. Set a date that is later than the one you entered in step 3b.

      Applications will be allowed to use the data value row from this date forward.
4. If you need to, make changes to the other values in the row.
5. If appropriate, attach the cloned row to the product or coverage.
6. Save your changes.

4.5.11 Working with Subobject Data Value Rows

If a component contains a subobject, you can attach any row in the component and proceed to add rows to the subobject, automatically creating a one-to-many relationship between them.

Context

Further, if the subobject has subobjects, you can do the same for them. When you access a subobject inside a standalone component, you aren't allowed to add data value rows to the subobject.

> **i Note**
>
> You must override subobject data value rows manually. Applying the `Override` command to the component's data value row overrides only the association for the component and the subobject rows. However, overriding at the subobject level causes the component data value row to become local.

Procedure

1. Open the component .

   You can open the component either standalone or within its product or coverage
2. Select the *Values* tab.
3. Select a data value row.

   The existence of the subobject is flagged in the component's title bar by the icon beside the name of the component. Each data value row also contains a plus sign icon.
4. Select the plus sign.
   The subobject's data value rows are displayed.
5. Select the data value row whose related subobject rows you want to access.

   The following actions occur:
   - The plus sign icon changes to a minus sign icon
   - All other data value rows are hidden
   - The subobject's *Values* tab appears beneath the data value row that you chose along with its related rows, if any.
6. View, add or edit rows, just as you would in the parent component's *Values* tab.

   Child rows that you add and that you haven't committed are deletable.

   Move between peer objects by selecting their tabs.
7. If the child object has its own child, a icon appears beside the object's name, and a plus sign icon appears in its data value rows. You can open further child objects, down to any level.
8. To close the lowest open object, choose the minus sign icon in its parent data value row; to close all open subobjects and return directly to the parent, choose the minus sign icon in the parent's data value row.

9. If you unattach a row that has local child rows, the child rows and their children automatically become unattached.

   When you save your changes, the system checks for anchored children that are inherited by or assembled into other products. If these exist, unattaching isn't allowed. The unattaching of the parent row is logged in the audit trail.
10. If you delete a row that has child rows, the child rows and their children are unattached but they aren't deleted. This isn't logged in the audit trail.

4.5.12 Copying Rules

You can copy and paste rules from one data value row to other rows anywhere in the Product Studio. This allows you to re-use the rule (under a new name) or to change it to suit your needs, thus speeding up the creation process.

Context

For example, form attachment rules are often similar across multiple forms; by copying and pasting a single, previously tested form attachment rule, you can more quickly create new ones, and also reduce errors.

Procedure

1. Navigate the *Values* tab to locate the data value row that contains the rule that you want to copy.
2. Select the rule's icon and choose *Copy*.
3. Choose *OK*.

   > → Tip
   >
   > The clipboard retains the copied rule for the duration of your session, allowing you to paste the same rule multiple times.
4. If necessary, navigate to the target component or product.
5. If necessary, locate or create the target row.
6. Select the rule cell and choose *Paste*.
   The *Rule Clipboard* appears, showing all copied rules that are of the same type as the current cell.

   > → Tip
   >
   > You can sort the contents of the clipboard by selecting any column header. You can also preview the rule from the clipboard.
7. Select the rule and choose *Select*.
   The *Rule Name* dialog appears.

8. Enter the name for the new rule.

   The name of the new rule must be unique in the system.
9. Choose *OK*.
10. Save your changes.

4.5.13 Deviating Rules

With rule inheritance deviation, you can deviate an inherited rule to modify the logic of different parts of the rule. This can simplify rule creation and maintenance.

Context

> **! Restriction**
>
> You should avoid using rule deviations, as this feature will be deprecated in a future version.

Procedure

1. Navigate the *Values* tab to locate the data value row that contains the rule that you want to deviate.
2. Select the rule's icon , and select *Deviate*.

   The rule's icon changes to , indicated the rule has been deviated. and the *Rule Painter* is displayed.

   > **i Note**
   >
   > The `Rule id` and `Rule Name` don't change when a rule is deviated.
3. To deviate a script step, right-click on the step and select *Deviate Step*.

   The step deviation icon appears on the step title bar.
4. If you want to modify the step, choose the *click to edit* section.
   The *Rule Editor* opens.
5. In the *Rule Editor*, select the rule or function you wish to change and choose the override icon .

   The *Rule Editor* displays the override icon in the left margin showing the area being locally modified with a white background. The remainder of the script is shown in gray, signifying the gray block of code is inherited from the parent rule and isn't modified locally.
6. Make any desired changes and save the script.
7. Build the rule.
8. Close the *Rule Painter*.
9. To undo the step deviation, right-click on the step and select *Delete Deviation*.

10. A deviated rule can be deleted, bringing back the original inherited rule:
    a. Right click on the rule and select *Delete*.

       A confirmation message appears.
    b. Choose *Yes* to confirm your decision.
    c. The system prompts you to rebuild the rule.

4.5.14 Enabling or Disabling Data Value Rows

If you have `Full Control` permission for a component , you can change a component's original definition (whether it is a "with-values" or a "without-values" component). This is accomplished on the *Properties* dialog.

4.5.15 Deleting Data Value Rows Across Multiple Pages

Context

When you choose the *Delete* icon in the *Values* tab, two options are displayed:

- *Delete Marked Data Row(s)*
- *Delete All Data Rows*

These options are disabled when no data value rows are selected.

When selecting *Delete All Data Rows*, a warning message displays advising that upon confirmation of the message:

- All local data value rows in all pages will be deleted
- All child records (cloned data value rows) with any existing embedded objects will be deleted
- All anchored relationships will be unattached

Choosing *No* cancels the deletion, and no data value rows are deleted.

Choosing *Yes* deletes all local, unlocked data value rows across all pages from the *Values* tab, and the changes are saved immediately.

4.6 Understanding the Value Components Import and Export Features

The component value export and import feature provides a convenient way to copy data between two components, whether they're in the same product, different products, or different repositories, thus removing the need for manual re-entry and reducing the risk of errors.

In addition, you can also use the import feature to bring in data created and managed outside of the Product Modeler. For example, in spreadsheets.

The following constraints apply when exporting/importing component values:

- The Export or Import file is in JAR format, packaged with data value rows in CSV format and embedded objects represented in an XML file format as well as anchoring information.
- On performing a *Values* tab export, if there are locked data value rows in the current component and any sub-objects, the following message will be displayed: `There are locked data rows in the current component and/or sub-components. Only the latest committed version of the locked data rows will be exported. Do you want to continue?`
  Select **`Yes`** to proceed with exporting, or **`No`** if you choose to work on the locked data before exporting.
- The system will also accept a CSV formatted import file for backward compatibility.
- The export process ignores any sort applied in the *Values* tab and uses the default sort order: grouping rows by ascending `PCD` value.
- You can't export data related to screens.
- Date values in the file and on the target component should use the same format.
- When performing a *Values* tab import (Append, Update, or Replace), if there are locked (gold) or uncommitted (green) data value rows in current *Values* tab and any sub-component you aren't able to import. A warning message is displayed: `Unable to import values due to locked data. Please commit / revert all locked data rows in the current component of this product and any child products and try again.`
  Select *OK* to continue.
- The import process provides no clean-up or synchronization: if you delete data value rows in the source component, then export the data and re-import it into a target component, parallel deletions aren't made on the target. You must repeat the deletions on the target manually.

  > ⚠ Caution
  >
  > The import feature doesn't validate the uniqueness of key column values. If you require uniqueness in a key column, you must ensure that the key column values are unique at their source.

- If a data value row in the target has no `PCD` value and uniqueness isn't maintained on a key column, the update is made to the first row that matches the key column.
- If you try to append data value rows from an import file that has a `PCD` attribute that matches existing data value rows in the target and those data value rows are marked as selected but don't have effective dates and expiration dates, the data value rows won't be imported and this won't be reflected in the log.

Related Information

Exporting Data from the Values Tab [page 133]
Importing Component Values [page 136]
Exporting and Importing Products [page 71]

4.6.1 Exporting Data from the *Values* Tab

Context

The Export function is available from the *Values* tab's toolbar for a standalone or assembled component.

The Export file is packaged in JAR format and contains the following data:

- Data value row values in CSV format
- Each embedded object of the data value rows in XML format
- Anchoring relationships of the data value rows in XML format
- Information of local selections of inherited data value rows in CSV format

In the export JAR file package, each component's data value rows values are stored in a CSV file and include the following information:

- The column names
- All unselected local data value rows
- All selected local data value rows
- All child rows
- The above data value rows from all pages

> **i Note**
>
> If filtering is applied, only the filtered data value rows are exported.

- All types of attribute values of the above data value rows
  - The export process collects all attributes of the following data types: Number, Text, Date, Boolean, Text Area, URL
  - Attributes name are exported as column headings
  - If the attribute is an embedded object (such as a rule or a screen), the object ID is displayed
  - `PCD` as a separate column
  - `Effective Date` as a separate column
  - `Expiration Date` as a separate column
  - UUID as a separate column

> **i Note**
>
> If the attribute is of type Data List, only the value will be presented. The rule in the `Data List Source` component won't be exported.

- Saved changes
- Committed changes
- The column order in the exported file is the same as the attribute order in *Values* tab
- A special column that indicates whether the row is an attached or unattached row; this column becomes the first column in the CSV file

The CSV file doesn't include:

- Inherited data value rows (unselected, inherently selected, locally selected)
- Public, and locally deleted data value rows
- Locally overridden data value rows
- Unsaved changes
- Uncommitted changes

> **i Note**
>
> The CSV file contains Unicode (UTF-8) characters. Your spreadsheet application may require some configuration to ensure the Unicode displays correctly.

In the export JAR file package, each embedded object of the local /overridden data value rows is represented as an XML file, including the following objects:

- Rule
- Validation rule, including the calling error code

The following embedded objects aren't included in exported files:

- Embedded objects in inherited data value rows
- Embedded objects in locally overridden data value rows
- Embedded objects in unsaved changes
- Embedded objects in uncommitted changes
- Deviated rules
- Screens

The anchoring relationship information for exported data value rows is stored in an XML file (control.xml), that contains tree information and the names of the component and its subcomponents. As well, it contains anchoring information (local selections of local and inherited rows) between data value rows on the component tree.

The system supports export of multiple levels of components and anchoring. If you select to export at the top-level component and all its subcomponents, the information is also included in the JAR file. Exporting at a subcomponent level won't export any anchoring relationship.

The export JAR file package contains local selections of inherited data value rows information of the current *Values* tab as a CSV file.

Procedure

1. Open the component .
2. Select the *Values* tab.
3. Filters are reflected in the export. If you want to filter the data, do so now.
4. Select *Menu* > *Export* from the *Values* tab menubar.
   The *Export* dialog displays.
5. If you want to export the data to create an XLIFF file for translation, select *For Translation* as the *Content Type* and choose *Export*.

   The language name is obtained from the currently selected language in *Values* tab (non-product level), or the primary language (product level). The following actions occur:

   - The values in the specified language from any multi-language enabled attributes are exported in an XLIFF file with file name: `Language_Export_<selected language name>_<product name>(<version number>.0)_<component name>(<version number>.0)_<timestamp>.xlf`
   - Each XLIFF file includes the language code, the unique ID (UUID) of the data value row, and the multi-language enabled values to be translated.
   - The attribute values for multi-language enabled in local data value rows, including selected, unselected and overridden rows are exported or included in the XLIFF file.
   - Blank entries are exported with unique ID (UUID) and an empty value.
   - If there are unsaved changes, a warning message displays asking if user wants to proceed without saving before displaying the *Export* dialog.
   - If there is no local data, an error message displays.
   - If there is no attribute with *Enable Multi-Language* checkbox selected, an error message displays advising there is nothing to export for translation.
6. If you want to export data value rows, their embedded objects, selections of external rows and anchoring relationships, select *Data Rows* as the *Content Type*.

   If the attribute's *Enable Multi-Language* checkbox is selected, you can enter values in different languages. When performing a data value row export, only data entered in the selected or primary languages is exported.
7. If you want to export the file to a local folder, choose the *Save to Local Folder* option in the *Exported File Location* area of the dialog and choose *Export*.

   This is the preferred mode when exporting a *Values* tab containing a small amount of records. For larger *Values* tab exports, you should export to the `CSI Home` folder.

   A message displays indicating that the export has begun and another message is displayed when the export is complete. This local download process needs to finish before you can continue working in FS-PRO.

   The system creates a file name based on the name and version of the component, and adds a unique id number.

   The *View Downloads* dialog displays

   a. Select *Save* to save it into your default download directory or select *Save As* to browse and choose a new location.

8. Alternately, choose the *Save to CSI Home Folder* option in the *Exported File Location* area of the dialog and choose *Export*.

   This is the preferred mode when exporting a *Values* tab containing many records.

   A message displays, indicating that the export file is being downloaded to the Design Time Administrative Console.

   The system creates a file name based on the name and version of the component, and adds a unique id number.

   a. Choose *OK*.
   b. Open your web browser 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.
   c. Choose *System* > *View Logs* from the menu bar.
   d. Select the file by going to *Log Viewer* > *PC* > *Log* > *PC Logs* > *ExportDataRow* in the tree.
      When the download is initiated, a log entry is created indicating that the export has started. When the download is completed, another log entry with the attached download file is created.
   e. Click the file to select it.
      The *View Downloads* dialog displays.
   f. Select *Save* to save it into your default download directory or select *Save As* to browse and choose a new location.

4.6.2 Importing Component Values

The Import function is available from the *Values* tab's toolbar for a standalone or assembled component. You can import data from a JAR file or from a CSV file.

Prerequisites

Users must have `WRITE` or `FULL` permission to the component or containing product in order to be able to import data value rows. Otherwise, the *Import* option is disabled.

Context

There are some points to consider before importing data value rows.

- Attribute names in the file should match those in the target; attributes with names that don't match exactly are excluded.
- If the file contains fewer attributes than the target component, the extra attributes on the target are left empty, or receive default values if those have been set.
- If the file contains more attributes than the target component, the extra attributes are excluded from the import.

- An import operation only removes data value rows if the `Replace` data option is selected.
- Child rows are identified and inserted based on `PCD` value and/or key column.
- If the target component or any of its data value rows are locked, during import a warning message appears.
- Whether you access the target component standalone or from within a product, the imported values have local status by default.
- Uncommitted data value rows won't be updated/appended/replaced.

If you select a JAR file as the upload file containing CSV files with data value rows information, the data value rows will be imported. If there is no sub-component, then the only CSV file in the JAR file will be imported to the current component. If there are sub-components, then the system will import to each sub-component with name that matches the CSV file name.

You can also import data directly from a spreadsheet. The requirements are as follows:

- You can choose to export the data and manually make changes, or create the CSV file from scratch. You can import the CSV directly, or contain it in a JAR file.
- The columns in your spreadsheet should have the same names as the attributes in the target component.
- You can only import values that can be converted to one the following data types: Number, Currency, Text, Date, Boolean, Text Area, or URL.
- Empty trailing columns in a spreadsheet may cause the CSV file to be invalid.
- When importing currency values, don't include currency symbols, such as "$", or formatting, such as commas (,).

If you want to set the selection status of the data value rows, name the first column in your spreadsheet **`Selection`** and input **`0`** for unselected rows and **`1`** for selected rows. The value in the `Selection` column only affects the row itself; its child rows must be selected separately.

> **i Note**
>
> Unselecting a row in the import file doesn't override the selection status of that row in the target. After import the row remains selected.

> **i Note**
>
> If you select a CSV file as the upload file, since a CSV file doesn't contain any embedded object and anchoring relationship, only data value rows will be imported.

You may have CSV files containing data value rows information that have been exported from previous versions of Product Modeler.

Changes are recorded in the log file, as follows:

- If a data value row in the source file isn't found in the current component, a message displays in the log file. The data for that data value row isn't imported.
- If an attribute in the source file isn't found in the current component, a message displays in the log file. The data for that attribute isn't imported.
- If a value of the source language in the source file isn't the same as in the current component, a message displays in the log file. The data for that value isn't imported.

If the component name in the source file is different than the current component, an error message displays. No data will be imported.

Note the following considerations when using the *Import for Translation* for the *Values* tab

- If there are unsaved changes, a warning message displays asking if you wish to proceed without saving before displaying the *Import* dialog.
- If there are no attributes with the *Enable Multi-Language* checkbox selected, an error message opens advising there is nothing to import for translation.
- Values from the file are imported in the language selected in the *Values* tab (non-product level), or to the primary language (product level).
- If the import file's language code is different from the selection, an error message displays.
- If the import file's multi-language attributes don't match those in the target, a warning message displays.
- If the import file's source language is the same as the target language, a warning message displays.
- Import only accepts XLIFF for components.
- After a successful import, the *Values* tab is refreshed and populated with the imported data.
- Only local data value rows will be updated with the imported language. Existing values for the import language might be replaced by the import.
- No new non multi-language data will be created or updated. Data value row selections won't change.

Multi-language support is handled in the following way for imports:

- If the import file is a CSV file, the values are imported in the language selected in *Values* tab (non-product level), or to the primary language (product level).
- If the import file is a JAR file, the system validates if the language code from the file is the same as the language selected in *Values* tab (non-product level) or the primary language (product level). If they are the same, the values are imported in the language specified. If they are different, an error message displays and no import occurs.
- When the import finishes, the *Values* tab shall be refreshed to show the content in the imported language.

There is criteria for matching `PCD` or key columns when the `Append` option is selected. If you select the `Append` option it adds new rows or new child rows but doesn't perform any existing data value row updates. If the data value row's `PCD` (or key value if no `PCD` is present) in CSV file matches that in *Values* tab and the UUID value is blank or isn't used, a child row is created for the matching row. If the same `PCD` is matched multiple times and the UUID value is blank or not used, all matching rows will be appended as child rows. If the data value row's `PCD` (or key value if no `PCD` is present) in CSV file doesn't match any of that in the current *Values* tab:

- If the `PCD` isn't in use by another data value row in the system and the UUID value is blank or isn't used, a new row is created.
- If the `PCD` is in use by a data value row in the system, no new row is created.

If no `PCD` or key value is provided for a data value row in CSV file and the UUID value is blank or isn't used, a new row is created.

There is criteria for matching `PCD` or key column when the `Update` option is selected. If you selected the `Update` option, it adds new rows/new child rows and updates existing data value rows. If the data value row's `PCD` (or key value if no `PCD` is present) and UUID in CSV file matches that in *Values* tab, the data value row's *Values* tab value and selection (but not deselection) is updated with the import file's value and selection. If the same `PCD` is matched multiple times and the UUID value is blank or isn't used, the subsequent matching rows will be updated as child rows. If the data value row's `PCD` (or key value if no `PCD` is present) in CSV file doesn't match any of that in the current *Values* tab:

- If the `PCD` isn't in use by another data value row in the system and the UUID value is blank or isn't used, a new row is created.
- If the `PCD` is in use by a data value row in the system, no new row is created.

If no PCD or key value is provided for a data value row in CSV file, a new row is created.

There is criteria for matching PCD or key column when the Replace option is selected. If you selected the Replace option, it deletes all existing data value rows and replaces the *Values* tab with new data value rows. It doesn't matter if the data value row's PCD (or key value if no PCD is present) is matching or not, all data value rows in *Values* tab (including their embedded objects) are deleted and replaced with those in the import file. If the same PCD is found on multiple rows in import file and the UUID value is blank or not used, the subsequent rows will be inserted as child rows. If the same PCD and/or UUID is in used by a data value row in the system, no new row is created.

> **i Note**
>
> If there are sub-components, all data value rows in current and all sub-components will be replaced.

The table below summarizes the outcome (assuming the UUID column in the import file is blank):

Behavior of the Import Action

Import Option	Does the PCD or Key Exist in File?	Does the PCD or Key Exist on the Target	Result in Target
Append	Yes	No	New row is created if specified PCD isn't in use by another row
	Yes	Matches local row	Child row is created
	Yes	Matches row elsewhere in system	No change. No new row is created
	Multiple rows exist	Matches local row	Multiple child rows are created
	No	-	New row is created
Update	Yes	No	New row is created if specified PCD isn't in use by another row
	Yes	Matches local row	Row is updated
	Yes	Matches row elsewhere in system	No change. No new row is created
	Multiple rows exist	Matches local row	Row and child rows (if existing) are updated. If more rows are in file, new child rows are created
	No	-	New row is created

Import Option	Does the PCD or Key Exist in File?	Does the PCD or Key Exist on the Target	Result in Target
`Replace`	Yes	Doesn't matter	Existing row is deleted and new row is created if specified **PCD** isn't in use by another row
	Multiple rows exist	Doesn't matter	Row and child rows are created if specified **PCD** isn't in use by another row
	No	-	New row is created

In summary, in order to update a data value row, the UUID provided in the import file should be blank, or the **PCD** and UUID should match in source and target.

In order to replace or add a new row (parent or child row), the UUID value should be blank or unused in the target.

If the value in the import file can't be imported as the target attribute type, then the data isn't imported. The table below summarizes the outcome.

Matching Attributes

Does the Attribute Exist in the File?	Does the Attribute Exist in the Target?	Result in Target
Yes	No	Data isn't imported
Yes	Matches name and type	Data is imported
Yes	Matches name only	No change for existing rows, blank for new or replaced rows
No	Exists	No change for existing rows, blank or default value for new or replaced rows

The following is the criteria for a data value row selected in the *Values* tab.

- A data value row selected in the *Values* tab won't be unselected (except for the `Replace` option) even though selection value for that data value row is set to 0 in import file.
- To import a selected child row, in the import file, selection must be = 1, and effective and expiration dates must be provided in a valid format. The parent row must be selected already and has an expiration date. Otherwise the child row won't be imported.
- The selection column will be ignored if you import to a standalone component, or a component that is a sub-component.

The table below summarizes the expected results.

Selections

Import Option	Selection in CSV ...	Selection in target ...	Result in Target
`Append`	0	Unselected	Unselected child row created
	0	Selected	Unselected child row created
	1	Unselected	Unselected child row created
	1 (with valid effective or expiry dates)	Selected (with valid expiry dates)	Selected child row created
	1 (with valid effective or expiry dates)	Selected (no valid expiry dates)	No child row is imported
	1 (no valid effective or expiry dates)	Selected (with valid expiry dates)	No child row is imported
	1 (no valid effective or expiry dates)	Selected (no valid expiry dates)	No child row is imported
`Update`	0	Unselected	Data value row remains unselected
	0	Selected	Data value row remains selected
	1	Unselected	Data value row becomes selected
	1	Selected	Data value row remains selected
`Replace`	0	Unselected	Data value row is replaced and unselected
	0	Selected	Data value row is replaced and unselected
	1	Unselected	Data value row is replaced and selected
	1	Selected	Data value row is replaced and selected

Procedure

1. Open the component .
2. Select the *Values* tab.

3. Select *Menu* > *Import*.
   The *Import* dialog displays with the title *Import to English* (with English being the currently-selected language from the *Values* tab for the non-product level or the primary language for the product level).
4. If you are importing data value rows, perform the following steps:
   a. Select the *Data Rows* radio button.
   b. Select an import type. Choose either the *Append*, *Update*, or *Replace* radio button.
   c. Choose *Browse...* at the *Upload File* field.
      The *Choose File to Upload* dialog displays.
   d. Select the file and choose *Open*.

      The file can be in JAR format. The system will also accept CSV format import file for backward compatibility.

      The *Import* dialog refreshes with the *Upload File* field populated with the selected file name.
5. If you are importing translated content, perform the following steps:
   a. Select the *For Translation* radio button.
   b. Choose *Browse* at the *Upload File* field.
      The *Choose File to Upload* dialog displays.

      Select an XLFF file with translation data in the same language as the selected language (for non-products) or the primary language (for products).
   c. Select the file and choose *Open*.
      The *Import* dialog refreshes with the *Upload File* field populated with the selected file name.
6. Choose *Upload*.

   Data value rows are imported as per the selected option and import file.

   The audit trail records all addition, modification, or deletion of data value rows.

   A *Status* dialog box is displayed when the import is completed.

   > **i Note**
   >
   > If no valid data is found in the import file, an error message is displayed.

7. If you want to download the import log, perform the following steps. Otherwise, proceed to the next step.
   a. Select the *Please click here* link to download the import log
      The *File download* dialog displays.
   b. If you want to view the log, click *Open*.
      The *data_Import _Identification* log window displays. View the log and then close the window.
   c. Choose *Save*.

      The *Save As* dialog displays.
   d. Click the dropdown arrow by the *Save in* field and navigate to the location to save the log file.
   e. Save your changes.
      The *Download complete* dialog displays.
   f. Choose *Close*.
8. Select *OK* on the *Status* dialog to continue without viewing or saving the import log.

Results

After the import is complete, the following takes place:

- The *Values* tab is refreshed to display latest values.
- The column order in the *Values* tab remains the same. The column order in the import file doesn't affect the column order in the *Values* tab.
- The sort order in the *Values* tab is refreshed to the default sort order. The sort order in the import file doesn't affect the sort order in the *Values* tab.
- All imported new data value rows will be locked by the current user.
- All imported existing data value rows won't be locked.

4.7 Publishing

Before a product can be used at runtime, it must first be published, promoting it to the Product Repository. A published product is available for use by other systems.

In the Product Modeler, you publish a product in order to make it available on other servers, typically for development, testing, or production, and for access by external systems.

You can only publish products and coverages, or the objects contained in them. Standalone components and questionnaires can't be published.

Publishing is actually an umbrella term for a pair of separate but complementary operations: packaging and deploying. The prerequisite to these operations is building.

> **i Note**
>
> Another related operation that occurs during publishing is that the product definition XML is generated.

What is Building?

Each object that you create in the Product Modeler requires a Runtime version that is known as the compiled version. As well, all objects require artifacts for use at Runtime. These artifacts vary from object to object. For example, when you build a rule, the Product Modeler creates the following artifacts: a Java File, a Class File, Rule XML, and Rule Definition XML.

In a product, you should perform a `Build` of an object whenever you add or change it, and you must build the object before you can run or deploy it. For example, if you edit an existing rule, you must use the `Build` command in the *Rule Painter*. Building generates XML for the relevant product area, and in some cases generates Java classes. In the modeling tool, you can build the following object types: Rules, Metadata, Questionnaire Views, and Screens.

When you select one of the `Build` commands from the modeling tool's *Build* menu, all objects of the selected type are compiled and their artifacts are generated. Unbuilt objects can't be packaged or deployed (that is, published).

What is Packaging?

The package operation collects all the outputs of a `Build` (compiled objects and their artifacts) from within your product and copies them into a JAR file. If any artifact is missing, the Product Modeler reports an error. If the JAR file doesn't exist, the package operation creates it (the JAR file is always saved locally). Thus the JAR file contains the Runtime version of your product. After an object has been built and packaged, it can be deployed.

> **i Note**
>
> There isn't a "packaging" command. Instead, packaging happens as part of the `Publish Product`, `Publish Below`, and `Update Object` commands.

What is Deploying?

The deploy operation copies the product JAR file to servers that you registered for this product, using the Administrative Console. For example, when a new product is ready for testing, you deploy its JAR file to your testing server, selecting from the list of registered servers.

> **i Note**
>
> You can only deploy the JAR file to servers previously registered to the product. For more information, see the Administration Guide.

What are Pre-Publishing and Post-Publishing Services?

Pre-publishing and post-publishing services are user-defined services that consist of custom Java classes containing business logic that you want executed either immediately before or after a product publish operation. For example, you could use a post-publishing service to send an email notification that the publish operation has completed.

In FS-PRO, by default pre-publishing and post-publishing is disabled. To enable it in the environment, go to *System* > *Edit Configuration Settings* in the Design Time Administrative Console. Then go to *PC* > *Env* > *PC Environment* in the *Configuration Manager* and enable the `PrePostPublishingEnabled` configuration variable.

For a product to use pre-publishing and post-publishing services, the product must inherit, or you must add in, the `Custom Publishing` component, which is one of the system objects. Each custom class that you want to run is recorded as a data value row in the component. One of the attributes enables you to specify whether the service is to run before or after publishing. You can also define a service as mandatory, in which case it always runs when a full product publish occurs. During a full product publish, the *Publish* dialog lists the publishing services contained in `Custom Publishing`, and you select or deselect the services that you want to run with the publish operation.

Multi-language Support and Publishing

Published product JAR files support multi-languages, provided multi-language support and the desired language are enabled in the Administrative Console. If these conditions are met the JAR file will contain the following:

- Language code
- Language description
- ABAP language code

The Different Ways to Publish

You control when and how publishing occurs, and thus you can increase the efficiency of your development effort. The ways to publish and their appropriate uses are outlined in the following table:

Publishing

	Product publishing	**Product branch publishing**
Command:	`Publish Product`	`Publish Below`
Uses:	To create the product JAR file. To create a complete and up-to-date version of the product.	To publish part of a product structure as you create a product, thus allowing other team members to access your work sooner.
Affects:	The entire product	A selected part of the *Product Tree*.
Prerequisites:	Build all objects	The product JAR file must already exist. You are prompted to build all the objects in the branch, or selected objects types in it; all objects are moved.
Packaging:	Creates or recreates the local JAR file and all the artifacts in it, taking into account all additions, edits, and deletions.	The target object and all its children are added to/replaced in the JAR file.
Deployment:	Before publishing, presents option to deploy the JAR file, showing the list of target servers registered to this product.	Before publishing, presents option to deploy the JAR file, showing the list of target servers registered to this product.

Related Information

Building All Objects of a Type [page 146]
Publishing a Product [page 147]

4.7.1 Building All Objects of a Type

Use the following procedure to build all objects of a given type in a product.

Context

The following object types are available: Metadata, Questionnaire View, Screen, and Rule.

Procedure

1. Open your web browser and log in to the Product Modeler at the following URL: `https://<pro_designtime_app_url>/csiroot/ii/pc/`
   The Product Modeler will open after a short delay.
2. Open the product .
3. Go to *Build* > *Build* `<object_type>`.
4. Confirm your choice.
5. If you chose *Build Screens* or *Build Rules*, by default, only changes that users have committed are included in the publish operation.

   To include the data value rows currently locked by yourself or other users, choose *Include uncommitted changes* and move users from the *Available Users* list to the *Selected Users* list.
6. Choose *Build*.

   > **i Note**
   >
   > If the build operation fails, the dialog indicates the nature of the problem. You must correct the objects indicated before rerunning the build function.
7. When the process finishes, choose *Close*.

4.7.2 Publishing a Product

Publishing (creating a product JAR file), deployment, indexing, and product specification documentation generation activities are all accessed from a single dialog.

Context

You can perform one or more of the following optional activities from the *Publish Product* dialog:

- Publish products
- Deploy products
- Index products
- Create product specification documentation

> → Tip
>
> At least one of the above activities must be selected to enable the *Execute Now* or *Execute Later* options. Until an activity is selected, only the *Close* option is enabled.

> ⚠ Caution
>
> The *Execute Later* option is not currently supported. Selecting this option will trigger the immediate execution of the publish task.

> i Note
>
> Additional options can be selected for these activities.

The following procedure describes how to optionally publish a product to the local product JAR file; optionally to deploy it to servers registered for the product; optionally index products and optionally create product specification documentation. If you elect not to publish from the *Publish Product* dialog, the other choices are still available.

If the product JAR file doesn't exist and publishing isn't selected, an error message is displayed. If the product has children, you have the option of publishing them as well.

If your product uses custom classes and you want to be able to manage them in an external source content management system, you can control whether these classes are included in the JAR file by a setting in the product's properties.

> i Note
>
> Before publishing a versioned (cloned) product, you must build its screens and rules.

Procedure

1. Open your web browser and log in to the Product Modeler at the following URL: `https://<pro_designtime_app_url>/csiroot/ii/pc/`
   The Product Modeler will open after a short delay.
2. Open the product .
3. Go to *File* > *Publish Product*.
   The *Publish Product* dialog displays.
4. In the *Products* panel, perform one of the following actions:
   a. Select the *Execute the selected activities on the current product only* radio button.
   b. Select the *Execute the selected activities on the current product along with selected child products* radio button.

      Select the checkbox for the appropriate child products.
5. In the *Publish Products* panel, the *Publish selected product* checkbox is selected by default.
   a. Deselect the *Publish selected product* checkbox if you don't want to publish.
   b. Select the *Include Product Metadata and DefaultData XML* checkbox if you want to include the product metadata and default XML in the published JAR file to create product documentation. Ensure this option is deselected if you aren't generating product documentation.
   c. Select the *Build DataDefinition* checkbox if you want to rebuild the data definition when publishing. Ensure that this option is deselected if you don't want to rebuild the data definition.
6. If you want to publish, perform one of the following actions:
   - Select the *Publish (Committed changes only)* radio button.
   - Select the *Publish (Include uncommitted changes)* radio button

   If publishing is selected, products will be published first, and then deployment and doc generation will be performed if the activities are selected as described in steps 7 and 9.

   If you elect not to publish the products, as described in step 4 above, and they have never been published, and you select deployment or doc generation activities an error message is displayed `JAR file for the products does not exist. Please publish the selected products and try again`.
7. In the *Pre and Post Publishing Services* panel, (if pre and post publishing services are available in your environment) select the checkbox for the appropriate options.

   By default the *Publish* checkbox is selected. If you are integrating this product with FS-PM then select both the *Send Structure Info to IFBC* checkbox and the *Upload & Deploy to Runtime* checkbox.
8. In the *Deploy Products* panel, select the checkbox for the appropriate server.

   > **i Note**
   >
   > If no server is selected, no product JAR file is deployed.

   The list of servers in this panel is created in the Administrative Console using *Edit Product Deployments*. For more information, see Working with Edit Product Deployments.

   Deploy Product activity is executed after publishing if publishing is selected.
9. In the *Index Products* panel, select the checkbox for the target databases to index the selected products.

   Indexing is executed at the same time as publishing if publishing is selected.

10. In the *Create Product Specification Documents* panel, select the checkbox for the appropriate view.

    If the *Product Specification* checkbox default is set to unchecked, select the checkbox and select the desired view from the *View* dropdown list.

    A Product Specification Document is generated from the product JAR file created from the publishing action, if publishing is selected. Or from previously published JAR file, if publishing isn't selected.

    > → Tip
    >
    > You must select a View in order to generate documentation.

    The Create Product Specification Documents activity is executed after publishing if publishing is selected.

    You are able to specify the default state of the *Product Specification* checkbox and view when publishing a single product or a family or products.

11. If you want to schedule the execution of the operation, select *Execute Later*.

    > ⚠ Caution
    >
    > The *Execute Later* option is not currently supported and will trigger the immediate execution of the publish task.

    The *Schedule Options* dialog displays.

    a. Enter the date and time that you want the operation to run.
    b. Select *Execute*.

12. If you want to schedule the execution of the operation, select *Execute Now*.
13. Selected activities are sent to the *Scheduled Activity Manager*.

    If single product is selected, one entry is created in the *Scheduled Activity Manager* for each selected task for the single product.

    If product family is selected, one entry is created in the *Scheduled Activity Manager* for each selected task for the whole family of products.

    > ⚠ Caution
    >
    > If the `PUBLISH` task for a product is in `COMPLETED WITH ERROR` status, no `DEPLOY` or `DOC GEN` task will be created.

    You are able to check task statuses and download generated documentation from the *Scheduled Activity Manager*. You are able to check task statuses and download generated documentation from the *Scheduled Activity Manager*.

Results

After completing this procedure, you can continue working or you can log off from the system. If you are logged in, a message notifies you when the publish operation is complete; if you are logged off when the request completes, at your next login you will receive the completion message.

You can view the publish queue at any time.

Related Information

Managing Publishing Requests [page 153]
Creating Objects [page 41]
Editing an Object's General Properties [page 42]
Setting a Default Documentation View [page 150]

4.7.3 Understanding Product Activity in the Audit Trail

For each of the product activities listed below, an entry is recorded in the Audit Trail:

- Publish Product
- Deploy
- Index
- Documentation generation

If a product family is selected, activity on each product will be displayed as one record in Audit Trail.

Related Information

Creating a Version of an Object as a Clean Inheritance of its Parent Product [page 60]

4.7.4 Setting a Default Documentation View

You can specify a default state of the *Product Specification* checkbox when publishing a single product or a family or products.

Procedure

1. Open your web browser 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* > *Edit Configuration Settings* ◀ from the menu bar.
3. Navigate to ▶ *Configuration* > *PC* > *Env* > *PC Environment* > *ProductSpecDocGenAfterPublishSelection* ◀
   - When `ProductSpecDocGenAfterPublishSelection` is set to Yes, the generate documentation checkbox is selected by default in the *Publish* dialog. You are still able to manually deselect it.
   - When `ProductSpecDocGenAfterPublishSelection` is set to No, the generate documentation checkbox is deselected by default in the *Publish* dialog. You are still able to manually select it.

The out-of-the-box `ProductSpecDocGenAfterPublishSelection` value is off (No value).

4. Select Yes in the *Override* field to turn it on.
5. The out-of-the-box `DefaultProductSpecDocView` value is blank. To specify another *View*, enter the name in the *Override* field.

Results

The specified `DefaultProductSpecDocView` value will be selected by default on the *View* dropdown list in the *Publish* dialog. You are still able to manually change the selection.

If the specified *View* isn't applicable for the current product/product family, no *View* will be selected by default on the dropdown list.

4.7.5 Publishing Product Branches

Use the following procedure to publish a specific area of a product's structure to the local product JAR file.

Context

Because this procedure rebuilds only the objects that have changed since the last build, you can use it to save time during development. To publish a branch, the local product JAR must already exist.

> **i Note**
>
> This procedure adds new content to the JAR file and replaces existing content.

Procedure

1. Open your web browser and log in to the Product Modeler at the following URL: `https://<pro_designtime_app_url>/csiroot/ii/pc/`
   The Product Modeler will open after a short delay.
2. Open the product  .
3. Right-click the object directly above the object that you want to publish, and select *Publish Below*.

   > **i Note**
   >
   > The object that you select as the parent must already have been published to the JAR, otherwise you must select an object earlier in the tree path that has been published.

   The *Publish Below* dialog appears.

4. If you want to apply the changes to inherited products, select the checkbox for the desired products in the *Product(s)* pane.
5. In the *Objects* section, select the object types to build.

   If no objects are selected, all objects will be built and published.
6. Ensure that the *Publish selected product(s)* and *Build All DataDefinition* checkboxes are selected.
7. By default, only changes that users have committed are included in the publish operation. To include the data value rows currently locked by users, select *Include uncommitted changes* and move users from the *Available Users* list to the *Selected Users* list.

   Your user ID is included in the *Selected Users* list by default.
8. In the *Deploy Product(s)* section, select or deselect the target servers.

   If no server is selected, the publish deploys to the local server only.
9. Choose *Publish Below*.

   The dialog closes and a new dialog displays the progress of the operation.

   > **i Note**
   >
   > If the publish operation fails, the dialog provides a link to the error message. You must correct the objects indicated, and then rerun the publish function.
10. Choose *Close*.

4.7.6 Deploying Without Publishing

The following procedure enables you to deploy a JAR file or the JAR files of a family of products to a registered server without having to republish.

Context

This is useful, for example, when you have finished all testing on your development server and want to deploy the JAR files to your testing server to begin the QA process.

Procedure

1. Open your web browser and log in to the Product Modeler at the following URL: `https://<pro_designtime_app_url>/csiroot/ii/pc/`
   The Product Modeler will open after a short delay.
2. Open the product .
3. Go to *File* > *Deploy Products* .
4. Do one of the following:

- If the product has no children, choose Yes to confirm your decision.
- If the product has children, indicate whether you want to deploy it alone or its children as well, and choose *Next*.

The *Deploy* dialog appears.

5. If you are deploying a single product:
   a. By default, only changes that users have committed are included in the deploy operation.
   b. Select or deselect the target servers in the *Deployment Servers* section.

      If no server is selected, deployment is to the local server only.
   c. Choose *Deploy*.
6. If you are deploying a family of products:
   a. Select or deselect the products in the *Product Tree* that you want to deploy (you need to have `Write` permission for any product that you select).

      Note that you can deploy any combination of products, and that you can deploy any child without deploying its parent.
   b. Select or deselect the target servers in the *Deployment Servers* section.

      If no server is selected, deployment is to the local server only.
   c. Choose *Deploy*.

4.7.7 Managing Publishing Requests

The Product Modeler provides a tool for viewing and managing the publish requests and other asynchronous activities.

You access this tool in the same way both from the *Studio* tab and from within a product.

Publishing requests can have any of the following statuses: `Pending`, `In Progress`, `Scheduled`, `Completed`, or `Completed with Errors`. You can view the requests by any of these statuses. You can delete pending requests, as well as view the logs of in-progress or completed requests (including those with errors).

Related Information

Managing Scheduled Activities [page 70]

4.8 Downloading Product JAR Files

Publishing a product creates a JAR file containing the product's XML. Use the product JAR download feature to copy this JAR file to your computer for backup, promotion, or debugging.

Procedure

1. Open your web browser and log in to the Product Modeler at the following URL: `https://<pro_designtime_app_url>/csiroot/ii/pc/`
   The Product Modeler will open after a short delay.
2. Open the product .
3. Go to *File* > *Download Product (JAR)* .
   A dialog appears.
4. Choose *Download*.
   A system dialog appears.
5. Choose *Save* and specify where you want to store the JAR file.

4.9 Coverage-Based Business Terms

The coverage-based data model is comprised of several specific elements. A better understanding of each element and its purpose will help ease the product configuration process.

Products

Monoline Products

Monoline products contain a single line of business.

Package Products

Package products bundle multiple lines of business. Within a package you can have contracts, which can have many coverages, and so on.

Contract

A contract is a grouping of coverages that are typically associated with a specific insurance Line of Business (LoB), such as Auto, Property, Liability, Marine, and so on.

In the case of a package product, the product model would contain a contract to represent each LoB within the package. For example, travel insurance contains the following LoBs: Personal Accident, Travel Inconvenience, and Personal Liability.

Each policy must have at least one contract.

Coverage

A coverage is the scope of protection provided under an insurance policy. For example, Comprehensive, Third Party Fire and Theft, Third Party Only Liability: General Liability, Directors and Officers, Errors and Omissions, Fiduciary are all types of coverages.

Coverage Bundle

Coverage bundles are groupings of coverages. This allows for attachments at the coverage bundle level, such as common clauses or financial reporting.

Sub-Coverage

A sub-coverage defines a specific insurance loss. For example, a peril covered under a coverage. Claims systems use sub-coverages to establish a mapping of the peril to the Claims Loss Tree, which establishes whether the loss is covered by the policy.

5 Data Models

The purpose of a data model is to define tables, and fields within tables, that will be used by your application/product. A data model is composed of data definition objects that contain related data elements and attributes. A data model can be created and configured through the following steps:

Creating the Data Model

Create a new object and set it's *Modifier Type* to `Data Model`. This acts as a container for the data definitions.

Creating the Data Definition Objects

Next, create your data definition objects and set their *Modifier Type* to `Data Definition`. These objects must be based on a source. They will inherit the attributes from the source. From the *Based On* dropdown on the *New Object* dialog, choose either `Data Definition Base` or your company data definition template. Afterward, define the table name and description in the `Table` and `Column` components of each data definition object. The `Column` component in the `Data Definition Base` object contains a set of fields required by all tables. These fields should always be selected. You can add additional fields to your new data definition objects as required. The `column name`, `description`, `data type`, and `data length` attributes are required for each new column. The `description` and `column name` must match.

Assembling the Data Model

Assemble the data definitions within the data model, as required. When you add in the data definition objects to the model, you should define the relationship between the tables. You can assemble a data definition object under another data definition object within the data model if you want to define a parent-child relationship.

Building and Publishing

The data model's tables and columns will need to be made available to objects, such as products and model views. This is accomplished by building and publishing the data model.

Open your data model in the FS-PRO *Product Studio*. Select *Build* > *Metadata*. Next, select *File* > *Publish Product*. Ensure that the *Build DataDefinition* checkbox is selected, and select *Execute Now*. Go to the *Scheduled Activity Manager* and verify that the publish completes without errors.

Related Information

Configuring the Application Data Model Product [page 157]
Associating Data Models to a Product [page 159]

5.1 Configuring the Application Data Model Product

If you will be using FS-PRO to define tables, you will need to create an application data model product. It will act as the container for all data models that define main tables in your repository.

Configuration

The application data model product can be configured like any other data model. You may also assemble other data models within the application data model product.

When creating the application data model product, you will need to perform two one-time configuration settings in the design time administrative console.

First, open the Administrative Console, and navigate to the configuration settings under *System* > *Edit Configuration Settings*. Navigate to the `ApplicationDataModelProduct` configuration variable under *Configuration* > *System* > *Env* > *Content Setting*, and ensure that the configuration variable's *Override* value is set to the name of your application data model product.

> **i Note**
>
> If you are deploying the application data model product, you will need to change the same setting in the Runtime Administrative Console before deploying.

Second, open the Design Time Administrative Console and go to *Product* > *Edit Product Deployment*. Select the application data model product and enter the server name, user name and password in the applicable fields. Save your changes. This will provide access to your application data model product's tables and columns in the Runtime instance.

> **→ Tip**
>
> Test the connection to verify that the settings are valid

Building and Publishing

In order to make the application data model product's tables and columns available to applications, it needs to be built and published.

Next, return to the *Product Studio* in FS-PRO and open your application data model product. Select *Build* > *Metadata*. Next, select *File* > *Publish Product*. Ensure that the *Build DataDefinition* checkbox is selected, and select *Execute Now*. Go to the *Scheduled Activity Manager* and verify that the publish completes without errors.

> **i Note**
>
> The application data model product must be published at the data model level. Data definition objects should not be built separately, and the *Publish Below* feature is not supported for the application data model product.

Use in Products

In order to use the application data model product in your products and model views, you will need to modify the value rows in the `Config` component of each product or model view, found under the `Configuration` folder.

Select the `Config` component, then select the *Values* tab. Next, override the `REFERENCE_PRODUCT` and `REFERENCE_PRODUCT_VERSION` data value rows. Set the name of your application data model product in the value field of the `REFERENCE_PRODUCT` data value row and set the version in the `REFERENCE_PRODUCT_VERSION` data value row.

> **→ Remember**
>
> You can configure standalone products to use an individual data model, instead of the application data model product.

Now you should have access to the tables and columns defined in your application data model product in your rules and views.

To generate the tables and table columns, you need to synchronize the process flowstore on the runtime instance For more information, see Synchronizing the Database to all Flowstores.

> **⚠ Caution**
>
> If you do not synchronize the process flowstores, the application won't work

5.2 Associating Data Models to a Product

Within the product, you need to indicate which data model will be utilized when designing the product and during Runtime.

Prerequisites

The data models must already have been defined.

Procedure

1. Set the main axis data model for the product.
   a. Go to `<product_name>` > *Configuration* > *Config* .
   b. Select the *Values* tab.
   c. Set the `REFERENCE_PRODUCT` key record to the main axis data model that you want to reference.
   d. Set the `REFERENCE_PRODUCT_VERSION` key record to the version number for the data model you want to reference.
   e. Save your changes.
2. If you have an extension data model, assemble the extension data model into the product.
   a. Go to `<product_name>` > *Data Model* .
   b. Assemble the extension data model.
3. If you have an extension data model, set the extension data model as a reference.

   > **! Restriction**
   >
   > This step is only applicable to coverage-based products where an extension is not assembled under the `Data Model` folder.

   a. Go to `<product_name>` > *Configuration* > *Config* .
   b. Select the *Values* tab.
   c. Set the `REFERENCE_PRODUCT` key record to the main axis data model that you want to reference.
   d. Set the `REFERENCE_PRODUCT_VERSION` key record to the version number for the data model you want to reference.
   e. Save your changes.
4. Specify whether the EDMX data model needs to be generated during publish. There is a flag in the `Config` component that instructs the Publish task to build the EDMX data model. This data model is required for the `OData UI5 Renderer`.
   a. Go to `<product_name>` > *Configuration* > *Config* .
   b. Select the *Values* tab.
   c. Select the `GenerateEAppEDMX` key record and set the value to `true`.

d. Save your changes.

6 Data Object Layer

6.1 Configuration Model Data Using Data Definition

In products, risk entities have common attributes such as Risk Id and Premium (roll up). LOBs, coverages and other entities also have their own common sets of attributes. This configuration model requires you to define these common attributes at each entity level separately. It allows you to access them using `<object name>.<attribute name>` notation in script rules. With this model you must repeat the common algorithms (implemented in rules or Java code) over and over again for each of the entities. This configuration model only supports an entity relationship database.

Data Object Layer New Function

Data Object Layer allows you to model data objects using object modeling techniques including composition and inheritance. In addition it allows you to provide ORM mapping.

Data Object Layer makes it possible to reuse the rules and Java code that operate on abstract objects thus saving time and increasing efficiency.

For persistent objects, features include:

- You can define hierarchical data objects and attributes, and map the data objects and attributes to physical database tables and columns.
- Data Definition object allows you to specify data object metadata and data object mapping to physical database tables.

For transient objects, features include:

- The separation of the business layer from the presentation layer allows for the development of business layers that fully define a business domain model and a presentation layer that may present the data with varying degrees of denormalization.

ORM Mapping:

- An object-relational mapping layer allows custom code or script rules to retrieve and store these objects in a database.

Data Object Layer Notes

- Data objects shouldn't be versioned (the physical database schema doesn't support versioning).
- You can use either the data definition model or the data object model in the same product but not both. The data definition table pickers won't work for the data object model.

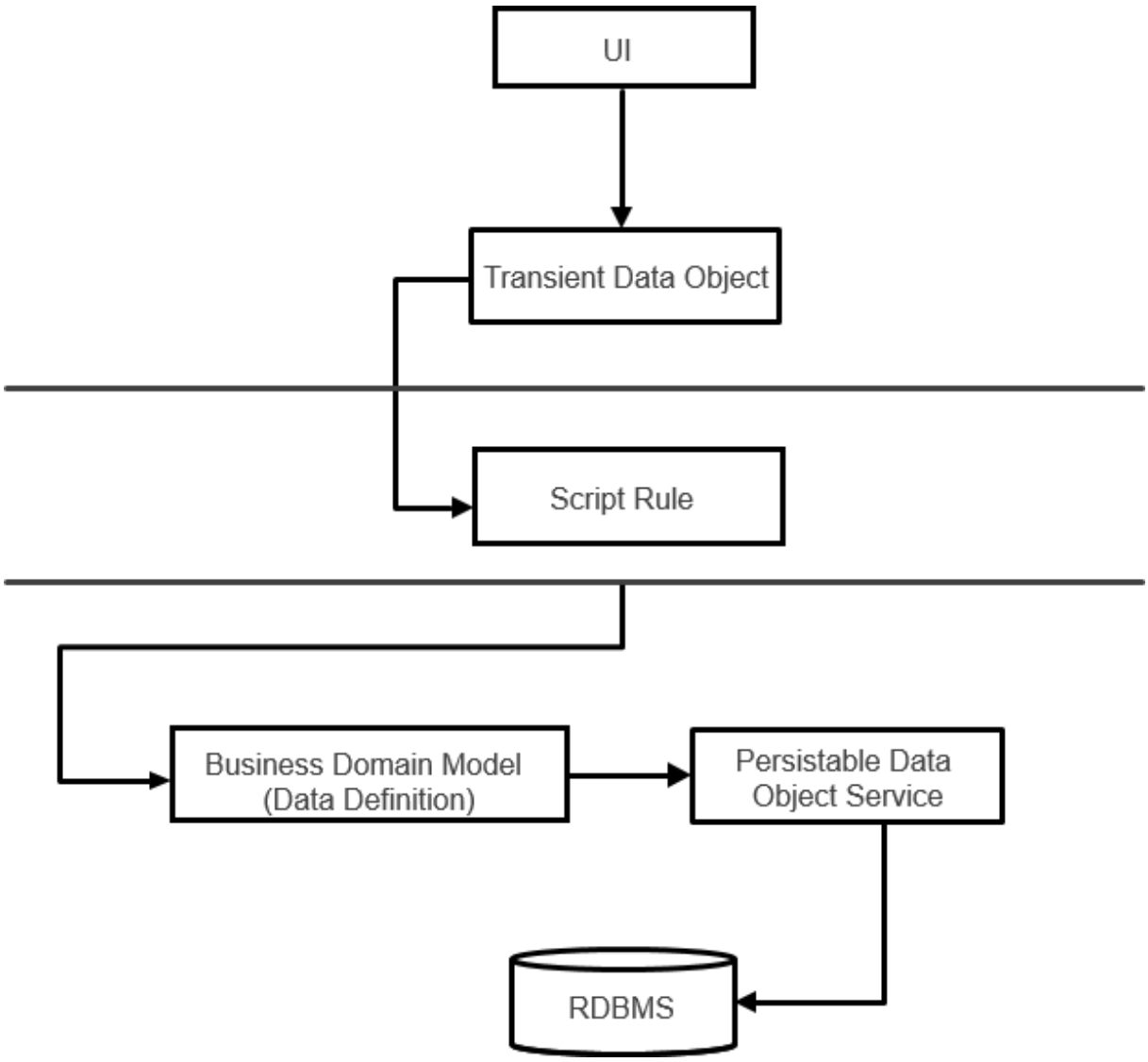
- The lazy select insert function is already included for the data object model. There is no need to switch on the `LazyFlowstoreInsert` configuration variable in Administrative Console. It is only applicable for data definition model.

6.2 Understanding the Object Relational Mapping Layer

Transformation of data from the UI object to the Business Model is accomplished by script rules that are used to map the data between the two.

In the browser data captured in the UI is constructed in the Transient Data Object. It is then passed to a script rule. The script rule then updates the Business Domain Model. When the UI is calling for data, the process can be reversed.

A representation is illustrated below:



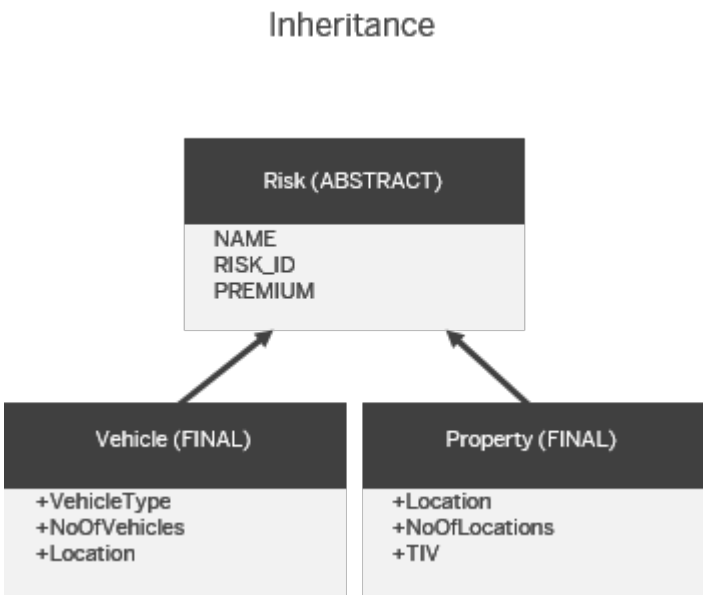
Related Information

Business Model Support for Composition or Inheritance [page 163]
Example: Data Definition Object Creation [page 165]
Example of a Packaged Product [page 166]

6.2.1 Business Model Support for Composition or Inheritance

Both composition and inheritance are supported for constructing your business model.

An example or inheritance is shown in the following diagram:



Business Model Definition

The business model definition is design-driven. In the Product Studio, the data definition object is used to create persistent and transient objects.

To use the data object layer feature, you have to manually create/modify your data model design. You need to set up data objects with inheritance structure, with `ABSTRACT` or `FINAL` values in the `TABLE_TYPE` attribute as discussed below.

Table Component Attribute Amendments

Amendments have been made to the `Table` component.

The TABLE_TYPE attribute is now used to define whether it is an ABSTRACT, FINAL or TRANSIENT data object.

- ABSTRACT persistent object can have multiple levels of inheritance.
- ABSTRACT persistent object can be inherited by multiple ABSTRACT or FINAL persistent objects.
- ABSTRACT or FINAL persistent object can only inherit ABSTRACT persistent object (i.e. ABSTRACT can't inherit FINAL; FINAL can't inherit FINAL).

Table Component Attribute Rules

The following rules apply to the Table attributes:

- The Table component's TABLE_TYPE attribute is used for you to specify if it is an ABSTRACT, FINAL or TRANSIENT data object.
- If a table name is ABSTRACT or FINAL, this indicates that it is a persistent object that is used when you want to create physical database and store data in it.
- If a table name is TRANSIENT, this indicates that it is a Transient object that can be used to display data in a structure different than where they are stored. The data stored in transient objects are temporary. No physical database table is created.
- Script rules or custom JAVA code can access either transient objects or persistent objects.
- The DESCRIPTION attribute is mandatory and you need to enter a value for it. Use camel case (for example, SampleDesc) for the value. Don't use spaces or special characters.
- On saving a data value row, the system validates if DESCRIPTION is blank.

The following takes place if an ABSTRACT data object's TABLE_NAME is specified:

- For normalized tables, the system persists the columns defined in the ABSTRACT data object and creates a table with the specified name and columns in the database.
- System creates inheriting FINAL tables in the database with its local columns.
- When a record is created for one of its inheriting FINAL tables, a record is also created in the ABSTRACT table, with reference to the FINAL table's record. No record is created in any other inheriting tables.
- The ABSTRACT and FINAL tables will have name as specified as in the TABLE_NAME attribute of the Table component.

Rules When ABSTRACT Data Object's TABLE_NAME Isn't Specified

The following takes place if an ABSTRACT data object's TABLE_NAME isn't specified:

For denormalized tables, the system creates FINAL data objects in the database with all locally defined columns, and columns defined in their inheriting ABSTRACT data objects.

Column Component Attribute Amendments

Amendments have been made to the Column component.

The following rules apply to attributes of the `Column` component:

- The `DESCRIPTION` attribute is mandatory and you need to enter a value for it. Use camel case (for example, SampleDesc) for the value. Don't use spaces or special characters.
- On saving a data value row, the system validates if `DESCRIPTION` is blank.

Data Object Layer Best Practice

If you are using data object layer, the following best practices are recommended for creating a business model definition.

- Define the inheritance of data objects (`FINAL` -> `ABSTRACT`) at the standalone level. This guarantees that no data object data value rows are overridden, and no attribute changes occur at product level. This is to ensure that each metadata is the same throughout the system and isn't modified per product.
- Assemble data objects into data models at standalone level. This ensures the same data model structure can be used across the same line of business.
- Assemble `ABSTRACT` data objects with the same structure as the intended `FINAL` data model, and refer to the assembly at template level (that is, not locally in a coverage group or product).
  - Having script rules at the template level using only `ABSTRACT` data ensures that the rules work across all coverage groups or products inheriting that template.

    > **i Note**
    >
    > Don't publish the `ABSTRACT` assembly to Runtime as it won't work.

  - Setting minimum and maximum cardinality for data objects is mandatory for Data objects.
- Assemble data model with `FINAL` data objects and refer to/assemble it in the actual business coverage groups or products. This ensures `FINAL` data objects with metadata specifically for the coverage groups or products can be used locally.

6.2.2 Example: Data Definition Object Creation

This topic provides an example to create an `ABSTRACT` data object and then create a `FINAL` data object that inherits from the created `ABSTRACT` data object.

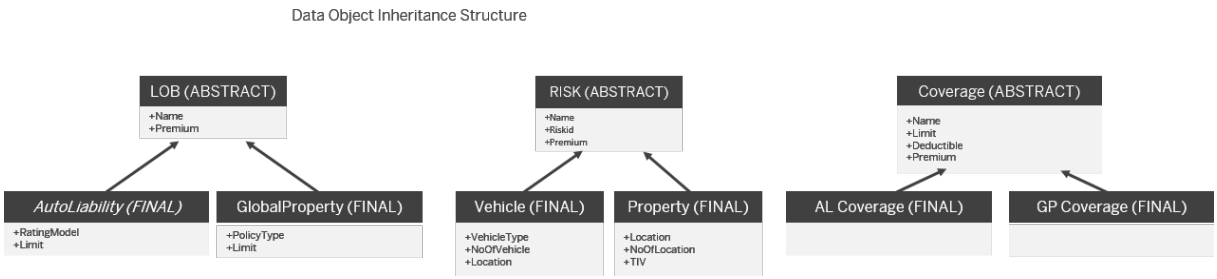
Procedure

1. In the Product Studio, create the Data Definition object in the appropriate folder.
2. In the `Table` sub-component, create data value row with **`TABLE_TYPE = ABSTRACT`** to make it an abstract data object.
3. In the `Column` sub-component, create any attributes needed for your abstract data object.
4. Create another Data Definition object by inheriting the Data Definition created above.

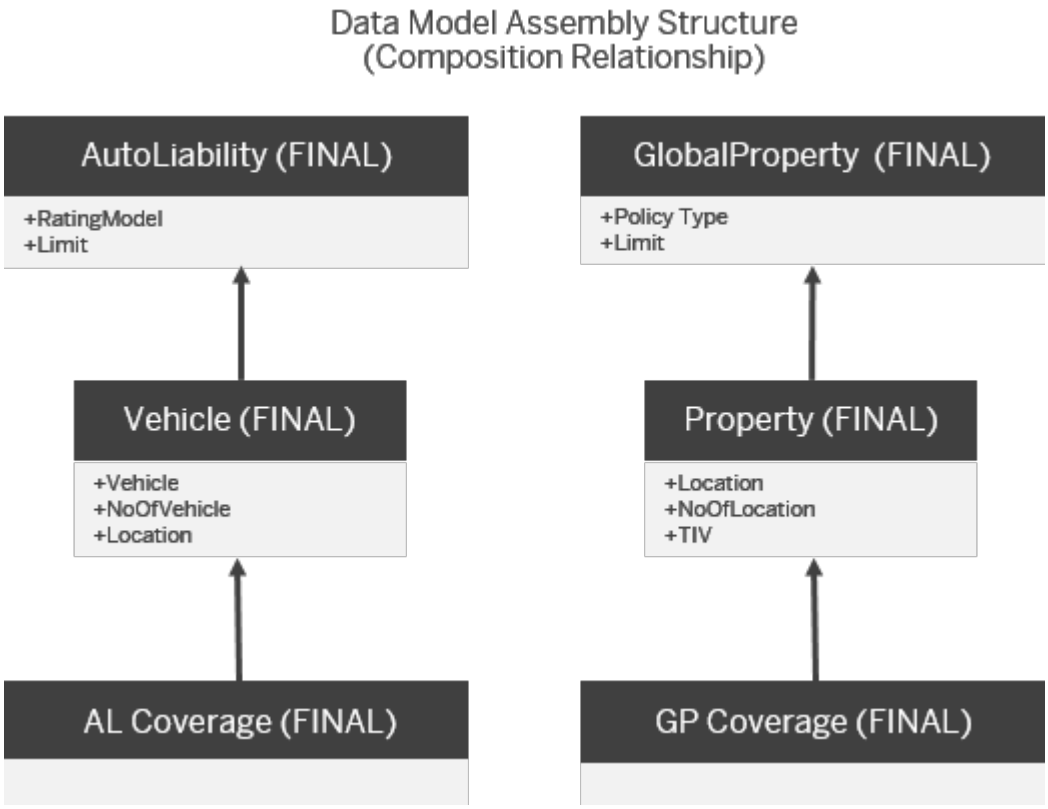
5. In the inheriting Data Definition, override the Table data value row (both values and association) from its parent and give the overridden row a new *Table Name* and *Description*.
6. Set the overridden data value row's **`TABLE_TYPE = FINAL`** (or blank) to make it a `FINAL` data object.
7. In the `Column` sub-component, create any attributes needed for your final data object:
   - If the `TABLE_TYPE` attribute is left blank, the persistent object is defaulted to be `FINAL`.
   - System accepts the `TABLE_TYPE` attribute value as non case-sensitive. For example, it can be **`ABSTRACT`** or **`abstract`** or **`Abstract`** or **`aBsTrAcT`**.
   - You are able to optionally specify the `TABLE_TYPE` attribute of an `ABSTRACT` persistent object (Affects the creation of physical database tables).

6.2.3 Example of a Packaged Product

The following diagram illustrates the Data object inheritance structure:



The following diagram illustrates the Data model assembly structure with composition relationship:



6.3 Using Custom Code in a Business Model Definition

Transient objects can be instantiated using custom code, such as a custom class created by you. The custom objects must be defined as transient objects in order to access through rules.

Example: Entitlements must be modeled as data objects in order to be accessed through rules.

Transient object definition will support composition and inheritance. By using custom code, you are able to manipulate data objects in the current composition model.

6.4 Publishing a Business Model Definition

When you publish, the system performs checks.

Validations and Warnings

The following list highlights the validation checks performed and the warning message displayed when using a data layer object.

- Validations occur when you select the *Metadata* option under the *Build* menu in the *Product Painter*.
- In the *Product Painter*, for each data definition assembled in the *Product Tree*, defined as data object, the system checks if the following situation occurs:
  - There is any invalid data object inheritance structure.
  - An invalid value (that is, a value other than `FINAL`, `ABSTRACT`, `TRANSIENT`, or blank) is entered in the `TABLE_TYPE` column of the data definition's table component an error message is displayed: `Invalid TABLE_TYPE value is found in current or parent data definition table. Only 'ABSTRACT', 'FINAL', 'TRANSIENT' or blank value is allowed. Please try again.`
  - During build metadata if an invalid data object hierarchy is found (for example, a `FINAL` or blank `TABLE_TYPE` value is found in a parent data object), a warning message is displayed in the build metadata dialog: `"FINAL" or blank TABLE_TYPE found in parent data definition object(s). If you are defining hierarchical Data Objects, please make sure all parent data objects have "ABSTRACT" as TABLE_TYPE, and then rebuild metadata.`
- Once build metadata is successful (and other criteria for successful publishing met), the product JAR is generated with XMLs representing the data object model (one XML for each `FINAL` Data Object, with `ABSTRACT` data object embedded in each), for the following actions:
  - Downloading a product JAR
  - Publishing a single product
  - Publishing a product family
- Publish below

Physical Data Model

Physical table mapping for all the final data objects and physical column mapping for all the attributes must be provided in the data definition objects in the Product Modeler. This will be validated during `Product Publish` and `Build Data Definition` actions in the Product Modeler.

Inheritance

The following two physical data model mapping alternatives will be supported to persist the attributes defined in abstract data objects.

- Normalized tables for abstract data objects

- Denormalized tables for final objects

Normalized Tables for Abstract Data Objects

If a physical table mapping is specified in the `Table` component of the abstract data definition, the system will persist those attributes in the specified table. Additional attributes defined in the inherited objects will be persisted in child (one-to-one) tables.

> **i Note**
>
> When multiple final data objects are inherited from an abstract data object and one final object is instantiated in Runtime, it won't create empty records in the other final object's tables.

Below is an example:

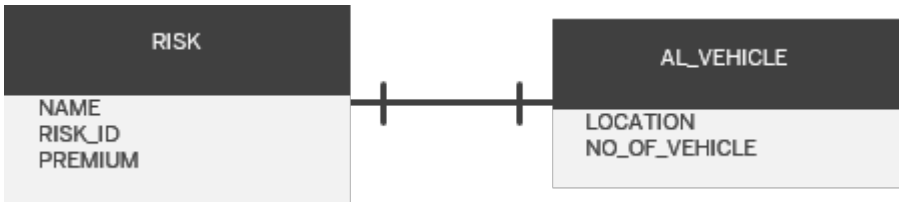
Abstract Data Definition: Risk

Table Mapping: **RISK**; Attributes: Name, RiskId, Premium

Final Data Definition: Vehicle

Inherits from: Risk, Table Mapping: AL_VEHICLE; Attributes: Location, NoOfVehicles

A diagram of the physical model is illustrated below:



Denormalized Tables for Final Objects

If the physical table mapping isn't specified in the `Table` component of the abstract data definition, the system will persist all the attributes of the final data objects in their own tables.

Below is an example:

Abstract Data Definition: Risk

Table Mapping: **None**; Attributes: Name, RiskId, Premium

Final Data Definition: Vehicle

Inherits from: Risk, Table Mapping: AL_VEHICLE;**Attributes: Location, NoOfVehicles**

Composition

The physical data model of a composition object is always persisted in a child table.

Child Data Definition:

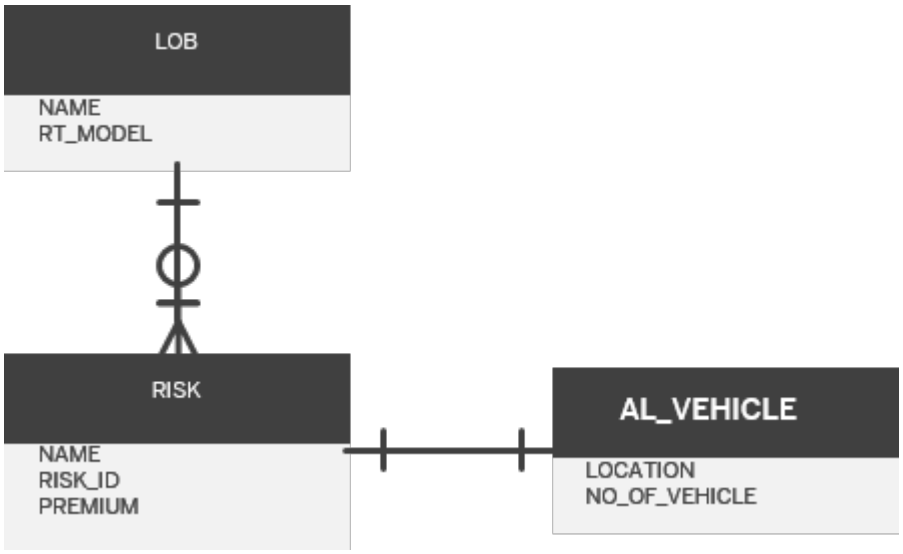
Abstract Data Definition: Risk

Table Mapping: **RISK**; Attributes: Name, RiskId, Premium

Final Data Definition: Vehicle

Inherits from: Risk, Table Mapping: AL_VEHICLE;**Attributes: Location, NoOfVehicles**

A diagram of the physical model is illustrated below:



Business Model Definition Synchronize Flowstore

The mapping solution will facilitate automated translation between the business abstraction and the physical data model based on the mapping provided by you. No provision will be made to define or influence the mapping strategy. Inheritance is supported.

The solution provides schema generation and management for areas of the model persisted within FS-PRO.

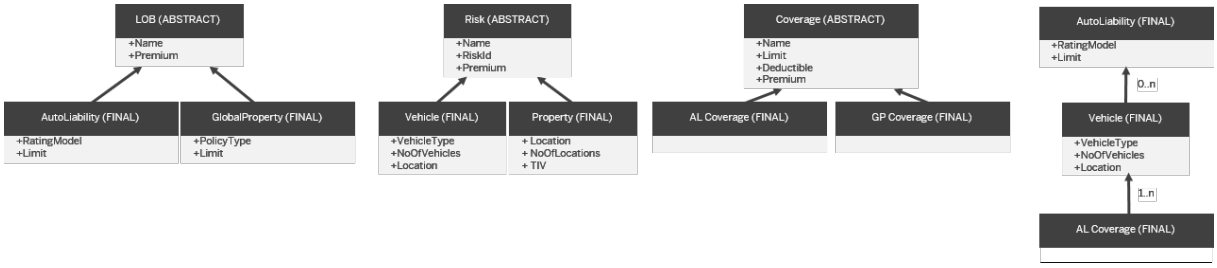
Using the *Product* > *Synchronize Product Flowstore* feature in the Design Time Administrative Console, the system identifies if a data object is defined as `ABSTRACT` or `FINAL`, and creates scripts to allow you to create tables in the database.

- The table name will be as defined in the `TABLE_NAME` attribute of the data definition persistent object.
- The system will still create a database table for any `FINAL` data object that has no local columns defined (i.e. it only has inherited columns from `ABSTRACT` parent).

6.5 Accessing a Business Model in Script Rules

Product services, represented by FS-PRO script rules as well as custom code, are able to refer to a business abstraction model.

The following diagram illustrates an example of a setup diagram.



As illustrated in the setup diagram above, with script rules you are able to access current and child tables using ABSTRACT and FINAL names.

```
n TYPE NUMBER
FOR EACH Risk IN LOB
n = n + Risk.Premium
END FOR

n TYPE NUMBER
FOR EACH Vehicle IN AutoLiability
n = n + Vehicle.Premium
END FOR
```

Related Information

Manipulating Transient Data Objects With Script Rules [page 171]
Changes in Script Rule [page 172]

6.5.1 Manipulating Transient Data Objects With Script Rules

Script rules allows for creating and manipulating transient data objects.

In script rule, you are able to convert and display a data object to a structure different than the current Business Model. For example, `<coverage>` is the parent of `<risk>`. In script rule, you are able to translate `<risk>` as the parent of `<coverage>`, using transient objects.

Keep the following in mind:

- No matter if it's normalized or denormalized, you are able to access the ABSTRACT data object in script rule.
- You are able to work with multiple input arguments in their script rules.
- If multiple XML/Database Table input arguments are used, you must specify the argument name when looping through object tables.
- Assignment remains functional when an input argument name is included. For example: `contract:Vehicle.Premium = 1000.0`

Syntax: `FIND`

- It's used for Database Table types.

- To use the `FIND` syntax, you must specify the input argument name. Otherwise rule building will fail.
- The `FIND` syntax can be used to find data in a specific child object at any level (that is to say, child, grandchild, great-grandchild ...).
- You can also call custom classes or execute rules within the `FIND` clause.

Syntax: `FIND ROW IN <DATA_OBJECT>:<TABLE> FOR <PRIMARY_KEY_VALUE>`, where the value of `<PRIMARY_KEY_VALUE>` should be the PK_ID of the table in the dataobject in which you are interested.

Example:

```
FIND ROW IN contract:LOB FOR key
    vLOBPrem = contract:LOB.Premium
    contract:LOB.Premium =1000.0
    dObj = CALL CustomMethod IN LIBRARY CustomClass WITH contract:LOB
    EXECUTE RULE rule WITH contract:LOB FROM Rating WHERE PCD=3434534
END FIND
```

Syntax: `DELETE`

- It's used for Database Table and XML types.
- To use the `DELETE` syntax, you must specify the input argument name. Otherwise rule building will fail.
- The `DELETE` syntax can be used can be used to delete an object in the data objects hierarchy.

This is illustrated below:

```
FOR EACH ROW IN contract:LOB
    IF contract:LOB.Name = "test"
    DELETE contract:LOB
    END IF
END FOR
```

Syntax: `WITH`

- It's used for Database Table and XML types.
- To use the `WITH` syntax, you must specify the input argument name.
- The `WITH` syntax can be used can be used to create a new object.
- You can also call custom classes or execute rules within the `WITH` clause

An example is shown below:

```
WITH NEW ROW IN contract:Vehicle
    contract:Vehicle.VehicleType ="Car"
    contract:Vehicle.Premium = 1000.0
END WITH
```

6.5.2 Changes in Script Rule

Supported syntax is shown in the following table:

Type Syntax	Database Table (such as URS, Data Object)	XML	Object Table (DataTable, used for XPath queries)
`APPEND`	No	Yes	No
`WITH`	Yes	Yes	No
`FIND`	Yes	Yes	No
`DELETE`	Yes	Yes	No
`FOR EACH`	Yes	Yes	No
`LOOP`	Yes	Yes	No

6.6 Retrieving Reference Data

Reference data (read only) may be persisted external to FS-PRO. Rules definitions have access to externally persisted data in a seamless fashion.

A new system object, `Reference Data` component, is created out-of-the-box under the following path in the Product Studio: *System Objects Repository* > *System Group* > *Base Data Object*.

Reference Data Component Rules

The following rules apply to the `Reference Data` component:

- Attribute: `Path` [TEXT, 255] - The path to the desired component, in xPath. This attribute is needed in order to handle the case when the same component appears more than once in the product
- Attribute: `Component` [TEXT, 50, Required] - The component name that contains the desired data
- Attribute: `Custom Class` [TEXT, 255, Required] - The custom class to look for the desired data instead.

Custom classes have to be uploaded to the Administrative Console's File System's `Custom Class` Folder.

The `Reference Data` component should be assembled in the product using it.

Using the Reference Data Component

In a script rule, you are able to query data in a component in Product Modeler. However, sometimes the desired data may be located externally.

- The `Reference Data` component allows you to specify custom classes to retrieve external data.

- If a selected data value row in the `Reference Data` component of a product is defined with the component name to look for, and a custom class that contains a query, the script rule (when executed by the system) will look for the desired data from the custom class' query instead of from the component.
- If the data value row isn't selected or doesn't exist, the system will look for the desired data from the specified component.

7 Questionnaire Models

Questionnaire Models serve as the starting point for a multilayered questionnaire that you use in eApp screens.

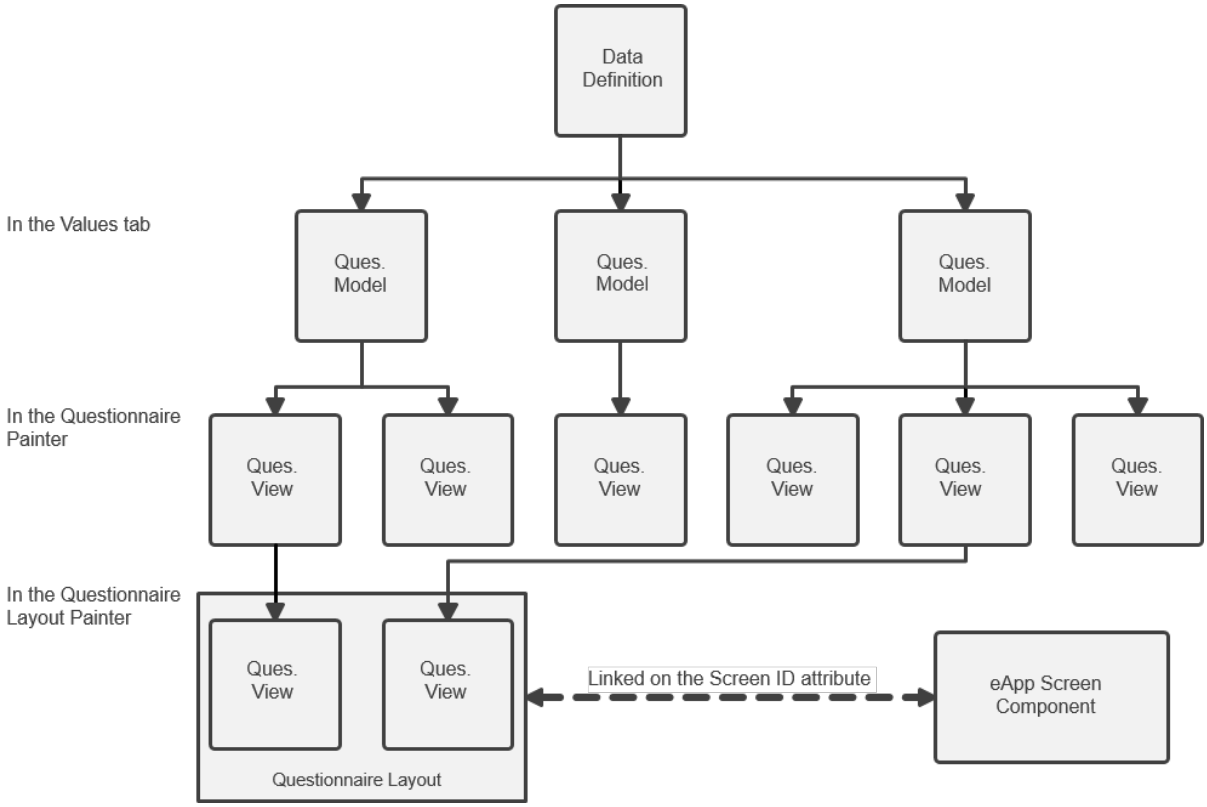
The three layers enforce separation of data, screen objects, formatting, and the final presentation in an actual eApp screen. And because you can add multiple versions at each level, from a single data definition you can define multiple questionnaires, with multiple presentations, for use in any number of eApp screens.

Each layer in the questionnaire has its own function, as follows:

`Questionnaire Model`	A type of component that you create in the Product Studio and add to your product. You then add questions and screen controls to the Questionnaire Model, defining their properties (such as rules and data masks) but not their formatting (which is accomplished in the Questionnaire View). You can link any number of Questionnaire Model components to a data definition.
`Questionnaire View`	You derive a Questionnaire View directly from the Questionnaire Model, selecting questions and controls from the Questionnaire Model as required. The placement of questions and controls on a screen is defined by the Questionnaire View, as is the formatting of the elements. Each Questionnaire View can be created from one or more Questionnaire Models.
`Questionnaire Layout`	A questionnaire layout is where you assemble the various Questionnaire Views, as you would like them to appear in an eApp screen. The questionnaire layout must use Questionnaire Models and Questionnaire Views from the same data definition, including any extensions.

> **i Note**
>
> Questionnaires aren't version managed. Therefore they don't use *Effective Date* and *Expiry Date* fields.

The following diagram shows the relationship between the data definition, the questionnaire layers, and the eApp screen:



You develop a questionnaire through the various levels as follows:

	Elements (Questions and Screen Controls)	Rules/Event Handlers
Model Layer	Create: • Questions • Parent/child questions • Triggers • Screen controls, including data grids	Create: • Filter rules for data grids • Product queries
View Layer	Add questions and screen controls from model and arrange on screen	Specify event handlers and actions for any element.
Layout	Add views and arrange in layout	Create rules for views

Notes on Questionnaire Models

When working with inherited Questionnaire Models and Questionnaire Views, you can add new elements, but you can't change the inherited questions and controls, you can only delete them.

The `Questionnaire Model` type is a configurable object type that derives from the standard questionnaire object, `Questionnaire Base` (as such, the `Questionnaire Model` type is categorized as a system object).

In Questionnaire Models, script rule names don't need to be unique.

Questionnaire Models and Multi-language Support

When entering values in a Questionnaire Model for a primary language, the following rules apply:

- The *Language* dropdown list isn't available in the *Values* tab.
- For object properties supporting Unicode, the fields display in the primary language selected for the current product, or blank if values haven't been entered.
- When selecting another primary language from the *Properties* dialog, those Questionnaire Model objects property fields are refreshed to display in the selected language (if available).
  - Initially, property fields are blank for user input for that property in the selected primary language.
  - For property fields with primary language values previously entered, the previously-entered values in the selected primary language will be displayed.
- All changes are recorded in the `Audit Trail`.

> **i Note**
>
> To change the language in a Questionnaire Model, go to the *Properties* dialog of the current object containing the Questionnaire Model.

Related Information

Creating Questionnaire Models [page 178]
Defining Screen Controls [page 180]
Defining Query List Columns [page 200]
Defining Query Lists [page 200]
Defining a Data Grid [page 201]
Working with Questionnaire Views [page 213]
Working with Questionnaire Layouts [page 229]
Supported Rich Text Formatting [page 232]

7.1 Creating Questionnaire Models

You create questionnaire models in the Product Studio, in your product's questionnaire folder . If your product doesn't have a questionnaire folder, you must first create it.

Context

> → Tip
>
> You can define multiple questionnaire models for a given data definition.

Procedure

1. Navigate in the *Studio Tree* to the product's questionnaire folder .
2. Choose the *New Object* icon in the *Object List* tab.

   The *New Object* dialog appears.
3. Enter a name for the model in the *Name* field. Keep in mind that if you want to create multiple models for the same data definition you may want to devise a naming scheme.
4. Select *Questionnaire Model* from the *Modifier Type* dropdown list.
5. Choose *OK*.

   The new object appears in the questionnaire folder.

   > i Note
   >
   > If you open the questionnaire model in standalone mode, only the *Manual* tab is available. To link the model to a data definition and add questions, you must add the model to a product.
6. Open your product and add your questionnaire model to the *Product Tree*.
7. After you add the questionnaire model, you define it by adding questions and controls. You do this using drag-and-drop style controls. Each question is automatically linked to a data definition column that you select. To add a question to a model:
   a. Navigate in the product to the questionnaire model and select it.
   b. Select the *Values* tab.
      The model appears as a tree structure.
   c. Choose the *Choose Column* icon in the toolbar.
      The *Master Table Column Chooser* appears.
   d. Navigate the table structure in the dialog's tree (if the table has extension tables, they appear in the structure as well). Select the column and drag it to the questionnaire container.

In the *Values* tab, you can use the *Move Up* icon and the *Move Down* icon from the toolbar to position questions and controls in the tree.

e. Save your changes.

When you are ready you can define the details of the question.

Besides questions, a questionnaire can contain many other elements, such as text, buttons and data grids. All these elements are represented by screen controls located in the Toolbox palette. The following table describes all the screen controls available in a questionnaire model.

Icon	Name	Description
	Questionnaire Container	Groups questions and screen controls
	Data Grid	Adds a detail list section that can be edited in-line
	Text	Displays text that you define
	Button	Adds a programmable button control

8. To add a screen control to a model, choose the *Toolbox* icon in the toolbar to show the palette, then select and drag the control to the appropriate place in the model tree.
9. Because a questionnaire model is a tree structure, you can place objects at different (horizontal) levels within the tree, so that some objects appear as children of other objects. To change an object's level, select the object and in the toolbar choose the *Increase Indent* icon or the *Decrease Indent* icon.

   > → Remember
   >
   > Only questionnaire containers can have child objects.

   To make a question into a child question, you indent it beneath another question, which then automatically becomes a parent question. At Runtime, the child question is hidden, and is only revealed when the user enters the correct trigger value. You define this value in the child question's `Trigger` property.
10. In a questionnaire model, you can delete any object below the root node of the tree. If the object has children, they are deleted as well. To delete an object from a questionnaire model, select the object to delete and choose the *Delete Record* icon and confirm your decision. The object is removed from the tree.
11. To search for an object in a questionnaire model, using any of the properties visible in the model tree, including *Column Name*, *Question Text*, *Control Type*, and *Label*. Select the *Search Data* icon ; the *Find* dialog appears. Enter your search string, select search options, and choose *Find Next*. If the search succeeds, the matched string is highlighted in the model tree.
12. You can show or hide table, column, and data type information for questions, and the type for screen controls. Select any node in the model tree and choose the *Toggle Column Display* icon .

13. You can use the *Page Size* field in the toolbar to control how many first-level nodes in the tree appear per page.

    For example, if your model contains seven first-level nodes and you set the value in *Page Size* to 3 and press the `Tab` key, only the first three nodes appear. The remainder move to pages 2 and 3.

Related Information

Defining Screen Controls [page 180]
Adding Elements to Questionnaire Views [page 226]

7.2 Defining Screen Controls

Individual questions have a variety of properties that you can define. These properties depend on the question's data type, control type, and use in the questionnaire.

You specify the details of screen controls using the *Properties* dialog. Each type of screen control has its own unique properties.

Related Information

Defining a Text Box Screen Control [page 181]
Defining a Text Area Screen Control [page 183]
Defining a Static Text Screen Control [page 185]
Defining a Number Screen Control [page 186]
Defining a Date Screen Control [page 189]
Defining a Check Box Screen Control [page 191]
Defining a Data List Screen Control [page 192]
Defining a Yes/No Radio Buttons Screen Control [page 195]
Adding Labels to the UI [page 197]
Adding Buttons to the UI [page 197]
Creating Questionnaire Models [page 178]

7.2.1 Defining a *Text Box* Screen Control

In the Product Modeler, a *Text Box* is a basic control that allows for entry/update with any character value.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the question.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Text Box* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.

4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Encrypted	Select if you want input to the field to be masked, for example, as for a password. The value will be replaced with asterisks in the display view.
Read Only	Select to make the field non-editable. Content display rules can be configured to override this setting and show this attribute in Runtime.
Hidden	Select to make the question and its field invisible at Runtime. Content display rules can be configured to override this setting and show this attribute in Runtime.
Show Notes	This attribute is not supported.
Clear Value on Hide	Clears the input if the user chooses to hide the question. You can hide it via the *Hidden* property, the *Trigger* property or a content display rule.
Column Name	Shows the table and column name and isn't editable.

5. Complete the properties in the *Advanced* tab.

   The properties that appear are dependent on the question's data type.

Default Value	The values set in the property will be applied when the screen is rendered and the attribute is blank.

Length	Defines the maximum number of characters allowed to be entered in the screen control. The value set here will only be applied provided that it is less than the value specified in the data model.
Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Parent Trigger	Used in a parent question to define the values will trigger the hidden child questions to appear. To use multiple answers, separate them with commas (,).
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported. If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

Related Information

Supported Rich Text Formatting [page 232]

7.2.2 Defining a *Text Area* Screen Control

In the Product Modeler, a *Text Area* is a scrollable text box. The control provides properties that enable you to customize the number of rows that appear on screen, the data length, etc.

Context

> **i Note**
>
> There is an additional parameter in the view that can be configured to specify the number of rows displayed in the Text Area memo field. Use the `Height` attribute to set this value.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the question.

   The *Properties* dialog appears.

   > **→ Tip**
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Text Area* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Read Only	Select to make the field non-editable. Content display rules can be configured to override this setting and show this attribute in Runtime.
Hidden	Select to make the question and its field invisible at Runtime.
Show Notes	This attribute is not supported.
Clear Value on Hide	Clears the input if the user chooses to hide the question. You can hide it via the *Hidden* property, the *Trigger* property or a content display rule.
Column Name	Shows the table and column name and isn't editable.

5. Complete the properties in the *Advanced* tab.

   The properties that appear are dependent on the question's data type.

Default Value	The values set in the property will be applied when the screen is rendered and the attribute is blank.
Length	Defines the maximum number of characters allowed to be entered in the screen control. The value set here will only be applied provided that it is less than the value specified in the data model.
Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Parent Trigger	Used in a parent question to define the values will trigger the hidden child questions to appear. To use multiple answers, separate them with commas (,).
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported. If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

Related Information

Supported Rich Text Formatting [page 232]

7.2.3 Defining a *Static Text* Screen Control

In the Product Modeler, a *Static Text* screen control is a basic control that displays attribute values only. There is no ability to enable the field on screen and update.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the question.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Static Text* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Show Notes	This attribute is not supported.

5. Complete the properties in the *Advanced* tab.

   The properties that appear are dependent on the question's data type.

Default Value	The values set in the property will be applied when the screen is rendered and the attribute is blank.
Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Parent Trigger	Used in a parent question to define the values will trigger the hidden child questions to appear. To use multiple answers, separate them with commas (,).
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported. If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed.

	When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

Related Information

Supported Rich Text Formatting [page 232]

7.2.4 Defining a *Number* Screen Control

In the Product Modeler, a *Number* screen control is a basic control that allows for entry/update with any numeric value.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the question.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the  *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Number* from the *Control Type* dropdown list.

The properties that will be displayed are determined by the type of screen control chosen.

4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Read Only	Select to make the field non-editable. Content display rules can be configured to override this setting and show this attribute in Runtime.
Hidden	Select to make the question and its field invisible at Runtime.
Clear Value on Hide	Clears the input if the user chooses to hide the question. You can hide it via the *Hidden* property, the *Trigger* property or a content display rule.
Currency Format	This attribute is not supported.
Column Name	Shows the table and column name and isn't editable.
Min value	For defining the screen control's minimum acceptable value (inclusive). If validation fails during Runtime, the field will be outlined in red, an error message will be displayed below the control and an error will be included in the validation summary.
Max value	For defining the screen control's maximum acceptable value (inclusive). If validation fails during Runtime, the field will be outlined in red, an error message will be displayed below the control and an error will be included in the validation summary.

5. Complete the properties in the *Advanced* tab.

The properties that appear are dependent on the question's data type.

Default Value	The values set in the property will be applied when the screen is rendered and the attribute is blank.
Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Parent Trigger	Used in a parent question to define the values will trigger the hidden child questions to appear. To use multiple answers, separate them with commas (,).
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported. If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed.

	When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Dimensions	This attribute is not supported.
Length	Defines the maximum number of characters allowed to be entered in the screen control. The value set here will only be applied provided that it is less than the value specified in the data model.
Precision	The number of decimal places for a numeric value. The value set here will only be applied provided that it is less than the value specified in the data model. The value for this property cannot equal or exceed the *Length* property.
Allow Negative	Select to allow negative values in a number field.
Display % sign	Displays the number, as entered, and appends a percent sign.
Currency	This attribute is not supported.
Group Enabled	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

Related Information

Supported Rich Text Formatting [page 232]

7.2.5 Defining a *Date* Screen Control

In the Product Modeler, a *Date* screen control is a basic control that allows for entry/update with any date value.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the question.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Date* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Date Format	This attribute is not supported.
Style	The style of the date rendered in the UI is governed by this parameter. If set to short, med, long or full, then the date format will be displayed based on the globalization style. **User Default** Derived based on your globalization settings. **Short Date** For example: 12/31/2015 **Medium Date** For example: Dec 31, 2018 **Long Date** For example: December 31st, 2018 **Full Date** For example: Monday December 31st, 2018
Read Only	Select to make the field non-editable.

	Content display rules can be configured to override this setting and show this attribute in Runtime.
Hidden	Select to make the question and its field invisible at Runtime. Content display rules can be configured to override this setting and show this attribute in Runtime.
Show Notes	This attribute is not supported.
Clear Value on Hide	Clears the input if the user chooses to hide the question. You can hide it via the *Hidden* property, the *Trigger* property or a content display rule.
Column Name	Shows the table and column name and isn't editable.
Min value	For defining the screen control's minimum acceptable value (inclusive). If validation fails during Runtime, the field will be outlined in red, an error message will be displayed below the control and an error will be included in the validation summary.

5. Complete the properties in the *Advanced* tab.

   The properties that appear are dependent on the question's data type.

Default Value	The values set in the property will be applied when the screen is rendered and the attribute is blank.
Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Parent Trigger	Used in a parent question to define the values will trigger the hidden child questions to appear. To use multiple answers, separate them with commas (,).
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported. If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.

Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

Related Information

Supported Rich Text Formatting [page 232]

7.2.6 Defining a *Check Box* Screen Control

In the Product Modeler, a *Check Box* screen control is a boolean control.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the question.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Check Box* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Read Only	Select to make the field non-editable. Content display rules can be configured to override this setting and show this attribute in Runtime.
Hidden	Select to make the question and its field invisible at Runtime.
Show Notes	This attribute is not supported.

Clear Value on Hide	For child questions, select to undo input if the user chooses to hide the question.
Column Name	Shows the table and column name and isn't editable.

5. Complete the properties in the *Advanced* tab.

   The properties that appear are dependent on the question's data type.

Default Value	The values set in the property will be applied when the screen is rendered and the attribute is blank. If you set a default value of either **`Y`**, **`yes`**, or **`true`**, Product Modeler will convert this value to **`1`**.
Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Parent Trigger	Used in a parent question to define the values will trigger the hidden child questions to appear. To use multiple answers, separate them with commas (,).
Mandatory Flag	This parameter doesn't apply to checkboxes.
Mandatory Flag Rule	This parameter doesn't apply to checkboxes.
Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

Related Information

Supported Rich Text Formatting [page 232]

7.2.7 Defining a *Data List* Screen Control

In the Product Modeler, a *Data List* screen control is a control that executes a query to displays results as a dropdown list or as radio button options.

Context

The display of the data list is governed by the *Control Type* property configured within the `View` properties. It is in the `View` properties where you will specify whether this control is displayed as a dropdown list or radio button options.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the question.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Data List* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.

   > i Note
   >
   > The determination of the control type (dropdown list or radio buttons) is assigned in the *Properties* dialog of the *Questionnaire Painter*.

4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Read Only	Select to make the field non-editable. Content display rules can be configured to override this setting and show this attribute in Runtime.
Hidden	Select to make the question and its field invisible at Runtime. Content display rules can be configured to override this setting and show this attribute in Runtime.
Show Notes	This attribute is not supported.
Clear Value on Hide	Clears the input if the user chooses to hide the question. You can hide it via the *Hidden* property, the *Trigger* property or a content display rule.
Column Name	Shows the table and column name and isn't editable.

5. Complete the properties in the *Data Source* tab, which defines the content of the data list.

Local List	Enter a set of values to populate the data list. The local list will only be used if no *Product Query* is defined. Each record needs to include both the code value and its corresponding description (that will be displayed at Runtime). Separate each record by semi-colons. The size of the list is limited to 250 characters.
Code List	This attribute is not supported.

Product Query	Use to create a product query rule that returns a data table. Select the icon and select *New Rule*. To create the rule using the wizard feature, choose the icon and select *Rule Wizard*. **→ Remember** When creating a product query for a questionnaire model to populate a dropdown list, you must provide both the key column and the description column.
Query Rule	This attribute is not supported.
Empty Option	For a dropdown list, select to have the list of values start with an empty row. This option is not applicable for radio button lists.
Sort	When selected, the control records are sorted in ascending order.

i Note

When defining a *Data List* in a questionnaire model for a data definition column, the contents of the data list must match the data type of the data definition column. For example, if the data list is for a numeric column but the data source returns a character string, the input data won't be validated in the questionnaire and an error will be generated when a value with a different data type is selected at Runtime.

6. Complete the properties in the *Advanced* tab.

The properties that appear are dependent on the question's data type.

Default Value	The values set in the property will be applied when the screen is rendered and the attribute is blank.
Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Parent Trigger	Used in a parent question to define the values will trigger the hidden child questions to appear. To use multiple answers, separate them with commas (,).
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported. If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory.

	If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

7. In the *Values* tab, choose *Save*.

Related Information

Supported Rich Text Formatting [page 232]

7.2.8 Defining a *Yes/No Radio Buttons* Screen Control

In the Product Modeler, a *Yes/No Radio Buttons* screen control is a boolean control switch, that displays *Yes* and *No* options.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the question.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the  *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Yes/No Radio Buttons* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field.

	Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Read Only	Select to make the field non-editable.
Hidden	Select to make the question and its field invisible at Runtime.
Show Notes	This attribute is not supported.
Clear Value on Hide	For child questions, select to undo input if the user chooses to hide the question.
Column Name	Shows the table and column name and isn't editable.

5. Complete the properties in the *Advanced* tab.

   The properties that appear are dependent on the question's data type.

Default Value	The values set in the property will be applied when the screen is rendered and the attribute is blank. If you set a default value of either **`Y`**, **`yes`**, or **`true`**, Product Modeler will convert this value to **`1`**.
Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Parent Trigger	Used in a parent question to define the values will trigger the hidden child questions to appear. To use multiple answers, separate them with commas (,).
Mandatory Flag	This parameter doesn't apply to the Boolean control switch.
Mandatory Flag Rule	This parameter doesn't apply to the Boolean control switch.
Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

Related Information

Supported Rich Text Formatting [page 232]

7.2.9 Adding Labels to the UI

In the Product Modeler, additional text or labels can be configured for a view.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Open the *Toolbox* control and select the text icon and drag it into the questionnaire model
3. Select the label.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

4. Complete the properties in the *General* tab.

Label	The actual text as you want it to appear to the end user.

5. Complete the properties in the *Advanced* tab.

Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

7.2.10 Adding Buttons to the UI

A button can be configured and added to a view to trigger a particular action. For example to save, refresh or rate.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Open the *Toolbox* control and select the button icon and drag it into the questionnaire model.

3. Select the button.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

4. Complete the properties in the *General* tab.

Label	The actual text as you want it to appear to the end user.

5. Complete the properties in the *Advanced* tab.

Trigger	An attribute will be hidden in the screen view until such time that the value specified in this property corresponds to the Runtime value on the parent attribute (*Parent Trigger*). To use multiple answers, separate them with commas (,).
Dimensions	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

Next Steps

Next, you need to assemble the button into the view and configure the button's event handler.

Related Information

Adding Custom Buttons to the Footer Bar in an eApp [page 198]

7.2.10.1 Adding Custom Buttons to the Footer Bar in an eApp

You can add custom buttons to the footer bar in an eApp.

Custom buttons can be configured using the eApp Custom Buttons component, which is a child of the eApp Layout component. It can only be used for custom buttons, standard buttons such as *Save* or *Review* are implicit and cannot be controlled using the eApp Custom Buttons component. All buttons configured in this component are added to the footer bar of the eApp.

The eApp Custom Buttons component contains the following attributes:

Button ID	Used for design-time error messages.

	This attribute is required. It only supports ASCII characters.
Button Name	The name displayed on the button in the UI. This attribute is required. It supports multi-language entries.
Execution Rule	The rule that is executed upon selecting the button. Error messages generated by rule are displayed the same way regular validation rules are displayed. This attribute is required.
On Success JavaScript Class Name	The name of the JavaScript file that is executed upon successful rule execution. The JS file can be uploaded in the FS-QUO Administrative Console. This attribute is required.
Display Order	The order in which buttons will appear in the footer bar between *Save* and other static buttons (*Review*, for example). Sort by display order, represented with an integer, then by button name, represented with letters. Custom buttons will be displayed left to right, in ascending order. Values don't have to be sequential or unique. This attribute is required.
Button Display Rule	The rule that is executed upon loading the eApp Layout that determines if a button is visible or not. The rule return value is Boolean, so if the result is `true`, then the button is visible. Otherwise, it is not. If the *Visibility Rule* column is empty, the button is displayed. **i Note** Since the display rule is executed upon loading the eApp Layout, refreshing the screen won't run the rule; a change in the eApp Layouts is required.

The Service API `getEAppLayoutCustomButtons` is automatically called upon loading an eApp. It reads the eApp Custom Buttons to determine which buttons are displayed. It has the following parameters:

Rule Name	getEAppLayoutCustomButtons
Return Type	Data Object (result of the query to the eApp Custom Buttons component)
Input	eAppLayoutName

→ Tip

This rule will need to be overridden if the standard ProducerApplicationPQM30 eApp layout is not being used within the product.

7.3 Defining Query List Columns

You can define the properties for each column in a query list.

Procedure

1. In the query list, select the column you want to define.
2. Complete the properties in the *General* tab as follows:

Question Text	Specifies the actual question as you want it to appear to the end user. You can add the question directly to the text field.
Column Name	Enter a label for the column header.
Sort Column	Select this option to allow the user to sort the grid on this column.
Sort by Data Type	Select an item from the dropdown list to indicate the column's data type.

3. Save your changes.

7.4 Defining Query Lists

After you add a query list control to a model, you can define the control's general properties and add the data columns.

Procedure

1. Select the query list control and select the *General* tab.
2. Enter a label to identify the query list in the *Question Text* field.

   In the remaining steps you specify the type of source you want to use for the query list's data, create or identify the source, and choose the columns.
3. If you want to specify the type of source for the query list's data, select *Custom Class* from the *Source Type* dropdown list.

   You will identify an existing Java class that creates the query.
4. Enter the name of the Java class in the *Class Name* field.
5. Choose the  icon located beside *Column Chooser* to open the column chooser tool appropriate to the source type you chose.

   The tool provides access to the columns of the result set.
6. If you want to add a column, click and drag it to the query list control in the questionnaire model.

> **i Note**
>
> Hold down the `Ctrl` key while clicking columns to select multiple columns for dragging to the grid control.

If necessary, use the *Move Up* icon and the *Move Down* icon from the toolbar to position the columns in the grid.

7. Save your changes.

Related Information

Defining Query List Columns [page 200]

7.5 Defining a Data Grid

Use a data grid in a questionnaire when you need to capture repeating data (the type that typically is stored in a child table).

Context

For example, the insured could own several vehicles and want to insure them all under the same policy. You could use a data grid within a questionnaire to capture and display this data.

> **→ Tip**
>
> If required, you may also display a data grid with data from a peer table or an ancestor's child table on the same screen.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Open the *Toolbox* control and select the data grid icon and drag it into the questionnaire model
3. Select the data grid.

   The *Properties* dialog appears.

   > **→ Tip**
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

4. Enter a label for the data grid in the *Question Text* field. which will be the title for the data grid displayed in the view.
5. To sort the grid records based on the specified table attribute, specify the column that you want to sort by in the *Sort By* field.

   Enter the column the name (without the table name prefix).
6. Select *Has Child Screen* if you want to allow the user to open a child table in a window, using + *Add* or › *Navigation* icons.
7. Specify a value in the the *Max Rows* field to define the maximum number of rows that can be added to the data grid.
8. Select the *Auto Insert Record* option to have a new record created automatically for the user, if there are no existing records.
9. Select the *Advanced* tab.
10. Enter values in the *Trigger* field to define what answers to the parent question cause the data grid to appear.

    To use multiple answers, separate them with commas (,).
11. To identify a grid section, by default the system assigns the name of the child table to the grid section object. But if you are using more than one grid for a child table, you should provide unique identifiers for all grid sections beyond the first. The string you enter in the *Script Suffix* field is appended to the grid's default identifier to make the grid unique.
12. To add a rule to control what data appears in the grid, choose the icon beside *Filter Rule* and select *New Rule*.

    The calculation rule returns a boolean value (that is, true or false).
13. Save your changes.
14. If you want to add data columns from the main axis table to the grid, perform the following steps:

    > ⚠ Caution
    >
    > Don't add system columns to a data grid. For example, PK_ID, PARENT_ID, DELETE_FLAG, etc.

    a. Choose to open the *Child Table Chooser* tool and select the main axis table for the records that you want displayed in the grid.

       Alternately, you can also enter the name of the table manually in the *Table* field.

    b. Choose the icon beside *Column Chooser*.
       The *Child Table Column Chooser* tool opens, providing access to the columns of the table you specified in the previous step.

    c. To add a column, click and drag it to the grid control in the questionnaire model.

       > → Tip
       >
       > Hold down the `Ctrl` key while clicking columns to select multiple columns for dragging to the grid control.

    d. If necessary, use the *Move Up* icon and the *Move Down* icon to establish the default display order of the columns when they are assembled in the view.
15. If you want to add data columns from an extension table to the grid, perform the following steps:

> ⚠ Caution
>
> Don't add system columns to a data grid. For example, PK_ID, PARENT_ID, DELETE_FLAG, etc.

a. Choose to open the *Child Table Chooser* tool and select the extension table for the records that you want displayed in the grid.

   Alternately, you can also enter the name of the table manually in the *Table* field.

b. Choose the icon beside *Column Chooser*.
   The *Child Table Column Chooser* tool opens, providing access to the columns of the extension table you specified in the previous step.

c. Click and drag it to the grid control in the questionnaire model.

   > → Tip
   >
   > Hold down the `Ctrl` key while clicking columns to select multiple columns for dragging to the grid control.

d. If necessary, use the *Move Up* icon and the *Move Down* icon to establish the default display order of the columns when they are assembled in the view.

16. Save your changes.

Related Information

Configuring a Detailed View for a Data Grid [page 203]
Adding the Validation Status Column [page 204]
Defining a Data Grid Column Text Control [page 205]

7.5.1 Configuring a Detailed View for a Data Grid

When you are designing your eApp, you need to take into consideration the amount of data that you will want to appear on the screens. A best practice is to configure detailed views for your data grids, which allows you to partition the display of data into logical chunks.

When you are designing the layout of your eApp, determine which data will be needed for each step in the process. If there won't be many fields on a screen, then in-line updates are acceptable. If there will be a large amount of data to be entered to satisfy a process, then the configuration of detailed views will offer a more efficient design. A detailed view will allow the user to select a data grid record in an eApp and open a screen where additional configuration/data entry can occur.

> → Tip
>
> It isn't recommend to have more than 5 columns of data being displayed on a data grid.

The configuration of detailed views for data grid is accomplished by selecting the *Has Child Screen* checkbox within the data grid and configuring the detailed view within the eApplication.

Optionally, a detailed view can be made exclusive to a data grid assembled in a specific parent screen. Set the `PARENT_SCREEN` attribute for the child view eApp screen record.

> **i Note**
>
> The detail view needs to use the same target table as the parent data grid

7.5.2 Adding the Validation Status Column

An column can be added to the data grid which will indicate whether there are validation issues in the grid or the detail view that need to be addressed.

Context

When the application or screen is validated, each data grid record will be flagged as Valid with a green identifier or Invalid with a red identifier.

Procedure

1. Go to the data definition for the data grid table.
2. Select the VALIDATE_STATUS column.
3. Save your changes.
4. Build the metadata for the product.
5. Within the questionnaire layout, navigate to the data grid control.
6. Choose the  icon located beside *Column Chooser* to open the column chooser tool.
7. Select the `VALIDATE_STATUS` column and drag it into the questionnaire layout for the data grid.
8. Optionally, open the *Questionnaire Painter* and add this field in the grid view.

   If you choose not to assemble the field to the grid view, grid records will still be flagged with a red or green identifier following the validation check.

7.6 Defining Screen Controls for Data Grid Columns

7.6.1 Defining a Data Grid Column Text Control

You can define the properties for each column in a data grid.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the column you want to define.

   The column must be assembled under the data grid.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Text* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Read Only	Select to make the field non-editable. Content display rules can be configured to override this setting and show this attribute in Runtime.
Hide Column	Select to make the question and its field invisible at Runtime. Content display rules can be configured to override this setting and show this attribute in Runtime.
Column Name	Shows the table and column name and isn't editable.

5. Complete the properties in the *Advanced* tab.

   The properties that appear are dependent on the question's data type.

Dimensions	This attribute is not supported.
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported.

	If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Grouping Enabled	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

7.6.2 Defining a Data Grid Column Date Control

You can define the properties for each column in a data grid.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the column you want to define.

   The column must be assembled under the data grid.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the  *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Date* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

Question Text The actual question as you want it to appear to the end user.

You can add the question directly to the text field.

Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.

Read Only Select to make the field non-editable.

Hide Column Select to make the question and its field invisible at Runtime.

Column Name Shows the table and column name and isn't editable.

Use 24-hour format This attribute is not supported.

Style The style of the date rendered in the UI is governed by this parameter.

If set to short, med, long or full, then the date format will be displayed based on the globalization style.

User Default

Derived based on your gloabilization settings.

Short Date

For example: 12/31/2015

Medium Date

For example: Dec 31, 2018

Long Date

For example: December 31st, 2018

Full Date

For example: Monday December 31st, 2018

5. Complete the properties in the *Advanced* tab.

The properties that appear are dependent on the question's data type.

Dimensions This attribute is not supported.

Mandatory Flag Determines whether the screen control is mandatory.

Select *Yes* or *No*. *Default* is not supported.

If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed.

When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen.

Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.

Mandatory Flag Rule A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory.

If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed.

When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen.

Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.

Grouping Enabled — This attribute is not supported.

Additional Properties — This attribute is not supported.

6. In the *Values* tab, choose *Save*.

7.6.3 Defining a Data Grid Column Number Control

You can define the properties for each column in a data grid.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the column you want to define.

   The column must be assembled under the data grid.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Number* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

   Question Text — The actual question as you want it to appear to the end user.

   You can add the question directly to the text field.

   Optionally, select the *Edit Content* button to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.

   Show Total — This attribute is not supported.

   Read Only — Select to make the field non-editable.

Hide Column	Select to make the question and its field invisible at Runtime.
Column Name	Shows the table and column name and isn't editable.
Min value	For defining the screen control's minimum acceptable value (inclusive). If validation fails during Runtime, the field will be outlined in red, an error message will be displayed below the control and an error will be included in the validation summary.
Max value	For defining the screen control's maximum acceptable value (inclusive). If validation fails during Runtime, the field will be outlined in red, an error message will be displayed below the control and an error will be included in the validation summary.

5. Complete the properties in the *Advanced* tab.

The properties that appear are dependent on the question's data type.

Dimensions	This attribute is not supported.
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported. If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Allow Negative	Select to allow negative values in a number field.
Display % sign	Displays the number, as entered, and appends a percent sign.
Currency	This attribute is not supported.
Length	Defines the maximum number of characters allowed to be entered in the screen control. The value set here will only be applied provided that it is less than the value specified in the data model.
Precision	The number of decimal places for a numeric value. The value set here will only be applied provided that it is less than the value specified in the data model. The value for this property cannot equal or exceed the *Length* property.

Grouping Enabled	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

7.6.4 Defining a Data Grid Column Data List Control

You can define the properties for each column in a data grid.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the column you want to define.

   The column must be assembled under the data grid.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the  *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Data List* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.

   > i Note
   >
   > The determination of the control type (dropdown list or radio buttons) is assigned in the *Properties* dialog of the *Questionnaire Painter*.

4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button  to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.
Read Only	Select to make the field non-editable. Content display rules can be configured to override this setting and show this attribute in Runtime.
Hide Column	Select to make the question and its field invisible at Runtime.

Content display rules can be configured to override this setting and show this attribute in Runtime.

Local List Enter a set of values to populate the data list.

The local list will only be used if no *Product Query* is defined.

Each record needs to include both the code value and its corresponding description (that will be displayed at Runtime).

Separate each record by semi-colons.

The size of the list is limited to 250 characters.

For example: `1=Cash;2=Check;3=Credit Card;4=EFT`

Code List This attribute is not supported.

Product Query Use to create a product query rule that returns a data table. Select the icon and select *New Rule*.

To create the rule using the wizard feature, choose the icon and select *Rule Wizard*.

> → Remember
>
> When creating a product query for a questionnaire model to populate a dropdown list, you must provide both the key column and the description column.

Query Rule This attribute is not supported.

Empty Option For a dropdown list, select to have the list of values start with an empty row.

This option is not applicable for radio button lists.

> i Note
>
> When defining a *Data List* in a questionnaire model for a data definition column, the contents of the data list must match the data type of the data definition column. For example, if the data list is for a numeric column but the data source returns a character string, the input data won't be validated in the questionnaire and an error will be generated when a value with a different data type is selected at Runtime.

5. Complete the properties in the *Advanced* tab.

The properties that appear are dependent on the question's data type.

Dimensions This attribute is not supported.

Mandatory Flag Determines whether the screen control is mandatory.

Select *Yes* or *No*. *Default* is not supported.

If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed.

When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen.

Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.

Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Grouping Enabled	This attribute is not supported.
Sort	When selected, the control records are sorted in ascending order.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

7.6.5 Defining a Data Grid Column Check Box Control

You can define the properties for each column in a data grid.

Procedure

1. In the product, select the questionnaire model and select the *Values* tab.
2. Select the column you want to define.

   The column must be assembled under the data grid.

   The *Properties* dialog appears.

   > → Tip
   >
   > If the *Properties* dialog is hidden, choose the  *Show/Hide Property* icon in the toolbar to reveal it.

3. Choose *Check Box* from the *Control Type* dropdown list.

   The properties that will be displayed are determined by the type of screen control chosen.
4. Complete the properties in the *General* tab.

Question Text	The actual question as you want it to appear to the end user. You can add the question directly to the text field. Optionally, select the *Edit Content* button  to open a Rich Text Format (RTF) editor to format the text. When you save your changes in the editor, the formatting will be applied to the content of the `Question Text` attribute field via tags.

Read Only	Select to make the field non-editable.
Hide Column	Select to make the question and its field invisible at Runtime.
Column Name	Shows the table and column name and isn't editable.

5. Complete the properties in the *Advanced* tab.

   The properties that appear are dependent on the question's data type.

Dimensions	This attribute is not supported.
Mandatory Flag	Determines whether the screen control is mandatory. Select *Yes* or *No*. *Default* is not supported. If set to *Yes*, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Mandatory Flag Rule	A rule that returns a Boolean response (true/false) to determine if the screen control is mandatory. If enabled, the screen control will have a red asterisk displayed next to it. The screen control will be outlined in red if validation is failed. When the validation check is executed, an error will be generated if this field isn't populated. The error will be listed in both the error grid and in the validation summary for the screen. Note that the execution of the *Mandatory Flag Rule* will take precedence over the setting of the *Mandatory Flag*.
Grouping Enabled	This attribute is not supported.
Additional Properties	This attribute is not supported.

6. In the *Values* tab, choose *Save*.

7.7 Working with Questionnaire Views

You use a questionnaire view to define how a questionnaire model appears at Runtime. You create and edit questionnaire views from within *Questionnaire Painter*, which you access from within your questionnaire model.

In a questionnaire view, you specify the following information:

- The number of tables and their sizes
- The placement of objects (also known as elements), including questions, screen controls, headings, fields, text and grids within the tables

- The formatting of objects
- Event handlers for objects

You perform these actions visually, by selecting elements and dragging and dropping them from the questionnaire model's tree into the tables. Elements can derive from any questionnaire model in the *Product Tree*.

You can define multiple questionnaire views for a questionnaire model, and use rules or scripts to determine which view to present to the user.

A view links to a model on a question `PCD` attribute. After you create the views, the relationship between model and views works as follows:

- Runtime, you may need to rebuild the viewIf you edit a question in the model, all the linked views automatically reflect your changes; however, to have them show in
- If you add a question to the model, you need to add it to each view
- If you delete a question from the model, the question is automatically deleted from each view

If you edit a question in the model, all the linked views automatically reflect your changes;To save time, you can also clone a view, which makes a copy of the view.

Understanding Responsive Layout

The `OData UI5 Renderer` displays all configured screen layouts using the Block Layout format employed by SAP UI5, which displays content in a section-based manner. It renders configured eApp views in a responsive manner where attributes, such as headers, labels, fields, buttons and data grids will be oriented within the view depending on the size of the screen.

When designing a view (via the *Questionnaire Painter*), the following layout principles should be understood and guidelines adhered to:

- The label and associated data field should be positioned next on each other in the same row.
- The attribute assembled to the left will be right-justified in Runtime. Its corresponding attribute, assembled to the right, will be left-justified.
- No more than 4 columns should be configured within a view.
- No more than 5 columns should be assembled within a data grid. Otherwise, a the result will be a compressed view, with very narrow columns.
- A Text screen control assembled in the 1st cell of any outer table (cell 1.1) will be rendered as a label, left-justified and formatted as a Heading1 (H1). Note that configured RTF tags will be ignored in this instance.
- A Text screen control assembled in the 1st cell of any inner table (cell 1.1) will be rendered as a label, left-justified but not formatted.
- The name of the data grid (specified in the screen control) will be displayed as the data grid label. It will be situated at the top left with a "Title" format (for example, left-justified, with a larger font size).
- Child fields that are dependent on a `Parent Trigger` are typically configured below the parent field but are offset to the right.
- The horizontal merging of cells will affect the display of the data in the merged cell, as well as impact the positioning of the following cells.
- Vertical merging of cells is not supported.

- If the last cell in the row is empty, if it is ignored.
- Empty rows will be rendered as blank spaces in Runtime.

Example1:

Heading	
Sub-Heading 1	
First Field Label	
Second Field Label	
Third Field Label	
Sub-Heading 2	
Fourth Field Label	
Fifth Field Label	
Sixth Field Label	

A single table with two columns and nine rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Sub-headings have no formatting.

The field labels are right-justified and the corresponding attributes are left-justified.

Example 2:

You have a primary table with 2 columns and 3 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Nested 2x4 tables are inserted in cells 2:1, 2:2, 3:2 and 3:3.

Sub-headings have no formatting.

The field labels are right-justified and the corresponding attributes are left-justified.

Heading	
Sub-Heading 1 First Field Label Second Field Label Third Field Label	Sub-Heading 2 First Field Label Second Field Label Third Field Label
Sub-Heading 3 First Field Label Second Field Label Third Field Label	Sub-Heading 4 First Field Label Second Field Label Third Field Label

Example 3

You have a primary table with 2 columns and 3 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Nested 2x4 tables are inserted in cells 2:1 and 2:2.

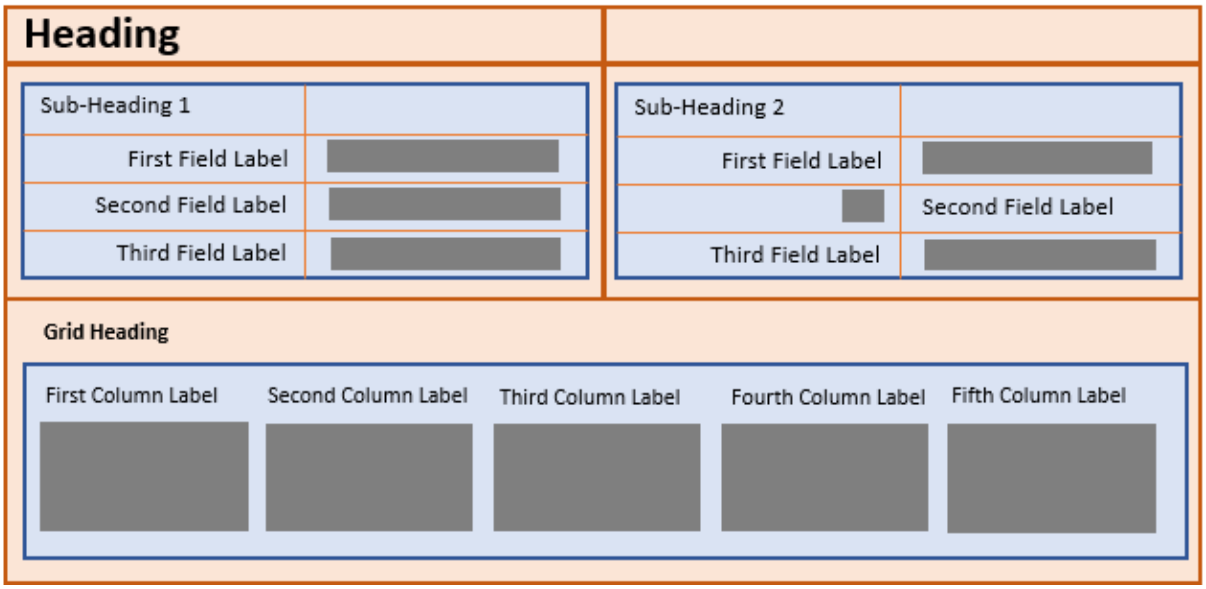
Sub-headings have no formatting.

The field labels are right-justified and the corresponding attributes are left-justified.

Cell 3:1 is extended into cell 3:2 (merging).

The grid control is assembled into 3:1.

The text specified in the *Question Text* field for the data grid control is displayed as a heading for the grid.



Example 4 a)

You have a table with 4 columns and 9 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Sub-headings have no formatting.

Child fields are assembled below the parent and are offset to the right with a blank cell to the left.

There are no blank cells between the field label and attribute.

Configuration with no blank cells between label and field

Heading			
Sub-Heading 1			
Parent Field Label			
	Child Field Label		
		Sub-Child Field Label	
Sub-Heading 2			
Fourth Field Label			
Fifth Field Label			
Sixth Field Label			

Rendered view

Heading

Sub-Heading 1

Parent Field Label

Child Field Label

Sub-Child Field Label

Sub-Heading 2

Fourth Field Label

Fifth Field Label

Sixth Field Label

Example 4 b)

You have a table with 4 columns and 9 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Sub-headings have no formatting.

Child fields are assembled below the parent and are offset to the right with a blank cell to the left.

The blank cells assembled between the field label and attribute will be ignored in Runtime.

Configuration with blank cells between label and field

Heading			
Sub-Heading 1			
Parent Field Label			
	Child Field Label		
		Sub-Child Field Label	
Sub-Heading 2			
Fourth Field Label			
Fifth Field Label			
Sixth Field Label			

Rendered view

Heading

Sub-Heading 1

Parent Field Label

Child Field Label

Sub-Child Field Label

Sub-Heading 2

Fourth Field Label

Fifth Field Label

Sixth Field Label

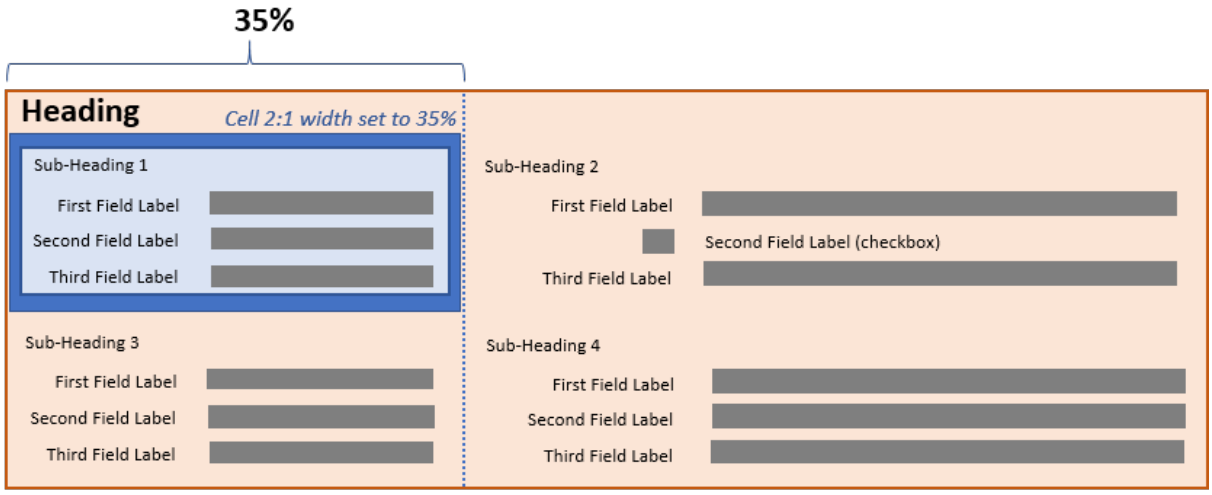
Example 5a

You have a table with 2 columns and 10 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Sub-headings have no formatting.

A width of 50% has been specified in Cell 7:2. Because of the width value being specified, all cells immediately below Cell 7:2 will inherit this setting unless otherwise specified.



Example 5b

You have a primary table with 2 columns and 3 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Nested 2x4 tables are inserted in cells 2:1, 2:2, 3:2 and 3:3.

Sub-headings have no formatting.

The field labels are right-justified and the corresponding attributes are left-justified.

A cell width of 35% is specified in the primary table cell, Cell 2:1. Because of the width value being specified, all cells immediately below Cell 2:1 will inherit this setting unless otherwise specified.

Configuration with blank cells between label and field

Heading			
Sub-Heading 1			
Parent Field Label			
	Child Field Label		
		Sub-Child Field Label	
Sub-Heading 2			
Fourth Field Label			
Fifth Field Label			
Sixth Field Label			

Rendered view

Heading

Sub-Heading 1

Parent Field Label

Child Field Label

Sub-Child Field Label

Sub-Heading 2

Fourth Field Label

Fifth Field Label

Sixth Field Label

Example 5c

You have a primary table with 2 columns and 3 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

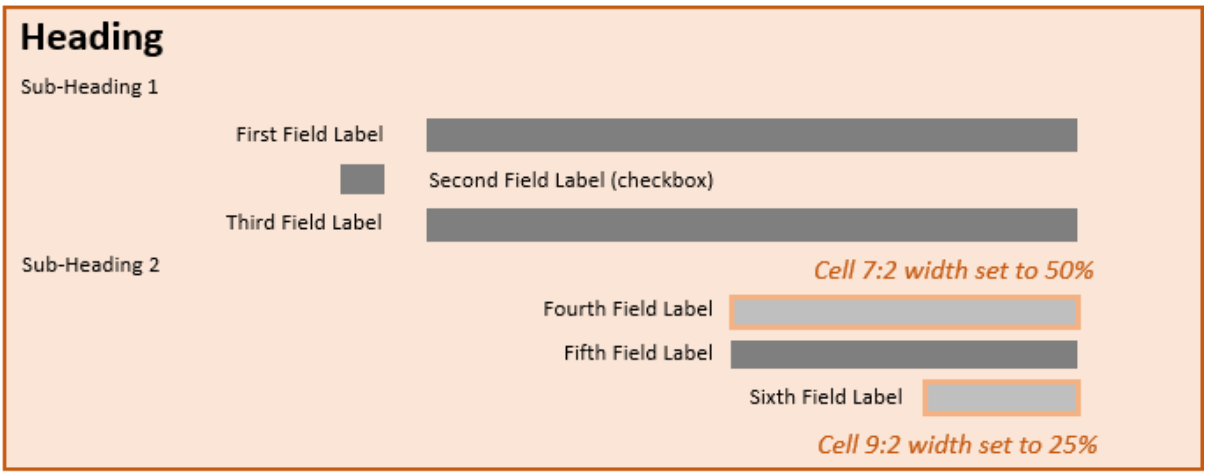
Nested 2x4 tables are inserted in cells 2:1, 2:2, 3:2 and 3:3.

Sub-headings have no formatting.

The field labels are right-justified and the corresponding attributes are left-justified.

A cell width of 35% is specified in the primary table cell, Cell 3:1. Because of the width value being specified, all cells immediately below Cell 2:1 will inherit this setting unless otherwise specified.

As no width value is specified in the preceding cells, two columns will be rendered that divide the space evenly (50% each).



Example 6a

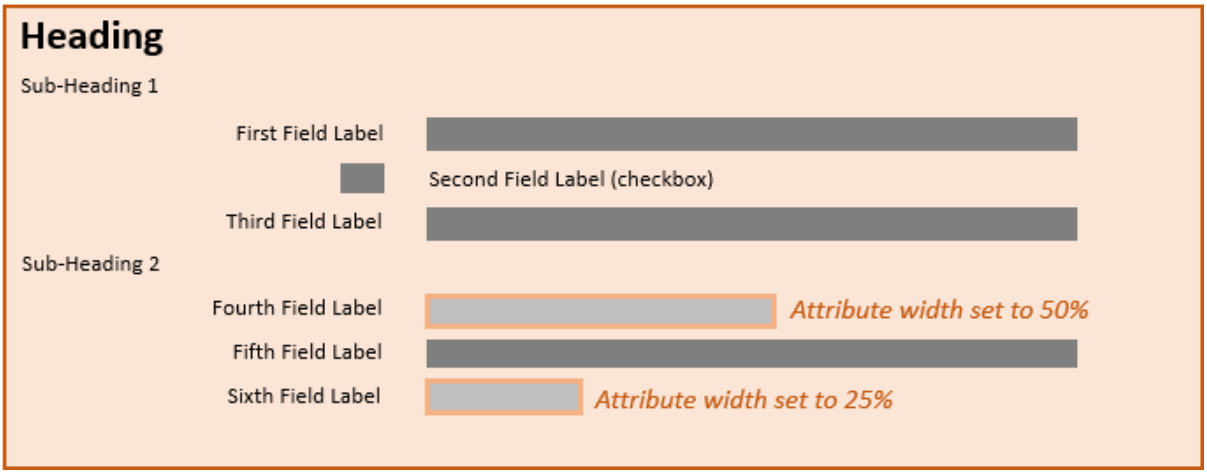
You have a table with 2 columns and 10 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Sub-headings have no formatting.

A width of 50% has been specified in the attribute assembled in Cell 7:2. The attribute will be apportioned 50% of the space available in that cell.

The cells below will not inherit that setting.



Example 6b

Using percentage, em or pixel size.

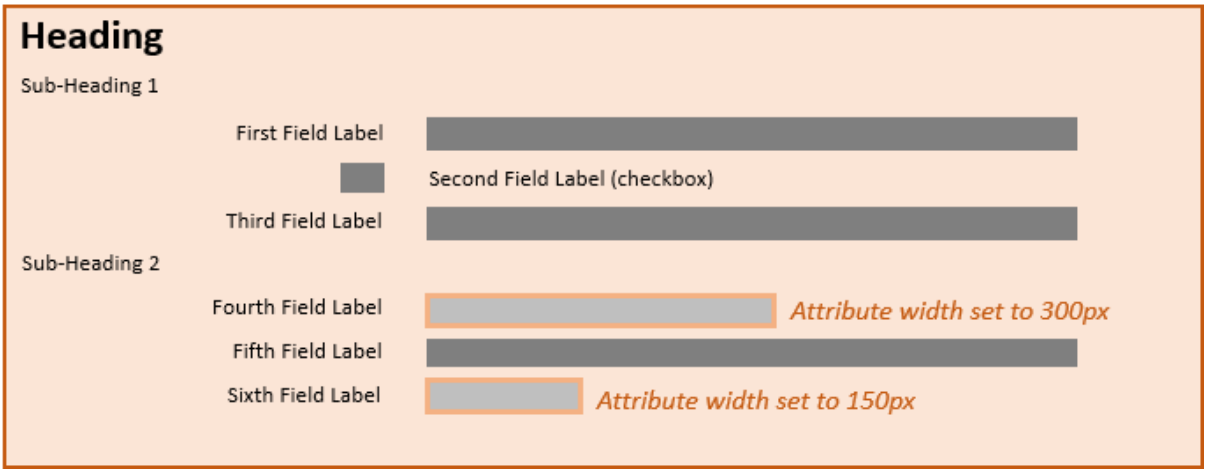
You have a table with 2 columns and 10 rows.

A Text screen control assembled in cell 1:1 is formatted as a Heading.

Sub-headings have no formatting.

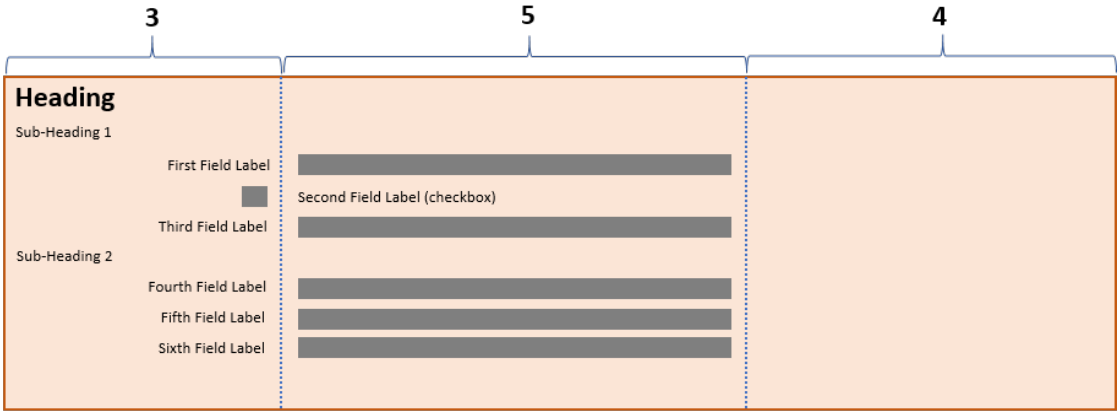
A width of 300px has been specified in the attribute assembled in Cell 7:2. The attribute will be apportioned 300px of the total pixel space available in that cell.

The cells below will not inherit that setting.

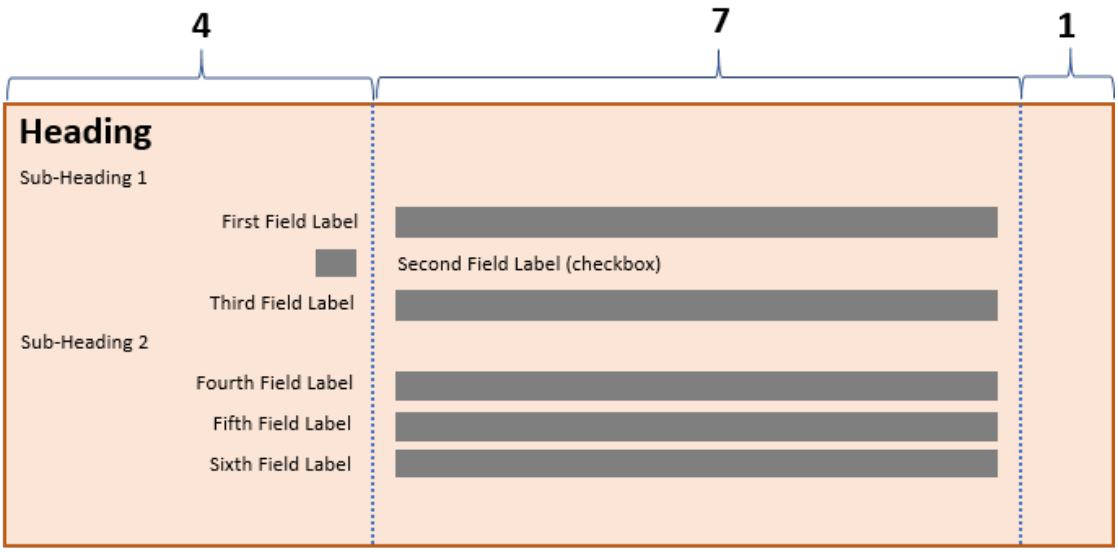


The positioning of attributes on the screen will be dependent on table configuration, defined widths and screen size:

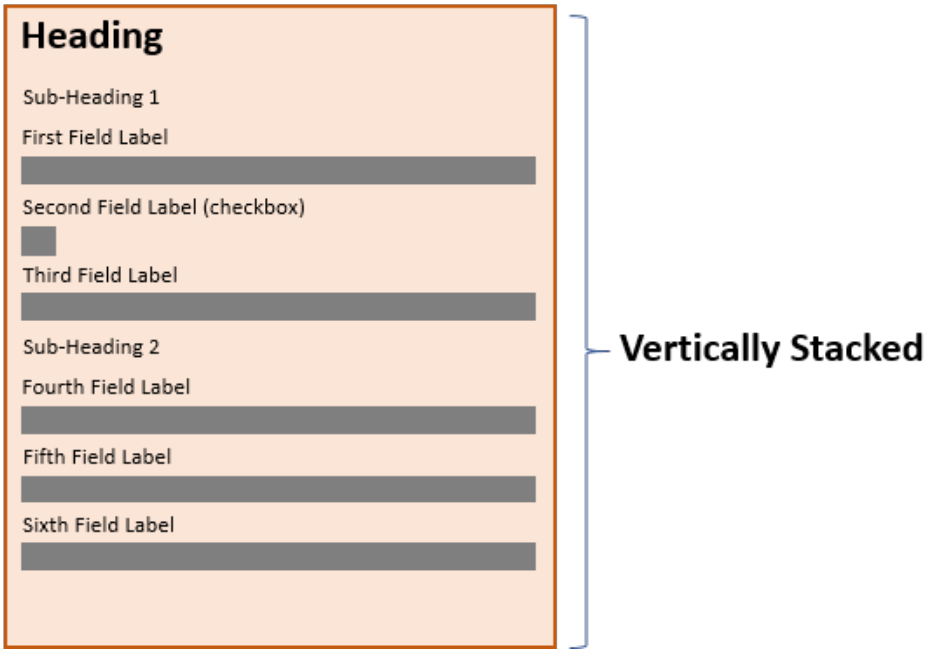
1. If you define a view with 1 table and 2 columns, with no defined cell widths, then the positioning of attributes on the screen will adjust depending on the browser screen size.
   - A large screen, with an effective width greater than 1050px, will display a 2 column view in a 3:5:4 ratio.



   - A medium screen, with an effective width between 650px and 1050px, will display a 2 column view in a 4:7:1 ratio.



- A small screen will vertically stack all attribute/elements. All width values for controls and attributes will be ignored.



2. If you define a view with 1 table and 2 columns, with defined cell widths as a percentage, then the positioning of attributes on large and medium screens will be based on the percentage. A small screen, with an effective width below 650px, will ignore all width values and vertically stack all attribute/elements.

For more information, see:

Block Layout

The table of contents (displayed to the left of the eApp screen view) will be displayed or collapsed depending on the browser window size.

For more information, see:

Side Navigation

Related Information

Creating a Questionnaire View [page 224]
Cloning a Questionnaire View [page 225]
Deleting a Questionnaire View [page 225]
Adding Elements to Questionnaire Views [page 226]

7.7.1 Creating a Questionnaire View

You create a questionnaire view in *Questionnaire Painter*. A view can consist of multiple tables, and these tables can also contain tables (up to two levels).

Procedure

1. Open the product, navigate to the questionnaire model, and select the *Values* tab.
2. Choose the *Questionnaire Painter* icon on the *Values* tab toolbar.
3. Choose *New* from the toolbar.
   The *New Questionnaire* dialog appears.
4. Enter a unique name in the *Name* field to identify the view.
5. Enter text in the *Description* field to explain the view's purpose.
6. Enter values in the *Row* and *Column* fields to define the size of the first screen table.

   If you don't specify values, the table defaults to four rows and five columns.
7. Choose *Submit*.
8. Save your changes.

Results

A questionnaire view appears as an empty grid of the size you specified.

7.7.2 Cloning a Questionnaire View

If an existing questionnaire view closely matches the requirements of a new view, you can save development effort by making a copy of the view, which you can then modify as needed.

Procedure

1. Open the product, navigate to the questionnaire model, and select the *Values* tab.
2. Choose the *Questionnaire Painter* icon on the *Values* tab toolbar.

   The *Questionnaire Painter* opens, listing all the views associated with a questionnaire model in the dropdown list in the toolbar.
3. Select the questionnaire view to be copied from the dropdown list.
4. Choose *Clone* from the toolbar.
   The *New Questionnaire* dialog appears.
5. Enter a unique name in the *Name* field to identify the view.
6. Enter text in the *Description* field to explain the view's purpose.
7. Choose *Submit*.

Results

A questionnaire view appears that is identical to the original and ready for you to edit.

7.7.3 Deleting a Questionnaire View

You can delete a questionnaire view.

Procedure

1. Open the product, navigate to the questionnaire model, and select the *Values* tab.
2. Choose the *Questionnaire Painter* icon on the *Values* tab toolbar.

   The *Questionnaire Painter* opens, listing all the views associated with a questionnaire model in the dropdown list in the toolbar.
3. Select the questionnaire view to be deleted from the dropdown list.
4. Choose *Delete View* from the toolbar.

   A confirmation message appears.

5. Choose *Yes* to confirm your decision.

Results

The view is deleted and the layout area appears blank.

7.7.4 Adding Elements to Questionnaire Views

After you create a questionnaire view, you have a blank table to which you need to add the elements that comprise the view.

Context

The basic steps in defining a view are as follows:

Procedure

1. Access the list bar and drag elements from the *Questionnaire Model Tree* panel and position them in the view. When selecting elements for dragging and dropping, click the element's icon, not its description; in some cases selecting the description causes the element to appear to be undraggable. You can only drag and drop items into empty cells; you can't drag and then insert between existing content.

   > **i Note**
   >
   > In the *Questionnaire Model Tree*, questionnaire containers appear as folders, as do parent questions. In addition, you can't have more than one of the same question on the same screen.

2. To add elements from other models, in the *Questionnaire Model Tree* panel use the dropdown list to select a different model in the *Product Tree*, causing the model's structure to appear. You can now drag these elements into your view.

   > **i Note**
   >
   > When dragging a questionnaire model field that is a parent-child pair, if you drag the pair unopened, a warning messages displays: `The question you are adding contains child question(s) which will not be added unless you expand the question. Do you want to add the parent question only?`

   - Choose *Yes* to add the parent question only.
   - Choose *No* to cancel without adding either. Then expand the question folder in the tree view and drag the desired questions.

3. To see a basic representation of how the view will look at Runtime, in the toolbar, choose *Preview*.

   In preview mode, the object labels will be displayed in the primary language selected for the current product, or blank if values haven't been entered. When selecting another primary language from the product's *Properties* dialog, the object labels will be displayed in the newly-selected primary language, or blank if values haven't been entered.

   > **i Note**
   >
   > To change the language in a questionnaire model, go to the *Properties* dialog of the current object containing the questionnaire model.

4. To reposition an element, select and drag it.

   You can't drag and drop an element over two existing elements in order to "auto insert." For example, if two columns are side by side within a data grid, and you want to place a different column between them, you must first either insert a new grid column to create a destination to drag the column to, or move the second column to free its position.

5. You perform many of the layout functions in a questionnaire view using the *Toolbox* palette. To launch the palette, choose the *Toolbox* icon in the toolbar. The commands available are as follows:

New Table	Adds a grid to the selected screen cell. When you choose , a sizing grid appears and you specify the table size by moving the pointer and clicking.
Delete	Deletes the selected element.
Delete Column	Deletes the selected column. Use caution, as any elements contained in the column are deleted without warning.
Delete Row	Deletes the selected row. Use caution, as any elements contained in the row are deleted without warning.
Insert Column Left	Adds a column to the left of the selected cell's column.
Insert Column Right	Adds a column to the right of the selected cell's column.
Append Column	Adds a column at the far right of a table.
Decrease Column Span	Spanning merges a cell with the next column to its right. Each click removes the last cell from the spanned cell. To remove multiple cells at once, enter the number in the field located between the buttons, then press `Tab`.

Increase Column Span	Spanning merges a cell with the next column to its right. Each click adds the next cell on its right. To add multiple cells at once, enter the number in the field located between the buttons, then press `Tab`.
Insert Row Above	Adds a row above the selected cell's row.
Insert Row Below	Adds a row below the selected cell's row.
Append Row	Adds a row at the bottom of a table.
Decrease Row Span	Spanning merges a cell with the row below. Each click removes the lowest cell from the spanned cell to the spanned cell. To add multiple cells at once, enter the number in the field located between the buttons, then press `Tab`.
Increase Row Span	Spanning merges a cell with the row below. Each click adds the next cell below to the spanned cell. To add multiple cells at once, enter the number in the field located between the buttons, then press `Tab`.

Next, define the properties of elements as necessary, using the *Properties* panel in the list bar.

6. Labels have the following properties:

Width	The width of the label in pixels.
LineBreak	Forces the label to the next line.
Wrapping	Forces a label's text to wrap, when applicable, to avoid long strings from being cut off.

7. The following properties apply to tables:

   ID A table can be assigned a unique variable “ID” which can later be utilized by a rule.
   For example, in a content display rule to hide the table.

   All other properties and event handlers are not relevant to tables.

8. The following properties apply to rows:

   ID A row can be assigned a unique variable “ID” which can later be utilized by a rule.
   For example, in a content display rule to hide the row.

   All other properties and event handlers are not relevant to rows.

9. The following properties apply to cells:

   ID A cell can be assigned a unique variable ID which can later be utilized by a rule.
   For example, in a content display rule to hide a cell.

   Width Adjusts the width of the cell in the Runtime view.
   Specified as a percentage.

Defined widths are not respected in vertically stacked mode (if the screen view is small and the elements are stacked vertically).

If width is defined on a cell, then the cells underneath it in the table column will inherit the setting.

All other properties and event handlers are not relevant to cells.

10. The following properties apply to screen controls:

    These properties apply to text box, text area, date, number, and data list (dropdown only) screen controls.

    Width Specify the object's width in percentage, em or pixels.

    Height This applies only to text area screen controls, and specifies the number of rows in the memo field..

11. Event handlers add flexibility to your questionnaires by enabling you specify responses to user actions within questions or other view elements. An action (such as save or refresh) is triggered by one or more of the standard events that you specify. Only the following event handlers are supported:

    `onChange()` This event handler is only applicable to text box, text area, number, date, checkbox, and Yes/No Radio Buttons and data list (dropdown only) screen controls.

    `onClick()` This event handler is only applicable to button, Yes/No Radio Buttons, checkbox and data list (dropdown only) screen controls.

12. To add an event handler to an element in a questionnaire view:
    a. Select the element in the list bar, open the *Event Handler* panel. If the element can have event handlers, they appear in the panel.
    b. You can use one of the provided event handlers or select further ones from the dropdown list. Each time you select an event from the dropdown list, choose *Add*.
    c. In each event that you want to use, add the name of an appropriate method in your JavaScript.

       For example `'this.refreshScreen();'` and `'this.save();'`.
    d. Choose *Apply*.
13. Save and build the view.

7.8 Working with Questionnaire Layouts

The final step in adding a questionnaire model to an eApp is to create a layout attached to a specific eApp screen component and then to add questionnaire views to the layout. You do this using the *Questionnaire Layout Painter*.

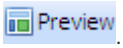
Procedure

1. Navigate the *Product Tree* to the eApp and select the eApp screen component .
2. To add a questionnaire layout to an eApp screen, perform the following steps:

a. Select the *Values* tab.
b. Add a data value row.
c. Enter a name for the screen in new row's `Screen Name` attribute, .
d. Right-click in the `Screen ID` attribute and select *New Questionnaire Layout*.
The *Create A New Grid* dialog appears.
e. Enter the number of rows and columns that you want for the screen grid.

Typically you would have one row for each view.

f. Choose *OK*.
The *Questionnaire Layout Painter* appears.

3. If you want to add a view to a questionnaire layout, perform the following steps:
a. Open the list bar (located at the far right) and open the *Questionnaire Views* panel.
A tree appears showing the folders that contain the questionnaire views in the product.
b. Select and drag each view that you want to add to the layout; position the view.
c. If you want to see a preview of how the view will look at Runtime, choose 

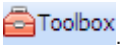
.

In preview mode, the object labels will be displayed in the primary language selected for the current product, or blank if values haven't been entered. When selecting another primary language from the product's *Properties* dialog, the object labels will be displayed in the newly-selected primary language, or blank if values haven't been entered.

> **i Note**
>
> To change the language in a questionnaire model, go to the *Properties* dialog of the current object containing the questionnaire model.

d. Customize the table as necessary.

> **i Note**
>
> Tables and their cells have the same properties as in *Questionnaire Painter*.

4. A layout can consist of multiple tables. These tables can also contain tables. To add tables and edit them, including the original grid:
a. Choose 

.

The palette contains the same commands as the one used for views.

b. Choose the  *New Table* icon; use the pointer to select the grid size, and click.
c. You can use the other commands in the toolbox to add or remove rows and columns and to delete tables.

5. You can specify the Runtime width of the screen for a questionnaire layout:
a. Select the `Body` object by choosing *[Body]* just above the main grid.
b. Access the list bar and open the *Body Properties* panel.
c. Enter the pixel value in the *Screen Width* field and choose the 

Save icon.

Width can alternatively be specified in percentage, em or responsive layout format (L,M,S) as well. For example:

- 100px
- 50%

- 10em
- L6 M6 S12

6. You can add event handlers to the elements in your questionnaire layout. You do this just as you would in a questionnaire view.
7. If you want to add a JavaScript directly to a questionnaire layout:
   a. Show the list bar and open its *Custom JavaScript* panel.
   b. Paste your script into the text box.
   c. Choose *Validate*.
   d. Choose *Apply*.
8. If you want to specify an external JavaScript file for the layout:
   a. Show the list bar and open its *Custom Files* panel.
   b. In the appropriate field, enter the path and name of the file.

      For example, `/classpath/com/testscript.js`. Example for files uploaded in the App Custom Web Root: `/custom/js/testscript.js`
   c. Choose *Apply*.
9. Complete the questionnaire layout:
   a. Select the 

Save icon and the 

Build icon.
   b. Close the *Questionnaire Layout Painter*.
   c. Close the painter.

      You are returned to the *Values* tab.
   d. Provide a value for the screen's position in the `Screen Order` attribute, .
   e. Select a table from the `Table Name` attribute.
   f. Save your changes.
   g. Publish your product.
10. If you want to edit a questionnaire layout:
   a. In the *Values* tab, right-click in the `Screen ID` attribute and select *Edit*.
   b. The *Questionnaire Layout Painter* appears.
   c. Make your changes.
   d. Select the 

Save icon and the 

Build icon.
   e. Publish your product.

Related Information

Adding Elements to Questionnaire Views [page 226]

7.9 Supported Rich Text Formatting

The Label/Text displayed on the eApp screen view is configured in the questionnaire properties of the control. The configuration allows for application of Rich Text Formatting (RTF) for selected text.

The following table indicates the RTF properties that can be applied in the Product Modeler and that are respected by in Runtime:

RTF Property	Example
Bold	`<b>Sample Text</b>If a format is not listed in the above table, it is not supported and should not be used in a property.`
Italics	`<I>Sample Text</I>`
Underline	`<u>Sample Text</u>`
Strikethrough	`<strike>Sample Text</strike>`
Superscript	`<sup>Sample Text</sup>`
Subscript	`<sub>Sample Text</sub>`
Foreground Color	`<span style="COLOR: #ff0000">Sample Text</span>`
Background Color	`<span style="BACKGROUND-COLOR: #ff0000">Sample Text</span>`

If a format is not listed in the above table, it is not supported and should not be used in a property.

Related Information

Defining Screen Controls [page 180]
Defining a Data Grid Column Text Control [page 205]
Defining a Data Grid Column Date Control [page 206]
Defining a Data Grid Column Number Control [page 208]
Defining a Data Grid Column Data List Control [page 210]
Defining a Data Grid Column Check Box Control [page 212]

8 eApps

eApps is an FS-PRO extension that enables you to encapsulate all the elements of a product such that it can be linked to an FS-QUO application template and work immediately, without customization.

In the Product Modeler, this collection of sections and screens is called the eApp (eApplication). For each product, you design the eApp by creating user interface objects at the product (policy), coverage group, and coverage levels. The objects to create include questionnaires, views, screens, and rules. Each of these levels have their own eApps. Together, these eApps form the entire user interface shown in the *Fill Out Application* section of FS-QUO. By specifying the user interface for each coverage group and coverage, you can reuse those sections and screens in different products and save development time. To use eApps in your product, you must inherit your product from a product that already uses eApps.

In FS-QUO, when a producer or underwriter clicks a policy, coverage group, and coverage section in *Fill Out Application*, the relevant, specified information and questions appear.

An eApp is a structure that consists of the following related objects:

`Data Definitions`	An object that stores the metadata describing a single data table. A questionnaire can contain questions from different data definitions.
`Questionnaires`	Questionnaires enable you to combine questions, rules, screen behaviors, and formatting in one object.
`Screens`	The standard screen used to host questionnaires
`Rules`	Rules enable you to add business logic to objects in FS-PRO.

You can define an eApp at any product level (product, coverage group, coverage).

- You can define multiple eApps within a single product, at any of the levels.
- In a marketable product, you can roll up all the Apps to function as one unified eApp that FS-QUO users can navigate through.
- You can use AJAX or other Web 2.0 technologies.
- You can customize the GUI through the extendable rendering engine.

Related Information

Risk-based and Coverage-based Product Models [page 234]
eApp Objects in Coverage-Based Products [page 234]
How eApps Call Each Other [page 239]
eApp Renderer [page 243]

8.1 Risk-based and Coverage-based Product Models

Depending on the marketable product that you want to develop, you can created risk-based or coverage-based product models. The eApp design differs according to these product models.

Risk-based Products

For risk-based products, you specify coverages under risks in the product model. The `Mock Commercial General Liability Monoline` product that is provided with FS-PRO is an example of a risk-based product.

> **i Note**
>
> The topics in this chapter describe the eApp configuration for risk-based products.

Coverage-based Products

For coverage-based products, you specify risks under contracts or coverages.

Related Information

eApp Renderer [page 243]

8.2 eApp Objects in Coverage-Based Products

This section describes the objects used to design eApps.

An eApp consists of two major parts: the layout part, and the content part.

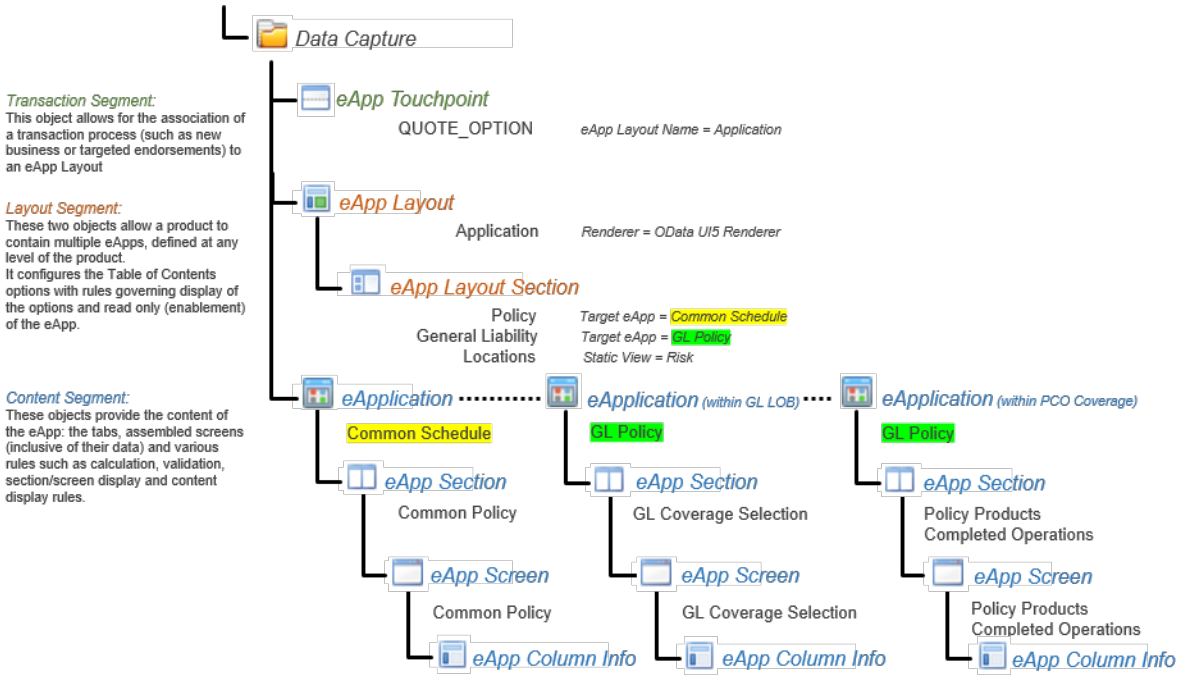
The layout is made of two objects: `eApp Layout` and `eApp Layout Section`. These objects define how users navigate through your eApps at the product and the group coverage levels.

The content segment is made up of the following objects:

`eApplication`	Used for navigation through the eApp.
`eApp Section`	Used for navigation through the eApp.
`eApp Screen`	
`eApp Column Info`	Used for the behavior (such as rules and validation) of questions.

These objects define the screens and data of an eApp.

The following diagram shows an example eApp structure:



> **i Note**
>
> For information about attribute details, such as data type and length, see the object's *Attribute* tab in the Product Modeler.

Related Information

Layout Objects in Coverage-Based Products [page 235]
Content Objects in Coverage-Based Products [page 237]

8.2.1 Layout Objects in Coverage-Based Products

eApps are created with the following layout objects:

- `eApp Layout`
- `eApp Layout Section`

These objects define top-level navigation.

The eApp Layout Object

The eApp Layout object creates the navigation features of an eApp. The eApp Layout object has the following attributes:

`eApp Name`	The name of the eApp. This is a required attribute.
`Renderer`	Supports extending the capabilities for rendering eApp content. The default UI5 renderer is *[Standard UI5 Renderer] pqm.eapp.splitapp*.
`Render Type`	Supports extending the capabilities for rendering eApp content. Appears in three different objects of the eApp structure, and there are differences in how these objects use it.
`Display Name`	The title of the eApp. It is displayed on the title bar at the top of the app. This attribute is used at the Product level. This is a required attribute.
`Read Only Rule`	Used to specify eApps to be rendered as read only. This is a required attribute.

The eApp Layout Section Object

Each eApp Layout Section object must have an eApp Layout Section. Each eApp Layout Section record defines a section of the layout, and links to an eApp (the one specified in the `Target eApp` attribute) that appears in the section.

The eApp Layout Section object has the following attributes:

`Section Name`	This is a required attribute.
`Target eApp`	This is a required attribute.
`Section Order`	This is a required attribute.
`Section Display Rule`	Determines whether the layout section will be shown or hidden.
`DescriptionRule`	
`SYSATTR_TABLE_NAME`	used by the system. This is a required attribute if there is a Rule Type attribute.

8.2.2 Content Objects in Coverage-Based Products

The following four objects define content and its presentation in an eApp:

- eApplication
- eApp Section
- eApp Screen
- eApp Column Info
- eApp Data Table
- eApp Data Attribute

The eApplication Object

The `eApplication` object stores the name of the eApp and specifies how the associated eApp Sections are rendered.

The `eApplication` object has the following attributes:

`eApp Name`	This is a required attribute.
`Renderer`	Select the *eAppContent* item.
`Render Type`	Determines the display used in the associated eApp Sections.
`Display Name`	The name that will be displayed on the left hand side master list. This is a required attribute.

The eApp Section Object

The `eApp Section` object stores the names of the eApp's sections, how you want them rendered, the order in which you want the sections to appear, and rules that control whether a section appears or is hidden.

The `eApp Section` object has the following attributes:

`Section Name`	This is a required attribute.
`Render Type`	Determines the display used in the associated screens.
`Section Order`	This is a required attribute.
`Section Display Rule`	Determines whether the layout section will be shown or hidden.
`SYSATTR_TABLE_NAME`	Used by the system. Required if there is a `Rule Type` attribute.

The eApp Screen Object

The eApp Screen object stores the name of the screen, its order within the section, an optional screen display rule, the screen itself, the name of the data definition table that the screen links to.

The eApp Screen object has the following attributes:

`Screen Name`	
`Screen ID`	This is a required attribute.
`Screen Order`	This is a required attribute.
`Screen Display Rule`	
`Content Display Rule`	
`Trigger All Display Rules`	If you want all the display rules for the eApp structure to run when the user saves this screen (whether explicitly or implicitly), in the `Trigger All Display Rules` attribute, enter **`1`** or **`Y`**. The default value is **`0`** or **`N`**. Note that even if `Trigger All Display Rules` is set to N, the first time the user navigates through the eApp sections and screens all the display rules will run.
`TABLE_NAME`	This is a required attribute.
`SYSATTR_TABLE_NAME`	Used by the system. Required if there is a `Rule Type` attribute.

The eApp Column Info Object

The `eApp Column Info` object stores validation and calculation rules, which you assign to specific columns, including their execution order and execution mode and whether a rule is currently active.

The `eApp Column Info` object has the following attributes:

`Column Name`	This is a required attribute.
`VALIDATION_RULEID`	This is a reference to a validation rule. A new rule can be defined from this attribute.
`CALCULATION_RULEID`	This is a reference to the calculation rule. A new rule can be defined from this attribute.
`CALCULATION_RULE_ORDER`	A number value which, if specified, it determines the order of execution for the rules. A new rule can also be defined from this attribute.
`SYSATTR_APP_TABLE_OBJ_ID`	This attribute is not being used at this time.

IS_VALIDATION_RULE_ACTIVE	If this attribute is set to Y or 1, the validation rule will be executed in its defined sequence. If this attribute is set to zero, the rule will not be executed.
IS_CALCULATION_RULE_ACTIVE	If this attribute is set to Y, the calculation rule will be executed in its defined sequence. If this attribute is set to zero, the rule will not be executed.
RULE_EXEC_MODE	This attribute determines when the rule will be executed. The following options are available: Pre&Form (1) — This rule will be executed when navigating to a screen, before a screen is rendered. It will also be triggered when saving the screen. This is the default value if no option is selected. Pre (2) — This rule will be executed when navigating to a screen, before a screen is rendered. Post (3) — These rule will be executed when navigating away from a screen, after saving. Form (4) — The rule gets triggered when you save the screen Pre&Post (5) — This rule will be executed when navigating to a screen, before a screen is rendered. This rule will also be executed when navigating away from a screen, after saving Once a type is selected from the dropdown list, the number value of the selected choice will be displayed.
SYSATTR_PROCESS_OBJ_ID	This attribute is not being used at this time.
SYSATTR_TABLE_NAME	Used by the system. Required if there is a `Rule Type` attribute.

8.3 How eApps Call Each Other

eApps provide for flexible linking, which means that you can create a group of these eApps in many different ways.

To link the eApps, the following mandatory attributes are used:

- `eApp Name`, in the `eApplication` object
- `Target eApp`, in the `eApp Layout Section` object

Both the `eApp Name` attributes serve the same purpose: they uniquely identify their object grouping. By adding an existing `eApp Name` value to the `Target eApp` attribute, you link the `eApp Layout Section` to the object named, regardless of where that object is in the *Product Tree*.

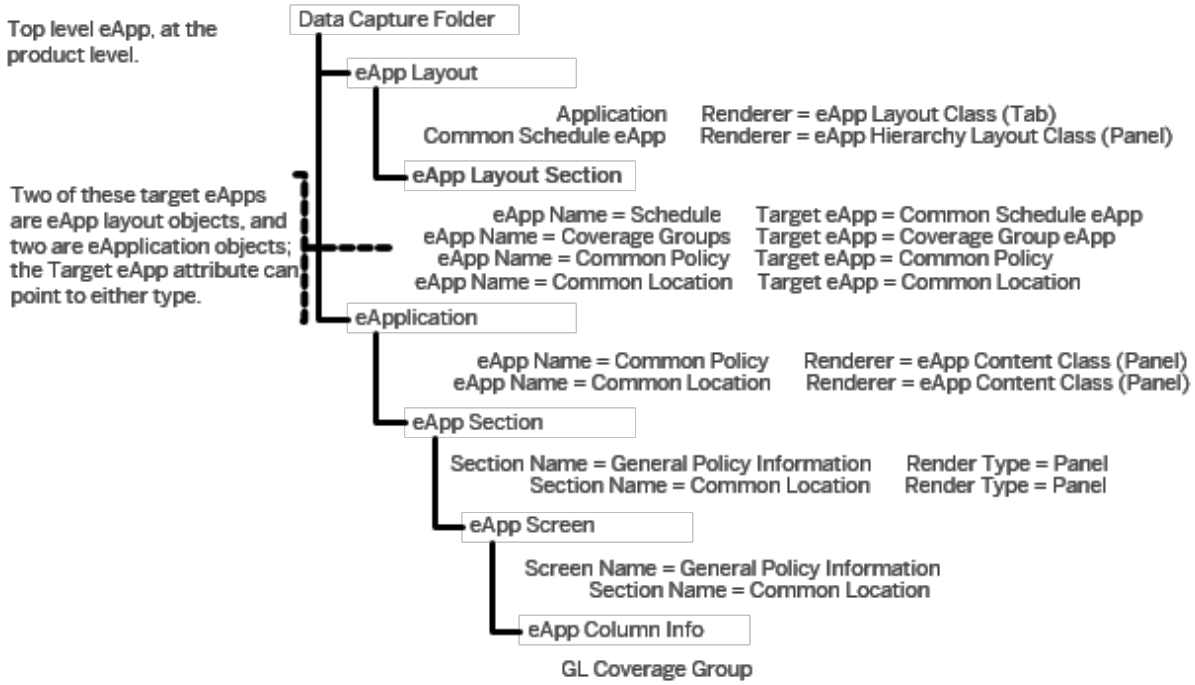
Notes:

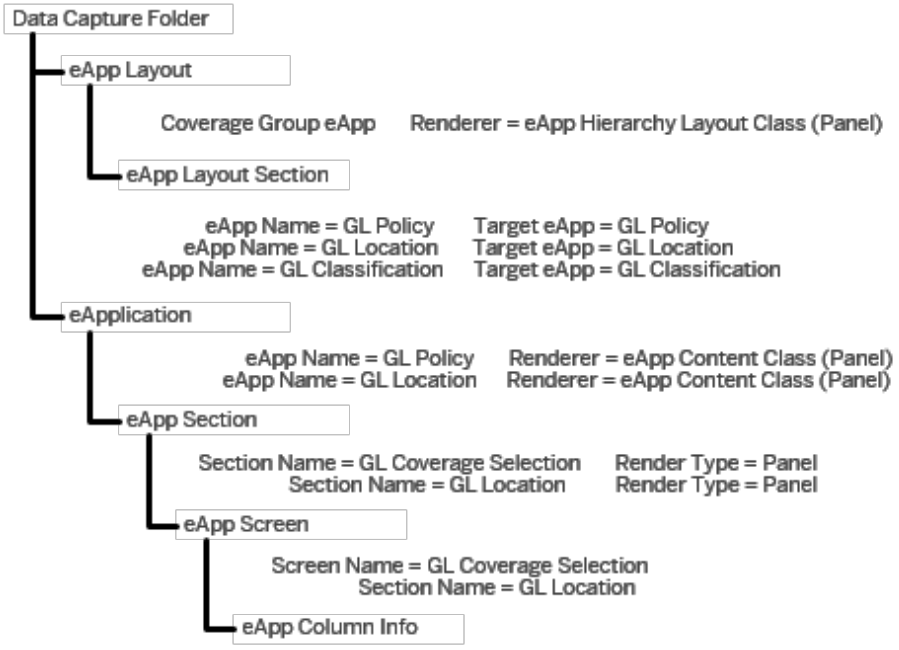
- The `Target eApp` attribute has the following restrictions:
  - The `Target eApp` attribute is only for use at the product level and the coverage group level.
  - Calling a coverage-level eApp Layout object isn't permitted.
  - In a product, if the `Target eApp` attribute isn't found, no content will display and run.
- As long as an eApp has the calling eApp name, FS-PRO traverses the entire *Product Tree*, finds the eApp, and displays it. Therefore, an eApp from any assembled coverage group or coverage can appear at a top-level tab.

Example: eApp Calling Structure

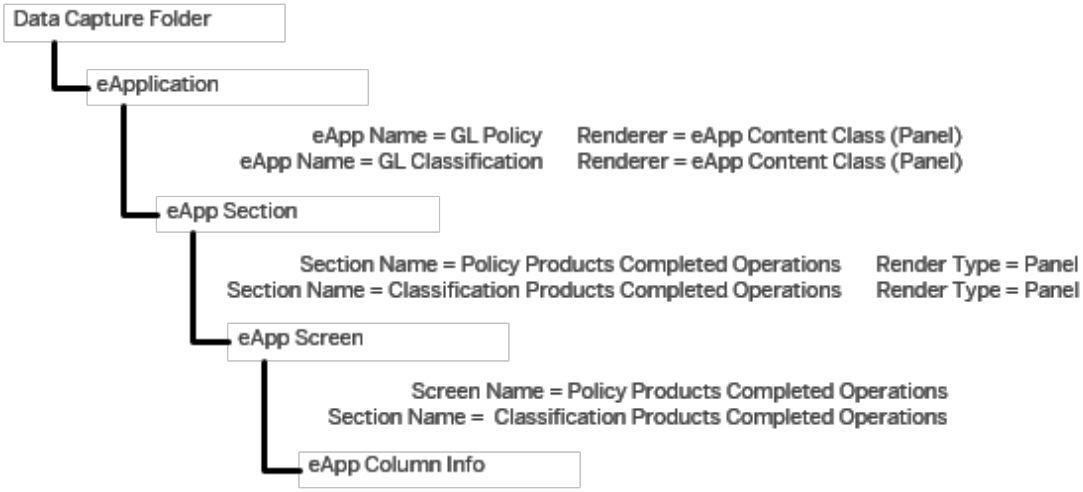
This example shows how the presentation layer uses the `Target eApp` attribute to connect to eApps.

Two of the coverage-level eApps are shown on the next figure. For full examples, see the templates in the Product Modeler.

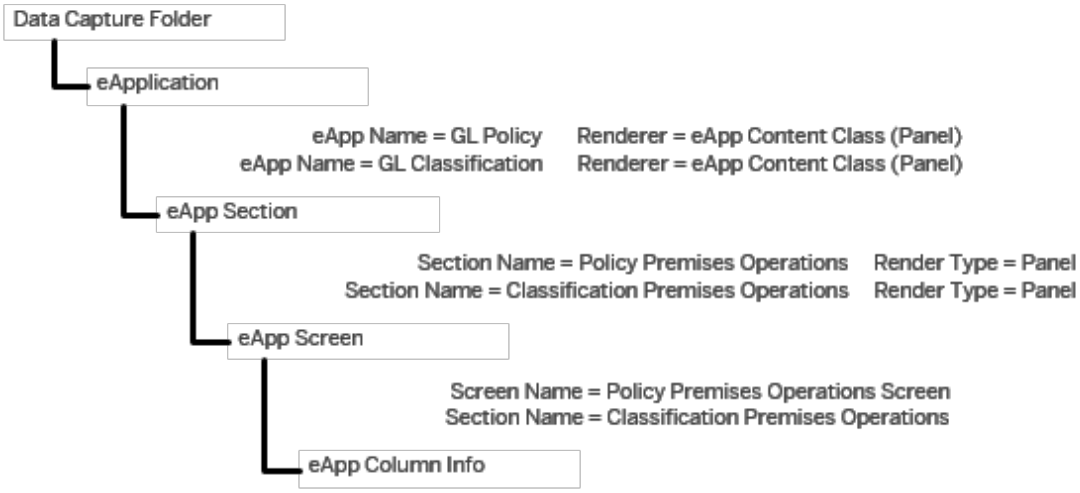

Top level eApp, at the product level.
Data Capture Folder
eApp Layout
Application
Renderer = eApp Layout Class (Tab)
Common Schedule eApp
Renderer = eApp Hierarchy Layout Class (Panel)
eApp Layout Section
Two of these target eApps are eApp layout objects, and two are eApplication objects; the Target eApp attribute can point to either type.
eApp Name = Schedule
Target eApp = Common Schedule eApp
eApp Name = Coverage Groups
Target eApp = Coverage Group eApp
eApp Name = Common Policy
Target eApp = Common Policy
eApp Name = Common Location
Target eApp = Common Location
eApplication
eApp Name = Common Policy
Renderer = eApp Content Class (Panel)
eApp Name = Common Location
Renderer = eApp Content Class (Panel)
eApp Section
Section Name = General Policy Information
Render Type = Panel
Section Name = Common Location
Render Type = Panel
eApp Screen
Screen Name = General Policy Information
Section Name = Common Location
eApp Column Info
GL Coverage Group


Data Capture Folder
eApp Layout
Coverage Group eApp
Renderer = eApp Hierarchy Layout Class (Panel)
eApp Layout Section
eApp Name = GL Policy
Target eApp = GL Policy
eApp Name = GL Location
Target eApp = GL Location
eApp Name = GL Classification
Target eApp = GL Classification
eApplication
eApp Name = GL Policy
Renderer = eApp Content Class (Panel)
eApp Name = GL Location
Renderer = eApp Content Class (Panel)
eApp Section
Section Name = GL Coverage Selection
Render Type = Panel
Section Name = GL Location
Render Type = Panel
eApp Screen
Screen Name = GL Coverage Selection
Section Name = GL Location
eApp Column Info

Products Completed Operations Coverage



Premises Operations Coverage



8.4 eApp Renderer

The eApp renderer controls the look and feel of your eApplication. Depending the renderer you select, it will interpret the objects differently. Risk-based and coverage-based products required different rendering functionality and need to use the renderer appropriate for their architecture.

Provided Rendering Classes

The rendering classes determine the options available in the various `Renderer` and `Render Type` attributes. The following classes are provided with FS-PRO:

The eAppLayout class	This class creates the product-level navigation tabs for the main `eApp Layout` object. Note that at product level the navigation features aren't configurable.
The eAppContent class	This class is used exclusively in the eApplication object's `Renderer` attribute and controls the appearance of sections and screens at this level. There are several options available for the `Renderer` attribute:

eAppHierarchyLayout	com.camilion.pa.core.eapp.renderer.EappHierarchy This renderer is not supported. Do not use.
eAppLayout	com.camilion.pa.core.eapp.renderer.EappLayout This renderer is not supported. Do not use.
eAppTabularHierarchyLayout	com.camilion.as.eapp.lightproduct.renderer.EappHierarchy This renderer is not supported. Do not use.
eAppTabularLayout	com.camilion.as.eapp.lightproduct.renderer.EappLayout This renderer is not supported. Do not use.
Multi-Level Risk Renderer	com.camilion.renderer.multilevelrisk.eappnav.view.EappLayout This renderer is not supported. Do not use.
PremiumAudit	com.camilion.custom.renderer.premiumaudit.PremiumAuditEapp This renderer is not supported. Do not use.
Standard UI5 Renderer	pqm.eapp.splitapp This renderer is for coverage-based product layouts.
OData UI5 Renderer	fs.pro.eapp This renderer is for risk-based product layouts.

Data List Base

The `Data List Base` is a system template. It is used to register your custom render classes, should you choose to create and use them. The render classes provided by SAP are already registered in this product.

The render classes and render types registered in this reference product appear in the dropdown lists of the `Renderer` and `Render Type` attributes.

The `Renderer` attribute references the `DLRendererSource` component, for the list of rendering classes available in the environment. The `DLRenderTypeSource` component contains the values referenced by the `Render Type` attribute (such as Panel and Tabs).

In the *Studio Tree*, the `Data List Base` (1.0) template is located at: *System Repository* > *System Group* > *Templates* > *Data List Base*

9 Risks

Insurance products provide coverage for individuals or assets. These are defined as risks and need to be configured within a product.

Context

You define the risks for each line of business (LoB). Each policy can have multiple risks. For example, you might want to add vehicles or drivers as risk types for a Commercial Auto product.

> **i Note**
>
> Risks cannot be shared between LoBs.

Procedure

1. Configure the risks within the main axis data model (including the `Application Data Model Product`).
   a. Configure the data definition for each risk.

      For example, LOCATION and CLASSIFICATION tables.
   b. Add a `DESC_TX` attribute column to the data definition table record.

      This attribute is utilized for storing the description of the risk and displaying it in runtime.
   c. Assemble the primary risk under the POLICY_QUOTE table within the main axis data model (including the `Application Data Model Product`).

      An example of a primary risk would be LOCATION.
   d. If needed, assemble any sub-risks under the parent risk in the data model.

      An example of a sub-risk would be CLASSIFICATION.
2. Configure the line of business (LoB) extension data definitions for the risks.
   a. Configure the data definition table for each risk.

      For example, GL_LOCATION and GL_CLASSIFICATION.

      Ensure that the `REF_NAME` attribute in the table record is specified and references the `REF_NAME` attribute in the corresponding main axis table.
   b. Within the extension data model, assemble the extension data definitions.
   c. Assemble the extension data model within the product in the `Data Model` folder.
3. Configure the hierarchy of a LoB.
   a. Ensure that the `Risk Hierarchy` component is assembled under the coverage group for each LoB.

b. Add the policy-level hierarchy record (POLICY_QUOTE) within the `Risk Hierarchy` component.
c. Specify the policy details.

   These include defining the `Risk_Level`, `Risk_Name`, `DataDef_Name` and `Display_Name` attributes.

   You will also need to define the following rules, if you have multiple ratebooks: the `Application_Context_Date_Rule`, `Reference_Object_Version_Rule` (ratebook rule), and the `State_Code_Rule`.
d. Add the risk-level record within `Risk Hierarchy` component.
e. Specify the risk details.

   These include defining the `Risk_Level` and `Risk_Name`, `DataDef_Name`, `Display_Name` and `Parent_Risk` attributes.

   You will also need to define the following rules, if you have multiple ratebooks: the `Application_Context_Date_Rule`, `Reference_Object_Version_Rule` (ratebook rule), and the `State_Code_Rule`.

4. Configure the static grid views for each risk.
   a. Within the `eApp Layout Section` object, create a record for each risk.
   b. Specify a static risk grid view for each `Layout Section` record, for the display and/or management of multiple risks.
   c. Specify the `Target eApp` attribute for each record.

      When the `NAVIGATION_TYPE` attribute is set to `static` and the `STATIC_VIEW_NAME` attribute is set to `Risk`, this is the eApplication that will be rendered when selecting or adding a record in the risk grid.

      It will display the sections and screens that have been configured within the eApplication.
   d. Specify the columns and filters that will be available in the static risk grid views via the OData annotations within the data definitions for each risk.

Related Information

Risk Hierarchy Objects [page 246]
Configuring the Grid for a Risk [page 248]
Adding the Validation Status Column [page 249]
Creating the Description field in the Risk Data Definition [page 250]
Setting the OData Annotations for Elements [page 250]

9.1 Risk Hierarchy Objects

The `Risk Hierarchy` component is located at `<product_name>` > *Coverage Groups* > `<product_name>` > *Identification* > *Risk Hierarchy*.

The component has the following attributes:

`Risk_Level`	Specifies the level of the risk in the hierarchy. Policy is 0, top level risks are 1, sub-risks are 2, 3, etc. This is a required attribute.
`Risk_Id_Column_Name`	This attribute is not utilized by the risk hierarchy.
`Risk_Name`	Enter the table name specified in the data definition for the hierarchy record. This is a required attribute.
`Display_Name`	This value will be displayed in the *Level* column within the following objects: • *Forms* grid • *Ratebooks* grid • *Referrals* grid • *Error* grid This value will also be displayed as the associated risk name in the risk grid.
`DataDef_Name`	Enter the table name specified in the data definition for the hierarchy record. This is a required attribute.
`Parent_Risk`	Enter the object name (See the `Risk_Name` attribute) for the parent object associated with this risk data value row. This value should be empty for the policy-level risk (where `Risk_Level` is 0).
`Primary_Option_Eapp`	This attribute is not utilized by the risk hierarchy.
`WA_Eapp`	This attribute is not utilized by the risk hierarchy.
`Primary_RISK`	This attribute is not utilized by the risk hierarchy.
`EXTENDED_RISK`	This attribute is not utilized by the risk hierarchy.
`EXTENDED_PARENT`	This attribute is not utilized by the risk hierarchy.
`DataDef_Name_DD`	This attribute is not utilized by the risk hierarchy.
`State_Code_Rule`	A decision table rule or a script rule. It is meant to return a string value that represents the state/province code. This isn't utilitized by the risk hierarchy, but may be required for multiple-ratebook configurations.
`Application_Context_Date`	A decision table rule or a script rule. It is meant to return a string value that represents the date that is set when a risk is added.
`Reference_Object_Version_Rule`	A decision table rule or a script rule. The rule that is executed to lookup the ratebook.

`Inherit_Parent_Reference_Indicator`	Determines if the ratebook version for that risk will be inherited from its parent.
`Addl_Insured_Table`	This attribute is not utilized by the risk hierarchy.
`Addl_Interest_Table`	This attribute is not utilized by the risk hierarchy.
`SYSATTR_TABLE_NAME`	This attribute is not utilized by the risk hierarchy.
`ExtensionTables`	This attribute is not utilized by the risk hierarchy.
`ICON`	Specifies the icon that will be displayed to the left of the risk details within the *Related To* popup menu for the *Level* column in grids in the eApplication. The format of this attribute must be as follows: `<font_group_name>//:<character_name>` This is an optional attribute.

9.2 Configuring the Grid for a Risk

After having created a risk, you need to configure the OData annotations for the risk grid.

Prerequisites

You must have already added a `DESC_TX` attribute column to the data definition table record.

Procedure

1. Open the risk data definition for the main axis data model.
2. Navigate to the `Table` component.
3. Select the *Values* tab.
4. Select the existing table record in the risk data definition.
5. Set an OData annotation for the SAP Label. This will be the name that will be displayed in the *Level* column within the `AppErrors` grid. For example, Classification or Location.
   a. In the `ODATA_ANNOTATIONS` attribute column, enter `sap:label={0}`
   b. In the `ODATA_ANNOTATION_VALUES` attribute column, enter the translatable label.
6. Set the OData annotation for the pageability. This setting determines the number of records loaded in the grid. Setting this value to true will specify that records are only loaded as needed during scrolling.
   a. In the `ODATA_ANNOTATIONS` attribute column, enter `sap:pageable=true`
7. Save your changes.
8. Navigate to the `Column` component.

9. Select the *Values* tab.
10. Select the existing column record in the risk data definition that will be either visible or filterable in the risk grid.
11. Set the OData annotation for the visibility of the column in the risk grid.
    a. In the `ODATA_ANNOTATIONS` attribute column, enter `sap:visible=true`
12. Set the OData annotation for the filter. This setting determines if the filter option will be enabled for this column in the risk grid.
    a. In the `ODATA_ANNOTATIONS` attribute column, enter `sap:filterable=true`
13. Set an OData annotation for the SAP Label. This will be the name that will be displayed for the grid column and filter within the risk grid.
    a. In the `ODATA_ANNOTATIONS` attribute column, enter `sap:label={0}`
    b. In the `ODATA_ANNOTATION_VALUES` attribute column, enter the translatable label.
14. Save your changes.

9.3 Adding the Validation Status Column

An column needs to be added to the data grid which will indicate whether there are validation issues in the grid or the detail view that need to be addressed.

Context

When the application or screen is validated, each data grid record will be flagged as Valid with a green identifier or Invalid with a red identifier.

Procedure

1. Go to the data definition for the data grid table.
2. Select the `VALIDATE_STATUS` column.
3. Configure the OData annotation to display the `VALIDATE_STATUS` column in the grid.
4. Configure the OData annotation to enable the `VALIDATE_STATUS` column in the filter bar.
5. Save your changes.

9.4 Creating the Description field in the Risk Data Definition

As part of the risk creation, you need to configure a column that will store the concatenated description for the risk grid. For example, the address, city, state and zip code for a Location risk.

Procedure

1. Open the risk data definition for the main axis data model.
2. Navigate to the `Column` component.
3. Select the *Values* tab.
4. Add a record
5. Set the `DESCRIPTION` attribute value to **`DESC_TX`**.
6. Set the `COLUMN_NAME` attribute value to **`DESC_TX`**.
7. Set the `DATA_TYPE` attribute value to **`TEXT`**.
8. Set the `DATA_LENGTH` attribute value to **`250`**.
9. Save your changes.

9.5 Setting the OData Annotations for Elements

You will need to set OData annotations for the following elements:

- Error Grid
- Risk Grid
- Association between main axis tables

There are two levels of annotations: one for columns and one for tables. The annotations determine the behavior of the grid, as well as the columns and tables that are available.

The OData annotations are already configured in the out-of-the-box products, but the following information is provided for your reference if you want to change the default configurations:

Error Grid

For the `ODATA_ANNOTATIONS` attribute of the `Column` component within the `SCREEN VALIDATION ERROR` data definition:

`sap:visible=true`	This setting determines the visibility of the column in the error grid.

`sap:filterable=true` This setting determines if the filter option will be enabled for this column in the error grid.

`sap:label={0}` This setting determines the name that will be displayed for the grid column and filter within the error grid.

For the `ODATA_ANNOTATIONS` attribute of the `Table` component within the `SCREEN VALIDATION ERROR` data definition:

`sap:pageable=true` This setting determines the number of records loaded in the grid. Setting this value to true will specify that records are only loaded as needed during scrolling.

For the `ODATA_ANNOTATIONS` attribute of the `Table` component within the data definition for the risk (`CLASSIFICATION` and `LOCATION`):

`sap:label={0}` This will be the name that will be displayed in the *Level* column within the `AppErrors` grid. For example, Classification or Location.

For the `ODATA_ANNOTATIONS` attribute of the `Table` component within the `POLICY_QUOTE` data definition:

`sap:label={0}` This will be the name that will be displayed in the *Level* column within the `AppErrors` grid. For example, Policy.

> **i Note**
>
> The `ODATA_ANNOTATIONS_VALUES` attribute is the container for the translatable name that the SAP Label within the `ODATA_ANNOTATIONS` attribute references.

Risk Grid

For the `ODATA_ANNOTATIONS` attribute of the `Table` component within the risk data definition for the main axis data model:

`sap:label={0}` This setting determines the name that will be displayed for the *Level* column within the `AppErrors` grid.

> **i Note**
>
> The `ODATA_ANNOTATIONS_VALUES` attribute is the container for the translatable name that the SAP Label within the `ODATA_ANNOTATIONS` attribute references.

`sap:pageable=true` This setting determines the number of records loaded in the grid. Setting this value to true will specify that records are only loaded as needed during scrolling.

For the `ODATA_ANNOTATIONS` attribute of the `Column` component within the risk data definition for the main axis data model:

`sap:visible=true` This setting determines the visibility of the column in the risk grid.

`sap:filterable=true` This setting determines if the filter option will be enabled for this column in the risk grid.

sap:label={0}	This setting determines the name that will be displayed for the grid column and filter within the risk grid.

Association between Main Axis Tables

Creates an association between main axis tables (such as ACCOUNT and POLICY_TXN), for navigation and querying purposes.

For the ODATA_USAGE attribute of the Column component within the POLICY_TXN data definition, for the ACCOUNT and POLICY records:

[2] - NavigationProperty

For the ODATA_ASSOCIATION attribute of the Column component within the POLICY_TXN data definition, for the ACCOUNT record:

Dependent=ACCOUNT,

Multiplicity=One,

PropertyRef=ACCT_ID,

DepPropertyRef=PK_ID

For the ODATA_ASSOCIATION attribute of the Column component within the POLICY_TXN data definition, for the POLICY record:

Dependent=POLICY,

Multiplicity=One,

PropertyRef=POLICY _ID,

DepPropertyRef=PK_ID

For the ODATA_USAGE attribute of the Table component within the POLICY_QUOTE data definition, for the SCREEN_VALIDATION_ERROR_SET record:

[2] - NavigationProperty

For the ODATA_ASSOCIATION attribute of the Table component within the POLICY_TXN data definition, for the SCREEN_VALIDATION_ERROR_SET record:

Dependent=SCREEN_VALIDATION_ERROR,

Multiplicity=Many,

PropertyRef=PK_ID

DepPropertyRef=MASTER_PK_ID

10 Creating a Product Using the `Sample Auto Product` Templates

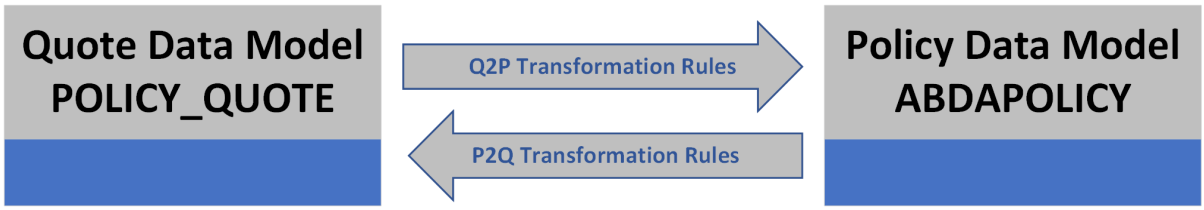
There are many ways to construct a product in the Product Modeler. This section demonstrates the best practices for using the sample Auto product (`SAP S4 PR Motor Vehicle Insurance`) templates.

How are these templates different?

In FS-QUO, the eApp navigation model and the user interface screens are tightly coupled with the underlying data model. As part of our integration solution with S/4 HANA we have created a separate UI data model (referred to as the Quote data model) from the FS-PM data model (sometimes referred to as the ABDA data model). This allows the configuration to support the UI requirements directly without a need to follow the prescribed structure presented by the FS-PM Policy data model. The eApps will utilize this Quote data model, as well as newly introduced sets of rules that will aid in the translation from the UI data model to the hierarchical ABDA data model

There are number of quotation services, rating, underwriting (Simulate and Submit), and policy issuance which rely on data in the FS-PM Policy data model. Therefore, it is necessary to transform the data from the Quote data model to the FS-PM Policy data model using script rules before calling those quotation services – these script rules used for transforming the data from the Quote data model to Policy data model will be referred to as `Q2P` (`Quote to Policy`) rules in this document.

When a quotation is calculated by calling FS-PM, the premium (i.e the ABDA data) is returned from FS-PM. This data needs to be transformed back to the Quote data model. This reverse transformation, which occurs when FS-PM is called for rating, uses another set of script rules that will be referred to as `P2Q` (`Policy to Quote`) rules in this document.



Step 1: Design the Quote Application Screens

Use standard architecting techniques to plan out an appropriate, simplified eApp structure.

The key is to be able to capture all the required information in as simple a manner as required for the typical user of the system.

Step 2: Design the UI Data Model

Based upon the requirements of the screen design, design a data model that can implement the screens.

The Product Modeler allows much more flexibility in creating individual and sharable (for common tables) data models based on the product groups. To support the UI flexibility and to allow for reuse, a separate base data model is created with the tables that are reused across all LOBs and one additional UI `<LOB>` data model is created for each LOB.

Common S4 Data Definitions

Tables that are reused across all LOBs have been implemented in the SAP PnC data model and the SAP LnA data model. These data models should be utilized as a basis for creating new data models.

Tables included:

- UI_QUOTE_EXT
- UI_QUOTE_EXT_POLHLDR
- UI_QUOTE_EXT_COIN_HEADER
- UI_QUOTE_EXT_COIN_SHARE (Child of UI_QUOTE_EXT_COIN_HEADER)

LOB- Specific Data Model

One data model should be created for each LOB by extending either SAP PnC data model or SAP LnA data model.

Column Naming Convention

Data needs to be transferred between the Quotation Model and the Policy Model.

Mapping the columns one by one in script rules for copying the data from one model to the other is not only tedious, but also error prone and poses maintenance problem. To avoid this, a standard naming convention and a function stem is made available to facilitate the copy of the data from one to the other at the entity level.

This naming convention is accomplished by matching column names used in the ABDA model when creating the UI tables, but with a specified 3 letter prefix.

> **i Note**
>
> The UI data model typically combines data from multiple policy tables to provide flexibility in the UI. For example, the columns POLPR_ABDAPREM.PAYFRQ_CD and ABDAPRMPAYR.COLLTYPE_CD have been added to the UI_QUOTE_EXT table along with other columns from ABDAPOLICY and ABDAPOLPR tables to allow for capture of such details on the main screen.

The 3-letter prefix consists of a 2 letter `Entity Type` code plus a 3rd letter `Transfer Direction` code as itemized below:

Entity Type	2 Letter Prefix
Policy	PL
Contract	CN
Coverage	CV
Sub Coverage	SC
Policyholder	PH

Entity Type	2 Letter Prefix
Premium Payer	PP
Premium	PR
Surcharge and Discount	SD
Insured Object	IO
Insured Person	IP
Benefit	BN
Fund	FD
Coinsurance Header	CH
Coinsurance Share Segment	CS
Coinsurance Commission	CC
Clause	CL
Limit	LM
Deductible	DD

Direction	3rd Letter prefix
From Quote data model to Policy data model	P
From Policy data model to Quote data model	Q
Both directions For example: Benefit Amount, depending on the calculation method (SI vs. Premium) SI based: the Benefit Amount is captured and send to Policy Model for premium calculation Premium Based: the Benefit Amount is calculated as part of the premium calculation and needs to be transferred back to Quote data model to display it on the screen.	B

Example: If a column is named UI_QUOTE_EXT.PRP_PAYFRQ_CD, the first two letters (PR) indicates the target column is in the Premium (ABDAPREM) table and the 3rd letter (P) indicates data needs to be copied from the Quote data model to the Policy data model.

PRO_ID Column

A standard column, *PRO_ID*, is created in all Policy data model tables to identify the relevant entities based on the `SAP PRO Id` component with the `Identification` folder within the product object. The *PRO_ID* column will be used in rules as an identifier instead of the FS-PM Template Id.

***SEQ_NO* and *PRO_SRC_ID* Columns**

In most cases, only a single instance of a policy model entity is required under a given parent entity.

For example, there can be only one instance of Third Party Liability (TPL) Coverage allowed under a given Auto Contract.

However, if multiple instances of a given policy model entity are allowed under a given parent, then *SEQ_NO* column value is initialized in the Quote data model. The value of the *SEQ_NO* column is transferred into the *PRO_SRC_ID* column in the appropriate Policy data model entity record to track the record between the two data models.

Example: An auto product UI may allow for the capture of multiple vehicles for a single policy. This information, however may be required to be set up in the policy data model to be a single contract per vehicle. In this case, the *PRO_SRC_ID* is used to track the vehicle in the specified contract.

This would allow for updating the policy data model table records in subsequent Q2P transformation rather than deleting and creating the records in the policy data model all over again.

VEHICLE.PK_ID	VEHICLE.SEQ_NO	ABDAPOLPR.PK_ID	ABDAPOLPR.PRO_SRC_ID
123	123		123
312	312		312

Step 3: Build the Product Objects

Build the product making use of the S/4i object templates.

SAP PRO Id Component

Each object currently has an SAP `<object type>` `Id` component that contains FS-PM related information. Introduced in the S/4 Product Modeling, is the SAP `PRO` `Id` component within each object.

This allows for rules to be related to an FS-PRO-specific identifier, rather than to the FS-PM `Template Id`. As the best practice for using the `Template Id` within FS-PRO is to override it for each object within the marketable object level, any rules needing to identify an object based upon this would also need to be overridden **or** a new object would need to be created. The introduction of the SAP `PRO` `Id` component allows for greater reuse of configured objects, as this component will not need to be overridden and thus, rules based upon this component will also not require overriding.

Defaulting

- Premium Payer is defaulted as Policyholder in the initialization rules in the relevant objects.

- Commission data is defaulted based on the Agent Business Partner captured in the *General Information* screen
- See Working with Initialization Rules [page 333] for defaulting Policy data model table data.

Transformation Folder

Each object hierarchy contains a `Transformation Folder` introduced within the SAP `Insurance UWA Product Template`.

This folder will contain up to 3 rules components:

a) SAP Initialization Rules

Initialization rules are found in all objects.

They are called to create the Policy data model data records

Typically, the rules contained within do not need to be overridden, with exception of the `initQuoteData` rule (in the Product Hierarchy)

The rules contained within are called at first by the Initialization Rule found on the QuoteEApp, but are also called from the transformation rules at the beginning of the rating process.

Call the `setDefaultValues` stem to capture the information from the `SAP Column Default Values` components.

a) SAP Transformation Rules

Transformation rules are found in all objects.

They synchronize Policy data model records. Rather than deleting and creating all records every time when data needs to be transferred from the Quote data model to the Policy data model, standard synchronize methods are implemented in all main axis level transformation rules to update the Policy data model using *PRO_ID* or *PRO_SRC_ID* to identify the matching records in the Policy data model.

Overrides are typical for these rules to comply with the differences that come from the specific UI configured for a particular product.

These rules are called as a precursor to any rating within FS-QUO. When rating is called from FS-PM, these rules are skipped, as there is no need for transformation.

c) SAP Utility Rules

These rules are only found in the Product Hierarchy.

The rules should **not** need to be overridden.

The two primary Utility rules are:

`get<ObjectType>Path`

These rules are used to create the path for the current object (from within the product's structure)

They use the *ABDA*.PRODUCT_BASE_ID* (populated with the `Template Id`) rather than the *Name* of the component, to allow for changing of object names (whether through inheritance or for other reasons).

The *PRO_ID* was not used for potential backwards compatibility reasons for products that we created without the `PRO_ID` component.

`set<Object Type>TemplateId`	These rules are used assign the `Template Id` from the object's SAP `<Object Type> Id` component into the appropriate *ABDA*.PRODUCT_BASE_ID* field. The rules make use of the *PRO_ID* to do this.

Questionnaire Model / View Configuration	LoB specific Questionnaire Model/View Objects are configured as stand-alone objects These objects are then assembled into the Contract Object hierarchy.
eApp Configuration	eApps are to be configured in the Product hierarchy at the Marketable Product, using the Questionnaire Views assembled into the Contracts. eApp configuration in the Sales Product allows for greater flexibility for the configuration of multi contract products.
Rule Stems	New rule stem functions made available:

Stem Name	Function Name	Description
`DataObjectAccessStem`	`convertSingleColumnCSVToDataObjec`	Returns a data table object with a single column. Each value in the CSV is listed as a row.
`DataObjectAccessStem`	`convertRawDataToDataObject`	Returns a data table object with the specified columns and values. Each row data should be separated by comma and column values should be separated by semicolon. For example: `'1;Yes,2;No','ID,DESC'`

STEP 4: IFBC Configuration

Follow the normal standards and practices when configuring the IFBC components for import into FS-PM.

The Auto ABDA data model uses the following tables which are specifice to the SAP Policy Management Add-On for Auto Insurance:

- AMDAPOLPR
- AMDACOV
- AMDANMDRVR

11 Creating a Product Using the `Sample Life Product` Templates

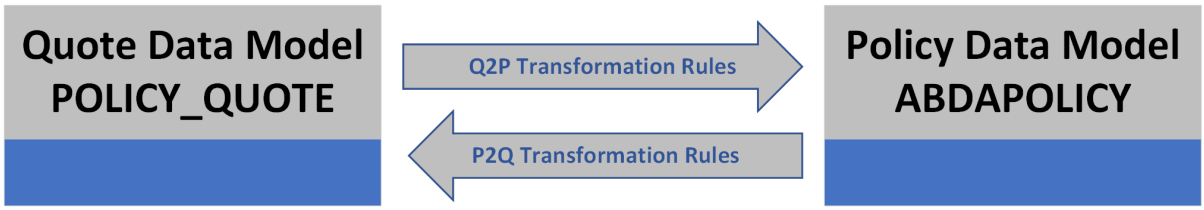
There are many ways to construct a product in the Product Modeler. This section demonstrates the best practices for using the sample Life product (`SAP S4 PR Life Capital`) templates.

How are these templates different?

In FS-QUO, the eApp navigation model and the user interface screens are tightly coupled with the underlying data model. As part of our integration solution with S/4 HANA we have created a separate UI data model (referred to as the Quote data model) from the FS-PM data model (sometimes referred to as the ABDA data model). This allows the configuration to support the UI requirements directly without a need to follow the prescribed structure presented by the FS-PM Policy data model. The eApps will utilize this Quote data model, as well as newly introduced sets of rules that will aid in the translation from the UI data model to the hierarchical ABDA data model

There are number of quotation services, rating, underwriting (Simulate and Submit), and policy issuance which rely on data in the FS-PM Policy data model. Therefore, it is necessary to transform the data from the Quote data model to the FS-PM Policy data model using script rules before calling those quotation services – these script rules used for transforming the data from the Quote data model to Policy data model will be referred to as `Q2P` (`Quote to Policy`) rules in this document.

When a quotation is calculated by calling FS-PM, the premium (i.e the ABDA data) is returned from FS-PM. This data needs to be transformed back to the Quote data model. This reverse transformation, which occurs when FS-PM is called for rating, uses another set of script rules that will be referred to as `P2Q` (`Policy to Quote`) rules in this document.



Step 1: Design the Quote Application Screens

Use standard architecting techniques to plan out an appropriate, simplified eApp structure.

The key is to be able to capture all the required information in as simple a manner as required for the typical user of the system.

Step 2: Design the UI Data Model

Based upon the requirements of the screen design, design a data model that can implement the screens.

The Product Modeler allows much more flexibility in creating individual and sharable (for common tables) data models based on the product groups. To support the UI flexibility and to allow for reuse, a separate base data model is created with the tables that are reused across all LOBs and one additional UI `<LOB>` data model is created for each LOB.

Common S4 Data Definitions

Tables that are reused across all LOBs have been implemented in the SAP PnC data model and the SAP LnA data model. These data models should be utilized as a basis for creating new data models.

Tables included:

- UI_QUOTE_EXT
- UI_QUOTE_EXT_POLHLDR
- UI_QUOTE_EXT_COIN_HEADER
- UI_QUOTE_EXT_COIN_SHARE (Child of UI_QUOTE_EXT_COIN_HEADER)

LOB- Specific Data Model

One data model should be created for each LOB by extending either SAP PnC data model or SAP LnA data model.

Column Naming Convention

Data needs to be transferred between the Quotation Model and the Policy Model.

Mapping the columns one by one in script rules for copying the data from one model to the other is not only tedious, but also error prone and poses maintenance problem. To avoid this, a standard naming convention and a function stem is made available to facilitate the copy of the data from one to the other at the entity level.

This naming convention is accomplished by matching column names used in the ABDA model when creating the UI tables, but with a specified 3 letter prefix.

> **i Note**
>
> The UI data model typically combines data from multiple policy tables to provide flexibility in the UI. For example, the columns POLPR_ABDAPREM.PAYFRQ_CD and ABDAPRMPAYR.COLLTYPE_CD have been added to the UI_QUOTE_EXT table along with other columns from ABDAPOLICY and ABDAPOLPR tables to allow for capture of such details on the main screen.

The 3-letter prefix consists of a 2 letter `Entity Type` code plus a 3rd letter `Transfer Direction` code as itemized below:

Entity Type	2 Letter Prefix
Policy	PL
Contract	CN
Coverage	CV
Sub Coverage	SC
Policyholder	PH

Entity Type	2 Letter Prefix
Premium Payer	PP
Premium	PR
Surcharge and Discount	SD
Insured Object	IO
Insured Person	IP
Benefit	BN
Fund	FD
Coinsurance Header	CH
Coinsurance Share Segment	CS
Coinsurance Commission	CC
Clause	CL
Limit	LM
Deductible	DD

Direction	3rd Letter prefix
From Quote data model to Policy data model	P
From Policy data model to Quote data model	Q
Both directions For example: Benefit Amount, depending on the calculation method (SI vs. Premium) SI based: the Benefit Amount is captured and send to Policy Model for premium calculation Premium Based: the Benefit Amount is calculated as part of the premium calculation and needs to be transferred back to Quote data model to display it on the screen.	B

Example: If a column is named UI_QUOTE_EXT.PRP_PAYFRQ_CD, the first two letters (PR) indicates the target column is in the Premium (ABDAPREM) table and the 3rd letter (P) indicates data needs to be copied from the Quote data model to the Policy data model.

PRO_ID Column

A standard column, *PRO_ID*, is created in all Policy data model tables to identify the relevant entities based on the `SAP PRO Id` component with the `Identification` folder within the product object. The *PRO_ID* column will be used in rules as an identifier instead of the FS-PM Template Id.

Step 3: Build the Product Objects

Build the product making use of the S/4i object templates.

SAP PRO Id Component

Each object currently has an `SAP <object type> Id` component that contains FS-PM related information. Introduced in the S/4 Product Modeling, is the `SAP PRO Id` component within each object.

This allows for rules to be related to an FS-PRO-specific identifier, rather than to the FS-PM `Template Id`. As the best practice for using the `Template Id` within FS-PRO is to override it for each object within the marketable object level, any rules needing to identify an object based upon this would also need to be overridden **or** a new object would need to be created. The introduction of the `SAP PRO Id` component allows for greater reuse of configured objects, as this component will not need to be overridden and thus, rules based upon this component will also not require overriding.

Defaulting

- Premium Payer is defaulted as Policyholder in the initialization rules in the relevant objects.
- Commission data is defaulted based on the Agent Business Partner captured in the *General Information* screen
- See Working with Initialization Rules [page 333] for defaulting Policy data model table data.

Transformation Folder

Each object hierarchy contains a `Transformation Folder` introduced within the `SAP Insurance UWA Product Template`.

This folder will contain up to 3 rules components:

a) SAP Initialization Rules

Initialization rules are found in all objects.

They are called to create the Policy data model data records

Typically, the rules contained within do not need to be overridden, with exception of the `initQuoteData` rule (in the Product Hierarchy)

The rules contained within are called at first by the Initialization Rule found on the QuoteEApp, but are also called from the transformation rules at the beginning of the rating process.

Call the `setDefaultValues` stem to capture the information from the `SAP Column Default Values` components.

b) SAP Transformation Rules

Transformation rules are found in all objects.

They synchronize Policy data model records. Rather than deleting and creating all records every time when data needs to be transferred from the Quote data model to the Policy data model, standard synchronize

methods are implemented in all main axis level transformation rules to update the Policy data model using *PRO_ID* or *PRO_SRC_ID* to identify the matching records in the Policy data model.

Overrides are typical for these rules to comply with the differences that come from the specific UI configured for a particular product.

These rules are called as a precursor to any rating within FS-QUO. When rating is called from FS-PM, these rules are skipped, as there is no need for transformation.

c) SAP Utility Rules

These rules are only found in the Product Hierarchy.

The rules should **not** need to be overridden.

The two primary Utility rules are:

`get<ObjectType>Path`	These rules are used to create the path for the current object (from within the product's structure) They use the *ABDA*.PRODUCT_BASE_ID* (populated with the `Template Id`) rather than the *Name* of the component, to allow for changing of object names (whether through inheritance or for other reasons). The *PRO_ID* was not used for potential backwards compatibility reasons for products that we created without the `PRO_ID` component.
`set<Object Type>TemplateId`	These rules are used assign the `Template Id` from the object's SAP `<Object Type> Id` component into the appropriate *ABDA*.PRODUCT_BASE_ID* field. The rules make use of the *PRO_ID* to do this.

Questionnaire Model / View Configuration

LoB specific Questionnaire Model/View Objects are configured as stand-alone objects

These objects are then assembled into the Contract Object hierarchy.

eApp Configuration

eApps are to be configured in the Product hierarchy at the Marketable Product, using the Questionnaire Views assembled into the Contracts.

eApp configuration in the Sales Product allows for greater flexibility for the configuration of multi contract products.

Rule Stems

New rule stem functions made available:

Stem Name	Function Name	Description
`DataObjectAccessStem`	`convertSingleColumnCSVToDataObjec`	Returns a data table object with a single column. Each value in the CSV is listed as a row.

Stem Name	Function Name	Description
`DataObjectAccessStem`	`convertRawDataToDataObject`	Returns a data table object with the specified columns and values. Each row data should be separated by comma and column values should be separated by semicolon. For example: `'1;Yes,2;No','ID,DESC'`

STEP 4: IFBC Configuration

Follow the normal standards and practices when configuring the IFBC components for import into FS-PM.

The Life ABDA data model uses the following tables which are specific to the SAP Policy Management Add-On for Life Insurance:

- ALDAPOLPR
- ALDACOV

12 Change Business

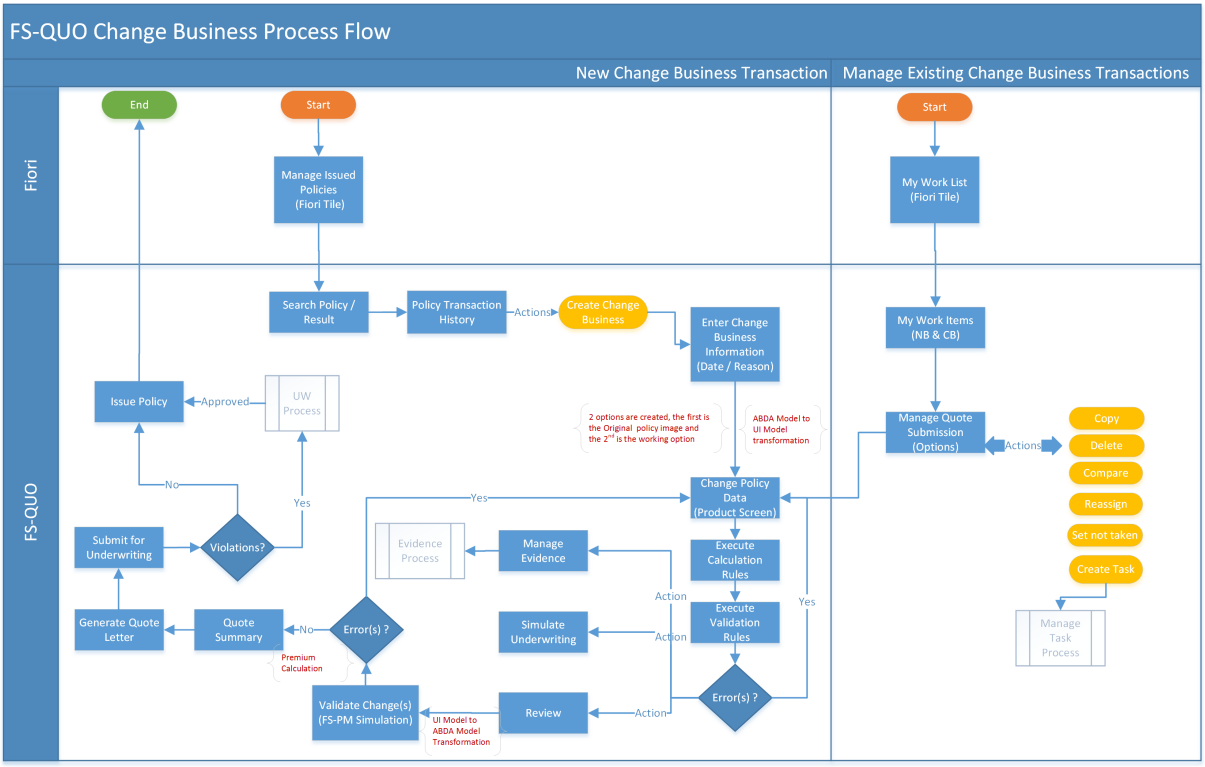
12.1 Configuring a Product to Allow Change Business Transactions

You can create a product that allows the creation of New Business and also allow Change Business transactions.

This functionality is exposed in FS-QUO Fiori Apps via a tile called *Manage Issued Policies*.

Change Business Process Flow

The following diagram indicates the process flow for a product that has been configured for the Change Business workflow:



Restrictions and Limitations

The Change Business workflow is available only for solutions that include an integration with FS-PM version 1909 and later. For more information, see the Integration Guide for Coverage-based Insurance Solutions

Support of change business is for coverage-based products only. This workflow is not available for risk-based products.

The following FS-QUO features are not supported in the Change Business workflow:

- Out of sequence changes (Undo/Redo)
- Cash Before Cover (CBC)
- Multi-currency
- Counteroffers

Creating a Product

You can quickly create a product that employs a Change Business workflow by using the sample Auto product (`SAP S4 PR Motor Vehicle Insurance`) templates. For guidance about creating a product, see Creating a Product Using the Sample Auto Product Templates [page 253].

> **i Note**
>
> In order to enable the Change Business functionality for a product, you need to set the `ChangeBusiness` attribute, under the *Extended Underwriting Application Configuration* > *Available Product* component.

Configuring RFCs

All of the RFC functions are located in *SAP S4 Global Reference Object* > *Configuration* > *Integration* > *RFC* > *Remote Function System*.

Functionality	Business Function Name	RFC Name
Search based on policy	SEARCH_POLICY	/PM0/ABT_SVC_POL_FIND
Search based on account (policy holder or representative)	SEARCH_ACCOUN_BANK	/PM0/ABB_BP_SEARCH_BANK_ES
	SEARCH_ACCOUNT_CCARD_DETAILS	/PM0/ ABB_BP_SEARCH_CCARD_DD_ES
	SEARCH_HELP	BAPI_HELPVALUES_GET
	SEARCH_ADR	/PM0/ABB_BP_SEARCH_ADR_Q_ES
	SEARCH_QUO_ADDRESSDATA	/PM0/ABB_QUO_ADDRESSDATA_GET
	SEARCH_ADR_DETAILS	/PM0/ ABB_BP_SEARCH_ADR_Q_ES_2

Functionality	Business Function Name	RFC Name
	SEARCH_ACCOUNT_ADR_DETAILS	/PM0/ABB_BP_SEARCH_ADR_DD_ES
Header details	READ_POLICY	/PM0/ABT_SVC_POL_READ
Transaction details	SEARCH_APPLICATION	/PM0/ABT_SVC_APPL_FIND

Configuration for Business Transaction (BTX) Execution

Every product will have business transactions (BTXs) that are possible on a policy configured in IFBC. A similar BTX configuration must be maintained for every product in FS-QUO.

1. Push the product to FS-PM.
2. Manually perform the BTX assignment in FS-PM's IFBC.
3. The same set of BTXs will need to be manually configured in the product in FS-QUO.

> **i Note**
>
> This process is not automated.

Business transactions are located in *SAP S4 PR Motor Vehicle Insurance* > *Configuration* > *Integration* > *BTX*.

There are three components in FS-QUO for the BTX Framework:

Available BTXs

This component is the FS-QUO-equivalent of the FS-PM assignment of business transactions in IFBC.

> **i Note**
>
> In IFBC, BTXs are assigned at different levels (policy, contract, coverage, etc). In FS-QUO, BTXs are all assigned at the product level.

BTX Configuration

This component allows users to specify the order of BTX execution, the operation that can be performed (example: update, delete, create) on entities, specify the table and the column set that are modifiable for the BTX

BTX Update Excluded Columns

This component allows users to specify the exclusion of columns in a BTX.

The excluded columns will not be sent to FS-PM during BTX execution.

For more information about working with the BTX Framework, see the Business Transaction (BTX) Framework

Related Information

Understanding the Change Business Workflow [page 268]
Business Transaction (BTX) Framework [page 271]
MVA Namespace [page 273]

12.1.1 Understanding the Change Business Workflow

This topic describes the workflow of a product that has been designed to include change business transactions. This functionality is exposed in the FS-QUO Fiori Apps via a tile called *Manage Issued Policies*.

> **i Note**
>
> Change Business functionality is demonstrated in products that were created using a non-hierarchical UI model. The `SAP S4 PR Motor Vehicle Insurance` templates are provided out of the box to help you get started. For more information, see Creating a Product Using the Sample Auto Product Templates

> **→ Remember**
>
> For Change Business functionality to be enabled in a product, you need to set the `ChangeBusiness` attribute, under the *Extended Underwriting Application Configuration* > *Available Product* component.

The Change Business workflow begins with the user searching for an policy that was issued in FS-PM, either via the policy number or account information.

Search based on policy	If the policy is found in the FS-PM system, the result will be displayed in the user's worklist.
Search based on account (policy holder or representative)	If the user searches for policies based on account information (the account must be a policy holder or a representative), all the policies belonging to the account will be retrieved and displayed in the worklist. A maximum of 100 results will be displayed on screen.

The user will select a policy from the worklist. Once the user selects the policy from the worklist, they then navigate to the next screen where the header details are displayed for the policy (this is a static screen).

For the displayed policy, a grid is displayed with details of all the transactions that were applied on the policy. If a Change Business transaction was created in FS-QUO, the submission number will be displayed in the grid.

> **i Note**
>
> If you have archived the submission numbers, it won't be displayed. The submission number must exist in the database.

A *Create Change Business* button will be available on the transaction grid. The button will be visible on the UI only under the following circumstances:.

- The product has been configured in FS-QUO with Change Business enabled
- There are no other transaction on the policy with a status of `in-process`

Otherwise, the button will be hidden on the UI.

When the user selects the *Create Change Business* button, a new screen is displayed where the user can enter the effective date of "change" to be applied and the reason for the change. The user then presses *Next* and confirms the information to proceed.

> **i Note**
>
> Once the information on the *General Info* screen is confirmed, no change of `Effective Date` is possible.

Once confirmed, a new submission is created in FS-QUO with two quote options:

- Option 1 is the baseline option
- Option 2 is the option on which the user will be making changes

> **! Restriction**
>
> If an open submission exists for a change then no new change business can be started.

An external change application is created in FS-PM, the data of the policy as on that effective date, is transformed and displayed in the FS-QUO UI / eApp. The change application is locked in FS-PM.

Next, the UI needs to be transformed from FS-PM's ABDA data model to FS-QUO's Quote data model via transformation rules. For example, the sample Auto product uses the `SAP Transformation Rules Component` -> `transformPolicyModelDatatoUIModel` rules.

To create a change external option in FS-PM, /PM0/ABT_SVC_CHGREQ_CRT RFM is called.

For the sample Auto product to demonstrate a consistent user experience between the New Business workflow and the Change Business workflow, a "partial universal change" approach has been implemented in the FS-QUO UI. This means that the eApp loaded for Change Business appears the same as New Business. Data is retrieved from FS-PM and the ABDA data model is transformed to the Quote data model.

Every product will have business transactions (BTXs) that are possible on a policy configured in IFBC. In FS-QUO for every product similar configuration has to be maintained for the product. For the sample Auto product, three BTXs are configured out of the box.

For a universal change approach, the UI for Change Business must be configured to ensure that for any changes made on the UI, there is a corresponding BTX that supports the change. If the UI that is used for New Business is implemented for Change Business and there are columns that are available (for example, insurable object) for which the BTX is not configured, then the columns must be made read-only on the UI. .

A new column `TXN_TYPE_CD` in the `POLICY_TXN` table has been introduced in the FS-QUO database. It is recommended that this flag be used to write rules for New Business vs Change Business.

When the user does **not** change the data on the eApp:

- If they press the *Save* icon, the premium will be recalculated and displayed
- If they press the *Review* button, an error or warning message will be displayed as no changes were made. The framework in FS-QUO identifies the changes made on the quote with the baseline quote option. If there are no differences found, then no BTXs will be derived and sent for simulation at review quote.
- If they press the *Simulate Underwriting* button, an underwriting decision will be given for the agent (the same as in the New Business workflow)

  > **i Note**
  >
  > Out of the box, the same underwriting rules are applicable for New Business and Change Business. You can choose to write different underwriting rules for Change Business (using the `TXN_TYPE_CD` flag).

  > **i Note**
  >
  > The only difference between the New Business and Change business underwriting workflow is that counteroffer is disabled for Change Business.

- If they upload an evidence, it will be uploaded at the policy level only. This restriction is for products created with the UI data model only.

When the user **does** change the data on the eApp:

- If they press the *Save* icon, the premium will be calculated and displayed
- If they press the *Simulate Underwriting* button, an underwriting decision will be given for the agent (the same as in the New Business workflow)

> **i Note**
>
> Out of the box, the same underwriting rules are applicable for New Business and Change Business. You can choose to write different underwriting rules for Change Business (using the `TXN_TYPE_CD` flag).

> **i Note**
>
> The only difference between the New Business and Change business underwriting workflow is that counteroffer is disabled for Change Business.

- If they upload an evidence, it will be uploaded at the policy level only. This restriction is for products created with the UI data model only.

If they press the *Review* button after a change has been made (for example, adds a coverage or changes the premium frequency), the following actions will occur:

1. FS-QUO will determine all the differences on the quotation compared to the baseline option (Option 1).
2. Once the delta is identified, FS-QUO calls an RFC (/PM0/ABT_SVC_CHGBTXPROPS_GET) and gets the information from FS-PM to know what are the columns that are modifiable for the BTX.
3. FS-QUO also looks for the IFBC configuration to check the fields that are modifiable.
4. Once the delta is identified, the BTX to be applied is identified, columns are identified for the BTX that are modifiable, and the actual BTX RFCs are called in FS-PM in "simulate mode". Only when the policy is being issued are all of the BTXs applied in "update mode" in FS-PM.

Copy Quote

When an option is copied for a quote option with transaction type `change business`, the new quote option is created with the data from the source option (that is, the option being copied from). The newly "copied" quote option will create a new external change option application in FS-PM during the copy process.

The principle is that all quote options for Change Business will have a corresponding external change option in FS-PM. The rest of the copy functionality is the same as for New Business

Compare Quote

The functionality remains the same as with the New Business workflow.

Delete Quote

The functionality remains the same as with the New Business workflow.

Generate Quote Letter

The functionality remains the same as with the New Business workflow.

Discard

The *Discard* button will be visible if there is an external change option open from FS-QUO when the user goes to a transactions screen within the *Manage Submission* screen. Once the user selects the submission and presses *Discard*, all quotes within the submission will be refused. Corresponding external change applications in FS-PM will also be refused.

12.1.2 Business Transaction (BTX) Framework

Every product will have business transactions (BTXs) that are possible on a policy configured in IFBC.

For every product in FS-QUO, a similar configuration has to be maintained for the product:

1. Push the product to FS-PM.
2. Manually perform the BTX assignment in FS-PM's IFBC.
3. The same set of BTXs will need to be manually configured in the product in FS-QUO.

> **i Note**
>
> This process is not automated.

Sample business transactions can be viewed in *SAP S4 PR Motor Vehicle Insurance* > *Configuration* > *Integration* > *BTX*.

> **→ Remember**
>
> For Change Business functionality to be enabled in a product, you need to set the `ChangeBusiness` attribute, under the *Extended Underwriting Application Configuration* > *Available Product* component.

There are three components in FS-QUO for the BTX Framework:

Available BTXs This component is the FS-QUO-equivalent of the FS-PM assignment of business transactions in IFBC.

The technical names and descriptions of the BTXs need to be configured in FS-QUO.

> **i Note**
>
> In IFBC, BTXs are assigned at different levels (policy, contract, coverage, etc). In FS-QUO, BTXs are all assigned at the product level. Also, if there is a change identified and there is no respective BTX configured, then an error will be displayed.

BTX Configuration	This component allows users to specify the order of BTX execution, the operation that can be performed on entities, specify the table and the column set that are modifiable for the BTX. Within the BTX Configuration for a product, the corresponding BTX operations need to be configured (such as, update, delete, create). Within the BTX operations, the corresponding BTX matching criteria and derived updates need to be configured. As an example, let's say that a BTX named Change Insured Object is configured within a BTX Configuration. And let's say that updates are made to the Insured Object columns via the eApp created with the sample Auto product: the Parking Area value is changed. Within the BTX operations, an update is only possible on the POLPR_ABDASUBJCT table. Since the Parking Area value is changeable on the UI, this column has to be specified under the BTX matching criteria. When the user changes the Parking Area value at Runtime, the BTX framework identifies the BTX operations (based on the BTX matching criteria) and the corresponding BTX to be executed. If you don't configure the Parking Area value in the BTX matching criteria and change the Parking Area value, then no business transaction can be determined and an error will be issued.
BTX Update Excluded Columns	This component allows users to specify the exclusion of columns per entity in a BTX. The excluded columns will not be sent to FS-PM during BTX execution. You can specify whether inserting or updating records is possible. For example, if the update record for the POLPR_ABDASUBJCT table is set to "0" and a user changes any of the insured object columns at runtime, then the system won't allow the change and an error will result. This configuration additionally allows users to define whether updates or inserts are possible on entities.

Transactions supported by the Change Business framework, with out-of-the-box configuration via the templates:

Change Payment Frequency	Create and Process Coverage	Edit Insured Object
Change Policyholder	Change Premium Payer	Execute Reversal
Change Surcharge/Discount	Change Clause	Change Main Due Date
Edit Limit	Change Deductible	Create and Process Contract

Transactions supported by the Change Business framework, with the appropriate manual configuration by customers:

Change Beneficiary	Change Discount Protection	Create and Process Coverage Package
Change Exchange Rate Data	Change Adjustment	End Adjustment
Exclude Coverage	Exclude Sub coverage	Change Adjustment
Change Creditor	Change Duration	Change Commission Participant
Change Benefit/Premium	Change CBC Amount Payer	

Unsupported transactions:

Product Change	Sales Product Change	Reverse Manual Rating
Execute Manual Rating	Process Premium Waiver	Annul Premium Waiver
Request CBC Amount	Determine CBC Category	

12.1.3 MVA Namespace

Any column in FS-PM that uses the `/MVA/` prefix must be mapped to their equivalent FS-PRO columns using the `Namespace Mapping` component. Normally, there is no namespace required for FS-PRO tables, but if a product in FS-PRO uses the FS-PM Auto Add On tables, an equivalent namespace configuration needs to be maintained for the runtime integration.

This is required to Push / create an application during the Review Quote process.

The component can be found in *SAP S4 Global Reference Object* > *Configuration* > *Integration* > *Namespace Mapping* and requires two attributes:.

QualifiedColumnName

Enter the column name from the FS-PM Auto Add On tables, if used.

NameSpace

For an auto product, the value for this attribute should be MVA.

Not all columns are mapped out-of-the-box, using the sample product.

Use of this component is required based on the product definition and entities being used.

13 Currency

13.1 Working with Multiple Currencies

In FS-PM, you can specify currency amounts in different countries so that different currency types are captured and displayed in the correct currency.

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

Any currency values captured on the screen (for example, the sum insured amount) are captured and displayed in the selected risk currency. The system converts the captured amount to local currency and perform calculations for calculation rules, validation rules, and so on.

Then, the system converts the calculation results back to risk currency and displays them on screen, stores them in database, or performs validations in local currency.

Types of Currency

There are three types of currencies in FS-PRO :

Local Currency	Local currency of the residence country of a company. This currency is used for product definitions and calculations in the product engine. The local currency is also used for product design (for example, customizing charges, deductibles, and dual control rules). In the eApplication, the `Local Currency Code` is read-only.
Risk Currency	Currency used for contractual agreements that are valid for a single insurance contract. Risk currency is used for data capture and storage.
Invoice Currency	Currency used for premium billing and in CD-documents. Invoice currency is also the currency in which the *Quote Summary* screen displays.

Supported Multi-currency Models

The multi-currency functionality is supported for the following levels:

Contract	Risk and invoice currency are used at the contract level and all sub-levels for data capture (for example, coverage, sub-coverage).

Coverage Risk and invoice currency are used at the coverage level and sub-levels (sub-coverage) for data capture.

> **ℹ Note**
>
> For Native APIF, multi-currency only works on products replicated from FS-PM.

Related Information

Multi-currency Exchange Rates [page 281]

13.1.1 Enabling Multi-Currency for Products

13.1.1.1 Currency Data Type

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

The CURRENCY data type can be used in a data definition. Some number data type columns are updated to the CURRENCY data type after a PBT import or a bootstrap import to the Product Modeler. The Runtime instance isn't affected, as the columns remain as NUMBER data type.

> **⚠ Caution**
>
> If you want to use multi-currency in your product, don't use rounding on the currency values in the rules. Otherwise, precisions will be lost during currency conversions.

13.1.1.2 Multi-currency Columns

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

In the FS-PRO data definition tables, new columns are required to store currency codes, exchange rate dates, and rates.

- For products with multi-currency support at the contract level, use the columns provided in the ABDAPOLPR table.

- For products with multi-currency support at the coverage level, use the columns in the ABDACOV tables (except for ABDAPOLPR.LOCCURR_CD and ABDAPOLPR.XRATEDETTYPE_CD which are settings at the contract level.

The following multi-currency columns are available:

Table and Column Name	Column Description	Data Type	Length
ABDAPOLPR.LOCCURR_CD	ISO Local Currency Code	TEXT	5
ABDAPOLPR.INVCURR_CD	ISO Invoice Currency Code	TEXT	5
ABDAPOLPR.RISKCURR_CD	ISO Invoice Currency Code	TEXT	5
ABDAPOLPR.XRATEDET-TYPE_CD	Exchange Rate Determination Type Code: 0 - No Selection 1 - Variable 2 - Fixed	NUMBER	1
ABDAPOLPR.RISKEXCH-RATE_DT	Risk Exchange Rate Date	DATE	8
ABDAPOLPR.RISKEXCH-RATE_VL	Risk Exchange Rate	NUMBER	(30, 14)
ABDAPOLPR.INVOEXCH-RATE_DT	Invoice Exchange Rate Date	DATE	8
ABDAPOLPR.INVOEXCH-RATE_VL	Invoice Exchange Rate	NUMBER	(30, 14)
ABDAPOLPR.RISKAP-PLEXCHRATE_DT	Application Risk Exchange Rate Date	DATE	8
ABDAPOLPR.RISKAP-PLEXCHRATE_VL	Application Risk Exchange Rate	NUMBER	(30, 14)
ABDAPOLPR.INVOAP-PLEXCHRATE_DT	Application Invoice Exchange Rate Date	DATE	8
ABDAPOLPR.INVOAP-PLEXCHRATE_VL	Application Invoice Exchange Rate	NUMBER	(30, 14)
ABDACOV.INVCURR_CD	ISO Invoice Currency Code	TEXT	5
ABDACOV.RISKCURR_CD	ISO Risk Currency Code	TEXT	5
ABDACOV.RISKEXCH-RATE_DT	Risk Exchange Rate Date	DATE	8

Table and Column Name	Column Description	Data Type	Length
ABDACOV.RISKEXCH-RATE_VL	Risk Exchange Rate	NUMBER	(30, 14)
ABDACOV.INVOEXCH-RATE_DT	Invoice Exchange Rate Date	DATE	8
ABDACOV.INVOEXCH-RATE_VL	Invoice Exchange Rate	NUMBER	(30, 14)
ABDACOV.RISKAPPLEXCH-RATE_DT	Application Risk Exchange Rate Date	DATE	8
ABDACOV.RISKAPPLEXCH-RATE_VL	Application Risk Exchange Rate	NUMBER	(30, 14)
ABDACOV.INVOAPPLEXCH-RATE_DT	Application Invoice Exchange Rate Date	DATE	8
ABDACOV.INVOAPPLEXCH-RATE_VL	Application Invoice Exchange Rate	NUMBER	(30, 14)

13.1.1.3 Importing Multi-Currency Columns

To use the multi-currency feature, you must import the columns provided for multi-currency.

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

To import multi-currency related columns, you need to import from FS-PM using the PBT import for both the Product Modeler Data Definition and the Runtime flowstore. For more information about PBT import, see Importing PBT Tables.

Alternately, import from latest bootstrap and flowstore.

13.1.1.4 Designing Questionnaires for Multi-currency Support

You should design the currency-related columns to capture currency-related information. You can design the columns in their questionnaires at the contract or coverage level, depending at which level your product supports multi-currency.

Context

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

In the Questionnaire model, questions have currency data type information displayed, although the control type remains as Number.

Procedure

1. Select the *Currency* checkbox in the *Number Properties* dialog box, to mark the question as currency.
2. Select the *Grouping Enabled* checkbox to group digits in this currency value.

   If a data list is designed to have currency values as list items, the values on the list won't be converted. They remain in the currency that you configured them in design time (most likely, local currency).

   > **i Note**
   >
   > In the question label, you might want to indicate the currency in which the dropdown list values are listed.

Related Information

Questionnaire Models [page 175]

13.1.1.5 Defining Available Currencies

You must define the available currencies to use the multi-currency feature.

Context

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

Procedure

1. Use the Risk Currency Code (for the `RISKCURR_CD` value) and Invoice Currency Code (for the `INVCURR_CD` value) domain tables to define currencies to be available for selection in runtime.
2. Define them for each contract if multi-currency is supported at the contract level.
3. Define them for each coverage if multi-currency is supported at the coverage level.

13.1.1.6 Setting Multi-currency Default Values and Settings

You must set the default values to use multi-currency.

Context

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

Procedure

1. Set the default exchange rate date to be used for risk currency and local currency.

   You can set the default values in the `SAP Column Default Values` component, or you can define them in the `eApp Column Info` rules. For example, you could set the exchange rate date to be the same date as the policy begin date, ABDAPOLPR.POLPRODBEG_DT.

2. Set the default values for local (`LOCCURR_CD`), risk (`RISKCURR_CD`), and invoice currency (`INVCURR_CD`).

   You can set the default values in the `SAP Column Default Values` component, or you can configure them in the `eApp Column Info` rules. For example, you could set the local currency as EUR, and then set the Risk and Invoice currency to be the same as the local currency.
3. Set the default value for exchange rate determination type (`XRATEDETTYPE_CD`):
   - 0 = No multi-currency selected.
   - 1 = Fixed exchange rate determination type.
     Exchange rate is manually set in New Business and fixed for the entire contract lifecycle.
   - 2 = Variable exchange rate determination type.
     Exchange rate is based on exchange rate values maintained in the central exchange rate table. The exchange rate is automatically determined each time a new contract version is created.
4. Set the multi-currency category control attribute, `MULTI_CURRENCY_CATEGORY`, in the contract's `IFBC Control Attributes` component. This attribute specifies the level at which multi-currency is supported:
   - 1 = Multi-currency at contract level.
   - 2 = Multi-currency at coverage level.
5. Set the `Multi Currency Mode` component in the product's `Configuration` folder:
   a. In the `Table Name` column, specify the value for the table name associated with the currency category (in the `Category Name` column).

      Current default values in the `Category Type` column are ABDAPOLPR for 1 (multi-currency at the contract level) and ABDACOV for 2 (multi-currency at the coverage level). You can use this component to specify other table names when defining multi-currency for other types of products. For example, risk-based products.

Results

Existing products with no multi-currency support continue to use the `CURRENCY_ID` to specify the currency used in the *Quote Summary* screen.

13.1.1.7 Quote Summary Enhancements

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

The `SAP Option Summary Grid Columns` component is enhanced to support displaying the *Quote Summary* screen with multi-currency.

- The `TableName` column specifies the table name associated with the multi-currency category.
- The rules in `ColumnDataRule` column look for the values for that category.
- The `getOptionSummaryGridColumnsMultiLevel` Service API queries the `SAP Option Summary Grid Columns` component filtered by `TableName`.

- In Runtime, the system displays premium breakdown in the *Quote Summary* screen.
- For multi-currency support at the contract level (ABDAPOLPR), the *Quote Summary* screen displays the premium breakdown for each contract in the selected invoice currency for that contract.
- For multi-currency support at coverage level (ABDACOV), the *Quote Summary* screen displays the premium breakdown for each coverage in the selected invoice currency for that coverage.

> **i Note**
>
> For policy-level currencies (ABDAPOLICY), the *Quote Summary* screen displays the premium value in the local currency.

13.1.2 Multi-currency Exchange Rates

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

Types of Exchange Rates

There are two types of exchange rates that determine how a new contract version is created:

Fixed	This exchange rate is captured manually in New Business and fixed for the entire contract lifecycle, until the contract is renewed. You can overwrite the default value (in New Business only). For all other business processes, the fixed rate from New Business is used and the fields will be displayed as read-only.
Variable	This exchange rate is based on exchange rate values that are maintained in the exchange rate table. This rate is automatically determined each time you create a new version of a policy or contract. You can't override variable exchange rates.

Exchange Rates Table

In the Exchange Rates table, you can store currency exchange rate information so that the information can be used for currency conversions. SAP provides a default exchange rates table with the product. You must populate and define the table to your specifications.

The new Exchange Rates table contains four columns:

FROM_CURRENCY	Source currency (Risk or Invoice currency). The data format is a 3-letter ISO code.

TO_CURRENCY	Target currency (Local currency). The data format is 3-letter ISO code.
EFFECTIVE_DATE	Date the exchange rate goes into effect. The data format is standard date data type and isn't time zone specific.
EXCHANGE_RATE	Exchange rate that you determine for your product. The data format is a floating point data type with a large number of decimal precision.

Importing and Retrieving Exchange Rates

Importing Exchange Rates	The database administrator should insert your companies' own exchange rates into the CURRENCY_EXCHANGE_RATE database table.
Retrieving Exchange Rates	The product developer can use the service API, `getExchangeRate`, to retrieve exchange rates from the database. The API can be extended to retrieve exchange rates from elsewhere.

Exchange Rate Lookup

If you need to look up exchange rates as part of a rule, use the stem `CurrencyExchRateLookup.getExchangeRate(fromCurrency, toCurrency, exchangeRateDate)`.

Using Exchange Rate Dates

For risk or invoice currency, you can either use the application date as the exchange rate or you can use a custom exchange rate date.

Related Information

Specifying the Exchange Rate Methods that Integrate with FS-PM [page 283]
Example: Using Exchange Rates for Contracts [page 284]

13.1.2.1 Specifying the Exchange Rate Methods that Integrate with FS-PM

For FS-PRO, the exchange rate will be always in the direct position. This topic is for customers who use FS-PM (TCURR + TCURF) to build up exchange rates for other downstream systems for integration purposes.

Context

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

In the Administrative Console, you must specify the following settings in runtime in order to use the multi-currency feature.

> **i Note**
>
> You don't need to change these settings if you aren't using the multi-currency feature.

Procedure

1. Go to *Application* > *AuthoritySuite* > *Env* > *FS-PM Integration Setting*.
2. For `BAPIExchangeRateGetFactorsMethod`, set the method used to get rate factor.
3. For `BAPIExchangeRateGetFactorsType`, set the exchange rate type used by the integrated FS-PM system.

Results

The APIF `DirectIndirectRateFlag` component is added to the *SAP Global Reference Object* > *Configuration* > *Integration* folder. This controls the direction exchange rates are sent out in the APIF call for different currencies, depending on the integrated FS-PM system. For example,

Direct	Exchange rates we usually see. For example, on http://www.xe.com/ .
Indirect	Exchange rates are sent inversed.

13.1.2.2 Example: Using Exchange Rates for Contracts

For contracts, you can specify whether or not you want to use multi-currency.

> **! Restriction**
>
> Support of multiple currencies is for coverage-based products only. This feature is not available for risk-based products. It is also not available for products based on the Sample Auto Product.

Electing Not to use Multi-currency

If you don't want to use multi-currency in the *Exchange Rate Type* field, select No Specification. The risk and invoice currency related fields become hidden.

Specifying an Exchange Rate Type

1. In the *Exchange Rate Type* field, select Fixed or Variable.

If you select Variable, the exchange rate date and exchange rate fields are read-only.

If you select Fixed, you can do the following:

- In the *Exchange Rate Type* field, select a different exchange rate date. You can now use the updated exchange rate date in your calculations.
- In the *Exchange Rate* field, select a different exchange rate. You can now use the updated exchange rate in your calculations.

> **i Note**
>
> If you choose Fixed or Variable and then go back to No Specification, all of the fields are reset and you must then redefine the values.

13.2 Understanding Currency Scaling

Currency scaling is a mechanism within the APIF XML export process that adjusts currency values to ensure the correct decimal place scaling within the source (FS-PRO) and target (FS-PM) systems.

> **! Restriction**
>
> The currency scaling feature is available for use with coverage-based products only. It is not supported for risk-based products.

The scaling adjustment that is applied is dictated by the exponent factor of a currency, as indicated in the ISO 4217 standard. For more information about currency code decimal numbers, see https://en.wikipedia.org/wiki/ISO_4217

This factor indicates the relationship between a currency's major and minor unit.

For example, USD (the United States dollar) is equal to 100 of its minor currency unit the cent. So the USD has exponent 2. The JPY (Japanese yen) is given the exponent 0, because its minor unit, the sen, although nominally valued at 1/100 of a yen, is of such negligible value that it is no longer used.

The majority of currencies have an exponent factor of 2. When transferring data between subsystems (such as FS-PRO and FS-PM) it is required that all currency values have an exponent factor of 2. No scaling adjustment is applied for these standard currencies.

For non-standard currencies, with an exponent factor not equal to 2, currency values needs to be converted to a factor 2 value prior to transfer to the target system. Once transferred the target system will reverse the scaling adjustment and restore the currency to it's original decimal place value.

For example, the South Korean won (KRW) has a exponent factor of 0. When booking a quote, all currency values stored in the FS-PRO record need to be divided by 100 within the APIF XML. The FS-PM system will consume the APIF XML and multiply the currency value by 100 to reverse the scaling adjustment.

Related Information

Setting Multi-Currency Functionality for Currency Scaling [page 285]
Overriding Currency Definitions [page 286]
Overriding the Fraction Decimal for Currency Scaling [page 289]

13.2.1 Setting Multi-Currency Functionality for Currency Scaling

The scaling factor applied to currency values when booking a submission is dependent on the multi-currency definition.

> **! Restriction**
>
> The currency scaling feature is available for use with coverage-based products only. It is not supported for risk-based products.

There are three multi-currency definitions which determine scaling behavior:

Policy Level	No MULTI_CURRENCY_CATEGORY is specified. All currency values within the submission will be scaled based on the currency specified in at the product level (ABDAPOLICY.CURRENCY_ID).
Contract Level	MULTI_CURRENCY_CATEGORY is set to 1 (*Contracts* > `<contract_name>` > *Configuration* > *Integration* > *IFBC Control Attributes*).

	This definition requires that a risk currency be specified at the contract level (ABDAPOLPR.RISKCURR_CD). Currency values within the contract and child tables (such as, premium and coverage) will be scaled based on the risk currency specified on the contract (ABDAPOLPR.RISKCURR_CD).
Coverage Level	MULTI_CURRENCY_CATEGORY is set to 2 (*Contracts* > `<contract_name>` > *Configuration* > *Integration* > *IFBC Control Attributes*). This definition requires that a risk currency be specified at the coverage level (ABDACOV.RISKCURR_CD). All currency values within the coverage and child tables (such as premium and sub-coverage) will be scaled based on the risk currency specified on the coverage (ABDACOV.RISKCURR_CD). A local currency also needs to be specified on the parent contract level (ABDAPOLPR.LOCCURR_CD). All currency values within the contract and contract level tables (such as contract premium) will be scaled based on the local currency specified on the contract (ABDAPOLPR.LOCCURR_CD).

Prerequisites within the Data Model for Currency Scaling

Currency columns within a data definition table are flagged with a CURRENCY data type.

> **i Note**
>
> Currency scaling is only applied to column fields with CURRENCY data types. NUMBER data type fields aren't scaled.

13.2.2 Overriding Currency Definitions

A override capability is available for currency values within lateral tables. With the override, currency values are scaled based on the currency value specified in the table, rather than by the default scaling definition.

Context

> **! Restriction**
>
> The currency scaling feature is available for use with coverage-based products only. It is not supported for risk-based products.

For example, the limit or deductible currency may differ from the parent contract or coverage object. The override option will allow for the scaling of these currency values based on the CURRENCY_ID specified in the lateral table.

> **i Note**
>
> The override capability is applicable only to multi-currency policies where the MULTI_CURRENCY_CATEGORY is set to 1 (contract level) or 2 (coverage level). Single currency policies will scale using the CURRENCY_ID value specified in the ABDAPOLICY table.

Procedure

1. Open your web browser 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 *Configuration* > *Application* > *AuthoritySuite* > *Env* > *FS-PM Integration Setting*.
4. Locate the *Currency_Code_OVERRIDE_ENABLED* row.
5. Set the value to **YES** in the *Override* column.
6. Save your changes.

Related Information

Overriding the CURRENCY_ID Reference Pointer [page 287]

13.2.2.1 Overriding the `CURRENCY_ID` Reference Pointer

This topic describes the steps needed to change the currency utilized for scaling in multi-currency policies.

Context

> **! Restriction**
>
> The currency scaling feature is available for use with coverage-based products only. It is not supported for risk-based products.

It may happen that a table contains multiple currency values with differing currency types. As a result, the currency values will need to be scaled based on the assigned currency type.

For example, if your ABDALIMIT table contained two LIMIT values: LIMIT_AM is US dollars (USD) and MAXLIMIT_AM is Korean Wan (KRW). Each LIMIT value needs to be scaled according to its associated currency type.

In FS-PRO, you will need to define the currency ID mappings that will be used in this situation. This is achieved in the `CurrencyID Domain` domain table. This mapping can occur at the product level or globally within a Reference Object.

The scaling service will call the `getCurrencyID` API rule, which will query the `CurrencyID Domain` domain table where currency-table.attribute associations can be defined. The service will attempt to find a matching association. If an association is found, the specified currency field will be utilized. If an association isn't found, the default [*] CURRENCY_ID will be used. The scaling service will initially attempt to derive overrides on the product, followed by the reference object.

> **i Note**
>
> This functionality is only triggered for multi-currency policies where the `Currency_Code_OVERRIDE_ENABLED` flag is set within the Runtime Administrative Console.

Procedure

1. If you want to associate the table or table.attribute to a particular currency type in a product, perform the following steps:
   a. Navigate to the product.
   b. Go to `<product_name>` > *Reference Data* > *Domain Tables* > *CurrencyID Domain*.
   c. Ensure that the default record (*,CURRENCY_ID) is selected.
   d. Add a data value row.
   e. Enter the table name, or table name and attribute combination, in the *Key* field of the new row.

      For example, POLPR_ABDALIMIT, POLPR_ABDALIMIT.LIMIT_AM or POLPR_ABDALIMIT.MAXLIMIT_AM.
   f. Enter the currency type attribute in the *CurrencyValue* field of the new row.

      For example, USD or KRW.
   g. Save your changes.
   h. Go to the *Configuration* > *Product Services* > *Service API* folder for the product.
   i. Ensure that the `getCurrencyID` API rule is selected.
   j. Build the metadata and rules for the product.
   k. Publish the product.
2. If you want to associate the table or table.attribute to a particular currency type in a reference object, perform the following steps:
   a. Navigate to the reference object to which your products refer.
   b. Go to `<reference_object>` > *Domain Tables* > *CurrencyID Domain*.
   c. Ensure that the default record (*,CURRENCY_ID) is selected.
   d. Add a row.
   e. Enter the table name, or table name and attribute combination, in the *Key* field of the new row.

      For example, POLPR_ABDALIMIT, POLPR_ABDALIMIT.LIMIT_AM or POLPR_ABDALIMIT.MAXLIMIT_AM.
   f. Enter the currency type attribute in the *CurrencyValue* field of the new row.

      For example, USD or KRW.

g. Save your changes.
h. Go to the *Configuration* > *Reference Services* > *Service API* folder for the reference object.
i. Ensure that the `getCurrencyID` API rule is selected.

> → Remember
>
> Reference object overrides won't be applied if the overrride is also enabled on the product.

j. Build the metadata and rules for the reference object.
k. Publish the reference object.

13.2.3 Overriding the Fraction Decimal for Currency Scaling

The fractional decimal for a specified currency that is defined in the ISO 4217 standard may not be the precision needed for currency scaling. An alternate precision can be defined within a CSV file.

Context

> ! Restriction
>
> The currency scaling feature is available for use with coverage-based products only. It is not supported for risk-based products.

The default location for the `CurrencyDecimalPlaceDef_default.csv` file is in the following FS-QUO `CSI_HOME` sub-directory: `<CSI_HOME>/as/custom/web/resource/CurrencyDecimalPlaceDef_default.csv`

When attempting to determine the precision, FS-QUO will initially attempt to lookup the currency fraction from the `CurrencyDecimalPlaceDef_default.csv` file. If a currency isn't found in the CSV then the currency fraction from the ISO 4217 standard will be returned and applied.

Procedure

1. Open your web browser 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*.
3. From the *Path* dropdown list, select *PA - Custom Web Root (Write)*.
4. Navigate to the *resource* folder.
5. Download the file.
6. Edit it to define the new fractional decimal value.

   A fractional decimal needs to be defined for each currency.

7. Save your changes.
8. Upload the updated file.

14 Surcharges and Discounts

In FS-PRO and FS-PM, you can define surcharges and discounts to be applied to quotations or counteroffers:

> **! Restriction**
>
> The surcharges and discounts feature is available for use with coverage-based products only. It is not supported for risk-based products.

During quotation or while creating a counteroffer, you can apply a surcharge or a discount. The value of the surcharge or the discount will be added at the contract or coverage level. Premiums will then be calculated to include the surcharge or the discount value.

After a surcharge or discount has been applied, it can be over-ridden at any time by an agent or underwriter until the quote has been issued.

This surcharge and discount functionality is supported for the contract and coverage levels.

14.1 Life and Household

14.1.1 Surcharge and Discount Columns for Life and Household Products

In the FS-PRO data definition tables for Life or Household products, columns are required to store the rates and amounts of surcharges and discounts.

> **! Restriction**
>
> The surcharges and discounts feature is available for use with coverage-based products only. It is not supported for risk-based products.

For Life or Household products with surcharges and discounts at the contract level, use the columns provided in the POLPR table.

For Life or Household products with surcharges and discounts at the coverage level, use the columns in the ABDACOV tables (except for `ABDAPOLPR.LOCCURR_CD` and `ABDAPOLPR.XRATEDETTYPE_CD` which are settings at the contract level).

The following surcharge and discount columns are available:

Available Surcharge and Discount Columns

Table and Column Name	Column Description	Data Type	Length
POLPR_ABDASUPDC.PRE-MFACT_TT	Surcharge or Discount	TEXT	5
POLPR_ABDASUPDC.PRE-MFACTUNIT_CD	Calculation Type	TEXT	5
POLPR_ABDASUPDC.PRE-MFACTOR_VL	Relative Value	NUMBER	5
POLPR_ABDA-SUPDC.AMOUNT_AM	Surcharge Or Discount Amount	NUMBER	1
POLPR_ABDASUPDC.PRE-MFACTCALC_TT	Reference Value Name	TEXT	8
POLPR_ABDASUPDC.PREM-RATE_AM	Proportnl Surcharg or Discnt	NUMBER	(30, 14)
POLPR_ABDASUPDC.PRE-MFACTBEG_DT	Start Date	DATE	8
POLPR_ABDASUPDC.CUR-RENCY_ID	Currency	TEXT	(30, 14)

14.1.2 Creating a Surcharge and Discount Object for Life and Household Products

The `Sample Household Extended` product includes a defined surcharge and a discount, that are assembled at the contract level:

Context

> **! Restriction**
>
> The surcharges and discounts feature is available for use with coverage-based products only. It is not supported for risk-based products.

SAP Installment Payment Surcharge	If an underwriter or agent chooses to add this surcharge, the system derives the predefined rates based on the payment frequency. For example, if the payment frequency is monthly, a surcharge rate of 5% is applied.

	Rate tables for the predefined rates are defined out of the box and can be modified.
SAP Recurring Discount	If an underwriter or agent chooses to add this discount, the system derives the rates based on the duration of contract. For example, if the duration of the contract is less than 5 years, a discount rate of 3% of the premium will be applied.

However, customized objects can also be created.

To create a surcharge/discount object, follow these steps:

Procedure

1. Navigate to your company library in the Product Studio.
2. Create a folder called **`Surcharges and Discounts`**.
3. Open the folder and choose *Create New Object*.

   The *New Object* screen opens.
4. Give the new surcharge/discount object a unique name.
5. Choose Surcharge and Discounts from the *Modifier Type* dropdown list.
6. Choose *Inherit From*, which is located next to the *Based On* field, and choose an object to determine the inheritance.
7. Choose *Select*.
8. Ensure that the *Inherit Folder Permission* checkbox is selected.
9. Choose *OK*.

Results

The new object is created and opened in the Product Modeler, ready to be configured.

14.1.3 Defining a Surcharge and Discount Object for Life and Household Products

As the surcharge and discount object is a product-specific object, it will have specific values and rules that will be used for the product. You can define these values and rules to fulfill your business needs.

> **! Restriction**
>
> The surcharges and discounts feature is available for use with coverage-based products only. It is not supported for risk-based products.

To define a surcharge object or a discount object in a Life or Household product, you'll need to perform the following tasks:

- Adding sections to the UI on the eApplication
- Creating questionnaire views
- Setting the values and rates in the domain tables
- Adding sections to the UI on the eApplication
- Defining the rules

After the definition is complete, build and publish the product to see the changes in Runtime.

Adding Questions to the Questionnaire

You'll need to add questions for the surcharges and discounts to the questionnaire model. A standard set of questions is provided at the `SAP Insurance UWA Surcharge Discount Template` level. You can see the questions under the `User Interface` folder, under *SAP Surcharge Discount Model View* > *Questions* > *SAP Surcharge Discount Questions*.

You'll need to define two 2 types of questions: questions for the *Detail* screens and questions for the grid.

> **Note**
>
> Don't alter, edit or delete the default questions. If the default questions don't satisfy your requirements, deselect them and create new ones.

Creating Questionnaire Views

Next, use the questions to create questionnaire views. You can see the questions under the `User Interface` folder, under *SAP Surcharge Discount Model View* > *Views* > *SAP Surcharge Discount Views*.

Two default views are provided:

- `SAP Surcharge Discount Detail - PQM30`
- `SAP Surcharge Discount Grid - PQM30`

> **Caution**
>
> Don't alter, edit or delete the default views. If the default views don't satisfy your requirements, create new ones.

Setting the Values and Rates in the Domain Tables

You'll need to define the logic for surcharges and discounts:

- What objects are eligible for a surcharge or discount?
- When should a surcharge or discount be allowed?
- How is the value of the surcharge or discount to be determined?

Add keys and values to domain tables to determine the rules and behavior of the surcharge or discount in the UI. Out-of-the-box, three domain tables are provided:

Surcharge Discount CalculationType Domain	Determines if the surcharge/discount will be applied as a fixed amount, a percentage, or a per mille value.
Surcharge Discount ReferenceValue Domain	Determines what object the surcharge or discount will be applied to.
Surcharge Discount SurchargeDiscountType Domain	Determines the type of surcharge/discount that will be applied.

You can choose to create and assemble your own tables in the `Reference Data` folder. Or you can use the default domain tables, where you can add new keys and values and override the existing ones.

> ⚠ Caution
>
> Don't delete the default keys and values from the domain tables

Once the surcharge and discount logic has been defined, the rates need to be set.

In the `Surcharges and Discounts` folder, go to *Configuration* > *Reference Data* > *Rate Tables* to customize the rates.

Code	Specifies the payment frequency of the premium.
Rate	Specifies is the factor that would be applied based on the payment schedule.

The default surcharge rate table is named `SampleInstallmentPaymentSurchargeRate`. The default discount rate table is named `SampleRecurringDiscountRates`.

Assembling the Object Under Marketable Products

The newly created surcharge and discount objects have to be assembled under the contract or coverage (as appropriate).

Adding Sections to the UI on the eApplication

Now that you've assembled the object, you'll need to add a new section or grid to the UI for the surcharges and discounts on the eApplication:

1. Go to *Sample Household Product Extended* > *Contracts* > *Sample Household Contract* > *User Interface* > *Data Capture* > *eApplication*.
2. Go to the *Values* tab.
3. Expand the *Producer Household* item in the tree, under the *EApp Name* item.
4. Expand the *Household Contract* item in the tree, under the *Section Name* item, in the *eApp Navigation Column* section.
5. Select the *Surcharge and Discount Details* item.
6. Select the button in the Screen ID column. On the resulting screen, drag and drop the *Detail* view created earlier and click *Save*.
7. Choose *Preview* to view how the UI will look with the new addition.

Defining the Rules

The out-of-the-box surcharge and discount object includes a set of rules to govern the behavior of surcharge and discount questionnaires. As the default rules won't cover all situations, you may define your own rules for your surcharges and discounts.

> ⚠ Caution
>
> Don't alter or delete the default rules. You can override any unwanted rule.

14.2 Auto-International

14.2.1 Surcharge and Discount Columns for Auto Products

In the FS-PRO data definition tables for Auto products, columns are required to store the rates and amounts of surcharges and discounts.

> ! Restriction
>
> The surcharges and discounts feature is available for use with coverage-based products only. It is not supported for risk-based products.

For Auto products with surcharges and discounts at the contract level, use the columns provided in the POLPR table.

The following surcharge and discount columns are available:

Table and Column Name	Column Description	Data Type	Length
POLPR_ABDASUPDC.PRE-MFACT_TT	Surcharge or Discount	TEXT	50
POLPR_ABDASUPDC.PRE-MFACTUNIT_CD	Calculation Type	TEXT	55 (3)
POLPR_ABDASUPDC.PRE-MFACTOR_VL	Relative Value	NUMBER	5
POLPR_ABDA-SUPDC.AMOUNT_AM	Surcharge Or Discount Amount	NUMBER	20
POLPR_ABDASUPDC.PRE-MFACTCALC_TT	Reference Value Name	TEXT	88 (50)
POLPR_ABDASUPDC.PREM-RATE_AM	Proportional Surcharge or Discount	NUMBER (CURRENCY)	(30, 14) (20)
POLPR_ABDASUPDC.PRE-MFACTBEG_DT	Start Date	DATE	8
POLPR_ABDASUPDC.CUR-RENCY_ID	Currency	TEXT	(30, 14) (5)

14.2.2 Creating a Surcharge and Discount Object for Auto Products

The `Sample Auto International` product includes a defined surcharge and a discount, that is assembled at the contract level.

Context

> **! Restriction**
>
> The surcharges and discounts feature is available for use with coverage-based products only. It is not supported for risk-based products.

Surcharge The underwriter or agent can choose to add a surcharge in the system as a value or as a percent. For example, 100 USD or 5%.

The `Free Surcharge` and `Different owner Surcharge` are available out-of-the-box for products based on the `Sample Auto International` product.

Discount The underwriter or agent can choose to add a discount in the system as a fixed value or as a percent. For example, 100 USD or 5%.

The `Free Discount`, `Employee Discount` and `Coupon Discount` are available out-of-the-box for products based on the `Sample Auto International` product.

However, customized objects can also be created.

To create a surcharge/discount object, follow these steps:

Procedure

1. Navigate to your company library in the Product Studio.
2. Create a folder called **`Surcharges and Discounts`**.
3. Open the folder and choose *Create New Object*.

   The *New Object* screen opens.
4. Give the new surcharge/discount object a unique name.

   Example: Free Discount.
5. Choose *Surcharge and Discounts* from the *Modifier Type* dropdown list.
6. Choose *Inherit From*, which is located next to the *Based On* field, and choose an object to determine the inheritance.
7. Choose *Select*.
8. Ensure that the *Inherit Folder Permission* checkbox is selected.
9. Choose *OK*.

Results

The new object is created and opened in the Product Modeler, ready to be configured.

14.2.3 Defining a Surcharge and Discount Object for Auto Products

As the surcharge and discount object is a product-specific object, it will have specific values and rules that will be used for the product. You can define these values and rules to fulfill your business needs.

> **! Restriction**
>
> The surcharges and discounts feature is available for use with coverage-based products only. It is not supported for risk-based products.

To define a surcharge object or a discount object in an Auto product, you'll need to perform the following tasks:

- Adding sections to the UI on the eApplication
- Creating questionnaire views
- Setting the values in the domain tables
- Adding sections to the UI on the eApplication

- Defining the rules

After the definition is complete, build and publish the product to see the changes in Runtime.

Adding Questions to the Questionnaire

You'll need to add questions for the surcharges and discounts to the questionnaire model. A standard set of questions is provided at the `SAP Insurance UWA Surcharge Discount Template` level. You can see the questions under the `User Interface` folder, under *SAP S4 MV Motor Vehicle Questions* > *SAP S4 QN MV Surcharge Discount Questions*.

You'll need to define two types of questions: questions for the *Detail* screens and questions for the *Grid*.

> **i Note**
>
> Don't alter, edit or delete the default questions. If the default questions don't satisfy your requirements, deselect them and create new ones.

Creating Questionnaire Views

Next, use the questions to create questionnaire views. You can see the questions under the `User Interface` folder, under *SAP Surcharge Discount Model Views* > *SAP Surcharge Discount Views*.

Two default views are provided:

- *SAP S4 Auto Surcharge View*
- *SAP S4 Auto Discount View*

> **⚠ Caution**
>
> Don't alter, edit or delete the default views. If the default views don't satisfy your requirements, create new ones.

Setting the Values in the Domain Tables

You'll need to define the logic for surcharges and discounts:

- What objects are eligible for a surcharge or discount?
- When should a surcharge or discount be allowed?
- How is the value of the surcharge or discount to be determined?

Add keys and values to domain tables to determine the rules and behavior of the surcharge or discount in the UI. Out of the box, three domain tables are provided:

SAP_S4_MV_Discount_Type Domain	Determines the type of Discount for Discount type
SAP_S4_MV_Surcharge_Type	Determines the type of Surcharge for Surcharge Type
SAP_S4_MV_Calculation_Type	Determines whether the Surcharge/Discount will be applied as a fixed amount or as a percentage

You can choose to create and assemble your own tables in the `Reference Data` folder. Or you can use the default domain tables, where you can add new keys and values and override the existing ones.

> ⚠ Caution
>
> Don't delete the default keys and values from the domain tables

Assembling the Objects Under Marketable Products

The newly created *Surcharge and Discount* objects have to be assembled under the contract or coverage (as appropriate).

Adding Sections to the UI on the eApplication

Now that you've assembled the object, you'll need to add a new section or grid to the UI for the surcharges and discounts on the eApplication:

1. Go to *Auto International* > *User Interface* > *Data Capture* > *eApplication*.
2. Go to the *Values* tab.
3. Expand the *QuoteEApp* item in the tree, under the *EApp Name* item.
4. Expand the *Policy Quote Section* item in the tree, under the *Section Name* item, in the *eApp Navigation Column* section.
5. Expand the *Vehicle eApp* Screen.
6. Select the *Surcharge and Discount Details* item.
7. Select the button in the *Screen ID* column. On the resulting screen, drag and drop the *Detail* view created earlier and click *Save*.
8. Choose *Preview* to view how the UI will look with the new addition.

Defining the Rules

The out-of-the-box surcharge and discount object includes a set of rules to govern the behavior of surcharge and discount questionnaires. As the default rules won't cover all situations, you may define your own rules for your surcharges and discounts.

> ⚠ Caution
>
> Don't alter or delete the default rules. You can override any unwanted rule.

15 Clauses

In FS-PRO and FS-PM, you can define clauses to be applied to quotations or counteroffers.

> **! Restriction**
>
> The clause feature is available for use with coverage-based products only. It is not supported for risk-based products.

During quotation or while creating a counteroffer, you can create a clause. In the pattern provided, a clause is added at the contract level.

After a clause has been applied, it can be over-ridden at any time by an agent or underwriter until the quote has been issued.

This clause functionality is supported for the contract and coverage levels.

> **i Note**
>
> This feature is configured by default in any products that were created using the sample Auto product (`SAP S4 PR Motor Vehicle Insurance`) templates. For products that were created using other templates, the feature will have to be configured manually.

Related Information

Clause Columns [page 302]
Creating a Clause Object [page 303]
Defining a Clause Object [page 304]

15.1 Clause Columns

In the FS-PRO data definition tables, columns are required to store the clause-related information.

For clauses at the contract level, use the columns created in the *POLPR_ABDACLAUSE* table.

The following clause columns are available:

Column Name	Column Description	Data Type	Length
POLPR_ABDA-CLAUSE.CLAUSENR_CD	Clause Number Code	NUMBER	5

Column Name	Column Description	Data Type	Length
POLPR_ABDA-CLAUSE.CLAUSENR_TT	Clause Name	TEXT	80
POLPR_ABDACLAUSE.BE-GIN_DT	Clause Begin Date	DATE	8
POLPR_ABDA-CLAUSE.END_DT	Clause End Date	DATE	8
POLPR_ABDA-CLAUSE.LONGTEXT_ID	Long Text Id	TEXT	22
POLPR_ABDA-CLAUSE.LONG-TEXT_DESC_T	Description/Long Text	CLOB	LONG TEXT

15.2 Creating a Clause Object

The `Auto International` product includes a defined `Clause` object, that is assembled at the contract level.

Context

The following types of clauses are available:

Clause Name/Number	Clause Type	Long Text	Processing Indicator	Cardinality
Inclusion Clause	Individual Agreement Inclusion	Yes	Manual Inclusion, values may be changed	0...1
Exclusion Clause	Individual Agreement (Exclusion)	Yes	Manual Inclusion, values may be changed	0...1
Glass breakage	Special Condition	No	Manual Inclusion, values may be changed	0...1
Compulsory Authorized Workshop Usage	General Contract Condition	No	Auto Inclusion, value may not be changed Inclusion condition: Always	0...1

If the existing clauses are not sufficient, you can create your own clause objects that fulfill your business needs.

To create a `Clause` object, follow these steps:

Procedure

1. Navigate to your company library in the Product Studio.
2. Create a folder called **`Clause`**.
3. Open the folder and choose *Create New Object*.
   The *New Object* screen opens.
4. Give the new `Clause` object a unique name.

   Example: Glass Breakage.
5. Choose *Clause* from the *Modifier Type* dropdown list.
6. Choose *Inherit From*, which is located next to the *Based On* field, and choose an object to determine the inheritance.
7. Choose *Select*.
8. Ensure that the *Inherit Folder Permission* checkbox is selected.
9. Choose *OK*.

15.3 Defining a Clause Object

As the `Clause` object is a product-specific object, it will have specific values and rules that will be used for the product. You can define these values and rules to fulfill your business needs.

To define a `Clause` object, you'll need to perform the following tasks:

- Adding sections to the UI on the eApplication.
- Creating questionnaire views.
- Setting the values in the domain tables.
- Adding sections to the UI on the eApplication.
- Defining the rules.

After the definition is complete, build and publish the product to see the changes in Runtime.

Adding Questions to the Questionnaire.

You'll need to add questions for the clause to the questionnaire model. A standard set of questions is provided at the SAP Insurance UWA Clause Template level. You can see the questions under the `User Interface` folder, under *SAP S4 MV Motor Vehicle.Questions* > *SAP S4 QN MV Clause Questions*.

You'll need to define two 2 types of questions: questions for the Detail screens and questions for the grid.

> **ℹ Note**
>
> Don't alter, edit or delete the default questions. If the default questions don't satisfy your requirements, deselect them and create new ones.

Creating Questionnaire Views

Next, use the questions to create questionnaire views. You can see the questions under the `User Interface` folder, under *SAP S4 MV Motor Vehicle Views* > *SAP S4 QN MV Clause Questions*.

Following views are created:

- *SAP S4 Auto Clause View*
- *Clause Child View*

> **⚠ Caution**
>
> Don't alter, edit or delete the default views. If the default views don't satisfy your requirements, create new ones.

Setting the Values in the Domain Tables

Add keys and values to domain tables to determine the rules and behavior of the `Clause` object in the UI. Out of the box, one domain table is provided:

SAP_S4_MV_Clause_Number	Determines the Clause Number. Example,: Inclusion or Exclusion etc.

You can choose to create and assemble your own tables in the `Reference Data` folder. Or you can use the default domain tables, where you can add new keys and values and override the existing ones.

Assembling the Objects Under Marketable Products

The newly created `Clause` objects have to be assembled under the contract level.

Adding Sections to the UI on the eApplication

Now that you've assembled the object, you'll need to add a new section or grid to the UI for the `Clause` object on the eApplication:

1. Go to *Auto International* > *User Interface* > *Data Capture* > *eApplication*.

2. Go to the *Values* tab.
3. Expand the `QuoteEApp` item in the tree, under the `EApp Name` item.
4. Expand the `Policy Quote Section` item in the tree, under the `Section Name` item, in the `eApp Navigation Column` section.
5. Expand the `Vehicle eApp Screen`.
6. Select the `Clause Details` item.
7. Select the button in the `Screen ID` column. On the resulting screen, drag and drop the *Detail* view created earlier and click *Save*.
8. Choose *Preview* to view how the UI will look with the new addition.

Defining the Rules

The out-of-the-box `Clause` object includes a set of rules to govern the behavior of clause questionnaires. As the default rules won't cover all situations, you may define your own rules for your `Clause` objects.

> ⚠ Caution
>
> Don't alter or delete the default rules. You can override any unwanted rule.

16 Planbooks

You can define a planbook that can be applied to quotations or counteroffers.

The Planbook serves as a template in which you can offer different Plan types of your product to your customers. For example, the *Best Personal Protection* sample product offers 3 Plan types: Basic, Advanced and Best.

During the lifecycle of your product, your product may undergo several changes. In order to accommodate these changes, you can create multiple planbooks for your product. During runtime, the system uses the *Effective Date* of your Planbook version to select the right version of your planbook.

Planbook functionality is configured out-of-the-box in the *Best Personal Protection* sample product, but can be configured manaully in other coverage-based products.

Related Information

Creating the Planbook (Reference Object) and Inheritance Structure [page 307]
Registering a Planbook [page 308]
Assembling Components in the Planbooks and the Service API Rule [page 309]
Displaying Benefits on eApps [page 310]
Linking a Planbook to a Marketable Product [page 310]
Versioning a Planbook [page 311]

16.1 Creating the Planbook (Reference Object) and Inheritance Structure

The following procedure describes how to create a planbook inheritance structure from a reference object.

Context

To create planbook (Reference Object) and its inheritance structure proceed as follows:

Procedure

1. Navigate to *Studio* > *Content Repository* > *SAP S4Library* > *Product Studio*
2. Create a new folder named *Reference Object* with the modifier type *Product Folder*.
3. In the *Reference Object* folder, create a folder named *Planbook* with the modifier type *Product Folder*.
4. In the *Planbook* folder, create your template object with the modifier type *Planbook*.
5. Create a new object.

   Example: Best Personal Protection Planbook V1 with modifier type *Planbook* inheriting from the Company Insurance PnC Planbook Template.

   > **i Note**
   >
   > The new object that you have created is from a inherited list. To view this inheritance list, right click on the object, choose *Properties* and then select *Inheritance List Tab*.

Results

You can view the *Planbook* object in the *Reference Object* folder.

16.2 Registering a Planbook

Your product has to be registered in the Design Time Administrative Console.

Context

To register a planbook, proceed as follows:

Procedure

1. Open your web browser and go to `<pro_designtime_app_url>/csiroot/admin/`.
2. Log in to the Administrative Console.

   The Administrative Console will open after a short delay.
3. Go to *Product* > *Edit Product Deployments*.

   The *Edit Product Deployments* screen appears.
4. Choose *Add New*.

The *Add New Product to Runtime Server* dialog appears.

5. Choose  for the product.
   The *Object Search* dialog appears.
6. Choose *Select* when you have found the correct product.
   Its name appears in the *Add New Product to Runtime Server* dialog.
7. Enter a name for the server in the *Server Name* field.
8. Enter the user ID in the *User Name* field.
9. Enter the password for the user in the *Password* field.
10. Save your changes.
    The new association appears in the list in the *Product Manager* screen.
11. Test the connection.
12. Choose *Update*.

    A confirmation dialog appears.
13. Choose *OK*.

16.3 Assembling Components in the Planbooks and the Service API Rule

You need to assemble the components of your product structure in your product repository and configure the Service API rules to access data from these components.

Context

> **i Note**
>
> If you are using the out-of-the-box solution of the Home Content sample product, you can skip steps 1-7.

Procedure

1. Go to ▶ *Studio* ▶ *Content Repository* ▶ `<company library>` ▶.
2. Open *Components*.
3. Create a component named *Plan Code* with the modifier type *Component Base*. Similarly create other components, as per your product structure.
4. Assemble the created components in *SAP Insurance Planbook Base* in a anchored fashion, as per your product hierarchy.
5. Upload the values in the assembled components in the *Home Content Planbook template*.

6. After uploading the data in the components, *Save* and then *Commit* all the data rows.
7. Select the entries in the components as per your product specification for activation.
8. To access the data from the components, you have to configure service API rules.

16.4 Displaying Benefits on eApps

You can configure your product to display benefits in the eApps.

Context

To display the benefits on the eApps proceed as follows:

Procedure

1. Go to *Studio* > *System Repository* > `<company library>`.
2. Open *Components*.
3. Create a component named *Plan Display* with modifier type as *Component Base*.
4. Upload the values in the assembled components in your Home Content plan book template.
5. After uploading the data in the components, save and commit all of your data value rows.
6. Select the entries in the components, as per your product specification, and activate.

Results

You will see the list of Benefits in the eApps during runtime.

16.5 Linking a Planbook to a Marketable Product

You will need to link the planbook with a marketable product.

Context

To link a planbook with a marketable product, proceed as follows.

Procedure

1. Open the *Underwriting Application Configuration*.
2. Navigate to *Available Products* > `<marketable product>` > *SAP Planbook Link*.
3. Configure the Key, Planbook Name, Planbook version, Effective Date and Expiry Date.

> **i Note**
>
> Ensure the planbook you have created is active by marking the corresponding checkboxes.

Results

Your planbook is now linked to your marketable product.

16.6 Versioning a Planbook

This topic describes how to version a planbook.

Context

When you make change to the planbook, such as changing or deleting reference objects, you are making a version of the planbook. You can create a new planbook version by copying your existing planbook.

To create a new planbook version proceed as follows:

Procedure

1. Navigate to *Studio* > *Content Repository* > *SAP S4 Library* > *Product* > *Reference Object* > *Planbook*.
2. Right click on the planbook and select *Copy* from the context menu.
3. Click the icon  in the *Object List Toolbar*.
4. Enter your new planbook version number in the pop-up screen to create your new planbook version.
5. Link your new Planbook with your marketable product (see Linking a Planbook to a Marketable Product [page 310]).

> **i Note**
>
> You can have as many versions as you want for your planbook. While creating multiple planbooks, ensure that you have set the correct *Expiration Date* in the previous version of your planbook. During

> runtime, the system identifies the applicable planbook based on the *Effective Date* you have set in your version.

Results

You will see the new version of plan book in the `Reference Object Planbook` folder and the new version of the planbook is linked to the marketable product.

17 Ratebooks

You can define a ratebook that can be applied to quotations or counteroffers.

The premium rates for your plan types is calculated in your ratebook.

During the lifecycle of your product, your product may undergo multiple changes. In order to accommodate these changes, you can create multiple ratebooks for your product. During runtime, system uses the *Effective Date* of your ratebook version to select the right version of your ratebook.

Related Information

Creating the Ratebook Book (Reference Object) and Inheritance Structure [page 313]
Registering a Ratebook [page 314]
Assembling Components in Ratebooks for Premium Rates and the Service API Rule [page 315]
Linking a Ratebook to a Marketable Product [page 316]
Versioning a Ratebook [page 317]

17.1 Creating the Ratebook Book (Reference Object) and Inheritance Structure

The following procedure describes how to create ratebook inheritance structure from a reference object.

Context

To create the ratebook (Reference Object) and its inheritance structure proceed as follows:

Procedure

1. Navigate to *Studio* > *Content Repository* > *SAP S4Library* > *Product Studio* .
2. Create a new folder *Reference Object* of modifier type *Product Folder*.
3. In the *Reference Object* folder, create a folder*Ratebook*" with Modifier type *Product Folder*.
4. In the `Ratebook` folder, create your template object with the modifier type *Ratebook*.
5. Depending on your product structure, create ratebooks for Contract, Coverage and Sub-Coverage reference objects.

6. Now create an object. Example: Best Personal Protection Ratebook V1 with modifier type *Ratebook*.

> **i Note**
>
> The new object you have created is from a inherited list. To view this inheritance list, right-click on the object, choose *Properties* and then select *Inheritance List Tab*.

7. Navigate to your *Home Content Ratebook Template* and assemble the reference objects that you newly created in Step 6.

> **i Note**
>
> Ensure you assemble the reference objects in the correct tab in the Product tree. Example: Contract Ratebook is assembled under the Contracts Tab, Coverage Ratebook is assembled under the Coverage Tab, etc.

Results

You can view the *Ratebook* object in the *Reference Object* folder.

17.2 Registering a Ratebook

Your product has to be registered in the Design Time Administrative Console.

Context

To register a ratebook, proceed as follows:

Procedure

1. Open your web browser and go to `<pro_designtime_app_url>/csiroot/admin/`.
2. Log in to the Administrative Console.

   The Administrative Console will open after a short delay.
3. Go to *Product* > *Edit Product Deployments*.

   The *Edit Product Deployments* screen appears.
4. Choose *Add New*.
   The *Add New Product to Runtime Server* dialog appears.

5. Choose  for the product.
   The *Object Search* dialog appears.
6. Choose *Select* when you have found the correct product.
   Its name appears in the *Add New Product to Runtime Server* dialog.
7. Enter a name for the server in the *Server Name* field.
8. Enter the user ID in the *User Name* field.
9. Enter the password for the user in the *Password* field.
10. Save your changes.
    The new association appears in the list in the *Product Manager* screen.
11. Test the connection.
12. Choose *Update*.

    A confirmation dialog appears.
13. Choose *OK*.

17.3 Assembling Components in Ratebooks for Premium Rates and the Service API Rule

You need to assemble the components of your product structure in your product repository and configure the Service API rules to access data from these components.

Context

> **i Note**
>
> If you are using the out-of-the-box solution of the Home Content sample product, you can skip steps 1-5.

Procedure

1. Go to *Studio* > *Content Repository* > `<company library>` > `<marketable product>` > *Components* > *Rating* >.
2. Inside the *Rating* folder, create your new component with modifier type of `Reference`.
3. Create the other components, as per your premium calculation requirement.
4. Assemble the created components in the ratebook that you have created, as per your premium calculation requirement.
5. Upload the values in the assembled components in the ratebook.
6. After uploading the data in the components, *Save* and *Commit* all the data value rows.

7. Select the entries in the components as per your product specification for activation.
8. To access the data from the components, you have to configure service API rules.

17.4 Linking a Ratebook to a Marketable Product

You will need to link the ratebook with a marketable product.

Context

To link a ratebook with marketable product, proceed as follows:

Procedure

1. Open the *Underwriting Application Configuration*.
2. Navigate to the path *Reference* > *Available Products* > `<marketable product>` > *SAP Ratebook Link*.
3. Configure the Key, RatebookName, Ratebook version, Effective Date and Expiry Date.

> **i Note**
>
> Ensure the ratebook you have created is active by marking the checkbox at the beginning of the column.

Results

Your ratebook is now linked to your marketable product.

17.5 Versioning a Ratebook

This topic describes how to version a ratebook.

Context

When you make change to the ratebook, such as changing or deleting reference objects, you version the ratebook. You can create a new ratebook version by copying your existing ratebook.To create a new ratebook version proceed as follows:

Procedure

1. Navigate to *Studio* > *Content Repository* > *SAP S4 Library* > *Product* > *Reference Object* > *Rate Book* >.
2. Right click on the *Ratebook* and select *copy* in the context menu.
3. Click the Icon  in the *Object List Toolbar*.
4. Enter your new ratebook version number in the pop-up screen to create your new ratebook version.
5. Link your new ratebook with the marketable product (see Linking a Ratebook to a Marketable Product [page 316]).

> **i Note**
>
> You can have as many versions as you want for your ratebook. While creating multiple ratebook, ensure that you have set the correct *Expiration Date* in the previous version of your ratebook. During runtime, the system identifies the applicable ratebook based on this *Effective Date* you have set in your version.

Results

You will see the new version of ratebook in the `Reference Object Ratebook` folder and the new version of the ratebook is linked to the marketable product.

18 Rules

18.1 Working with Product Rules

Rules enable you to add business logic to objects in FS-PRO. The main element of a rule is a rule step, which consists of inputs and input ranges (the conditions for making decisions), and output expressions (the outcomes when the conditions are met).

> **i Note**
>
> This section contains advanced material and assumes that you are familiar with the fundamentals of FS-PRO.

You create rules in the modeling tool using the *Rule Painter*. They have a data type of RULE and they use a local data definition.

Product rules enable you to apply business logic to a product. For example, you could create a rule that performs premium calculations for property and casualty insurance products. Product rules are typically used for:

- Ratings
- Referrals
- Attachments
- Attachment interdependencies

You add product rules to components . The rules are applied when the end user accesses the product. To have rules in a component, you create an attribute and assign it one of the rule data types. After creating the rule attribute, you can add a rule to any of the component's values, using the *Rule Painter*.

Related Information

Creating Rules [page 319]
Creating Rules from a Template [page 320]
Viewing Rules [page 320]
Editing Rules [page 321]
Deleting Rules [page 321]
Using the Rule Template Manager [page 321]
Deviating from Rule Inheritance [page 323]
Building all Rules in a Product [page 324]

18.1.1 Creating Rules

The following procedure describes how to create a standard rule.

Procedure

1. Open the product or coverage and navigate to the component .
2. If the component already has a rule attribute created, proceed to step 9.
3. Select the *Attributes* tab.
4. Add an attribute row and name the attribute `SYSATTR_TABLE_NAME` and give it the type **`TEXT`** and a length of **`250`**.
5. Add a data value row.
6. Enter a name for the rule attribute in the *Attribute Name* field.
7. Select RULE from the *Data Type* dropdown list.
8. Save your changes.
9. Select the *Values* tab.
   The attribute rule appears as a normal value column.
10. Right-click in the cell to which you want to add a rule and select *Create*.
    The *Capture Rule Info* dialog appears.
11. Enter a unique name for the rule.
12. Enter the name for the rule in the *Rule Name* field.
13. Enter the name for the original step in the rule in the *First Step Name* field.
14. Select the type of step you want to use from the *Step Type* dropdown list.
15. Select the return type from the dropdown list from the *Rule Return Type* field.
16. Choose *OK*.
    Another *Capture Rule Info* dialog appears, permitting the selection of a data definition for the rule.
17. You can optionally select a data definition to be associated with the rule. Choose *Continue* to skip, if desired.
    The *Rule Painter* appears.
18. Create the rule.

    > **i Note**
    >
    > If in the previous step you associated a data definition with the rule, you can use form. to access data from the parent table, or data. to access data from a child table (in Decision Table steps).

19. Save the rule and exit the *Rule Painter*.
    A rule icon appears in the cell, indicating that the data value row has a rule attached.

18.1.2 Creating Rules from a Template

You can create rules from a template.

Procedure

1. Open the product in the Product Studio and select the component. The only prerequisite to adding rules to a component is that it have an attribute with the data type RULE.
2. Select the *Values* tab.
3. Add a data value row.
4. Select the appropriate rule cell and select *Create*.
5. If you want to use a template as a starting point for the rule, choose *Search* by the *Create Using Template* field.
   The *Rule Template Manager* dialog appears.
6. If using a template, select *My Templates* to use a personal saved template, or perform a search for a saved system template.
7. Choose *View* to review the chosen template, and *Select* to use the template.
8. Enter a *Rule Name* and choose *OK*.
   A dialog opens asking you to select a data definition for the rule, or to choose *Continue* to create the rule without a data definition. The *Rule Painter* screen opens.

Related Information

Using the Rule Template Manager [page 321]

18.1.3 Viewing Rules

You can view any rule (as read-only) in the *Product Tree*'s *Values* tab.

Procedure

1. Open the product or coverage and navigate to the component .
2. Select the *Values* tab.
3. Click in the rule's cell and select *View*.
   The rule opens in the *Rule Painter*.
4. Close the *Rule Painter* when finished.

18.1.4 Editing Rules

You can open a rule in the *Rule Painter* to make changes.

Procedure

1. Open the product or coverage and navigate to the component .
2. Select the *Values* tab.
3. Click in the rule's cell and select *Edit*.
   The rule opens in the *Rule Painter*.
4. Edit the rule.
5. Save your changes.

18.1.5 Deleting Rules

From the *Values* tab, you can delete a rule.

Procedure

1. Open the product or coverage and navigate to the component .
2. Select the *Values* tab.
3. Click in the rule's cell and select *Delete*.
4. If you are prompted to remove associations from the rule, and you select *No* (to keep the associations), you can't reuse the rule name. If you select *Yes* (to remove the associations), you can reuse the rule name.
5. Choose *Yes* to confirm rule deletion.

   The icon is removed from the cell.
6. Save your changes.
   The red flag is removed and the deletion is committed.

18.1.6 Using the Rule Template Manager

With the *Rule Template Manager*, you can select a template as the starting point for creating rules.

It can be accessed two ways:

- By choosing *Search* next to the *Create Using Template* field in the *Capture Rule Info* dialog. This method doesn't permit template editing.

- By selecting *Rule Template Manager* from the *Tools* menu in the Product Studio. This method does permit template editing.

You can search by *Template Name* and *Template Type* from the dropdown list:

- Calculation
- Form Attachment
- Interaction
- Rating
- Referral
- Underwriting
- Look Up
- Screen Content
- Validation
- Other

You can view a list of templates you created by selecting *My Templates*.

You can view Rule Template details by selecting a template under *Available Templates*. It populates the *Rule Template Detail* section, providing the following template details:

- Name
- Type
- Description

In the *Rule Template Detail* section, the following options are available:

View	Opens the appropriate editor, permitting you to view the rule in detail.
Edit	Opens the appropriate editor, permitting you to edit the rule.
Delete	Prompts you to delete the rule, or cancel and return.
Save Detail	Permits you to save the *Rule Template Detail* entries with the rule.

> **i Note**
>
> When editing, you can only change the *Name* and *Description*. Choosing *Save Details* saves those changes.

Rule templates won't compile. Edits must be made to variable, table, and column references after the rule is created using the template corresponding to the product's data table structure. Text in black or blue (non-script rule language keywords) should be modified.

Related Information

Creating Rules from a Template [page 320]

18.1.7 Deviating from Rule Inheritance

Products in FS-PRO use an inheritance feature, allowing you to create new products or coverages based on existing ones.

> **! Restriction**
>
> You should avoid using rule deviations, as this feature will be deprecated in a future version.

The new product or coverage is extended or customized, building on previous work rather than starting from scratch. Rules enable you to add business logic, perform database lookups, or run data functions providing default values, validations, and calculations to objects in FS-PRO.

With rule inheritance deviation, you can deviate an inherited rule to modify the logic of different parts of the rule. This can simplify rule creation and maintenance. For example, in ISO rating logic, the full rating is provided at the country-wide level. However, in some states the ratings may be deviated. These deviations may include:

- Changed lookups
- Changed logic in a small section of code
- Changed overall logic

Without rule inheritance deviation, in these scenarios you must override the full rule, making it difficult to maintain. Even if a simple lookup is different for an individual region or state, you would be forced to maintain a different rule for all regions or states.

The following conditions apply to rule inheritance deviation:

- Rule inheritance deviation applies only to rules with Script-type steps, and isn't available for other step types
- Rule inheritance deviation applies to rules inside Product-type objects (such as products, coverage groups, coverages, and reference products).
- Input arguments must be identical for the parent rule and the deviated rules
- Rule inheritance deviation doesn't apply to standalone components (since they typically don't contain rules)
- Rule inheritance deviation doesn't apply to Questionnaire rules (Product Query rules and Filter rules)
- New steps can't be added in a deviated rule

Deviating and Modifying an Inherited Rule

When working in a product inheritance structure, you can partially override and modify the logic of different parts (the body and/or function) of an inherited rule. The following rules apply:

- When overriding and modifying the logic in the body of the inherited rule, the inherited rule executes the local (modified) version of the body with the original (parent) function.
- When overriding and modifying the logic in the functions of the inherited rule, the inherited rule executes the local (modified) version of the function.
- The function name, input arguments, and return type can't be changed without creating a new function. If these are changed, there is a system notification (syntax error) stating that the changes aren't permitted and that the rule won't be saved.

- You can't delete the original (parent) function. To prevent use of the parent function, create a new function and modify the body of the inherited rule to call this new function.
- You can modify the logic in the body and existing functions and add new functions.
- In all cases of partial overrides, the parent rule isn't impacted by modifying the body and/or functions of the inherited rule. The parent rule continues to execute the local (original) of the body and/or functions.

Modifying a Parent Rule

When working in a product inheritance structure, you can modify the logic of different parts (the body and/or function) of a parent rule. The following rules apply:

- Changes to the parent rule are propagated to all inherited rules unless the change is in an overridden section of the inherited rule.
- When building the modified parent rule, the system checks for and builds all deviated and merged rules (as a result of IPS) as needed.
- A parent rule can't be deleted if it is deviated in the inherited products. In this scenario, an error message generates advising the deviated rules must be deleted before the parent rules.

Auditing Deviated Rules

For audit purposes, the system captures all rule deviations and deleted rule deviations. This information is available by selecting *Tools* > *Audit Trails* from the Product Modeler, selecting the *Values* tab and filtering on Deviate Rule and Delete Deviation in the *Action* field.

18.1.8 Building all Rules in a Product

You validate and build rules individually as part of creating them in the *Rule Painter*. However, the modeling tool contains a feature that enables you to build or rebuild all the rules in a product at once.

Context

In particular, you would want to rebuild all rules in the following situations:

- You have upgraded FS-PRO to a new version that changes the way rules are built.
- You have lost the previously built rules or their XML.
- You migrate to a new application server and need to recreate all the rules.

If a product contains inherited rules, there is also an option to build local rules only.

Procedure

1. Open the product .
2. Select *Build* > *Rules* from the menubar.
   The *Build All* dialog appears with radio buttons permitting selection of *Include committed changes only* or *Include uncommitted changes*.
3. If you want to build local rules only (including deviated rules), select the *Force build local Rules only* radio button.
4. Choose *Build*.
5. When the process is complete, select *Close*.

> ⚠ Caution
>
> Though the `Build Rule` command notifies you if an error occurs and the process stops, `Build Rule` continues building even if errors occur. Errors are highlighted in red, along with the path. It is your responsibility to resolve all the errors and build those rules prior to publishing the product.

18.2 Working with Calculation Rules

A calculation rule is a configurable rule that automatically performs a calculation to determine the value of a field each time the screen is processed.

A calculation rule is configured in the eApp and is attached to a particular screen view via columns in the data definition. The logic is triggered depending on the type of rule defined: a precalculation rule, a post-calculation rule or a form calculation. The execution of the calculation rule is defined in the `eApp Column Info` object

Related Information

Creating Calculation Rules [page 325]

18.2.1 Creating Calculation Rules

You can define calculation rules can be attached to any column in a data definition.

Context

Calculation rules are decision table or script rules that will execute when rendering the view.

Procedure

1. Open the target template.
2. Navigate to *Data Capture* > *eApplication* > *eApp Section* > *eApp Screen* > *eApp Column Info*.
3. Expand the anchored *eApplication* record, *eApp Section* record and *eApp Screen* record.
4. Expand the *eApp Column Info* record.
5. Add a data value row.
6. Specify the column name.
7. Select the `CALCULATION_RULEID` column for the row and choose *Create* from the popup menu.
   The *Capture Rule Info* dialog opens.
8. Create the rule.
9. Build the rule.
10. Exit the *Rule Painter*.
11. If you have multiple calculation rules on this table, in the `CALCULATION_RULE_ORDER` attribute, enter a non-decimal value to indicate the order in which this rule should execute in relation to the other calculation rules.
12. If you want to disable a calculation rule, enter **`0`** in the `IS_CALCULATION_RULE_ACTIVE` attribute.
13. Determine when the rule runs by setting an option in the `RULE_EXEC_MODE` attribute:

Pre&Form (1)	This rule will be executed when navigating to a screen, before a screen is rendered. It will also be triggered when saving the screen.
Pre (2)	This rule will be executed when navigating to a screen, before a screen is rendered.
Post (3)	These rule will be executed when navigating away from a screen, after saving.
Form (4)	The rule gets triggered when you save the screen
Pre&Post (5)	This rule will be executed when navigating to a screen, before a screen is rendered. This rule will also be executed when navigating away from a screen, after saving

    Once a type is selected from the dropdown list, the number value of the selected choice will be displayed.
14. Specify the main axis table for the specified column name in the `SYSATTR_TABLE_NAME` attribute.
    This is a required attribute.
15. Save your changes.

Results

A rule icon appears in the cell, indicating that the data value row has a rule attached.

If the eApp screen where this rule is attached is not displayed in Runtime, the rule will not be executed.

18.3 Working with Underwriting Rules

An underwriting rule is a configurable rule that can be attached at different levels within a coverage-based product to test an underwriting condition and assess the insurance risk.

Based on the conditions in the given policy, these rules provide outcome whether the risk is Accepted, Declined, or Referred.

Underwriting rules are a part of the system underwriting process that identifies violations and provides recommendations.

Related Information

Defining Underwriting Rules [page 327]
Specifying Checklist Items [page 328]
Exchanging Information Between FS-PRO and the FS-QUO Front End [page 329]
Orchestrating Underwriting Rules [page 330]
Defining Underwriting Groups [page 331]
Defining Underwriting Case Permissions [page 332]

18.3.1 Defining Underwriting Rules

You can define multiple underwriting rules at different hierarchical levels in the coverage-based product structure.

For example, you can define rules at the following levels:

- Policy
- Contract_Bundle
- Contract
- Coverage
- Sub-coverage

Underwriting rules can also be defined for the assembled components such as insured object, policyholder.

> **i Note**
>
> The levels and components listed above are provided out-of-the-box. Underwriting rules can be defined for additional levels and assembled components depending on customizations your company made to the product.

Underwriting rules are boolean rules that receive input as a data object such as ABDAPOLICY, ABDAPOLPR, or ABDACOV. This is based on whether the rules are defined at product, contract, or coverage level.

On the *Underwriting Rules Component* screen, you can do the following tasks:

- Define rules in the *Rule Painter*.
- Associate a violation code with an underwriting rule that you can map to a message in multiple languages.
- In the `Rule Decision` and `Rule Negative Decision` columns, you can choose any of the following values: Accepted, Declined, or Referred. One of the following occurs:
  - If the boolean rule returned the result as True during execution, the underwriting decision for this rule is chosen from the *Rule Decision* field.
  - If the boolean rule returned the result as False, then the underwriting decision will be from the `Rule Negative Decision` attribute.
- In the `Associated Level` column, you can chose an associated level for the underwriting rule.

18.3.2 Specifying Checklist Items

A Checklist is a related information item for the underwriter to examine in case violation of an underwriting rule occurs.

You can do the following tasks:

- Associate multiple checklist items with an underwriting rule.
- Specify the descriptions of codes separately for checklist items.
- Define a checklist item as Mandatory or Optional. Mandatory checklist items must be completed by the underwriter during manual underwriting.
- Specify violation and checklist descriptions.

A set of reference components are assembled in the `SAP Insurance UWA Product Template`, inherited by all products, to provide the descriptions mapping for the component codes in underwriting. These components are:

Violation Item	Provides the descriptions for Violation Codes.
Check Item	Provides the descriptions for Check List Codes.

On the Violation Item domain table you can select *Enable Multi-Language* to provide descriptions for underwriting violations and checklists in different languages. You can then assemble these mapping components into products that support different languages and adapt their text to corresponding language.

You can also create Service APIs for mapping descriptions. You can also create Service APIs in product templates that help to map system codes for Underwriting Violations and Check List items to their language-specific description..

A set of Service APIs are provided in the `SAP Insurance UWA Product Template` and inherited by all products so you can get the details of the difference reference components that were defined. This information is specified in the `Product Services` folder of the template (*SAP Insurance UWA Product Template* > *Product Services* > *Service API*). These APIs include the following:

- `getRuleCategory`
- `getRuleObectType`
- `getUnderwritingGroup`
- `getViolationMessageList`

18.3.3 Exchanging Information Between FS-PRO and the FS-QUO Front End

You can exchange information between FS-PRO and the FS-QUO front end server apps. Each Rule execution creates a node for `UWRuleProcessingResult` and the associated checklist items.

The orchestration rules return the following responses from the underwriting rules:

Rule_Name Rule name that was executed.

Rule_Id Rule ID for the rule that was executed.

Rule_Category Returns the value associated with `Rule Category` attribute selected in the `Underwriting Rules` component.

Rule_Level Returns the value associated with `Associated Level` attribute selected in `Underwriting Rules` component.

Rule_Violation_Indicator Indicates the result of the rule execution, which is true or false.

It is true if the decision was Referred or Declined.

Rule_Decision Returns the decision for the underwriting rule.

If `Rule_Violation_Indicator` is true, the value of this field is taken from the chosen value for `Rule Decision` attribute in the `Underwriting Rules` component.

If `Rule_Violation_Indicator` is false, the value of this field is taken from the chosen value for `Rule Negative Decision` attribute in the `Underwriting Rules` component.

Rule_Object_Path Specifies the path of the object in the policy data object on which the given rule was executed.

The structure of the path is `<PKID>-<Object Type>/<PKID>-<Object Type>/` and so on.

Executed_Object_TemplateID Specifies the template ID or the product base ID of the data object in Policy Data on which the rule was executed.

Executed_Object_ID Represents the PK_ID field of the object on which rule was executed.

For example, if there are multiple coverages, the PK_ID is the coverage on which a rule was executed.

Executed_Object_Type Represents the type of the object on which the rule was executed. For example, coverage, contract, and insured object.

Rule_Violation_Code Returns the value associated with the `Violation Code` attribute selected in the `Underwriting Rules` component.

Checklist Represents a node for holding a list of check list items associated with an underwriting rule.

CheckListItem If `Rule_Violation_Indicator` is true then there are zero or more entries of check list items depending on whether a check-list item is associated with a rule in the `Underwriting Rules` component.

Code	Returns the value associated with the `Code` attribute selected in the `Check List Items` component.
Role	Returns the value associated with the `Role` attribute selected in the `Check List Items` component. Currently this field isn't being used.
Mandatory	Returns the value associated with the `Mandatory` attribute selected in the `Check List Items` component.

UnderwritingDescision One node of underwriting decision is returned that describes the overall underwriting decision

Decision	Provides the overall decision of the underwriting process. The valid values are Accepted, Declined, and Referred.
Decision_Reason	Provides a message describing the overall decision of the underwriting process.
UWGroup	Provides the name of the underwriting group that should handle the system underwriting result.

18.3.4 Orchestrating Underwriting Rules

When you orchestrate underwriting rules, the following occurs:

1. Rule orchestration starts from a Service API called `executeUnderwritingRules` in the `Service API` component.
2. This issues the execution of child services that are defined inside the underwriting rules executor component at different levels of the product structure.
3. The rules defined inside the underwriting rules executor cause the execution of the underwriting rules defined inside the `Underwriting Rules` component.
4. Depending on the underwriting rules defined, this method will receive an XML result based on the MessageOutput schema having the following items:
   - The attributes for the `Underwriting Rules` component
   - The rule result of the underwriting rules defined in the `Underwriting Rules` component.
   - The rule decision associated with the underwriting rule.
   - The checklist items associated with underwriting rules depending on the result of a particular underwriting rule

ExecuteProductUnderwritingRules	Identifies the name of the product from the templateId in the ABDAPOLICY data object
UnderwritingRulesIterator	Helps execute all the rules defined at product level
ExecuteContractUnderwriting	Repeats process at contract level and below
CreateUnderwritingDecision	Creates the final Decision after rules execution

UnderwritingRulesIterator	Identifies all the underwriting rules defined at the specified level, captures information and executes all these rules

5. This method will iterate the XML result and determine the rule decision and rule result of each XML node as one of the following:
   - If the rule decision for a node is Declined, then the overall underwriting decision will be Declined and the underwriting message will state Declined as at least one rule result was declined. In this case, the underwriting group will correspond to the system group.
   - If the rule decision for a node is Referred then the overall underwriting decision will be Referred and the underwriting message will be Referred as at least one rule result was referred. In this case, the underwriting group will correspond to the user-defined group.
   - If either of the first two scenarios don't occur, then the underwriting decision will be Accepted and the underwriting message will be Accepted as there was no underwriting violation. In this case, the underwriting group will correspond to the system group.
6. Based on the rule decision, an underwriting decision node is added to the XML result and returned as a response to the calling service.

> **i Note**
>
> Underwriting decisions of Declined or Referred are treated as violations, and checklist Items are attached in the node's XML response.

18.3.5 Defining Underwriting Groups

A reference component for underwriting group is available in the product. In this component, you can define the system group (denoted by SYS, the group for all Declined and Accepted underwriting decisions) and any other user-defined group (denoted as HHLD, the group for all Referred underwriting decisions).

> **i Note**
>
> The system group indicator can be Y or N indicating whether or not the group is a system group.

When you define underwriting groups, the following occurs:

- Group assignment logic checks for the underwriting decision.
- If the decision is Accepted or Declined, then a system group is chosen.
- If the decision is Referred, then a user-defined group is chosen.

You can enhance this logic by adding more groups and then creating new decision criteria for group assignments.

> **i Note**
>
> A Service API `getUnderwritingGroup` is available in the product that specifies the system and user-defined groups defined for the product.

18.3.6 Defining Underwriting Case Permissions

You can apply business-specific logic in a marketable product to assign different authority levels to users. This permission determines which underwriting groups can make underwriting case decisions and is configured via a product rule in the `SAP Insurance UWA Product Template`.

Underwriting cases will be automatically assigned to the underwriter groups that are derived from the business rules in the marketable product. Only the underwriters that are part of the specified underwriter group will be allowed to make the underwriting decision for the underwriting cases.

Other underwriter and agent groups, configured in the marketable product, will have restricted access to the underwriting case for assisting the authorized underwriter to make the decision on the underwriting case. For example, they will be able to provide details and supporting documents (evidences).

You enable underwriting case permissions in the following rule: `<product>` > *Configuration* > *Product Services* > *Service API* > *getUWGroupPermissions*.

The `getUWGroupPermissions` rule is run after the *Submit for Underwriting* option is selected in the app. It gets the insured amount for a case, and compares that to the range defined in the `<product>` > *Reference Data* > *Domain Tables* > *Underwriting Group Permissions* component. The insured amount range is determined in the `Sum Insured Min` and `Sum Insured Max` attributes.

The `Group Name` attribute determines the Underwriting Group being evaluated. The `Permission` attribute will determine the access that will be provided for the group members. There are 3 levels of permission:

UWReadOnly	Allows read-only access to underwriting cases.
UWContribute	Allows access to a limited feature set for underwriting cases.
UWFullAccess	Allows full access to underwriting cases.

The following list indicates the affected screen controls in the `My Underwriting Worklist` app:

- The `Confirm item` action in the *Checklist* tab on the *Underwriting Item* screen.
- The *Add* item action on the *Evidences* tab
- The *Delete* item action on the *Evidences* tab
- The *Accept* button
- The *Decline* button
- The *Notes* button
- The *Reassign* button
- The *Counteroffer* button
- The *Needs Improvement* button

The `My Underwriting Worklist`'s screen controls will be enabled, disabled or hidden for a user depending on the permissions that they have been granted.

For more information about designing data visiblity, see the Coverage-based Development Guide.

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

18.4 Working with Transformation Rules

A transformation rule is a configurable rule that synchronizes Policy data model records.

> **i Note**
>
> The use of transformation rules is only relevant to coverage-based products.

Transformation rules are found in all objects, with each object hierarchy containing a `Transformation` folder introduced within the `SAP Insurance UWA Product Template`.

Rather than deleting and creating all records every time when data needs to be transferred from the Quote data model to the Policy data model, standard synchronize methods are implemented in all main axis level transformation rules to update the Policy data model using *PRO_ID* or *PRO_SRC_ID* to identify the matching records in the Policy data model.

Overrides are typical for these rules to comply with the differences that come from the specific UI configured for a particular product.

These rules are called as a precursor to rating within FS-QUO. When rating is called from FS-PM, these rules are skipped, as there is no need for transformation.

18.5 Working with Initialization Rules

An initialization rule is a configurable rule that is called to create the Policy data model data records when transforming the data from the Quote data model to the FS-PM Policy data model.

> **i Note**
>
> The use of initiliazation rules is only relevant for coverage-based products.

Initialization rules are found in all objects, with each object hierarchy contains a `Transformation` folder introduced within the `SAP Insurance UWA Product Template`. Typically, the rules contained within the `Transformation` folder do not need to be overridden, with the exception of the `initQuoteData` rule (in the Product Hierarchy)

The rules contained within the `Transformation` folder are called at first by the Initialization Rule found on the QuoteEApp, but are also called from the transformation rules at the beginning of the rating process.

Call the `setDefaultValues` stem to capture the information from the `SAP Column Default Values` components.

18.6 Understanding Utility Rules

Utility rules are provided by FS-QUO specifically for solutions that integrate SAP Commerce, Financial Services Accelerator with SAP for Insurance. You won't need to create your own utility rules.

> **i Note**
>
> The use of utility rules is only relevant for coverage-based products.

Utility rules are only found in the Product Hierarchy.

The rules should **not** need to be overridden.

The two primary utility rules are:

`get<ObjectType>Path`	These rules are used to create the path for the current object (from within the product's structure) They use the *ABDA*.PRODUCT_BASE_ID* (populated with the `Template Id`) rather than the *Name* of the component, to allow for changing of object names (whether through inheritance or for other reasons). The *PRO_ID* was not used for potential backwards compatibility reasons for products that we created without the `PRO_ID` component.
`set<Object Type>TemplateId`	These rules are used assign the `Template Id` from the object's `SAP <Object Type> Id` component into the appropriate *ABDA*.PRODUCT_BASE_ID* field. The rules make use of the *PRO_ID* to do this.

18.7 Working with the *Rule Painter*

The *Rule Painter* is where you create and edit rules for FS-PRO products. Rules encapsulate business logic. They take inputs, process them, and return a result. A result can be used to make a decision, modify existing data on screen, or be used as an input to other rules.

Rule Elements

Every rule consists of a return data type, a version number, an activation date, and one or more rule steps. A rule step represents a single unit of logic that yields a result based on a combination of inputs. In turn, a rule step is composed of several containers:

Specification	The business logic of the step
Input	The inputs used in the Specification container
Output	Identifies the destination for the result of the specification

Property	Lists properties available to apply to the output

Return Data Types

Each rule returns the data type that you specify when first creating the rule. The original step in the rule defaults to this type. Any further steps that you add to the rule can use different return data types. The return types are:

- Boolean
- Database Table
- Date
- Number
- Object Table
- String
- XML
- Void (This return type is applicable only to decision steps, and only if it's not the last step)

> **i Note**
>
> A restriction when adding steps to a rule is that any step that you add below the rule's original step must have the same return data type as the rule. Steps that you add above the original step can have any return data type. Put another way: the last step in a rule must always have the same return data type as the rule.

Rule Step Types

The rule step types allow you to express business logic using a broad set of means. A rule step can be any of the following types:

Script Steps	Creates complex expressions using scripting language This step type supports all return data types
Assignment Step	Provides for a direct assignment where conditions aren't needed This has a return data type of NUMBER only
Decision Table Step	Contains a decision table, which is a way of expressing conditions without using IF/THEN statements This step type supports all return data types

Expressions

Because rule expressions use a dot notation syntax, the rule object is known as a stem. For example, a product rule starts with the stem `PRODUCT.`, a path rule starts with the stem `FLOW.`, and a text rule with the stem

`STRING.`. The stem `step.` refers to the result from a previous rule step in a multi-step rule. The stem `form.{<fieldname>}` refers to a data definition field.

Output expressions identify default values or define calculations that return a value after the expression executes.

In the case of rule steps that depend on the results of other rule steps, they can be linked in a multi-step rule. Typically, this is used with calculation rules, to deal with complex rules that can be broken down into simpler steps.

Custom stems

When creating custom stems, don't use any of the system stems (for example, `DATA.`, `TREE.`, `FORM.`, `QUERY.`, `RULE.`) as part of the stem name.

Related Information

Opening the Rule Painter [page 336]
Rule Objects [page 338]
Using Operators in Rules [page 339]
The Expression Editor [page 340]
Rule Unit Testing [page 362]
Rule Tracing [page 363]
Testing XPaths [page 366]
Rule Result Caching [page 367]

18.7.1 Opening the *Rule Painter*

There are various ways to open the *Rule Painter*.

You can open the *Rule Painter* from the *Values* tab of any component that has attributes with data types defined to one of the rule types. For example, from the `eApp Column Info` component, which can contain attributes defined as `Calculation Rule` and `Validation Rule`.

You can also open the *Rule Painter* from the questionnaire *Properties* dialog.

18.7.2 Rule Painter Screen

18.7.2.1 *Rule Painter* Toolbar

The functions contained in the *Rule Painter* toolbar apply to the rule and all the steps contained in it.

Icons on the Toolbar

Icon	Name	Description
	Build Rule	Compiles the rule, generating its final code and activating the rule.
	Save as Template	Saves the rule as a template.
	Template Manager	Opens the *Rule Template Manager*.
	Print	Prints a report containing all details of the rule.
	Show/Hide Property Window	Displays the property panel for the rule at the bottom of the screen.
	Refresh	Reloads the browser cache and forces the *Rule Painter* screen to redraw.
	Show/Hide Documentation	Toggles the display of the documentation area in all the steps contained in the rule.
	Technical View/Business View	Toggles the display between the Technical View (the default view) and the Business View (each step's name and specification).
	Show/Hide Steps	Toggles the display of the details of the steps in the rule.
	Rule Unit Testing	Runs the rule unit testing facility.

18.7.2.2 Rule Step Parts

A rule consists of one or more steps. Each step appears as a group of headings and fields.

Each step is a self-contained unit consisting of several standard parts:

Step name	A title identifying the purpose of the step

Documentation	Detailed comments describing the purpose of the step, plus any other useful information.
Specification	The core logic of the step, its form dependent on the step's type.
Input	The data the step requires to perform its function.
Output	The data the step produces.
Property	There is a single property available, *Rounding*, which allows you to round up or round down numbers. This property is only applicable to Assignment steps.

Related Information

Creating Assignment Steps [page 350]

18.7.3 Rule Objects

You can define rules based on the following objects:

Form	Enables you to select a field from the master, detail, extension, or reference table. You invoke the Form rule object by entering `form.` in the expression field. A list of the column names from the master table is displayed. You can use this object when you are creating validation and calculation.
Tree	Enables you to view all the tables and fields of a data definition in a tree format; allows you to use "set" functions (for example, Sum, Count, or Min) on any child table relative to the master table. You invoke the Tree rule object by entering `tree.` in the expression field. A list of the tables and column names that appear in the data definition is displayed in a tree format. You can use this object to access all the rows of any table in the data definition, regardless of the table the rule belongs to. For example, you could apply `Sum()` against all the rows of a table, whether it is a parent or a child relative to the rule's location.
Data	Enables you to view all the tables (the master, detail, extension, and reference tables) and fields of a data definition in a tree format; allows you to use "set" functions (for example, Sum, Count, or Min) on any child table relative to the current table, and to access single values inside a parent table. You invoke the data rule object by entering `data.` in the expression field. A list of the tables and column names that appear in the data definition is displayed in a tree format. You can use this object to access all the rows of a child table relative to the table the rule belongs to, or to access a single row in a parent table.

> **i Note**
>
> Though `tree.` and `data.` are almost identical in function, there are subtle differences between them: `tree.` has greater scope, allowing functions to operate on all the rows in

> a child table, regardless of what table its rule is located in. On the other hand, `data.` is constrained by the rule's location.

Function Stem

Enables you to select from a list of functions, grouped according to category, such as number, date, string, and aggregate functions.

You invoke the function rule objects by typing the appropriate stem in the expression field.

Alternately, you can select  to open the *Rule Stem* popup, which displays all the available stems.

For more information, see the Rule Painter Functions [page 469] section.

Selecting a stem places it in the expression field. Adding a period (.) after the stem invokes its function list, from which you can select the function you need.

Product Services

Enables you to invoke a user-defined product service in Product Services from a rule.

You invoke the Product Services object by entering `ProductService.` in the expression field. Use this rule object to invoke user-defined product services from a rule.

Step

Enables you to execute any other step or previous steps in the rule.

You invoke the step rule object by entering `step.` in the expression field. You can use this object to pass results from a previous step in a rule. This ability is particularly useful when combined with an input step.

Input

Enables you to select an input or create a new input for a rule.

You invoke the Input rule object by entering `input.` in the expression field. An input chooser dialog appears. You use this object when you are creating a rule.

18.7.4 Using Operators in Rules

Operators enable you to define conditions or perform calculations. You can use the following operators in expressions in the *Rule Painter*:

`()` Encapsulates a greater than or less than notation. For example, (5:*) designates greater than 5.

`[]` Encapsulates a greater than or equals to, or less than or equals to notation. For example, `[5:*]` designates greater than or equal to 5.

The two bracketing methods can be combined. For example, `[0:5]` designates less than 5 but greater than or equal to 0, and `[5:*]` designates 5 and greater.

`>` Left operand is greater than right operand.

`<` Left operand is less than right operand.

`=` Left operand is equal to right operand.

`<>` Neither operand is the specified value.

`>=` Left operand is greater than or equal to right operand.

`<=` Left operand is less than or equal to right operand.

AND A Boolean function which is true only if all its arguments are true.

OR	A Boolean function which is true if any of its arguments are true.
/	Divides the left operand by the right operand.
*	Multiplies the operands.
-	Subtracts the right operand from the left operand.
+	Adds operands.
%	Result of dividing the left operand by the right operand.
IN	A Boolean function which is true only if its input is contained in a specified range.
NOT	A Boolean function which is true only if its input is false.
space	Inserts a space.

18.7.5 The Expression Editor

The *Expression Editor* pops up when you select inside a step's *Specification* field. The editor enables you to define expressions using rule objects, operators, and alphanumeric strings.

The *Expression Editor* consists of a tools area, from which you choose the data fields, functions, or operators to create an expression, and a field that displays the expression.

The tool icons are as follows:

Icon	Name	Description
	*Revert Unsaved Changes*	Undoes changes since last save
	*Apply*	Validates the expression and adds it to the decision table

Related Information

Expressions [page 340]
Automatic Data Type Conversion [page 341]

18.7.5.1 Expressions

You use expressions to identify default values or to define calculations that return a value after the expression executes. Expressions consist of one or more of the following: constants, variables, operators, and functions.

The most basic use of an expression is with default values. For example, when you identify a default value, you can specify an attribute of a table (such as userID) or a function (such as `getCurrentDate`).

You can also use expressions to define complex calculations that consist of fields and/or functions, as well as combine them with operators.

Related Information

Using Operators in Rules [page 339]

18.7.5.2 Automatic Data Type Conversion

The *Rule Painter* automatically converts certain data types when values are assigned in expressions and within Assignment rule steps, removing the need to insert conversion functions.

For example, in an assignment step, if the Specification container held the statement: `RESULT = "12.30.2020"` in the Output container, you could enter the following expression: `form.EffectiveDate` and even though `EffectiveDate` is a date field, you wouldn't need to wrap it in a conversion function.

The following table summarizes the data types affected by the automatic conversion feature:

Original Data Type	Destination Data Type
TEXT	DATE or NUMBER
NUMBER	TEXT
DATE	TEXT

When automatically converting data to the date type, you have the choice of date formats described in the following table:

Format	Example
MM-dd-yyyy	12-30-2020
MM-dd-yyyy HH:mm:ss	05-22-2020 15:30:20
MMM-dd-yyyy	Jan-20-2020
MMM-dd-yyyy HH:mm:ss	Feb-18-2020 03:15:18
MM/dd/yyyy	12/30/2020
MM/dd/yyyy HH:mm:ss	05/22/2020 15:30:20
MMM/dd/yyyy	Jan/20/2020
MMM/dd/yyyy HH:mm:ss	Feb/18/2020 03:15:18

18.7.6 Working with Rules

18.7.6.1 Creating Rules

Rules can either be created manually or by using a template. This topic details how to create one manually.

Procedure

1. Click and select *Create* from the popup menu.
   The *Capture Rule Info* dialog appears.
2. Enter the name for the rule in the *Rule Name* field.

   The rule name must be unique in the system.

   The only characters allowed for a rule name are A-Z, a-z, 0-9, underscores (_) and blank spaces.
3. Enter the name for the step in the *First Step Name* field.
4. Select the data type that you want the rule to return from the *Rule Return Type* field.
5. Choose *OK*.

Results

The *Rule Painter* appears and you can start to develop the rule.

18.7.6.2 Creating Rules from a Template

Rules can either be created manually or by using a template. This topic details how to create one from a template.

Procedure

1. Select a rule attribute value field and select *Create*.

   The *Capture Rule Info* dialog opens.
2. Enter the name for the rule in the *Rule Name* field.

   The rule name must be unique in the system.

   The only characters allowed for a rule name are A-Z, a-z, 0-9, underscores (_) and blank spaces.
3. Enter a search term into the *Create Using Template* field.

4. Choose *Search*.
   The *Rule Template Manager* dialog opens.
5. Select the template from the *Available Templates* section.
   Template details appear in the *Rule Template Details* section.
6. Choose *Select*.
   The *Rule Template Manager* dialog closes.
7. Choose *OK*.

Results

The *Rule Painter* appears and you can start to develop the rule.

18.7.6.3 Renaming Rules

This procedure details how you can rename an existing rule.

Procedure

1. Open the rule in *Rule Painter*.
2. Open the *Properties* panel:
   - Choose .
   - Go to *View* > *Property Window* .
3. Edit the value in the *Rule Name* field.

   The rule name must be unique in the system.

   The only characters allowed for a rule name are A-Z, a-z, 0-9, underscores (_) and blank spaces.
4. Save your changes.

18.7.6.4 Viewing Rule Information

This procedure details how to view a rule's information.

Procedure

1. Open the rule in *Rule Painter*.
2. To change the view mode, choose the *Technical View/Business View* icon.
   Rule Painter provides two ways to view a rule:

Technical View	All step containers appear.
Business View	Only step names and specifications appear.

3. To hide or show all the information about the steps in a rule, choose the *Show/Hide Steps* icon.

18.7.6.5 Adding Comments to Rules

The *Rule Painter* provides a documentation area inside every rule and each step in the rule. You can use these fields to comment your rules.

Context

These fields are hidden by default.

The *Comments* text editor supports multiple languages.

Procedure

1. Open the rule in *Rule Painter*.
2. Choose the *Click to Add* link.
   A text editor appears.
3. To view existing comments, choose the *Show/Hide Documentation* icon.
4. Choose *OK*.

> **i Note**
>
> In the *Rule Painter*, your changes are saved automatically.

Results

The editor closes and your comments appear in the rule documentation area.

18.7.6.6 Building and Activating Rules

You can build and activate a rule.

Procedure

1. Open the rule in *Rule Painter*.
2. Build the rule.
   - Choose the *Build Rule* icon
   - Go to *Rule* > *Deploy*.

Results

The rule is built.

To view the generated Java code, choose *Display Details*.

18.7.6.7 Printing a Rule

You can print a rule.

Procedure

1. Open the rule in *Rule Painter*.
2. Choose the *Print* icon.
   The *Print preview* window appears.
3. Choose *Print*.

18.7.6.8 Rules and Inheritance

Build the metadata first so rules using the data will build successfully. For a rule with a deviation or merge, metadata must be built at all levels before the rules at all levels can be successfully built. Otherwise, the rules will fail when built.

Since building all rules doesn't build deviated or merged rules in lower-level products, if a parent rule has been modified you must rebuild all deviated and merged rules to get the latest rule inheritance data. The rules should be built from the parent rule in the *Rule Painter* or from the products containing the deviated/merged rules.

18.7.7 Working with Rule Steps

18.7.7.1 Adding Rule Steps

A rule can contain multiple steps. You can add a step either before or after an existing step.

Procedure

1. Open the *Rule Painter*.
2. Right-click the name bar of the step above or below which you want the new step to appear, and select *Add Step Before* or *Add Step After*.

   > **i Note**
   >
   > Any step that you add after (below) the rule's original step must have the same return data type as the rule. Steps that you add before (above) the original step can have any return data type.

   The *Step Details* dialog appears.
3. Enter the name for the new step in the *Step name* field.

   The name must be unique in the rule.
4. Select the type of step you want to add from the *Step Type* dropdown list.

   Your choice may alter the remaining fields in the dialog.

   > **→ Tip**
   >
   > A rule can have any combination of step types.
5. Select the data type the step outputs from the *Step return type* dropdown list.
6. If you are adding a decision step, in the *No of Conditions* field, indicate how many input rows the table requires.
7. Choose *Apply*.

Results

The new step appears in the rule.

18.7.7.2 Renaming Rule Steps

Context

> → Tip
>
> To hide all the information in a step other than the name bar, right-click the name bar of the step that you want to expand or collapse, and select *Expand Step* or *Collapse Step*.

Procedure

1. Open the rule in *Rule Painter*.
2. Select the step name.
   The *Expression Editor* appears.
3. Change the name as necessary.
4. Press `Enter`.

18.7.7.3 Moving Rule Steps

Rule steps are executed from top to bottom. Be sure that you arrange the steps to execute in the order required by their internal logic.

Context

If the steps are in the wrong order you will receive an error message.

> → Tip
>
> To hide all the information in a step other than the name bar, right-click the name bar of the step that you want to expand or collapse, and select *Expand Step* or *Collapse Step*.

Procedure

1. Open the rule in *Rule Painter*.
2. Right-click the name bar of the step that you want to move, and select *Move up* or *Move down*.

> **i Note**
>
> If a rule step contains a `step.` stem, the rule step must remain below the rule step that the stem references.

18.7.7.4 Deleting Rule Steps

This procedure details how to remove a step from a rule.

Context

> **→ Tip**
>
> To hide all the information in a step other than the name bar, right-click the name bar of the step that you want to expand or collapse, and select *Expand Step* or *Collapse Step*.

Procedure

1. Open the rule in *Rule Painter*.
2. Right-click the name bar of the step that you want to delete and select *Delete step*.

Results

The step is removed from the rule.

18.7.7.5 Adding Properties to a Rule Step

Some step types have a *Property* container, which holds a list of properties appropriate to the data type the step returns. For example, if the step returns a text value, the text-related properties appear in the container.

Context

The only property available is Rounding, which you can use to specify the number of decimal places in numeric outputs. Enter **0** for no decimal places, or a number to indicate how many places should appear; for example, enter **2** to have two decimals.

> → Tip
>
> To hide all the information in a step other than the name bar, right-click the name bar of the step that you want to expand or collapse, and select *Expand Step* or *Collapse Step*.

Procedure

1. Open the rule in *Rule Painter*.
2. Select the technical view. If necessary, choose the *Technical View/Business View* icon.
3. Locate the step's *Property* container.
4. Beside the property you want to use, select the empty field.
   The *Expression Editor* appears.
5. Enter the appropriate property setting.
6. Save your changes.
7. Close the *Expression Editor*.

Results

The new setting appears in the property.

18.7.7.6 Adding Comments to Rule Steps

You can add documentation to a rule step.

Context

> → Tip
>
> To hide all the information in a step other than the name bar, right-click the name bar of the step that you want to expand or collapse, and select *Expand Step* or *Collapse Step*.

Procedure

1. Open the rule in *Rule Painter*.
2. If you want to see the existing documentation in a step, right-click the step's name bar and select *Show Comments*.
3. To add a comment to a step, choose the *Show/Hide Documentation* icon.
4. Choose the *Click to add Step Description* link.
5. Add your comments.
6. Choose *OK*.

18.7.7.7 Creating Assignment Steps

Use an assignment step to create simple mathematical expressions and assign their results to an output. The expressions can only use addition, subtraction, multiplication, division, and exponents.

Context

The only data type that an assignment step returns is number.

Procedure

1. Open the *Rule Painter*.
2. Select the area under the step name.
   The *Assignment Rule Painter* appears.

3. In the *Result* field, enter the value you want to assign and choose *Apply*.

   Every assignment step requires an output, which assigns a destination for the result of the logic in the specification.
4. Specify the output.
   a. In the *Output* container, select *Click to add*.
      The *Data Object Attribute Picker* dialog appears.
   b. Navigate to the data definition and select the field to which you want to assign the result.
5. If you want to define rounding for the step, you need to set the property.

Related Information

Automatic Data Type Conversion [page 341]

18.7.7.8 Understanding Decision Steps

Decision steps are structured differently from the other step types. In a decision step, the Input containers that appear in other step types instead appear as rows within the decision table.

A decision table returns a result on the basis of a combination of inputs. The inputs are on the left side of the decision table; the outputs are on the right side. A decision table is another simple way of representing a complex nesting of `IF/ELSE` program statements. A decision step can return the follow data types: number, text, Boolean, date, data table.

Decision tables consist of the following items:

- Inputs and input ranges (the conditions for making decisions)
- Outputs and output expressions (the outcomes when the conditions are met)
- An assignment (except for validation rules)

Inputs are data that come from outside the activity. Inputs can be entered by end users or supplied by external systems. They can also be conditions or circumstances that trigger a rule.

You define the actions that are to be performed on the data that is entered. Inputs are used in defining validation and calculation rules.

In validation rules, for example, you can specify that the input must be within a certain range to be valid. If the input isn't within the specified range, an error condition occurs. This is the output of the rule.

One or more inputs can also trigger a calculation rule. For example, entries in the State and Price fields can trigger a calculation that displays the price of the item or service including the appropriate state tax.

> **i Note**
>
> Not all calculation rules require inputs. For example, if you are simply adding value A to value B and displaying the sum in another field, there is no need to explicitly specify an input. The entry of the values triggers the calculation.

You can use multiple input ranges to determine multiple outputs. The following example contains two inputs: Gender and Age. For the Gender input, there are two ranges: Male and Female. For the Age input, there are three ranges: 0-24, 25-55, and 56-x.

Input 1	Input 2	Output
Male	Age 0-24	Plan A
	Age 25-55	Plan B
	Age 56-x	Plan C
Female	Age 0-24	Plan D
	Age 25-55	Plan E
	Age 56-x	Plan F

The three ranges that apply for Input 2 are identical for each Input 1 range. In the *Rule Painter*, when you enter a secondary range for a primary input, that secondary range is automatically applied to all primary inputs.

In terms of syntax, the previous table operates like a series of `IF... THEN...` statements. In other words, `IF (Input 1 = Female) AND (Input 2 = [25:55]), THEN (Plan = E)`.

Assignments in decision tables

An assignment allows you to write a decision table's output value to a field in a form. For any rule other than a validation rule, the assignment row automatically appears as the last row in the decision table. To assign an output destination, select the last row of the output column (the assignment cell).

The *Data Object Attribute Picker* dialog appears, displaying the fields of the table to which the rule belongs, and you pick the data definition field you want to assign to. The field name then appears in the *Assign To* cell.

Decision step popup menus

You invoke decision table commands through popup menus accessed in various parts of the decision table.

Related Information

Creating Decision Steps [page 353]

18.7.7.8.1 Creating Decision Steps

The Specification container in a decision step hosts a single decision table. You define the specification by structuring the table (adding inputs and ranges) and adding values and expressions to it.

Procedure

1. Open the *Rule Painter*.
2. Create the decision step.

   It will have a default decision table, appropriate to the return type you specified when creating the step.
3. Right-click the *Name* bar and select *Add Condition*.
   A condition column is added.
4. If your decision table requires multiple conditions, you need to insert columns to it. Do this before you start to define the step. To insert a column to a table, perform the following steps:
   a. Right-click the *Name* bar and choose *Add Condition*.
      The new condition column appears to the left or right of the existing column.
   b. If you want to add more conditions to a decision table but want to determine their placement, right-click the header of an existing conditions column and choose either *Add Condition Before* or *Add Condition After*.
      The new column appears to the left or right of the existing column.
   c. If you want to change the sequence of condition columns, right-click the header of the condition column that you want to move, and choose either *Move Condition Left* or *Move Condition Right*.
      The new column appears to the left or right of the existing column.

   Now, begin to build your conditional expression by adding rows for the OR clauses. Add a separate row for each OR clause.
5. Right-click on the *Condition* header and select *Add Rows*.

   > → Tip
   >
   > If you are specifying ranges for multiple conditions, define the complete set of ranges for the first condition before specifying the addition of a set of ranges for the second condition.
6. Choose *Apply*.
   The rows are added to the condition.
7. Click in a clause row to open the *Expression Editor* to define the condition and return values.

   > ℹ Note
   >
   > Condition ranges can take the form of numerical or alphabetical values or selected or cleared checkboxes. For checkboxes, zero (0) represents unchecked, one (1) represents checked.
8. Save your changes.
9. Select another row to continue building your conditional expression.
10. If you want to add another clause row to the end of the expression, perform the following steps:
    a. Right-click on the condition name bar and choose *Add Rows*.
       The *Add Rows* dialog opens.

   b. Enter the number of rows.
   c. Choose *Apply*.
11. If you want to insert a new clause row before a specific row in the table, perform the following steps:
   a. Select the row and right-click.
   b. Choose *Add Rows Before* from the menu.
      The *Add Rows* dialog opens.
   c. Enter the number of rows.
   d. Choose *Apply*.
12. If you want to insert a new clause row after a specific row in the table, perform the following steps:
   a. Select the row and right-click.
   b. Choose *Add Rows After* from the menu.
      The *Add Rows* dialog opens.
   c. Enter the number of rows.
   d. Choose *Apply*.
13. Specify the output. Every decision step requires an output, which assigns a destination for the result defined in the specification. Perform the following steps:
   a. Select *Click to add* in the *Output* container.
      The *Data Object Attribute Picker* dialog appears.
   b. Navigate to the data definition and select the field to which you want to assign the result.
      Selecting the field assigns it to the output.

18.7.7.9 Creating Script Steps

Use a script step to create complex expressions using scripting language.

Context

The result can be assigned to boolean, database table, date, number, object table, string or XML data type outputs via the `RETURN` statement in the expression.

Procedure

1. Open the *Rule Painter*.
2. Right-click any existing step and select *Add Step Before* or *Add Step After*.
   The *Script Rule Painter* appears.
3. Specify the external inputs that will be utilized by the script expression in the *Input Arguments* section .
4. Specify the *Return Type* in the *Return Type* section.
5. If necessary, specify the *XML Object Name* for XML return types.

18.7.7.10 Understanding Rule Grammar for Assignment Steps

When adding business logic to Assignment steps, you must follow the required conventions.

Logical AND and logical OR

In the specifications for Assignment steps, logical `AND` and `OR` are handled as follows:

- Logical `AND` is indicated by nested `IF`'s
- Logical `OR` is indicated by subsequent (peer) `IF` statements, that behave as `ELSE IF` statements

Grammar Elements Defined

The following table describes the grammar for Assignment steps.

Element	Definition
RuleExpression	`{ ConditionalExpression | StatementExpression }`
ConditionalExpression	`{ IF Condition THEN RuleExpression } * [ElseExpression]`
ElseExpression	`{ ConditionalExpression | RuleExpression }`
Condition	`{ { <Placeholder> | <LibFunction>(<Args>) | RESULT } ConditionalOperator { ConditionConstant | <Placeholder> | <LibFunction>(<Args>) } }`
ConditionConstant	`{ NumberLiteral | StringLiteral | DateLiteral | BooleanLiteral }`
NumberLiteral	`{ [+-][0-9] + [K|M|B] }`
DateLiteral	`{ <yyyy>-<mm>-<dd> }`
BooleanLiteral	`{ TRUE | FALSE | <BooleanSynonym> }`

Element	Definition
ConditionalOperator	{ *IS EQUAL TO* | *IS NOT EQUAL TO* | *IS GREATER THAN* | *IS NOT GREATER THAN* | *IS LESS THAN* | *IS NOT LESS THAN* }
StatementExpression	*RESULT* = ArithmeticStatement | BooleanLiteral
ArithmeticStatement	{ { ArithmeticConstant | <Placeholder> | <LibFunction>(<Args>) | (ArithmeticStatement) } ArithmeticOperator { ArithmeticConstant | <Placeholder> | <LibFunction>(<Args>) | (ArithmeticStatement) } }
StatementConstant	{ NumberLiteral | StringLiteral }
ArithmeticOperator	{ + | - | * | / | % | ** }
<Placeholder>	[A-Z, a-z] [A-Z|a-z|0-9|space]* **Note** Placeholders are represented in the step's Output container.
<LibFunction>	[A-Z,a-z] [A-Z|a-z|0-9|space]* **Note** Library functions are defined in a reference product, and can be selected using a *Function Picker*.
<Args>	[<Placeholder> | BooleanLiteral [, <Placeholder> | BooleanLiteral]*]
<BooleanSynonym>	[A-Z, a-z] [A-Z|a-z|0-9|space]* **Note** Boolean Synonyms are defined in a reference product. Example: "Attach Form" could be a synonym for *TRUE*

The following table describes the grammar notation.

Symbol	Meaning
ElementName	Name of an element that is defined here
{ ... }	Mandatory element
[...]	Optional element
|	Separator for options in a list of elements
{ ... }*	One or more
[...]*	Zero or more
UPPERCASE BOLD	Keyword

18.7.7.11 User-Defined Functions

User-defined functions are custom functions that any product's rules can use.

You create user-defined functions in a dedicated component that is stored in a special-purpose product. A Product Modeler server can have only one such product. When you publish the product, all the user-defined functions in it are available to all the products on the server. Thus the product is effectively a function library.

Note that the location of the functions library product is site-specific; for more information, see your FS-PRO administrator. If the product already exists and you know its location, to see its functions do the following:

1. Open the functions library product and select the `Rule Functions Library` component.
2. Select the *Values* tab. The services appear as data value rows.

> **i Note**
>
> To run user-defined functions on a Product Modeler server and a Runtime server, you must configure server settings in the Administrative Console. See the Administration Guide.

User-defined functions work with Assignment rule steps.

For functions that require input arguments, you define the inputs in a sub-component within the functions library product.

Examples of user-defined functions

User-defined functions enable you to make rules more user-friendly. For example, you can create functions that substitute technical terms with mnemonics that clarify the purpose of a rule.

In an Assignment rule step, a mnemonic is a function name that makes the purpose of the function clear to users. For example, in a conditional rule step, instead of specifying:

```
if age > 30
```

```
RESULT = True
```

You could write:

```
if age > 30
RESULT = attachForm
```

where `attachForm` is the name of a rule in a user-defined function. The rule returns the Boolean value `True`, but the user immediately sees that the effect is to attach a form. In the `Rule Functions Library` component, the values assigned to this function would appear as follows:

- Service Name = `attachForm`
- Return Type = `Boolean`

Related Information

Creating the Functions Library Product [page 358]
Creating User-Defined Functions [page 359]
Calling User-Defined Functions from Steps [page 361]
Using Call Back Functions with User-Defined Functions [page 362]

18.7.7.11.1 Creating the Functions Library Product

You can have one function library product per Product Modeler server. The only purpose of this product is to store the `Rule Functions Library` component, and to make the functions that it contains available to all products on the server.

Procedure

1. In the Product Studio, determine an appropriate location for the product, keeping in mind that it may be called by any product on the server and that other product stakeholders will need to access it. Typically the location will be within a `Configuration Group` folder such as Company Library.
2. Within the chosen location, inherit a product from the base function library that comes with your Product Modeler installation.
3. Open your function library product and add the `Rule Functions Library` component  to it.
4. Select the *Attributes* tab for the `Rule Functions Library` component.
5. Add an attribute and define it, with a description, internal name and data type.

   You now need to create the sub-component needed by the `Attributes` component. This sub-component allows functions to use input arguments.
6. Select *Product Studio*.

If necessary, determine an appropriate location for your company-wide components. Typically this will be within a `Configuration Group` folder such as Company Library.

7. Create a component of the type Service API and name it **`Rule Functions Args`**.
8. Open the component and add the following attributes to it:

Description	Data Type	Length
`Name`	**`VariableName`**	**`50`**
`Type`	**`TEXT`**	**`50`**
`Description`	**`TEXT`**	**`250`**
`Expression`	**`TEXT`**	**`50`**

9. Save your changes.
10. Return to your functions library product.
11. Add the `Rule Functions Args` sub-component to the Rule Functions Library component .
12. Build the product.

Results

You can now add user-defined functions to the library.

Related Information

Creating User-Defined Functions [page 359]

18.7.7.11.2 Creating User-Defined Functions

You create a user-defined function by adding a data value row to the `Rule Functions Library` component, and then completing the values, including a rule.

Procedure

1. Open the `Rule Functions Library` component .
2. Select the *All Values* tab.

   Any existing services appear as data value rows.

3. Add a data value row.
4. Choose the *Service Name* cell and enter the name of the function.

> **i Note**
>
> The function name must be unique within the server, and should follow the naming standard for Java methods: the names should begin with a lowercase letter; second and subsequent words in the name should begin with a capital letter but not be separated with underscores. For example: `getCountryCode`.

5. If you want to add a rule to the function, right-click in the *Service Rule* cell and select *Create*.
   The *Capture Rule Info* dialog appears.
6. Create the rule.

> **i Note**
>
> Rules in user-defined functions can't use the `form.`, `data.`, and `tree.` stems, but can use the `input.` stem.

7. Build the rule.
8. Close the *Rule Painter*.
   The icon appears in the *Service Rule* cell.
9. In the `Return Type` attribute, enter one of the following:
   - **`TEXT`**
   - **`NUMBER`**
   - **`DATE`**
   - **`BOOLEAN`**
10. Add a brief explanation of what the rule does in the *Description* field.

> **i Note**
>
> Don't use the *Effective Date* and *Expiration Date* fields.

11. Save your changes.
12. If you specified inputs for the function, you should now add them to the `Rule Functions Args` sub-component:
    a. Select the `Rule Functions Args` sub-component.
    b. Select the *Values* tab.
    c. Select the anchor value of the function that you want to add an input value to.
    d. Choose *Next*.
       The values page for the selected function loads.
    e. Enter the name of the input in the *Name* cell. Enter it exactly as it appears in the function's rule in the `Rule Functions Library` component .
    f. Specify the data type for the input in the *Type* cell.
    g. Enter a name for the input in the *Description* cell.
    h. Enter the expression that provides the default value for the input in the *Expression* cell.
       For example: `input."age"`
    i. To attach the input to the product, select the checkbox adjacent to the row header.

j. Save your changes.

Next Steps

As with any product, to make your functions library available to all the products on the Product Modeler server, you must publish its product.

Related Information

Creating Rules [page 319]

18.7.7.11.3 Calling User-Defined Functions from Steps

The following procedure describes how to link a step in a rule to a user-defined function.

Procedure

1. Open the rule in *Rule Painter*.
2. Choose the step and select the Specification container to open it.
3. For an Assignment step, perform the following steps:
   a. Create the step logic as you normally would, until you reach the place where you want to add the user-defined function.
   b. To add the user-defined function, type its `PCD` value. For example, in an assignment step, where the `PCD` value is salaryIndex, the Specification container would contain: `RESULT = salaryIndex`

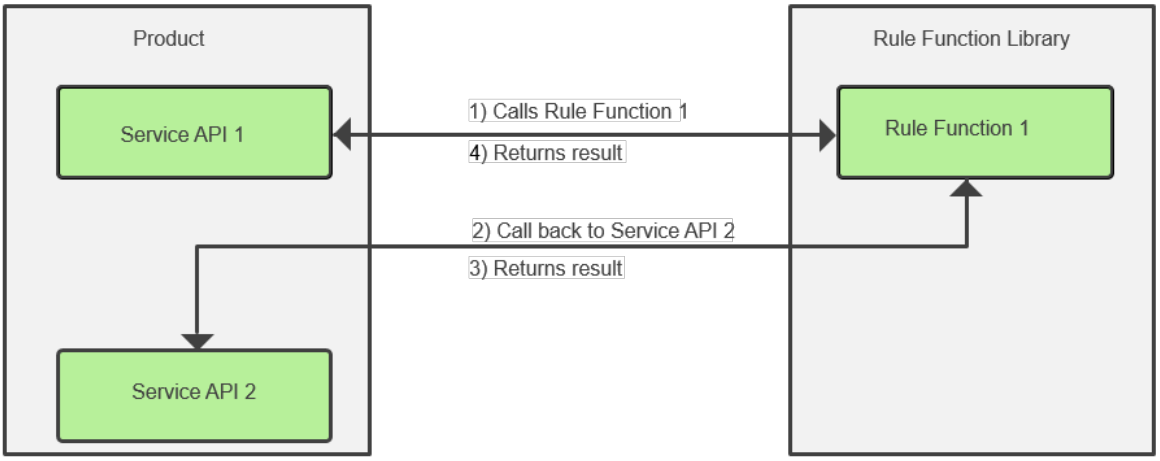
   > **i Note**
   >
   > The `PCD` value must be a text value. Typically, this is the name of the function.
4. Choose *Apply*.
5. Build the rule.
6. Exit the *Rule Painter*.
7. Save your changes.

18.7.7.11.4 Using Call Back Functions with User-Defined Functions

Call back functions enable a library function to send a "call back" to the product when the function needs product information. Rules access call back functions using the `ProductService` stem. Call back functions include the following:

- `runCallBackNumberService()`
- `runCallBackTextService()`
- `runCallBackDataTableService()`

The following diagram shows a basic calling pattern between a product and a rule functions library:



> **i Note**
>
> When you are developing the calls between the rule functions library and the product that uses the library, both products must be published every time you add or change a rule, so that the rules will be in the JAR file. That is, at all times both the callers and the called must be published. This means publishing the entire product, not just the rules.

18.7.8 Rule Unit Testing

The rule unit testing feature enables you to test rules. When you run a rule test, a dialog appears and prompts you to manually enter the required values.

Context

After the rule processes the values, the result appears. If the rule calls other rules, you are prompted to enter the required values of the child rules as well. To use rule unit testing, you don't need to configure any system settings.

You can use rule unit testing with most types of rule and with all rule step types. You can use all rule stems with rule unit testing. If you use the `query.` stem, the required database table must be present.

> → Remember
>
> If you have changed the rule since it was last compiled, you must compile it again before running unit testing.

Procedure

1. Open the rule in *Rule Painter*.
2. Choose the *Rule Unit Test* icon on the *Rule Painter* toolbar.
   The *Rule Testing* dialog appears.

   The *Rule Testing* dialog shows any data pertinent to the rule, including from data definitions and inputs. The dialog only presents inputs required by the rule; other inputs are ignored.
3. Enter the required inputs to all the data fields that appear.

   To see the complete set of functions, go to the *Rule Painter* and access the *Functions List* for the `System` stem.

   If the rule requires detail table inputs, in the section bar choose *Add* once for each row that you want to create.
4. Choose *Run*.
   The dialog changes to show the result of the rule.
5. Perform one of the following actions:
   - Choose *Back* to access the input values to edit them and rerun the rule.
   - Choose *Trace* to access rule tracing.
   - If the dialog presents a child rule, enter the required values and press *Run*.
   - Choose *Cancel* to exit the *Rule Testing* dialog.

Related Information

Rule Tracing [page 363]

18.7.9 Rule Tracing

Rule tracing enables you to trace any rule in Runtime, at both the rule step level and the expression level, providing you with the rule's specification, as well as Runtime results, including data tables.

If the rule calls another rule, the details and results of the second, and any subsequent rules, are available.

Executed rules are kept as objects in memory, but the memory is cleared when certain actions are performed.

For risk-based products, the objects are cleared from memory under the following circumstances:

- When you select *View* or *Edit* on the *Quote Summary* page and go into the eApp.
- When you create a new business or endorsement.
- When you pick a policy from the worklist

For coverage-based products, the objects are cleared from memory when you open a new screen.

Traceable Rules

Rule tracing works with the following kinds of rules:

- Calculation rules at the master table level and in any child table that contains a grid
- Validation rules at the master table level and in any child table that contains a grid
- Default value rules at the beginning of process creation and at child record insert
- Forms rules accessed through an API call during form attachment
- Referral rules accessed through an API call during referral attachment
- Show/hide rules:
  - Screens
  - Screen content
  - eApp sections
- Rating rules (for example, translations and filter rules)
- Content generation rules for screens (for example, dropdown lists)
- Content generation rules for screen sections (for example, dropdown lists.)

> **i Note**
>
> You can't trace cached rules.

Traceable Rule Steps

A rule is composed of one or more rule steps, which come in various types. You can trace both the Input and Step Result in Assignment steps.

In decision steps, you can trace the Step Result, as well as the Input and Output, as described in the following table:

Container Type	Element Type
Input	Variable
	Boolean expression
	Mathematical expression

Container Type	Element Type
	Another rule
Output	Constant
	Variable
	Boolean expression
	Mathematical expression
	Another rule
Decision table cell or range	Constant
	Expression
	Boolean expression
	Another rule

Related Information

Tracing Individual Rules [page 365]
Working with the Rule Painter [page 334]

18.7.9.1 Tracing Individual Rules

In a Runtime application, rule tracing makes available all the rules executed between two workflow activities, grouping the rules by activity, and arranging them in their order of execution. The following procedure describes how to trace a single rule.

Context

> → Tip
>
> To trace rules in Runtime, you must enable settings in the Administrative Console. For more information, see the Administration Guide.

Procedure

1. Open your application in Runtime.
2. Press CTRL + ALT + SHIFT + D.
   A window appears.
3. Choose *Rule Tracing* from the right-hand panel.

   The *Rule Tracing* dialog appears.

   By default, for each rule the Rule Context line indicates the type of rule and the Rule Name line shows the name that was provided at rule creation. When *Show/Hide Rule Details* is enabled, the name of the database table that the rule belongs to appears, along with the PK ID and the column name.
4. Locate the rule that you want to trace, and choose its *Trace* link.

   The *Rule Trace Report* window appears, with the *Trace View* tab selected, displaying the rule's details.

   > **i Note**
   >
   > If the rule returns a data table, to review the data, select any of the "data table" links to open the *Data Table Report* dialog.
5. If you want to see a list of the rules that you are tracing, choose *Tree View*.
   The rule and any child rules that it calls appear in the *Rule Tree*.
6. If you want to see the details of any rule in the *Rule Tree*, select the rule's node.
   The view changes to the *Trace View* tab.

   > **i Note**
   >
   > If you are viewing a child rule, you can change the view to the parent rule; from the toolbar, click the *Back to Parent* icon.

18.7.10 Testing XPaths

The *Rule Painter* provides a facility for validating and debugging the XPaths that you use in your rules.

Prerequisites

Before using XPath Tester, publish the product.

Procedure

1. Open the *Rule Painter*.
2. Go to *View* > *XPath Tester*.

The *XPath Tester* dialog appears.

3. In the *Product Name* and *Product Version* fields, the current product and its version appear by default. Optionally, you can change these to meet the needs of your test.
4. Enter the XPath that you want to test in the *Xpath* field.
5. Optionally, select *Clear Cache* if you want to clear the XPath cache before running the query.
6. Choose *Run Query*.

Results

The results returned by the query appear in the lower part of the dialog.

18.7.11 Rule Result Caching

The purpose of the rule result caching feature is to speed up the process of testing rules.

By enabling rule caching, you specify that the constant values evaluated by a rule remain in memory (that is, are cached) so that the next time the same rule is executed with the same inputs, the calling rule can take the value from the cache. This is much faster than having to rerun the called rule. Note that if the inputs change, the rule is rerun, and the new inputs are added to the cache.

> **! Restriction**
>
> Rule result caching is only available for calculation rules; also, the rule must use the `.input` stem only.

Related Information

Enabling Rule Result Caching [page 368]
Viewing the Rule Result Cache [page 368]
Clearing the Rule Result Cache [page 369]

18.7.11.1 Enabling Rule Result Caching

You can enable rule result caching for any calculation rule.

Procedure

1. Open the *Rule Painter*.
2. Select the *Show/Hide Property Window*.
   The *Rule Properties* panel appears.
3. Select *Yes* for the *Enable Cache* option.

   > → Remember
   >
   > If you are testing a sequence of rules, you should enable the cache at the lowest level of the hierarchy of rules.

18.7.11.2 Viewing the Rule Result Cache

Use the following procedure to examine the contents of the rule result cache.

Procedure

1. Open the *Rule Painter*.
2. Go to *View* > *Rule Result Cache Report* .
   The *Rule Result Cache* dialog appears.
3. Select the cache you want to view from the *Cache* dropdown list.
   The key/value pairs appear in the dialog.
4. If you want to filter cache to display only the results for one product, enter the name of the product in the *Filter* field.

18.7.11.3 Clearing the Rule Result Cache

The rule result cache can be cleared implicitly or explicitly.

Context

The cache is automatically cleared in any of the following instances:

- Building a rule clears the cache for that rule ID
- Publishing a product clears the rule results cache for all rules within that product
- Publishing a product remotely clears the cache for that product in the remote server
- When the server is restarted.

To explicitly clear the rule result cache, perform the following steps:

Procedure

1. Open the *Rule Painter*.
2. Go to *View* > *Rule Result Cache Report*.
   The *Rule Result Cache* dialog appears.
3. Select the cache you want to clear from the *Cache* dropdown list.
4. Select the X beside the *Cache* dropdown list.

18.8 Working with Scripting Rules

> **i Note**
>
> In this section, any language element followed by a question mark is optional. For example, `VARIABLE ? x TYPE NUMBER` indicates that the reserved word `VARIABLE` can be omitted.

The general features of the rule language include the following:

- Local variables and literals
- Looping
- Array and data set manipulation
- Function calls
- IF/ELSE statements, including compound statements
- Assignment statements
- Data access

- Look-ups
- Complex math formulas, such as used in Life and Annuities

Rules use inheritance. This means that the rules your product inherits from its parents are children of the rules in those parents, and changes made to the parent rules are automatically inherited by the children, unless you override them.

The system provides a *Rule Editor* with color-coding and a *Rule Painter*, where you edit and compile rules, and can view the generated Java code.

Related Information

Creating Scripting Rules [page 370]
Solving the Binary Floating-point Arithmetic Problem with the HighPrecisionCalcMode Setting [page 371]
Rule Scripting Language Syntax [page 372]
Custom Error Code and Message Syntax [page 388]
Example Script Rules [page 390]
Script Rule Exception Handling [page 392]
Audit Trail Information for Script Rules [page 399]

18.8.1 Creating Scripting Rules

Context

Rules can be created from scratch or based on a template. Templates can be saved under *My Templates* associated with a given user account, or can be stored as system-wide templates, available to all users. The following procedure describes creating and working with rules.

To create a scripting rule from scratch, perform the following steps:

Procedure

1. Open the product .
2. Select the component.
   The only prerequisite to adding rules to a component is that it have an attribute with the data type RULE.
3. Select the *Values* tab.
4. Add a data value row.

5. Select the appropriate rule cell and choose *Create*.
   The *Capture Rule Info* dialog appears.
6. Enter a name in the *Rule Name* field.
7. Select *Script* from the *Step Type* dropdown list.
8. Select a *Rule Return Type* from the dropdown list.
9. Choose *OK*.
   A dialog opens.
10. Either select a data definition for the rule or to choose *Continue* to create the rule without a data definition.
    The *Rule Painter* appears and you can start to develop the rule.

Related Information

Opening the Rule Painter [page 336]

18.8.2 Solving the Binary Floating-point Arithmetic Problem with the `HighPrecisionCalcMode` Setting

Context

In any programming language there is a known rounding issue referred to as the 'Binary floating-point arithmetic' problem. Most decimal fractions can't be represented exactly as binary fractions. Consequently, any decimal floating-point numbers you enter are only approximated by the binary floating-point numbers actually stored in the machine. This issue is resolved using the BigDecimal type calculation. The `HighPrecisionCalcMode` setting is available in the Administrative Console to parse all double type values to BigDecimal type in the back-end for all script rule step calculations.

> ⚠ Caution
>
> This function is limited to script rule steps.

To access the `HighPrecisionCalcMode` setting, perform the following steps:

Procedure

1. In the Administrative Console, verify that you are in the correct server.
2. Select *System* > *Edit Configuration Settings* > *Configuration* > *System Environment* > *Env* > *System Environment* > *HighPrecisionCalcMode*.
   `HighPrecisionCalcMode` setting has available values of Yes or No.

The default setting is Yes. The system will parse all double values to BigDecimal in the back-end for all script rule step calculations. All numbers are converted to BigDecimal type in all script rule steps going forward.

> ⚠ Caution
>
> It isn't recommended to switch it back to No and revert back to use double type values.

3. Recompile all existing rules, to apply the use of BigDecimal number type in existing script rules steps.

   No manual changes in the rules are required.

18.8.3 Rule Scripting Language Syntax

The rule scripting language is case sensitive: uppercase and lowercase letters differentiate words, including the reserved words. White space (the space, tab, and end-of-line characters) serves to separate tokens, and otherwise is ignored. No token can extend past the end of a line. Spaces may not appear inside any token except string literals.

> ℹ Note
>
> Rule changes aren't saved in the *Script Editor* if there are syntax errors. You will need to fix the syntax errors in order to successfully save the rule.

Related Information

Language Basics [page 373]
Literals [page 374]
Operators and Assignments [page 375]
Flow Control [page 378]
Functions [page 382]
Other Statements [page 383]
Calling Inner Rules [page 386]
Calling Custom Stems [page 387]

18.8.3.1 Language Basics

This topic provides an explanation of language basics for rule scripting.

Reserved Words

The rule scripting language reserves the following words:

Reserved Words

ALL	AND	ANY	ASC
ATTACH	AVG	BOOLEAN	BREAK
BY	CALL	CONTINUE	COUNTER
DATAOBJECT	DATE	DEBUG	DECIMALS
DESC	DETACH	DIMENSION	EACH
ELSE	ELSEIF	END	ERROR
EXECUTE	FATAL	FILTER	FOR
FROM	FUNCTION	IF	IN
INFO	INTERPOLATE	IS	ISEMPTY
LIBRARY	LOCATE	LOOKUP	LOOP
MAX	MESSAGE	METADATA	MIN
NEW	NOTATTACHED	NULL	NUMBER
OBJECTTABLE	OF	ON	OR
ORDER	PRINT	PRODUCT	QUERY
RETURN	RETURNS	ROUND	ROW
RULE	SERVICE	SERVICECOMMAND	SET
STEP	STRING	SUM	TO
TRACE	TYPE	UNDER	USING
VALUEOF	VAR	VERSION	WARN
WHERE	WHILE	WITH	XMLOBJECT

> **i Note**
>
> The `ServiceCommand` parameter is no longer required, but is backwards compatible.

Reserved words can't be used as identifiers and must be in uppercase.

Comments

A single line comment begins with two forward slashes and extends to the end of the line:

```
// a single line comment
```

Multiple-line comments start with `/*` and end with `*/`:

```
/* comment line 1
comment line 2
comment line 3 */
```

Note that comments don't appear in the generated Java rule code.

Reference Object

The reference object is a special purpose identifier introduced by the Script Rule Language and refers to column names and takes the following form: `<Tablename.Columnname>`

Note that if the table or column has a description attribute, then the column name will be referred to by its description. For example, if a column is named as follows:`GL.MULT_PREM_CONSTRC`

And if the column has a description, for example: `GL.MultiplePremiseRiskConstructed` the description must be used in the script rule instead of the actual column name.

18.8.3.2 Literals

A literal is a number, text, or the booleans true and false. A number literal consists of one or more digits and can also be in decimal format. A string literal is a sequence of zero or more printable characters enclosed by a pair of double quotation marks ("and ").

Identifiers

Identifiers start with a letter or underscore and contain any number of letters and digits. This applies regardless of whether the identifier is public, local, or a parameter. Identifier declarations follow Java variable declaration

rules and conventions. Don't use Java keywords (such as do) or keywords from the Rule Scripting Language as identifiers.

Identifiers are declared as follows: `VARIABLE? <identifier> TYPE <data type>`

The data types are:

NUMBER	Defined as a double in Java
STRING	The value must be within double quotes
BOOLEAN	The valid values are 'true' and 'false'
DATE	`java.util.Date`
DATAOBJECT	An interface representing DataTable.

> **i Note**
>
> The DATAOBJECT type is used to return Data Tables. You can't initialize a DATAOBJECT identifier; you must assign the results from a `QUERY` statement to the identifier.

The following examples show how to declare identifiers (note that the keyword *VARIABLE* is optional):

```
VARIABLE i TYPE NUMBER
s TYPE STRING
d TYPE DATE
datobj TYPE DATAOBJECT
```

The initial value can't be assigned to the identifier at the time of declaration. All identifiers are assigned a default value by the Rule Code Generator

18.8.3.3 Operators and Assignments

Assignment Statement

The assignment operator is "=". For example:

```
age=23
step=-34
MyString="ABC"
DueDate=DATE 2010-08-06
Result=true
```

Script rules support the following syntax for assignment statements:

```
<variable> = NULL
<tableName.ColumnName> = NULL
<tableName.ColumnName> = expression
<variable> = expression
<variable> = <text_value>
```

```
<tableName.ColumnName> = <text_value>
<tableName.ColumnName> PLUS = expression
<variable> = dateTypes
```

> **Note**
>
> NULL represents null whereas `<text_value>` represents the type STRING.

Operators

The valid operators are:

- + (add)
- - (subtract)
- * (multiply)
- / (divide)
- ^ (exponent)

> **Note**
>
> By default, script rules do **not** follow order of operations when using the default `BigDecimal` type. Thus, the expression 2 + 3 × 4 evaluates to 20, instead of 14, which is the expected result if normal order of operations is followed like so: 2 + (3 × 4). To always ensure that you get the desired result, format mathematical expressions with appropriate parentheses.

Expressions

For binary operators, both operands must be the same. For assignment, both the left side and the right side must have the same type. Expressions can be combined and nested to any level.

The following are valid assignment statements:

```
X=5
X =x+5
X =6+6
X = Table1.colum1
Table1.column1 = Table1.column2 + Table1.column3
X = Y ^ X
X  = ( a+b)^(c+d)
X = ( Y + Z ) * A
```

Relational Operators

The following relational operators are allowed in IF statements:

- = (equal to)
- > (greater than)
- >= (greater than or equal to)
- < (less than)
- <= (less than or equal to)
- != (not equal to)

In statements using relational operators, there's no need to consider the data type of operands. The following statements are valid:

```
d TYPE DATE
x TYPE NUMBER
s TYPE STRING
t TYPE BOOLEAN
T TYPE NUMBER
IF GL.DELETE_FLAG = t
IF GL.CovSection = t
IF x = GL.YearInClaimsMade
IF GL.YearInClaimsMade = x
IF GL.TRIAExpirationDate = GL.ProductsWithdrawalCutOff
IF GL.TRIAExpirationDate < GL.ProductsWithdrawalCutOff
IF GL.TRIAExpirationDate > GL.ProductsWithdrawalCutOff
IF GL.TRIAExpirationDate >= GL.ProductsWithdrawalCutOff
IF GL.TRIAExpirationDate <= GL.ProductsWithdrawalCutOff
IF GL.Subline = "Premises/Operations"
```

Logical Operators

Script rules support two types of logical operators, AND and OR. The syntax of the logical operators is as follows and can be nested to any level:

```
IF <condition> AND <condition>
<statement>
...
IF <condition> OR <condition>
<statement>
```

Examples of logical operators:

```
true = TRUE
IF GLtable.column1 = true AND GLtable.columnExpireDate <= GLtable.columnDueDate
...
IF GLtable.column1 = true OR GLtable.columnExpireDate <= GLtable.columnDueDate
...
```

Conditions

A condition is any combination of relational and logical operators. The basic syntax is as follows:

```
<element> <logical_operator> <element>
```

More complex conditions can be created by linking basic conditions with relational operators:

```
<element> <logical_operator> <element> <relational_op> <element>
<logical_operator> <element>
```

Note that a condition can optionally be encapsulated in parentheses:

```
(<element> <logical_operator> <element> <relational_op> <element>
<logical_operator> <element>)
```

Some example conditions:

```
amountOne TYPE NUMBER
amountOne = 1000
IF tableName.column1 > amount
...
IF tableName.column2 = "Travel"
...
IF tableName.column1 > amount AND tableName.column2 = "Travel"
...
```

Furthermore, multiple elements can also be combined within a single condition by adding parentheses and separating the elements with commas or the word "or." The syntax for this expression is as follows: `(<element> , <element> or <element>)`

For example:

```
IF tableName.column2 = ( "A" , "B" , "C" , "D" )
...
IF tableName.column2 = ( "A" or "B" or "C" , "D" or "E" )
...
```

18.8.3.4 Flow Control

IF Statement

The syntax for the IF statement is as follows:

```
IF <condition>
    <statement>
ELSE
    <statement>
ELSE IF <condition>
    <statement>
END IF
ELSEIF <condition>
    <statement>
END IF
```

The following code samples show the use of the IF statement:

```
RULE
IF GLClass.PremOpsPremiumBasis = ("Admissions" or "Area" or "Cost of Work" or
"Gallons" or "Gross Sales" or "Payroll" or "Total Cost" or "Vehicles")
GLClass.PremOpsExposureCalc = GLClass.PremOpsExposure /1000
ELSE
GLClass.PremOpsExposureCalc = GLClass.PremOpsExposure
END IF

END RULE;

RULE
x TYPE NUMBER
y TYPE NUMBER

IF CA_ExpRtMod.ExpNumberYearsReq = 1
    x=1.3455*56.7888 ROUND TO 2 DECIMALS
ELSE IF CA_ExpRtMod.ExpNumberYearsReq = 2
    x=12
    y=34
ELSE IF POLICY CA_ExpRtMod.ExpNumberYearsReq = 3
    x=123
ELSE IF CA_ExpRtMod.ExpNumberYearsReq = 4
    x=1234
    y=5678
ELSE IF CA_ExpRtMod.ExpNumberYearsReq = 5
    x=12345
ELSE
    x=123456
END IF

RETURN x

END RULE;



RULE
x TYPE NUMBER

IF POLICY_STATUS_LOG.POLICY_ID = 1
    x=1.3455*56.7888 ROUND TO 2 DECIMALS
    ELSE IF POLICY_STATUS_LOG.POLICY_ID = 2
        x=12
        ELSE IF POLICY_STATUS_LOG.POLICY_ID = 3
            x=123
            ELSE IF POLICY_STATUS_LOG.POLICY_ID = 4
                x=1234
                ELSE IF policy_status_log.policy_id = 5
                    x=12345
                    ELSE
                        x=123456
                END IF
            END IF
        END IF
    END IF
END IF

END RULE;
```

The following example shows the use of ELSEIF:

```
RULE
IF (REFERRAL.SUBJECT= ("Premises/Operations" or "Products/Completed
Operations"))
    AND REFERRAL.SUBJECT = "Yes"
    RETURN true
```

```
    ELSEIF (REFERRAL.SUBJECT = ("Products/Completed Operations"
        or "Premises/Operations and Products/Completed Operations"))
        AND REFERRAL.SUBJECT = "Yes"
        RETURN true
    ELSEIF (REFERRAL.SUBJECT = "Liquor"
    OR REFERRAL.SUBJECT  = "Owners and Contractors"
    OR REFERRAL.SUBJECT  = "Pollution"
    OR REFERRAL.SUBJECT  = "Underground Storage Tank")
    AND REFERRAL.SUBJECT  = "Yes"
    RETURN true
    ELSE
    RETURN false
END IF

END RULE
```

Condition read from XML file

When creating a Script Rule with an IF statement, the rule can't be saved if it contains an inner bracket inside the IF statement and the values inside the condition are read from XML files. For example: `IF ((x +Y)) >5)`. The rule should be rewritten with the following syntax: `IF (true AND(x +Y) >5)`.

Negation operator

When writing a Script Rule with the negation operator!, the following syntax is now acceptable: `IF (!(n =10 OR m =8) AND !(m = 0))`

IF boolean variable

New syntax of `IF (boolean)` is now available as illustrated in the following rule fragment:

```
RULE
.......
.......
IF (CoverageIsEntered ) // CoverageIsEntered is a variable
CoverageIsEntered = false
END IF
RETURN CoverageIsEntered
END RULE
```

Looping Statements

The language supports two kinds of looping statement, based on LOOP and FOR EACH. The syntax of the first type is as follows:

```
LOOP USING COUNTER counter FROM minVal TO maxVal
…
END LOOP
```

An example of this type of looping statement:

```
// variable declared in the middle of the script rule
VARIABLE sum TYPE NUMBER

LOOP USING COUNTER count FROM 1 TO 100
```

```
    sum = sum + count
END LOOP
CommercialProperty.ExposureTestAmount = sum
```

The second kind of loop uses FOR EACH. The syntax is as follows:

```
FOR EACH <element> IN <element>
…
END FOR
```

An example:

```
FOR EACH ROW IN childTable1
    VARIABLE count TYPE NUMBER;
    IF Table1.column1 = Table1.column2
        count = count + 1
    END IF
END FOR
```

For example:

```
FOR EACH ROW IN CommPropStruct
    IF (CommPropRadioOrTVAntennas IS ATTACHED)

        FOR EACH ROW IN CommPropAntennasDetail

            CommPropAntennasDetail.Amount = SUM(AntennasDetail.Limit/1000)
            CommPropAntennasDetail.Amount = CommPropAntennasDetail.Amount +
            CommPropStruct.Amount + CommProp.ExposureTestAmount

        END  FOR
    END IF
END FOR
```

> **i Note**
>
> In the above example, CommPropStruct can only be an immediate child table, not a grandchild table. Only an immediate child table is allowed at the first level of the loop. However, nested loop statements require grandchild and great grandchild tables.

WHILE Statement

The syntax of the WHILE statement is as follows:

```
WHILE <condition>
    <statement>
END WHILE
```

In the following example of a WHILE statement, x, y, a, and b are number variables:

```
a TYPE NUMBER
b TYPE NUMBER
```

```
a=10
b=5

WHILE (x+y < a) OR (x/y > b)
    c=45
    x=x+1
END WHILE
```

18.8.3.5 Functions

Aggregate Functions

The following functions are built-in to the language, but you can also implement them differently by defining your own local function or by having a FOR EACH loop within the rule's main body:

- SUM ("TableName.ColumnName")
- MIN ("TableName.ColumnName")
- MAX ("TableName.ColumnName")
- AVG ("TableName.ColumnName")

> **i Note**
>
> TableName is the child table of the current table that is the DATAOBJECT.

An example of an aggregate function:

```
CommPropAntennasDetail.ExposureAmount = SUM("CommPropAntennasDetail.Limit"/
1000)
```

Internal Functions

The syntax for internal function declarations is as follows:

```
FUNCTION function1 ARG a1 TYPE text, ARG a2 TYPE number
RETURNS number
    <Statement>
    <Statement>
END FUNCTION
```

Note that formal parameters and return types are optional. The rule statement is also optional and valid though it is meaningless. For example:

```
FUNCTION function1 WITH arg1 TYPE NUMBER, arg2 TYPE NUMBER
RETURNS NUMBER
```

```
    VARIABLE count TYPE NUMBER
    VARIABLE count1 TYPE NUMBER
    count1=23.555
    count = count1*arg1/1.2345
    RETURN count
END FUNCTION
```

You would call the above function within a rule as follows: `fnReturnVal1 = CALL function1 WITH fnInputVal1 ,fnInputVal2`

External Functions

A script rule allows the user to call any static method in a Java class provided the class is registered in the function library. The syntax for calling an external library is as follows:

```
REFOBJnumVar1 = CALL <funtion name> ceil IN LIBRARY <library name> NUM WITH
arg1, arg2
IDENTIFIER CALL <funtion name> IN LIBRARY <library name> WITH
arg1,arg2
```

Note that `IN LIBRARY <library name> NUM` is optional when the method name is unique within the function library; NUM is the name given to the `NumberFunctionLibrary` in the Administrative Console.

An example of a rule that calls an `externalNumberFunctionLibrary` function:

```
i = CALL getDaysAfter IN LIBRARY DATEFN WITH
POLICY_QUOTE.EXPIRATION_DT,POLICY_QUOTE.EFFECTIVE_DT

POLICY_QUOTE.EXP_YEAR  =CALL getYear IN LIBRARY DATEFN WITH
POLICY_QUOTE.EXPIRATION_DT
```

Note that `IN LIBRARY DATEFNNUM` is optional if the method names `getDaysAfter` or `getYear` is unique within the entire function library. DATEFNNUM is the name given to the `DateTimeFunctionLibraryNumberFunctionLibrary` in the Administrative Console.

18.8.3.6 Other Statements

ROUND Statement

The syntax is `ROUND TO INT DECIMALS` where INT is an integer. It is the same as using the `ROUND_HALF_UP_TO` syntax. The following example illustrates rounding the answer in two different ways:

```
RULE
totalPremium TYPE NUMBER
totalPremium1 TYPE NUMBER
```

```
FOR EACH ROW IN CP_LOCATION
totalPremium = (totalPremium + CP_LOC.FINAL_RATED_PREM_AMT ) ROUND TO 2 DECIMALS
totalPremium1 = totalPremium1 + CP_LOC.FINAL_RATED_PREM_AMT  ROUND TO 2
DECIMALS
END FOR
RETURN totalPremium
END RULE;
```

> **i Note**
>
> The parentheses in the first statement are optional and affect the order of operations and hence cause a different result from that of the second statement.

Additional Rounding Syntaxes

The following additional syntaxes are available:

ROUND_UP TO	Rounding mode to round away from zero.
ROUND_DOWN TO	Rounding mode to round towards zero.
ROUND_CEILING TO	Rounding mode to round towards positive infinity.
ROUND_FLOOR TO	Rounding mode to round towards negative infinity.
ROUND_HALF_UP TO	Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up. This is the default rounding strategy being used. Using ROUND_TO yields the same result.
ROUND_HALF_DOWN TO	Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down.
ROUND_HALF_EVEN TO	Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor.

QUERY and LOOKUP Statements

In the rule scripting language, QUERY and LOOKUP statements are similar to SELECT statements in SQL. You use QUERY and LOOKUP statements to return values from FS-PRO component objects. A returned value can be a single column, a row, or a data table consisting of multiple rows.

Internally, the system transforms the QUERY / LOOKUP statement into an XPath query. If the component name isn't unique within the product, the QUERY and LOOKUP statements are complemented with an UNDER clause. The order of the folders or components in the clause is bottom up and is translated internally into a top-down XPath statement.

The following query statement returns a DataObject.

```
D1= QUERY CODE, DESCRIPTION (There can be any number of columns)
FROM   QUALIFIER
FILTER BY DATE YYYY-MM-DD
WHERE REQUIRED = "Y"
```

```
LOCATE
IN PRODUCT <Product Name>  (Optional)
IN VERSION  <Product Name>  (Optional)
(UNDER COMPONENT1  (or FOLDERNAME )
WHERE  COMPONENT.CODE =  arg1 AND COMPONENT.REQUIRED is Y)
```

The following lookup statement returns a type such as String, double, boolean or java.util.Date. The return type depends on the data type of D2.

```
D2= LOOKUP CODE (There should be only one column name)
FROM QUALIFIER
FILTER BY DATE YYYY-MM-DD
WHERE REQUIRED = "Y"
LOCATE
IN PRODUCT <Product Name>  (Optional)
IN VERSION <Product Name>  (Optional)
(UNDER COMPONENT1
WHERE  COMPONENT.CODE = arg1 AND COMPONENT.REQUIRED is Y)
```

XPATH Syntax

The XPATH syntax only handles simple string query arguments and can't handle a more complex query like those involving string concatenations and variable substitution.

For example the following will work:

```
stateDescription TYPE STRING
stateDescription = QUERY DESCRIPTION FROM XPATH "/*/Reference/
StateCode[STATE_CODE='OH']"
```

The following will display a syntax error on saving the rule:

```
stateCode TYPE STRING
stateDescription TYPE STRING
stateCode = "OH"
stateDescription = QUERY DESCRIPTION FROM XPATH "/*/Reference/
StateCode[STATE_CODE='''" + stateCode + "']"
```

PRINT Statement

The rule scripting language introduces the PRINT statement in order to include Java System.out statements in generated rules for printing messages in the application log. The output goes to the SystemOut.log of the server. The syntax is: `PRINT printDefinition`

Where printDefinition is `PrintStatement (PLUS PrintStatement)*`

PrintStatement is defined as either String or an expression.

Examples:

```
PRINT "Print this message in system out"

PRINT "Answer is" + <x>
```

Where <x> is a variable.

```
PRINT <y>
```

Where <y> is a variable.

> **i Note**
>
> PRINT works with variables of type STRING, NUMBER, and BOOLEAN.

```
PRINT "Print me " + <i> + "Print me again " + <j> + "One more time " + <b> +
"Premium Amount" + CP_LOCATION.FINAL_RATED_PREM_AMT
```

Where <i>, <j>, and <b> are variables and CP_LOCATION.FINAL_RATED_PREM_AMT is a column in the CP_LOCATION table.

18.8.3.7 Calling Inner Rules

You can have your script rule call and execute a script rule, using the following syntax:

```
VariableName = EXECUTE RULE RuleColumnName
FROM ComponentName
WHERE ConditionalStatement
LOCATE
IN PRODUCT  ProductName  (Optional)
IN VERSION  ProductVersion  (Optional)
(UNDER ComponentName
WITH ConditionalStatement )
```

The variable name and script rule return type should match; the system validates for this only at Runtime.

Examples of calling a script rule as a sub rule:

```
VAR1 = EXECUTE RULE RatingRule WITH x, y
FROM RATING_RULE
WHERE PCD = "1689357"
LOCATE
IN PRODUCT  "Mock_Commercial_Property"
IN VERSION  "1"
UNDER RATING
```

In this example, x and y are variables in the calling rule and are defined in the properties page of the called rule.

```
d2 = String
prodName  ="Mock_Commercial_Property"
version  ="1"

d2 = EXECUTE RULE RatingRule2
FROM Rating_Rule
WHERE PCD = "1689357"

d2 = EXECUTE RULE Rule
FROM Rating_Rule
WHERE PCD = "1689357"
LOCATE
UNDER RATING

d2 = EXECUTE RULE Rule
FROM Rating_Rule
WHERE PCD = "1689357"
LOCATE
IN PRODUCT  prodName
IN VERSION version
UNDER RATING

d2 = EXECUTE RULE Rule
FROM Rating_Rule
WHERE PCD = "1689357"
LOCATE
IN PRODUCT  "Mock_Commercial_Property"
IN VERSION "1"
UNDER RATING
```

> **Note**
>
> To call a script rule from a standard rule, you don't need to specify the arguments; they are passed automatically.

18.8.3.8 Calling Custom Stems

You can call custom stems in the *Rule Painter* dialog the same way that you call external functions in the rule scripting language.

It is recommended as a best practice that the variable `<serviceCommand>` now be used in script rules when calling custom stems. For more information, refer to the *Custom stem class example script rules* and *Creating custom stem classes to handle BigDecimal type in script rules* topics in the *Application Development* section of the *Administration Guide*.

18.8.4 Custom Error Code and Message Syntax

MESSAGE ON ERROR Syntax

The script rule syntax to invoke a custom error code/message from the `Rule Error Messages` component is as follows: `MESSAGE ON ERROR "<errorCode>"`

The `MESSAGE ON ERROR` syntax can be defined in two levels, rule and operation. MESSAGE ON ERROR syntax is used to invoke custom defined error codes.

Rule Level (catch error that occurs within a rule)

```
RULE MESSAGE ON ERROR "errorCode"

    rule body

END RULE
```

Operation Level (catch error that occurs within an operation)

```
RULE

dataObject TYPE DATAOBJECT
dataObject = QUERY Test
FROM eApp_Column_Info
MESSAGE ON ERROR "errorCode"

END RULE
```

If there are multiple errors, the system will only catch one error and displays the error which occurs first.

Exception Handling Methods

The following methods support exception handling.

QUERY

Syntax is mandatory for QUERY. For example, if you want the system error to be caught, specify the following after your QUERY: `MESSAGE ON ERROR "SYSERR_QUERY"`

LOOKUP

Syntax is optional for LOOKUP and defaults to system error code SYSERR_LOOKUP if syntax isn't used in the script rule.

The script rule syntax for LOOKUP allows you to set a default value if an exception occurs but an exception will never be thrown: `DEFAULT <value>`

- If Boolean value is expected: `DEFAULT <true|false>`
- If Date value is expected: `DEFAULT DATE <YYYY-MM-DD>`
- If Number value is expected: `DEFAULT <number value>`
- If String value is expected: `DEFAULT "<string value>"`

The syntax above is used as follows:

```
RULE
factor TYPE STRING
factor = LOOKUP f DEFAULT -1
…
…
END RULE
```

Calling Inner Rules

The script rule syntax to invoke a custom error code/message from the `Exception Handling Config` component for EXECUTE RULE is as follows: `MESSAGE ON ERROR "<errorCode>"`

Syntax is optional and defaults to system error code SYSERR_EXECUTE_RULE if syntax isn't used in the script rule.

INTERPOLATE

The script rule syntax to invoke a custom error code/message from the `Exception Handling Config` component for INTERPOLATE is as follows: `MESSAGE ON ERROR "<errorCode>"`

For this INTERPOLATE rule, the system will default to SYSERR_INTERPOLATE (not SYSERR_EXECUTE_RULE) if MESSAGE ON ERROR syntax is (not used)/omitted.

In the example below, `<value>` in `n2=value` is an Input Argument that is of Type Number.

```
RULE
n1 TYPE NUMBER
n2 TYPE NUMBER
n2 = <value>
n1 = LOOKUP NVALUE2
FROM ComponentTwo
INTERPOLATE NVALUE1 FOR n2
LOCATE
IN PRODUCT "Script Rule Exception Handling"
IN VERSION "1"
MESSAGE ON ERROR "USRERR_LOOKUP_INTERPOLATE_ERROR"
RETURN n1
END RULE
```

Related Information

Script Rule Exception Handling [page 392]

18.8.5 Example Script Rules

This topic contains simple examples of script rules.

Using the MAX Function on Child Rows

This example returns the highest NUMBER_COL column value from the current record's child rows. The rule return type is Number. The argument is aoDataObject, with the type DataObject.

```
//Returns the highest NUMBER_COL column value from the current record's children.
RULE
VARIABLE result TYPE NUMBER
result = MAX(CHILD_TABLE.NUMBER_COL)
RETURN result
END RULE;
```

Using the SUM Function on Child Rows

This example returns the sum of all NUMBER_COL values from the current record's child rows. The rule return type is Number. The argument is aoDataObject, with the type DataObject.

```
RULE
VARIABLE result TYPE NUMBER
result = SUM(CHILD_TABLE.NUMBER_COL)
RETURN result
END RULE;
```

Using the AVERAGE Function on Child Rows

This example returns the average NUMBER_COL column value from the current record's child rows. The rule return type is Number. The argument is aoDataObject, with the type DataObject.

```
RULE
VARIABLE result TYPE NUMBER
result = AVG(CHILD_TABLE.NUMBER_COL)
RETURN result
END RULE;
```

Updating Child Rows with FOR ROW EACH and LOOKUP

The following example loops through each child record and updates the TEXT_COL column with a matching value from the component AQComp06 based on the child record's NUMBER_COL value. The return type is Void. The argument is aoDataObject, with type DataObject.

```
RULE
VARIABLE stringVar1 TYPE STRING
FOR EACH ROW IN CHILD_TABLE
BEGIN
stringVar1 = LOOKUP DESC_COL FROM AQComp06
WHERE KEY_COL = CHILD_TABLE.NUMBER_COL
LOCATE IN PRODUCT "AQProd26" IN VERSION "1"
UNDER AQProd26
CHILD_TABLE.TEXT_COL = stringVar1
END
END RULE;
```

LOOP USING COUNTER

The following example demonstrates a counter loop. If x is the current record's TEST_NUMBER_COL value, the rule returns x*1*2*3. The rule's return type is Number. The argument name is aoDataObject with type DataObject.

```
RULE
VARIABLE result TYPE NUMBER
result = PRODUCT_RULES.TEST_NUMBER_COL
LOOP USING COUNTER countVar FROM 1 TO 3
BEGIN
result = result*countVar
END
RETURN result
END RULE;
```

Calling Custom Functions

The following example shows how to call a custom function. The rule return type is Text. The rule uses no arguments.

```
//
RULE
VARIABLE result TYPE STRING
VARIABLE param TYPE STRING
param = "testParam"
result = CALL testFunc IN LIBRARY AQStem WITH param
RETURN result
END RULE;
```

18.8.6 Script Rule Exception Handling

Configurable error handlers, at the rule level, are provided in script rule language to allow for processing of unexpected errors and unhandled business errors. The following topics provide details on Script Rule exception handling:

Related Information

Rule Error Messages Component [page 392]
Custom Error Codes and Messages [page 395]

18.8.6.1 Rule Error Messages Component

The `Rule Error Messages` component enables error handling for script rules.

Error handling is enabled by assembling the `Rule Error Messages` component into a product object (for example, coverage group, coverage, and reference object). If the `Rule Error Messages` object isn't found in the product object, the additional debugging information isn't provided in the error report or error response (in the case of product web services).

The `Rule Error Messages` component is located in the `Objects` folder under the path *Product Studio* > *System Repository* > *System Group*.

The `Rule Error Messages` component contains the following predefined attributes:

`ErrorCode`	A message is displayed that corresponds to a specific error encountered by the system. All SAP ErrorCodes begin with SYSERR_ as shown in the table below. Type: TEXT Length: 50 Required: No
`ErrorMessage`	A detailed message indicating particulars of the error detected. Error messages can be customized using system provided parameters. Type: CLOB Length: N/A Required: No

A set of default system error codes and error messages are prepopulated in the standalone `Rule Error Messages` component as shown in the following table:

Default System Error Codes and Error Messages

ErrorCode	ErrorMessage	Description
SYSERR_QUERY	No results returned from QUERY: {ruleName} :{lineNumber} ==> QUERY"{contextInfo.columnName}" FROM"{contextInfo.fromComponent}" FILTER BY "{contextInfo.filterAttribute}" WHERE {contextInfo.whereStatement} LOCATE IN PRODUCT "{contextInfo.productName}" IN VERSION "{contextInfo.productVersion}"	If a QUERY results in an error, the Error message displays with various parameters to facilitate debugging.
SYSERR_LOOKUP	No results returned from LOOKUP: {ruleName} :{lineNumber} ==> LOOKUP "{contextInfo.columnName}" FROM "{contextInfo.fromComponent}" FILTER BY "{contextInfo.filterAttribute}" WHERE {contextInfo.whereStatement} ORDER BY "{contextInfo.orderColumn}" LOCATE IN PRODUCT "{contextInfo.productName}" IN VERSION "{contextInfo.productVersion}"	If a LOOKUP results in an error, the Error message displays with various parameters to facilitate debugging.
SYSERR_INTERPOLATE	Error from INTERPOLATE: {ruleName} : {lineNumber} ==> INTERPOLATE {contextInfo.intValue1} FOR {contextInfo.intValue2}	If INTERPOLATE results in an error, the Error message displays with various parameters to facilitate debugging.
SYSERR_EXECUTE _RULE	No results returned from EXECUTE RULE: {ruleName} : {lineNumber} ==> EXECUTE RULE"{contextInfo.ruleName}" FROM "{contextInfo.fromComponent}" FILTER BY "{contextInfo.filterAttribute}" WHERE {contextInfo.whereStatement} LOCATE IN PRODUCT "{contextInfo.productName}" IN VERSION "{contextInfo.productVersion}"	If EXECUTE RULE results in an error, the Error message displays with various parameters to facilitate debugging.
SYSERR_USER_CODE _NOT_FOUND	User defined error code not found: {userErrorCode} : {lineNumber}	If a USER CODE results in an error, the Error message displays with various parameters to facilitate debugging.

ErrorCode	ErrorMessage	Description
SYSERR_UNEXPECTED	An unexpected error occurred in rule: {ruleName} : {lineNumber}	If an UNEXPECTED error occurs that isn't covered by one of the above ErrorCodes, then this error serves to catch other error.

You can assemble the `Rule Error Messages` component into your base templates and then select the rows in the *Values* tab. Alternatively, you can assemble the `Rule Error Messages` component directly into a Product Base type object.

The `Rule Error Messages` component is to be assembled under the `Configuration` folder.

Related Information

Adding the Rule Error Messages Component [page 394]
Custom Error Code and Message Syntax [page 388]
Adding Folders to the Product Tree [page 92]

18.8.6.1.1 Adding the `Rule Error Messages` Component

Follow this procedure to add the `Rule Error Messages` component.

Context

> **i Note**
>
> This is the minimum requirement for Script Rule exception handling.

The following example illustrates adding the `Rule Error Messages` component to a marketable product but it can be assembled into any `Product Base` type object.

To add a `Rule Error Messages` component, perform the following steps:

Procedure

1. Open your product.
2. Right-click the `Configuration` folder and select *Add*.

   The *Object Picker* dialog displays.

3. Enter **`Rule Error Messages`** in the *Object Name* field.
4. Choose *Search*.
5. Select *Rule Error Messages* from the *Object Search Result* panel.
6. Choose *Select*.
   The `Rule Error Messages` component is added to the tree view in the `Configuration` folder.
7. Select the *Values* tab.
8. Select All from the *Views* dropdown list.
9. Select all the rows.

   > ⚠ Caution
   >
   > You must select all system error codes provided out-of-the-box. Unselection of system error codes isn't supported.

10. Save your changes.

Results

After assembling the `Rule Error Messages` component, no further action is required unless you wish to add custom error codes.

18.8.6.2 Custom Error Codes and Messages

In addition to the default system error codes and error messages (described above), you can create custom error codes and messages. This would enable you to account for additional errors but this is an optional step and isn't mandatory. It is recommended that the naming of custom error codes should be unique in the system, otherwise the system will use the first one it finds.

Parameters that can assist in writing custom error messages together with a Map Index Key based on type are shown in the tables below. In addition, an example of adding a custom error code is shown below as a guide.

The following list illustrates parameters that can be used in custom error messages.

`ruleName`	The name of the rule causing the exception.
`ruleId`	The ID of the rule causing the exception.
`lineNumber`	The line number where the exception occurred.
`productName`	The name of the product calling the rule.
`productVersion`	The version of the product calling the rule.
`xPath`	The xPath to the rule.

When creating custom error messages, the Map Index Key for parameters that can be used by type are denoted by the "X" in the respective table rows.

Map Index Key

Map Index Key	EXECUTE RULE	INTERPOLATE	LOOK UP	QUERY	Description
columnName			X	X	The names of the columns. Multiple values comma separated.
filterAttribute	X		X	X	The FILTER BY value.
fromComponent	X		X	X	The FROM value.
intValue1		X			Integer value.
intValue2		X			Integer value.
orderColumn			X		The ORDER BY value.
productName	X		X	X	The name of the product.
productVersion	X		X	X	The version of the product.
ruleName	X				The name of the rule to execute.
whereStatement	X		X	X	The WHERE statement (first WHERE statement after fromComponent).

Related Information

Custom Rule Error Message Example [page 396]

18.8.6.2.1 Custom Rule Error Message Example

A rule can be added to any component and therefore exception handling can be used. But the custom error message must be added to the `Rule Error Messages` component.

In order to illustrate creating a custom error message we created a test Product called 'Script Rule Exception Handing'.

In the `Configuration` folder we assembled and defined our `Rule Error Messages` component.

In addition, we added a `Product Services` folder containing the `Custom Publishing` component and a `Product Web Services` folder with `Product Web Services` component. This would allow you to see an XML representation of an error caught by the system in Product Web Services as an option.

> **i Note**
>
> If you are using `Product Web Services` then both Product Services and Product Web Services must be specified in your Product object along with the `Rule Error Messages` component.

Finally, we added a `Components` folder with a component labeled "ComponentOne". In this component we added data value rows with valid state codes of: AK, AL, AR, AZ and CA.

Over and above the prepopulated system error codes and error messages, we would like to see a custom error message when our rule is run and a valid state isn't found.

For consistency and to distinguish between a custom and system error code we have named our custom error code starting with USRERR_ rather than SYSERR_.

It is important the `ErrorCode` name is unique. For example, USRERR_STATE_NOT_FOUND.

As described below and for testing, we added a ServiceName and Rule to our `Product Web Services` component. In addition, we added our custom error code and error message to the `Rule Error Messages` component.

Adding a Custom Error Message to the Rule Error Messages Component

1. Navigate to the `Configuration` folder and locate the `Rule Error Messages` component in the tree view.
2. Select the *Values* tab.
3. Select the row heading of the last data value row and choose *Add*.
4. Select the *ErrorCode* column, and enter the following value: **`USRERR_STATE_NOT_FOUND`**
5. Select the *ErrorMessage* column and then choose the down arrow.
   The *Text Editor* displays.
   Error object information can be retrieved from the script rule expression by using the following {} around the attribute name. The syntax for contextInfo map is `{contextInfo.<…>}` where <...> denotes a map index key.
   In our example, we will use `{contextInfo.whereStatement}`
   - Error object information can only be retrieved for the error object that contains the error details (no parent error object information).
   - If the map index key name is misspelled, the system won't know to replace the value.

   > **i Note**
   >
   > The system is case-sensitive.
6. Enter the following in the *Text Editor*: **`{contextInfo.whereStatement} was not found. Please verify your component contains this data.`**
7. Select *OK*.
8. Select the row and save your changes.

Creating a Rule with Error Message Exception Handling

In our example we have created our rule in the `Product Web Services` component so we can make use of Product Web Services but it can be added to any component.

1. Select the `Product Web Services` component in the tree view.
2. Select the *Values* tab.
3. Select the white box of the *ServiceName* field and enter a name. For example, **`stateLookup`**.
4. Select the white box of the *Rule* field and choose *Create*.
   The *Capture Rule Info* dialog displays.
5. Enter **`stateLookup`** in the *Rule Name* field.
6. Select Script from the *Step Type* dropdown list.
7. Select String from the *Rule Return Type* dropdown list.
8. Choose *OK*.
9. Select the *Properties* tab and choose the *Add a new Input Argument* icon.
10. Double-click below *Name* and enter a name. For example, **`state`**.
11. Double-click below the *Type* field and select String from dropdown list.
12. Select the *Click to Edit* link.
    The *Rule Editor* dialog displays.
13. Write the rule expression:

    ```
    RULE MESSAGE ON ERROR "USSRERR_STATE_NOT_FOUND"
    val TYPE NUMBER
    val = LOOKUP VALUE FROM ComponentOne WHERE STATE=state
    RETURN val

    END RULE
    ```

    Our example is written at the rule level so any error caught between `RULE` and `END RULE` will be captured.
14. Select *Save Script* icon.
    A message displays to indicated the changes have been saved.
15. Select *OK*.
16. Close the *Rule Editor*.
17. Select the *Build Rule* icon.
    The Rule Deployment dialog displays.
    The Rule compiler runs and message displays to indicate the rule complied successfully.
18. Close the dialog.

Testing the Rule and Viewing the Rule Error Report

You are now ready to test your rule and see how your custom error message is shown in the Rule Error Report.

1. Select *Rule Unit Test* icon.
   The *Rule Testing* dialog displays.
2. At state enter a valid state as added to our component labeled "ComponentOne" for example "AK"(case sensitive). This will confirm the rule runs successfully with out an error message exception.
3. Select *Run*.

The *Rule Testing* dialog refreshes with no error and a Rule Result of AK.

4. Select the *back* button.
5. At state enter a value not added to our component labeled "ComponentOne" e.g. FL. This will show our custom error message exception.
6. Select *Run*.
   The FS-PRO Error Report displays.
7. Select *Rule Error Report* link on the right panel to display details.

18.8.7 Audit Trail Information for Script Rules

The audit trail records all actions users perform on a script rule.

> **i Note**
>
> A newly created rule must be saved before any information will be included in the audit trail. The audit trail only includes changes to an existing script rule. It will not include changes made during creation. Note that the *Database Migration Tool* must have been run during installation or upgrade in order for this feature to work properly.

When you edit a step or input argument within a script rule, the change will be visible in the Audit Trail.

After you have saved your changes and exited the *Rule Painter*, open the *Audit Trail* tool.

When you click the filter button, it will display the following details about the change:

- Which user made the change.
- A high level description of the action.
- The exact change that was performed.

You can export the audit data (using the current filter) from the selected tab into a CSV file.

For more information about the Audit Trail in general, see the *Viewing Object Histories and Audit Trails* topic.

Related Information

Viewing Object Histories and Audit Trails [page 61]

18.9 Default Validation Rules in the `Best Personal Protection` Product

When an application or screen for in the `Best Personal Protection` product is validated, the validation checks will be executed. Any validation errors will be listed in the error grid within the table of contents and the validation summary for the currently displayed screen.

The table below displays the default validations that are preset in the `Best Personal Protection` product. You can open the validations in the *Rule Painter* to customize the values.

Validation Rule Name	Path	Default Check
`mutualExclusiveCoverage`	*Product folder* > *Contract* > *Configuration* > *Integration* > *FSPM Validation*	Verifies that the Interior Glazing coverage and the Building and Interior Glazing coverage are mutually exclusive
`checkPostalCode`	*Product folder* > *Contract* > *Insured Objects* > *SAP IO House* > *Configuration* > *Integration* > *FSPM Validation*	Verifies that the postal code is in the correct format for Germany.
`checkDiscountAmt`	*Product folder* > *Contract* > *Surcharges and Discounts* > *Dimensions* > *Configuration* > *Integration* > *FSPM Validation*	Verifies that the annual premium before tax is a negative amount.
`CheckMinPremiumForPaymentFrequency`	*Product folder* > *Contract* > *Premiums* > *SAP PRM PnC Common Recurring* > *Configuration* > *Integration* > *FSPM Validation*	Verifies that the annual premium after tax is less than 60 Euros for monthly payment.
`CheckTariffPremium`		Verifies that the tariff premium is less than 30 Euros.
`CheckPaymentFrequency`		Verifies that the payment frequency is not specified or invalid

Related Information

Working with Product Rules [page 318]
Using the Sample Content [page 26]

19 Custom Object Types

FS-PRO lets you define certain object types for use in products and create custom object types.

The objects in FS-PRO are categorized into the following three types:

Static	System objects created by SAP that you can't edit. These are often labeled as base objects (for example, `Component Base`, `Product Base`, and so on).
Configurable	System objects created by SAP that you can edit to some extent. These are object types that are user customizable, such as the `Forms Library` and `Forms Mapping` type components.
Custom	Objects that you create from scratch or by deriving them from the SAP base object types.

Custom objects are user-defined variations on the SAP base object types. As with any object, you create custom objects in the Product Studio. Before you can create a custom object, you must define its type in the *Object Modifier Type Manager*, specifying elements, such as the base object type that it derives from, whether its attributes are modifiable, whether it can have data or sub-objects, and so on. A custom object can only derive from one of the following base objects: product, component, or questionnaire. The following table summarizes the relationship between base types and custom object types:

Use this base object type ...	To create these custom object types ...
product	product
	coverage
	data definition
component	component
questionnaire	questionnaire model

In Design Time, a custom object type is subject to the same restrictions as its base object type—you can only create instances of the object within the same folder type that the base object type requires. For example, you can only create a new instance of a custom data definition inside a product folder, or of a custom questionnaire inside a questionnaire folder. The following table outlines these restrictions:

These custom object types...	Can only be created in this folder type...
product	product
coverage	coverage
data definition	data definition
component	component

These custom object types...	Can only be created in this folder type...
questionnaire	questionnaire

You can create your own categories for organizing custom object types. When you create a new custom object type, you assign it to one of these categories. You can use categories to quickly search for custom object types.

Related Information

Object Attributes Dialog [page 402]
Searching for a Custom Object Type [page 405]
Creating a Custom Object Type [page 406]
Editing the Attributes of a Custom Object Type [page 407]
Deleting a Custom Object Type [page 407]
Creating a Custom Object Types Category [page 408]
Editing a Custom Object Type Category [page 409]
Deleting a Custom Object Type Category [page 409]

19.1 Object Attributes Dialog

The *Object Attributes* dialog lets you specify a custom object type's properties.

This dialog displays when you create a new custom object type or when you edit an existing custom object type.

Base Type

Specifies the base object type that you want your custom object type to derive from.

The attributes appearing in the dialog change to match the base object type.

A custom object type inherits many of the fundamental attributes and behaviors of its base type. You can only change a subset of these attributes and behaviors.

For example, in the *Product Studio* each base object type has its own folder type, which is the only place where you can create or store it. A custom object inherits this behavior from its base object.

This procedure describes the attributes that you can customize.

Modifier Code

Defines an identifier for the custom object type. This value must be unique among all custom object types, regardless of base type.

Limits: Up to 20 characters

> **i Note**
>
> An object's *Modifier Code* can be used in XML queries, and in creating association rules.

Modifier Type Name

Defines the name for the object type.

In the *Product Studio*, this value appears in the *ModifierType* column in the detail panel.

Created By

Indicates whether the base object type is System or Custom.

You can't edit this field.

For a new custom object, the value in this field is always Custom.

Category

Specifies the custom object category that you want to assign the object type to.

Can Have Data

This field displays for object types derived from the component base object.

Controls whether a new object of this type is allowed to contain data.

> **i Note**
>
> At Design Time, you can override the *Can Have Data* setting.

Number of Data Rows

Indicates the default number of rows that can be created in the *Values* tab.

The following options are available:

One	Specifies that the new object can have one row.
Many	Specifies that the new object can have multiple rows.

Image

Specifies an icon to represent the object type wherever it appears in Design Time.

You can use an icon provided by FS-PRO or your own custom.

The image size must be sixteen by sixteen pixels.

The image must be in one of the common graphical formats used in browsers: PNG, GIF, or ICO.

Can Have Public Data

This field displays for object types derived from the component base or questionnaire base object.

Specifies when you can add data to the object during Design Time.

The following options are available:

Yes	Indicates that you can add data to an object when accessed standalone.
No	Indicates that you can add data to an object only when it is part of a product.

Can Have Children

Specifies whether you can add sub-objects to the object during Design Time.

The following options are available:

Yes	Indicates that you can add sub-objects to the custom object.
No	Indicates that you can never add sub-objects.

Association Rule

This field displays if you select Yes in the *Can Have Children* field.

Creates a conditional rule that determines the sub-objects that you want to allow this custom object type to have.

> **i Note**
>
> A custom object type can have both an association list and an association rule active at the same time. If both an association rule and an association list are active, the rule takes precedence.

Association List

This field applies if you selected Yes in the *Can Have Children* field.

Specifies the sub-objects that you want to allow this custom object to have.

Attributes Based On

This field displays for object types derived from the component base object.

Specifies a set of default attributes.

At Design Time, all new objects of this type will have these attributes.

Can Extend Attributes

This field displays for object types that derive from the component base object.

Specifies whether you can add or modify attributes during Design Time.

The following options are available:

Yes Indicates that you can add or modify attributes when accessing the object in standalone mode.

No The object's *Attributes* tab in Design Time becomes read-only.

List of Tabs

Enables or disables the *Component Painter* tabs available at Design Time for this object type.

The following options are available:

- Manual
- Attributes
- Values
- Comments
- Custom Messaging
- All

Can Publish

This field is available for object types that derive from the `Product Base` object.

Specifies whether you can publish the object during Design Time.

The following options are available:

Yes Indicates that you can publish the object.

No Indicates that you can't publish the object.

Hidden

Specifies whether the object type is available at Design Time.

The following options are available:

No Indicates that the object type is available at Design Time.

Yes Indicates that the object type is hidden.
The object type isn't available at Design Time.

Enable Unit Test

This field is no longer used.

Unit Test URL

This field is no longer used.

Parent Can Be Substituted

Specifies whether the reference object type has a parent that can be substituted.

The following options are available:

Yes Indicates that you can substitute the parent. This is the default value for `Reference Objects`.

No Indicates that you can't substitute the parent.

When modifying the *Parent Can Be Substituted* option in the *Properties* dialog it can't be changed from Yes to No if a node of that custom object type has that option set to Yes
A warning message is displayed: `The Parent Can Be Substituted option cannot be changed to "No". The option is set to "Yes" in nodes using this modifier type. Update the value in the following nodes first`. A list of nodes is presented in alphabetical order.

19.2 Searching for a Custom Object Type

To view or edit a custom object type, you can first search for it in the *Object Modifier Type Manager*.

Procedure

1. In the Product Modeler, go to *Tools* > *Object Modifier Type Manager*.
   The *Object Modifier Type Manager* opens.
2. Specify the name of the modifier that you want to search for in the *Modifier Type* field.
3. Specify the organizing group that you want to search in in the *Category* field.
4. Choose *Search*.
   The *Search Results* pane lists the objects that match your search criteria.
5. Select the object that you want to view.

Results

The object details display in the *Object Attributes* pane.

19.3 Creating a Custom Object Type

You create custom objects in the Product Studio. However, before you can create a custom object, you must define its type in the *Object Modifier Type Manager*.

Context

A custom object can derive from one of the following SAP base objects types:

- Product
- Component
- Questionnaire

Procedure

1. In the Product Modeler, go to *Tools* > *Object Modifier Type Manager*.
   The *Object Modifier Type Manager* opens.
2. Choose *New*.
   The *Search Results* dialog displays.
3. Complete the fields.
4. Save your changes.

Results

The custom object type is created.

Related Information

Object Attributes Dialog [page 402]

19.4 Editing the Attributes of a Custom Object Type

You can edit a custom object type with the modifier type of Custom.

Prerequisites

Only custom object types with a Created By type of Custom can be edited. If the `Created By` type is System, you can't edit the object's attributes. Some attributes can only be defined at object creation. You can't edit all attributes.

Procedure

1. In the Product Modeler, go to *Tools* > *Object Modifier Type Manager*.
   The *Object Modifier Type Manager* opens.
2. Search for the object type.
3. Select the object type that you want to edit in the *Search Results* panel.
   The object details display in the *Object Modifier Type Attributes* pane.
4. Choose *Edit*.
   The *Object Modifier Type Attributes* dialog opens.
5. Edit the fields.
6. Save your changes.

Related Information

Object Attributes Dialog [page 402]

19.5 Deleting a Custom Object Type

If you no longer need a custom object type, you can delete it.

Prerequisites

Only custom object types with a Created By type of Custom can be deleted.

Procedure

1. In the Product Modeler, go to *Tools* > *Object Modifier Type Manager*.
   The *Object Modifier Type Manager* opens.
2. Search for the object type.
3. Select the object type that you want to delete in the *Search Results* panel.
   The object details display in the *Object Modifier Type Attributes* pane.
4. Choose *Delete*.
5. Choose *OK*.

Results

The object type is deleted.

19.6 Creating a Custom Object Types Category

You can organize your custom object types into categories. These categories help in searching and reviewing object types.

Procedure

1. In the Product Modeler, go to *Tools* > *Object Modifier Type Manager*.
   The *Object Modifier Type Manager* opens.
2. Select the down arrow button beside the *Category* dropdown list.
3. Choose *Create New*.
   A dialog displays.
4. Define an identifier for the category in the *Code* field.

   This field has a maximum value of ten characters.

   > **i Note**
   >
   > After you close the dialog, you can't change the *Code* field, which becomes read-only.

5. Define a name for the category in the *Category Description* field.

   This field has a maximum value of 255 characters.

   This value appears in the *Category* dropdown list.
6. Choose *OK*.

Results

The custom object type category is created.

19.7 Editing a Custom Object Type Category

If you need to change a custom object type category, you can edit it in the *Object Modifier Type Manager*.

Procedure

1. In the Product Modeler, go to *Tools* > *Object Modifier Type Manager*.
   The *Object Modifier Type Manager* opens.
2. Select the down arrow button beside the *Category* dropdown list.
3. Choose *Edit*.
   A dialog displays.
4. Edit the name for the category in the *Category Description* field.
5. Choose *OK*.

19.8 Deleting a Custom Object Type Category

If you no longer need a custom object type category, you can delete it.

Procedure

1. In the Product Modeler, go to *Tools* > *Object Modifier Type Manager*.
   The *Object Modifier Type Manager* opens.
2. Select the down arrow button beside the *Category* dropdown list.
3. Choose *Delete*.

Results

The object type category is deleted.

20 Reference Objects and Reference Config Objects

In the Product Modeler environment, marketable products represent a real world product, with all its features, data storage, and business logic. A marketable product is made up of various objects:

- Coverage groups
- Coverages
- Components
- Questionnaire models
- Data definitions
- Forms library objects
- Data model objects
- Configuration objects
- Reference objects

You create a marketable product in the Product Modeler by assembling component, questionnaire, coverage group, and other types of objects, into a product object.

`Configuration` objects are used to control the behavior of FS-PRO and FS-QUO.

`Reference` objects (originally called reference products) provide a central repository for enterprise-wide data. They can hold information about rating reference tables, application dropdown reference tables (domain tables), and product data tables, rating, forms, coverage selection and show/hide data. A `Reference Config` object provides a link between the marketable product and reference objects. By maintaining this shared data in reference objects and then linking to other objects, only the reference objects need be republished. When changes are made in the reference object there is no need to retest the marketable product, coverage group or coverages.

> **i Note**
>
> You can have multiple `Reference` objects.

By creating `Reference` objects a variety of products can reference enterprise data instead of duplicating that data over and over for each product. Reference objects increase the re-usability of data and objects throughout the entire product catalog.

For example, you might have a reference object named "Commercial Auto" that contains the rating, forms, and rate tables that apply to all Commercial Auto products for the entire country. Modeling of reference objects for multiple jurisdictions (such as states, counties or cities) is also possible, allowing for the configuration and return (in Runtime) of forms and rate tables per specified risk state. State-level ISO models can be configured, however there are no out-of-the-box ISO templates.

Related Information

Reference Object Modeling [page 411]
Adding a Reference Config Object [page 412]
Adding the Reference Object Line [page 413]
Adding a Reference Object [page 414]
Adding the Reference Object Link [page 414]
Modifying the Rule to Look Up Reference Objects [page 415]
Defining the Reference Object in the Marketable Product [page 416]

20.1 Reference Object Modeling

Companies that configure products with deviation layers at the country (Countrywide) and/or state levels can utilize the predefined `Underwriting Application Reference Object Template` template that is available out-of-the-box from the `Underwriting Application Bootstrap`.

The bootstrap also includes the `Reference Object Configuration` template, with preassembled `Reference Object Line` and `Reference Object Link` components that will be utilized to model the reference object. All available reference objects (differentiated by Line of Business, jurisdiction/state and effective expiration dates) will need to be listed.

> **i Note**
>
> Non-standard deviation layers such as Underwriting Company, Market Segment, County or City can also be accommodated, however additional configuration will be required to model them.

The following steps need to be be executed to create and model the reference objects, and associate them to the applicable marketable product.

1. Create a new company specific `Reference Config` object based on the existing `Reference Object Configuration` template.
2. Add a `Reference Object Line` object to the `Reference Config` object. This specifies the Coverage Groups (LOBs) that the `Reference Config` object applies to.
3. Create new reference objects for each deviation layer (state, city, etc.), based on the `Underwriting Application Reference Object Template` template.
   For each template, specify the settings for each deviation layer, such as Forms Library, Rate Tables, UI Lookup Tables, Validations, Calculation Rules, Stat Coding, etc.

   > **i Note**
   >
   > Common settings should be configured in the marketable product, Line of Business or Coverage-specific templates. All deviation layer-specific settings should reside in the corresponding reference object and then referenced from the marketable product as needed.

4. Add a `Reference Object Link` object to the `Reference Config` object. List all of the reference objects and specify the LOB and deviation layer (e.g. state) it pertains to.

5. Modify the `getReferenceLinkInfo` API rule, which is used to look up a specific reference object. Modify the rule to specify the criteria to be used for the query.
6. Associate the new company-specific `Reference Config` object to the marketable product by specifying the `Reference Product` and `Reference Product Version` settings in the `Config` object within the marketable product.

Related Information

Adding a Reference Config Object [page 412]
Adding the Reference Object Line [page 413]
Adding a Reference Object [page 414]
Adding the Reference Object Link [page 414]
Modifying the Rule to Look Up Reference Objects [page 415]
Defining the Reference Object in the Marketable Product [page 416]

20.2 Adding a `Reference Config` Object

Create a new company-specific `Reference Config` object based on the existing `Reference Object Configuration` template.

Context

It is best practice to create the `Reference Config` object in the following location: *Content Repository* > `<company_library>` > *Configuration Objects* .

In the path above, `<company_library>` refers to your company library.

Procedure

1. Navigate to *Content Repository* > `<company_library>` > *Configuration Objects* in the Product Studio.
2. Create a `Reference Config` object based on the `Reference Object Configuration` template.
3. Add the `Reference Config` object to the *Edit Product Deployments* in the Design Time Administrative Console.

   This is required to allow the publishing of this product template to the Runtime server.

   For more information, see Adding an Association Between a Product and a Remote Server

4. Build and publish the reference object template.

Related Information

Defining the Reference Object in the Marketable Product [page 416]

20.3 Adding the Reference Object Line

Specify the Coverage Groups (LOB's) that the `Reference Config` object applies to.

Procedure

1. Navigate to the `Reference Config` object in the *Product Studio* and open it.
2. Expand the `Reference Config` folder and select the `Reference Object Line` component.
3. Select the *Values* tab.
4. Right-click at the top row on the cell closest to the right and select *Add* from the pop up menu to add rows for the lines of business that your company writes (for example, General Liability; Commercial Property; Commercial Auto).
5. Select the *Code column* attribute and enter the code abbreviation for the first line of business.

   For example, **`GL`**.
6. Select the *Description column* attribute and add a complete description.

   For example, **`General Liability`**.
7. Select the checkbox.
8. Save your changes.

Next Steps

Repeat steps 4-8 for all other Lines of Business that the marketable products will be referencing.

20.4 Adding a Reference Object

Create a new `Reference Object` object for deviation layers (for example, state) based on the existing `Underwriting Application Reference Object Template` template.

Context

It is best practice to create the `Reference Object` object in the following location: *Content Repository* > `<company library>` > *Reference Objects*.

In the path above, `<company library>` refers to your Company Library.

Procedure

1. Navigate to *Content Repository* > `<library>` > *Reference Objects* in the Product Studio.
2. Create a `Reference Object` object based on the `Underwriting Application Reference Object Template` template.
3. Add the `Reference Object` object to the *Edit Product Deployments* in the Design Time Administrative Console.

   This is required to allow the publishing of this product template to the Runtime server.

   For more information, see Adding an Association Between a Product and a Remote Server

   .
4. Build and publish the reference object template.

20.5 Adding the Reference Object Link

Within each `Reference Config` object, list all of the reference objects and specify the LOB and deviation layer (e.g. state) it pertains to.

Procedure

1. Navigate to the `Reference Config` object in the *Product Studio* and open it.
2. Expand the `Reference Config` folder and select the `Reference Object Link` component that is anchored to the `Reference Object Line` component.

3. Select the *Values* tab.
4. Right-click on the top row and select *Add* from the pop up menu to add rows for the deviation layers your company writes.
5. Select the *State Code* column and enter the abbreviation for the state you write business in.

   For example: **`OH`**.
6. Select the *Reference Object Name* column and enter the name of the reference object template.

   For example, **`Mock GL Reference OH`**.
7. Select **`ALL`** from the `Transaction Type` attribute.

   A reference object can be configured for specific transactions types, such as renewal or endorsement. This will require additional configuration within the `getReferenceLinkInfo` API rule.
8. Save your changes after adding each row.
9. Repeat steps 4-8 to add rows for all states and deviation layers.
10. Return to the `Reference Object Line` component.
11. For each Line of Business added, select the  plus sign icon.
12. Ensure that none of the checkboxes on the right are selected.
13. Select the checkbox for each *Reference Object Name* that appears in the *Reference Object Link* dialog that is applicable for the Line of Business.
14. Use the navigation icons at the bottom to select from additional pages if required.
15. Save your changes.
16. Repeat steps 10-15 for the remaining Lines of Business.

20.6 Modifying the Rule to Look Up Reference Objects

You can edit the query criteria for the `getReferenceLinkInfo` API rule within a `Reference Config` component, to look up a specific reference object.

Context

When the `getReferenceLinkInfo` API rule executes, it will query the anchored `Reference Object Line` component and search for the reference object that corresponds to the query criteria. For example, LOB code and state code. If non-standard ratebooks (such as for counties or cities) are present, the API rule will need to be modified to return the applicable ratebook based on revised criteria.

Procedure

1. Navigate to the `Reference Config` object in the *Product Studio* and open it.
2. Expand the `Config API` folder and select the `Application Configuration API` component.

3. Select the *Values* tab.
4. Edit the service rule for the `getReferenceLinkInfo` record.
5. Save your changes.
6. Build the rule.

20.7 Defining the Reference Object in the Marketable Product

After all the objects that you created are defined in the `Reference Config` object, you must specify that `Reference Config` object in the product. This setting links the product with the appropriate reference object.

Procedure

1. Navigate to `<product_name>` > *Configuration* > *Config* .
2. Select the *Values* tab.
3. Select the `appConfigProduct` record and override the row.
4. In the `Value` attribute, specify the name of the `Reference Config` object that lists all the reference objects.
5. Select the `appConfigProductVersion` record and override the row.
6. In the `Value` attribute, specify the version number of the the `Reference Config` object that lists all the reference objects.
7. Save your changes.

Related Information

Working with Component Values [page 111]
Adding Data Value Rows [page 115]
Editing Data Value Rows for Components [page 116]

21 Insurable Objects

In a coverage-based product, an insurable object is an item that can be insured, such as a building, vehicle, etc.

In the case of a house, it has certain attributes, such as an address, square footage, construction date, etc. These attributes apply to the object itself, regardless of how it's used in an insurance policy.

An insurable object can be used by different policies and by different types of policies.

The goal of the Insurable Object Framework is to provide in Global Reference Object that can reuse in different product layers.

Related Information

Configuring a Global Reference Object for Insurable Objects [page 417]
Configuring Insurable Object Rules in a Coverage-based Product [page 419]

21.1 Configuring a Global Reference Object for Insurable Objects

You need to configure a global reference object for an insurable object to make it available to all coverage-based products.

Context

This information must be entered manually.

Procedure

1. Open the `SAP Insurance UWA Reference Object Template`.
2. Navigate to *SAP Global Reference Object* > *Configuration* > *Integration* > *Insurable Object* > *Insurable Object Category*.
3. Select the *Values* tab.
4. Add new row and enter the following information:

a. Enter a name for the insurable object in the `Name` attribute column.
b. Enter the code for the insurable object as defined in FS-PM in the `FSPM Code` attribute column.
c. Enter the name for the insurable object as defined in FS-PM in the `FSPM Name` attribute column.
d. Enter a description of the insurable object in the `Description` attribute column.
e. Save your changes.

5. Navigate to *SAP Global Reference Object* > *Configuration* > *Integration* > *Insurable Object* > *Insurable Object Type*.
6. Select the *Values* tab.
7. Add new row and enter the following information:
   a. Enter a name for the insurable object in the `Name` attribute column.
   b. Enter the code for the insurable object as defined in FS-PM in the `FSPM Code` attribute column.
   c. Enter the name for the insurable object as defined in FS-PM in the `FSPM Name` attribute column.
   d. Enter the table name of the FS-PM insured object component in the `Main Table` attribute column.
   e. Enter the name of the address table of the FS-PM insured object component in the `Address Table` attribute column, if applicable.
   f. Save your changes.
8. Navigate to *SAP Global Reference Object* > *Configuration* > *Integration* > *Insurable Object* > *Insurable Object Field*.
9. Select the *Values* tab.
10. Add new row and enter the following information:
    a. Enter a name for the insurable object in the `Column Name` attribute column.

       The information in this attribute must match the equivalent item as defined in FS-PM.
    b. Enter the name of the RFC from the column in the main table in FS-PM in the `RFC Field Name` attribute column.

       The information in this attribute must match the equivalent item as defined in the main table in FS-PM.
    c. Enter the type from the column in the main table in FS-PM in the `RFC Data Type` attribute column.

       The information in this attribute must match the equivalent item as defined in the main table in FS-PM.
    d. Enter **`1`** or **`0`** from the column in the main table in FS-PM in the `Mandatory Flag` attribute column.

       The information in this attribute must match the equivalent item as defined in the main table in FS-PM.
    e. Save your changes.
11. Navigate to *SAP Global Reference Object* > *Configuration* > *Integration* > *Insurable Object* > *Insurable Object Address Field*.

    This is only applicable if an address table exists for the component.
12. Select the *Values* tab.
13. Add new row and enter the following information:
    a. Enter a name for the insurable object in the `Column Name` attribute column.

       The information in this attribute must match the equivalent item as defined in FS-PM.
    b. Enter the name of the RFC from the column in the address table in FS-PM in the `RFC Field Name` attribute column.

       The information in this attribute must match the equivalent item as defined in FS-PM.

c. Enter the type from the column in the address table in FS-PM in the `RFC Data Type` attribute column.

   The information in this attribute must match the equivalent item as defined in FS-PM.

d. Enter **`1`** or **`0`** from the column in the address table in FS-PM in the `Mandatory Flag` attribute column.

   The information in this attribute must match the equivalent item as defined in FS-PM.

e. Save your changes.

14. Navigate to *SAP Global Reference Object* > *Configuration* > *Integration* > *Insurable Object* > *Insurable Object Extension Field*.
15. Select the *Values* tab.
16. Add new row and enter the following information:
    a. Enter a name for the insurable object in the `Column Name` attribute column.

       The information in this attribute must match the equivalent item as defined in FS-PM.

    b. Enter the name of the RFC from the column in the address table in FS-PM in the `RFC Field Name` attribute column.

       The information in this attribute must match the equivalent item as defined in FS-PM.

    c. Enter the type from the column in the address table in FS-PM in the `RFC Data Type` attribute column.

       The information in this attribute must match the equivalent item as defined in FS-PM.

    d. Enter **`1`** or **`0`** from the column in the address table in FS-PM in the `Mandatory Flag` attribute column.

       The information in this attribute must match the equivalent item as defined in FS-PM.

    e. Save your changes.

21.2 Configuring Insurable Object Rules in a Coverage-based Product

You need to alter the display rule configuration to allow updates to insurable objects.

Context

This means updating the `Content Display Rule` to allow all fields to be enabled on the insurable object screen.

Procedure

1. Open the product .
2. Navigate to *Contract* > *Coverages* > *User Interface* > *Data Capture* > *eApplication* > *Values* > *eApplication* > *eApp Section* > *eApp Screen* > *Insured Object Screen*.

3. Update `Content Display Rule` to allow all fields to be enabled on the insurable object screen.
4. Update the `createSampleHHInsuredObject` rule to call the stem with conditions below:
   - `InsuObjLookup` stem: has create and update methods, while the main table in FS-PM is the current table
   - `FSPMInsurableObjectStem` stem: has create and update methods, whether or not the main table in FS-PM is the current table. This stem includes the `categoryName` and `typeName` parameters.

22 Quote Letters

You can configure products to generate a customized quote letter in the FS-QUO individual workflow that can be emailed to the policy holder.

Prerequisites

The Adobe Documents Service (ADS) must be enabled in SAP Cloud Platform.

Your FS-QUO system must have a connection to the Adobe Documents Service (ADS).

Your system administrator must have configured the appropriate *Application* > *AuthoritySuite* > *Env* > *Adobe Documents Service Connection* settings in the Runtime Administrative Console.

Your system administrator must have configured the appropriate *Application* > *AuthoritySuite* > *Env* > *SAP Documents Service Connection* settings in the Runtime Administrative Console.

To have the ability to email quote letters to customers, your system administrator must have configured the appropriate *Application* > *AuthoritySuite* > *Env* > *SMTP Server Connection* settings in the Runtime Administrative Console.

For more information, see the following topics:

- Enabling the Adobe Documents Service
- Configuring an SMTP Server Connection
- Configuring the Document Storage System
- Configuring an SMTP Server Connection
- Application > AuthoritySuite > Env > Adobe Documents Service Connection
- Application > AuthoritySuite > Env > SAP Documents Service Connection

Quote Letter Templates

A generic quote letter template (`SampleQuoteLetter.xdp`) is available out-of-the-box, but there are also templates designed specifically for each line of business:

- Auto: `SampleVehicleQuoteLetter.xdp`
- Life: `SampleLifeQuoteLetter_EN.xdp`
- Household:
  - `SampleHouseholdQuoteLetter_ZF.xdp`
  - `SampleHouseholdQuoteLetter_KO.xdp`
  - `SampleHouseholdQuoteLetter_EN.xdp`

The templates are located in the `Authority Suite - Custom Template Folder (Write)` path in FS-PRO.

To customize the content of the quote letter, you can edit the templates using Adobe LiveCycle Designer to include the desired data.

> **i Note**
>
> A template must be specified for each marketable product.

Workflow

The quote letter is generated by selecting the *Generate Quotation Letter* button on the FS-QUO Fiori Apps. A quote letter can be generated during New Business, Change Business or Renewal workflows.

The letter will be built according to the template assigned in the product and will be populated with data retrieved from a quote option.

Out of the box, a generated quote letter is valid for 30 days from the date of issuance. However, the expiration period of the quote letter is configurable via the `QuoteLetterExpirationDays` configuration variable at the product level.

Selecting the *Generate Quotation Letter* button on the FS-QUO individual workflow results in the following actions:

- Triggers a product service to transform the quote option data to document generation.
- Triggers an API rule which sets default values such as the expiry date.
- Triggers an API rule which will retrieve the name of the template that is specified in the product.
- Generates the letter in PDF format and displays it.
  - The service that is used to generate the quote letter can be changed. Out of the box, a default SAP service will be used.
  - The XDP template to generate the quote letter can be changed. Out of the box, a generic quote letter template will be used.
  - The quote letter PDF will be stored in your system's CMIS.
  - The *Send Quote Letter* button is available on the popup dialog that appears after the *Generate Quotation Letter* button is selected. This button allows the agent to email the quote letter to the customer.
- Changes the status of the quote to `Quoted`.

Once a quote letter has been created, clicking the *Generate Quote Letter* button won't generate a new quote letter; it will open the existing quote letter.

Once the quote letter has been emailed to the customer, the *Send Quote Letter* button changes to become the *Resend Quote Letter* button. This allows the quote letter to be emailed by the agent again, if needed. Note that the quote letter is not regenerated; it is simply resent.

After an application is quoted and a quote letter is produced, the user has the option to issue the quote or flag it as `Not Taken`.

> **i Note**
>
> The previous iteration of the quote letter feature that used XLS files and schemas is no longer supported. After upgrading to FS-QUO 2.0 FPS1 or later, you will need to customize the new templates and use SAP Cloud Platform Forms by Adobe to generate quote letters.

22.1 Configuring Quote Letters in your Product

22.1.1 Quote Letter Templates

This setting identifies the template to be used when generating the quote letter PDF.

A template must be specified for each marketable product. A generic template has been provided (`SampleQuoteLetter.xdp`), but out-of-the-box quote letters have been configured for the various lines of business:

- Auto: `SampleVehicleQuoteLetter.xdp`
- Life: `SampleLifeQuoteLetter_EN.xdp`
- Household:
  - `SampleHouseholdQuoteLetter_ZF.xdp`
  - `SampleHouseholdQuoteLetter_KO.xdp`
  - `SampleHouseholdQuoteLetter_EN.xdp`

> **→ Remember**
>
> When creating your custom quote letter templates, you will need to create a separate template file for each language. The file name of the template must include the appropriate two digit language code, as per the sample files listed above.

The templates are located in the `Authority Suite - Custom Template Folder (Write)` path in FS-PRO.

Template: `SAP Insurance UWA Product Template`

Go to *Configuration* > *Config*.

Key: QuoteLetterXdpTemplate

Value: the template XDP name (for example, `SampleVehicleQuoteLetter.xdp`)

22.1.2 Quote Letter Expiration Date

This rule sets the number of days that the quote letter is valid.

Common Configuration (UWA)

Out-of-the-box configuration has a default value of 30 days.

Template: `SAP Insurance UWA Product Template`

Go to *Configuration* > *Config*.

Key: QuoteLetterExpirationDays

Value: 30

This setting is utilized by the API rule (`updateQuoteDataForPrint`) when setting quote letter defaults.

22.1.3 API Rule to Set Quote Letter Default Values

Generating a quote letter triggers an API rule that sets default values.

Template: `SAP Insurance UWA Product Template`

Go to *Configuration* > *Product Services* > *Service API*.

API rule: `updateQuoteDataForPrint`

The out-of-box rule has been configured to set the *Quote Expiration* date. For example, when a quote letter is generated, it's only valid for 30 days.

22.1.4 API Rule to Transform Quote Data for Printing

Generating a quote letter triggers an API rule that transforms quote data for printing.

Template: `SAP Insurance UWA Product Template`

Go to *Configuration* > *Product Services* > *Service API*.

API rule: `transformQuoteDataForPrintUsingADS`

This service API is overridden with product-specific implementations.

22.1.5 API Rule to Trigger the Illustrations Calculation

Generating a Life quote letter triggers an API rule is that prompts the illustrations calculation.

Life Capital Product Configuration

This rule checks to see if there are existing illustration calculation results. If there are no illustrations calculation results, it will trigger the illustrations calculation.

Template: `SAP LnA Product Template`

Go to *Configuration* > *Product Services* > *Service API*.

API rule to override: `calculateIllustrations`

22.1.6 API Rule for Send Quote Letter

Selecting the *Send Quote Letter* or *Resend Quote Letter* button allows the agent to email the quote letter to the customer, so that the customer can view and confirm the policy issuance. A service API determines the sender, recipient and content of the email.

> → Remember
>
> According to General Data Protection Regulation (GDPR) regulations, sending automated emails to customers requires obtaining customers' consent.

Template: `SAP Insurance UWA Product Template`

Go to *Configuration* > *Product Services* > *Service API*.

API rule: `sendEmailWithQuoteLetter`

This service API is overridden with product-specific implementations.

Default Values

The out-of-the-box rule sets the following values:

- Current quote option ID
- Sender:
  - The sender information for the quote letter email is comprised of two values: the sender's email address and the reply email address.
  - The sender's email address (`fromAddress`) will be an email address used by the SMTP server that is allowed to send the email. The default value in the rule will need to be overwritten with your own company email address.

    - The reply email address (`replyToAddress`) will be automatically populated with the email address of the Commission Participant that is selected on the *General Information* screen of the *Create Insurance Quote* app flow.
- Recipient:
    - For individual policies, the default recipient is the policy holder. In the case of an auto product, the default recipients are the policy holder and the named driver
    - For a master individual (multi-individual quote), the default recipient is the Master Policy policy holder
    - For a group quote, the default recipient is the Group Default policy holder (i.e. the group policy holder)
- Email content:
    - The default value for the subject line of the email is "Insurance Policy - Quote Letter #`<quote_number>`".
    - The default value for the file name of the attached quote letter is "`<customer_name>`.pdf".
    - The default value for the body text of the email is:

```
Hello <customer_name>,

It was a pleasure speaking to you. Please find the attached Quote letter for
your review.

If you have any questions or concerns, please feel free to contact our
Customer Service Department at <cusotmer_service_phone>, Monday to Friday
from 9-5pm and they will be happy to assist you further.

Thanks again for choosing <company_name> Insurance Services - we look
forward to serving you.

Regards,

<agent_name>
<company_name> Insurance Representative
<company_name>
<company_phone>,<agent_phone>,

<agent_email>
```

23 Inheritance Path Substitution

A key concept to understand is inheritance, a feature that enables you to build on previous work by re-using and extending objects instead of recreating them from scratch. Inheritance is often described as a parent-child relationship, because the child inherits the features of the parent.

An inheritance path is the order of the nodes (objects) from the root of an inheritance tree down to a particular node.

Inheritance lets you create a new product, coverage, or reference object based on an existing one. The new product automatically uses all the objects contained in the existing product object. The objects inherited include the coverages, components, questionnaires, and products.

You can customize the new product by overriding, extending, or disabling inherited objects, while creating or adding further components, values, questionnaires, and products.

Inheritance reduces your maintenance effort: when you update an object contained in a product, your changes are automatically available to all the products that inherit that object.

Most products and reference objects are defined with the following inheritance layers (or levels):

1. Base
2. Industry Definitions
3. Company Base
4. Company Deviations

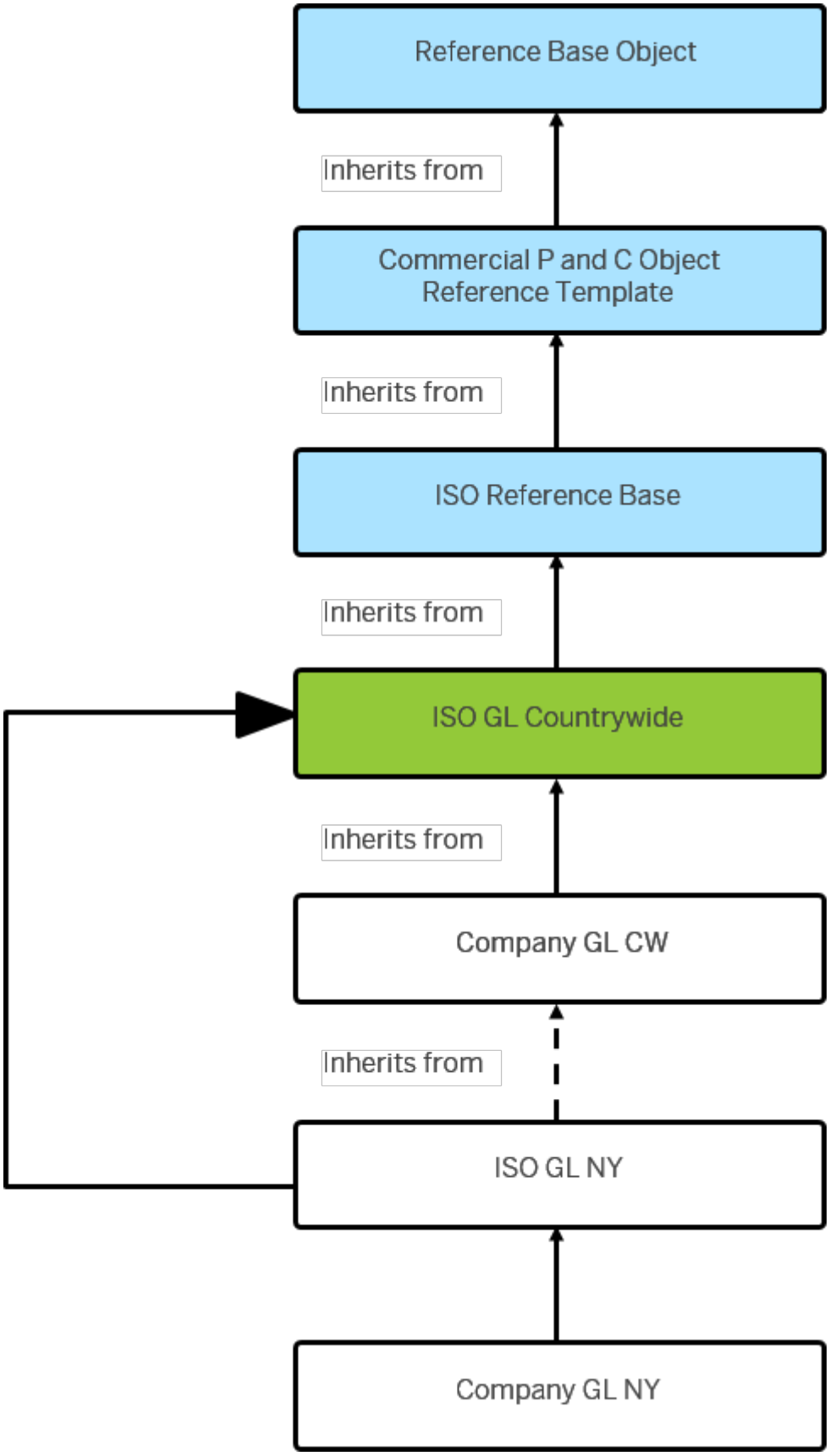
Previously an inheritance path was linear. Now with Inheritance Path Substitution (IPS), you can now substitute paths in an inheritance tree. This allows you to flag a node to indicate the *Parent Can Be Substituted*. When creating subsequent products inheriting from that node, you can choose to substitute a branch into its inheritance path.

> **! Restriction**
>
> IPS is only used for reference objects. It isn't used for marketable products.

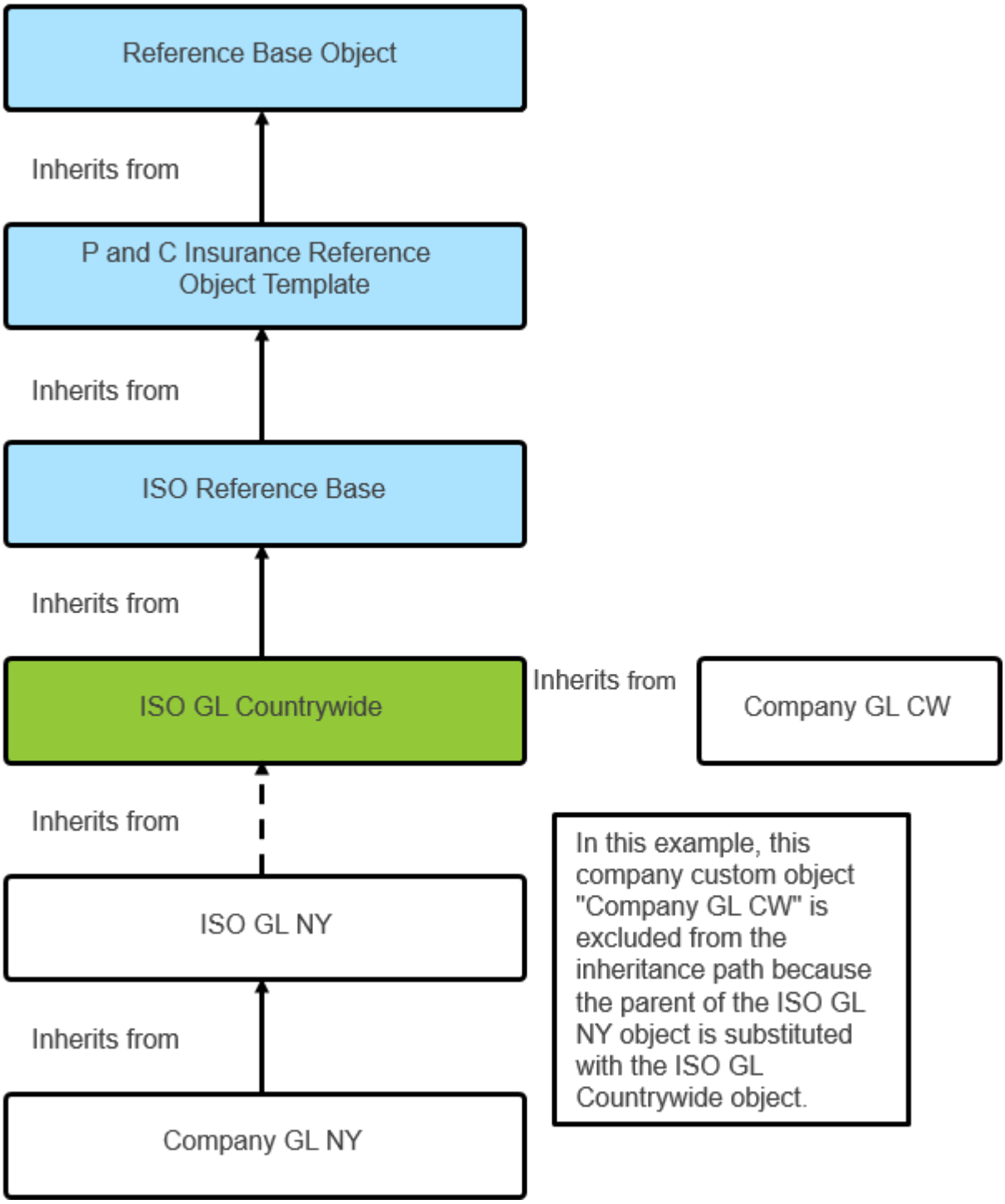
> **i Note**
>
> The change inheritance function is available for objects with Base Type = Product that have a parent product.

Inheritance allows a reference object to contain information about a base object and information from custom layers. By applying IPS as a onetime definition change you can create multiple variations to a line of business by creating separate branches in the inheritance tree thus generating different types of data.

The following diagram provides an illustration of substitution in an inheritance path.


Reference Base Object
Inherits from
Commercial P and C Object Reference Template
Inherits from
ISO Reference Base
Inherits from
ISO GL Countrywide
Inherits from
Company GL CW
Inherits from
ISO GL NY
Company GL NY

Another way to show this inheritance tree is illustrated below:



With IPS you can build out programs to run companies, lines, states, and so on. You can create more than one variation to a line of business by creating a separate branch in the sub-tree per variation.

Related Information

Planning Inheritance in the Company Product Model [page 430]
Creating a Reference Object Whose Parent can be Substituted [page 432]
When Changing the Inheritance Path Substitution Creates Orphaned Data in the Node [page 433]
Creating a Reference Object for Company Base Layers [page 433]
Creating a Reference Object for Company Deviation Layers [page 434]
Modifying a Reference Object with an Inheritance Path Substitution [page 437]
Modifying the Based On Option in the Properties Dialog [page 440]
Modifying the Parent Can Be Substituted Option [page 441]
Deleting a Reference Object with a Parent that can be Substituted [page 442]
Deleting a Reference Object that has Inheritance Path Substitution [page 442]
Publishing Products with an Inheritance Path Substitution [page 443]
Exporting and Importing Products with an Inheritance Path Substitution [page 443]
Copying Products with an Inheritance Path Substitution [page 443]
The Product Studio [page 30]
Changing Inheritance for a Product [page 36]

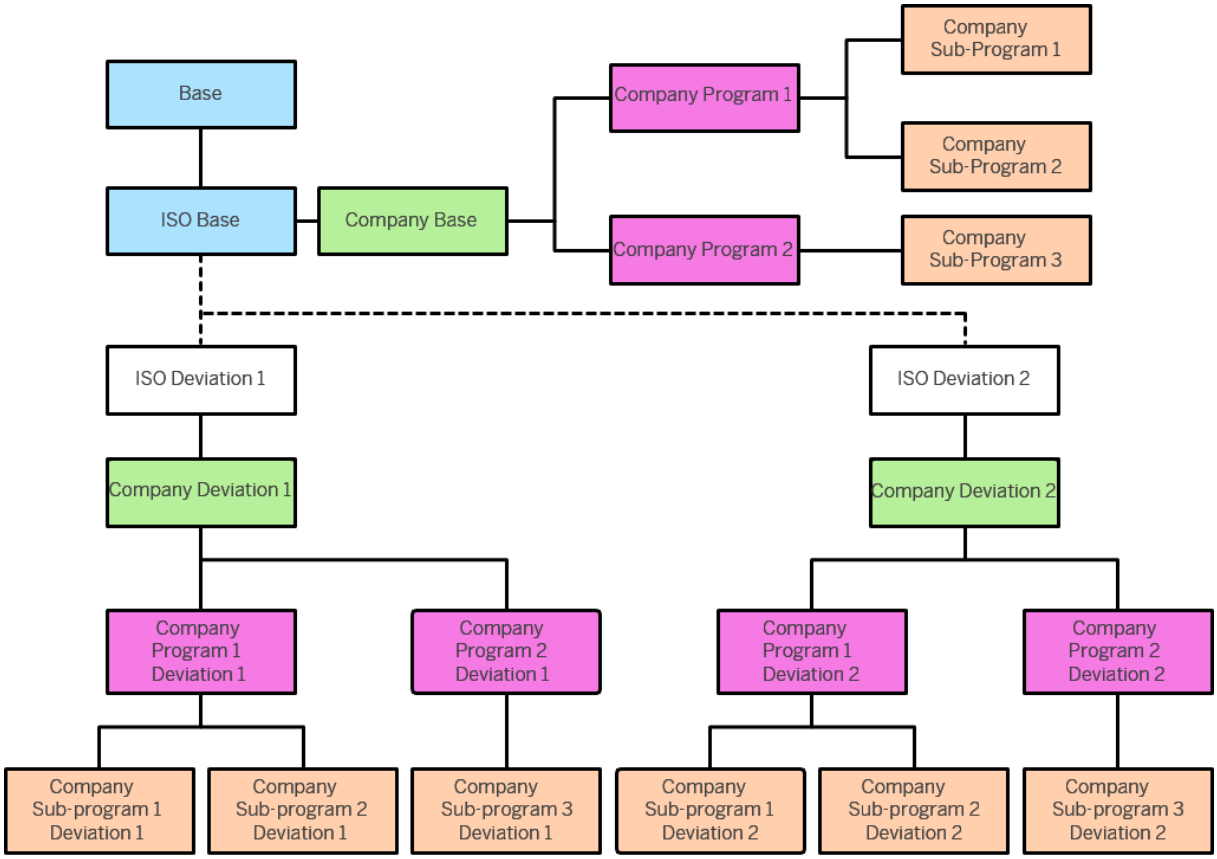
23.1 Planning Inheritance in the Company Product Model

Inheritance is most effective when you start by carefully planning the levels of your inheritance tree, defining what you want to achieve at each level, and keeping in mind how it benefits the next level.

Start by creating a diagram of the inheritance tree for your company product model. You can then follow this diagram when you define reference objects.

Company Reference Object Model Layers

Below is an example of a company reference object model illustrating Company Base Layers and Company Deviation Layers.



Multiple Deviation Layers

More than one company deviation to a line of business can now be modeled by creating a separate branch in the sub-tree per variation.

When defining the `Reference Object` for Company Base Layers, the *Parent Can Be Substituted* field is set to No.

For an ISO Company with ISO Deviation Layers the *Parent Can Be Substituted* field is set to Yes out of the box as depicted in the diagram above.

For a Non-ISO Company you need to determine on which reference object to set this flag. If you deviate by state then this flag will be at the state layer. If you deviate by writing company, then this flag will be at the writing company layer.

There can only be one reference object with the flag set to Yes in the inheritance path.

No Migration Path

If you have an existing reference object structure, you can't automate the conversion to a structure with inheritance path substitution. There isn't a migration path. You must make any changes manually, re-defining the entire inheritance tree.

Planning Questions

The following questions can help you plan the model:

- What custom layers do you need to add?
- Which objects do the custom layer objects inherit from?
- What types of information do you want to get from the reference object?
- What layers need to be substituted (included) to get the information you need?

An inheritance path can only have one `Reference Object` whose parent can be substituted. By substituting that reference object's parent, you can structure a different tree to get data.

23.2 Creating a Reference Object Whose Parent can be Substituted

Procedure

1. Open the Product Studio
2. Navigate to the *Reference Objects* folder where you want to create the `Reference Object`.
3. Select the *New Object* icon .

   The *New Object* dialog appears.

   > **i Note**
   >
   > The the *Parent Can Be Substituted* field is hidden.

4. Enter the name for the `Reference Object` in the *Name* field.
5. Select Reference Object from the *Modifier Type* dropdown list.
6. Choose the *Based On* icon.

   The *Object Picker* dialog displays.
7. Enter a name in the *Object Name* field for the reference object that you want to base the inheritance on for this new `Reference Object`.
8. Choose *Search*.
9. Highlight your selection in the *Object Search Result* panel .
10. Choose *Select*.
    The *New Object* dialog is refreshed. The *Parent Can Be Substituted* field is now displayed and set to the default value of No.
11. Set the *Parent Can Be Substituted* radio button to Yes.

12. Choose *OK*.
    Your changes are saved.

Related Information

When Changing the Inheritance Path Substitution Creates Orphaned Data in the Node [page 433]

23.3 When Changing the Inheritance Path Substitution Creates Orphaned Data in the Node

If changing the Inheritance Path Substitution creates orphaned data in the node, child nodes or products composed of the node a confirmation prompt is displayed:

```
Change in inheritance/inheritance path will cause local data to be deleted. See
log file for details. Do you want to continue?
```

Select *Yes* to continue or *No* to cancel.

Select the link to the log file to open it and view the path to orphaned data.

The system will check the following for orphaned data:

- Local folders and components
- *Manual* tab content
- Message template
- Local attributes
- Data value row values (data and selections including Questionnaires, Rules, and Screens)
- Questionnaire components and questionnaire views

23.4 Creating a Reference Object for Company Base Layers

When defining the reference object for Company Base Layers the procedure is the same as outlined above under *Creating a Reference Object Whose Parent can be Substituted* with one exception. You set the *Parent Can Be Substituted* radio button to No.

Related Information

Creating a Reference Object Whose Parent can be Substituted [page 432]

23.5 Creating a Reference Object for Company Deviation Layers

An ISO and Non-ISO Company model can include deviation layers. For illustration purposes we will describe the deviation for an ISO Company model.

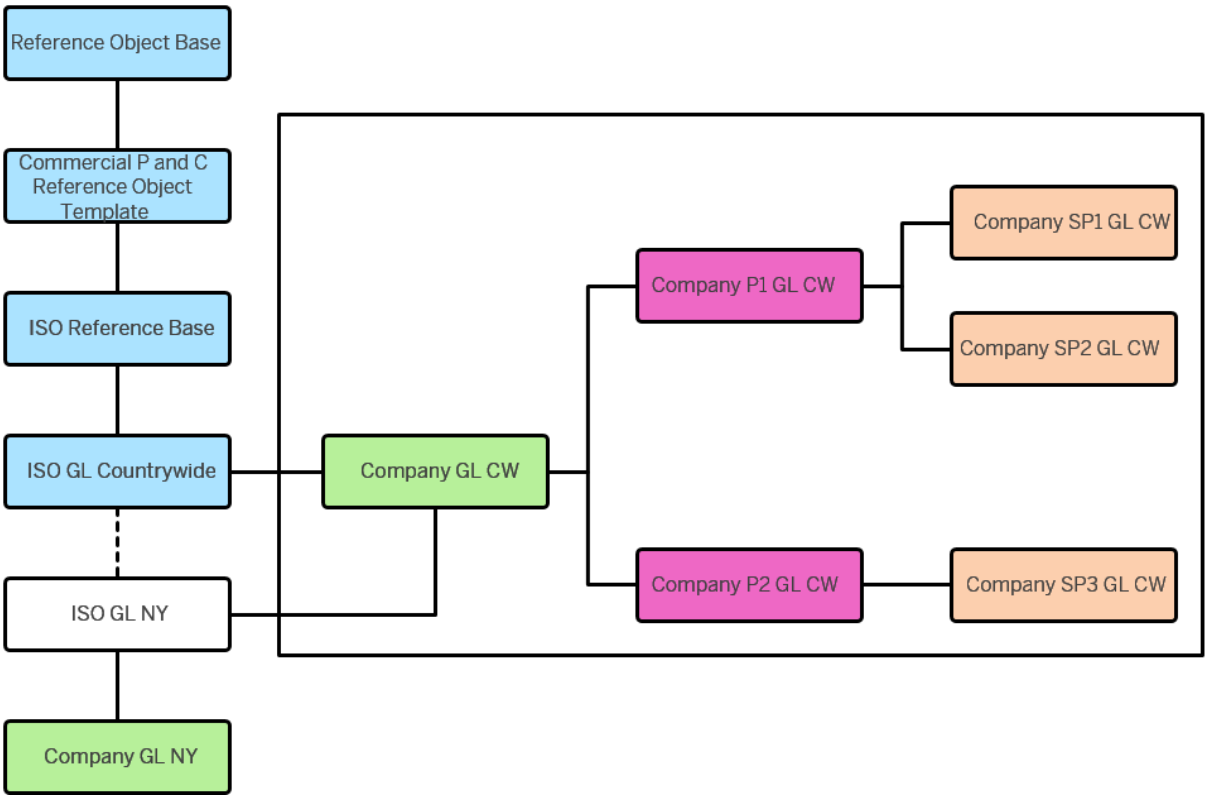
Context

We want to create a deviation layer for programs offered in the state of Ohio. For illustration purposes we have named our programs and sub-programs or deviation layer as "Company P1 GL CW" and "Company P2 GL CW". Each have sub-programs labeled SP1, SP2 and SP3. The abbreviation GL refers to General Liability, P1 refers to Program 1 and P2 refers to Program 2 and the abbreviation SP refers to sub-program.

Typically, a company model includes company deviated state objects in the inheritance tree. These objects contain company data that applies to specific states. In an ISO example, custom company deviated state objects always inherit from the base ISO state objects.

Generally, the node whose parent can be substituted is at a state layer, and custom deviated state objects inherit from state nodes. Therefore, when you create custom deviated state objects in the company state layers, you can select a substitution path.

In creating our Company Program General Liability State deviation layer there is a precondition that must be observed: you have previously implemented all Company base layers (as outlined in red below) and structured as illustrated in the inheritance tree diagram below.



Procedure

1. Navigate to the `Reference Object` folder where the deviation is to be placed.
2. Select the *New Object* icon .

   The *New Object* dialog appears.
3. Enter your Reference Object name in the *Name* field.

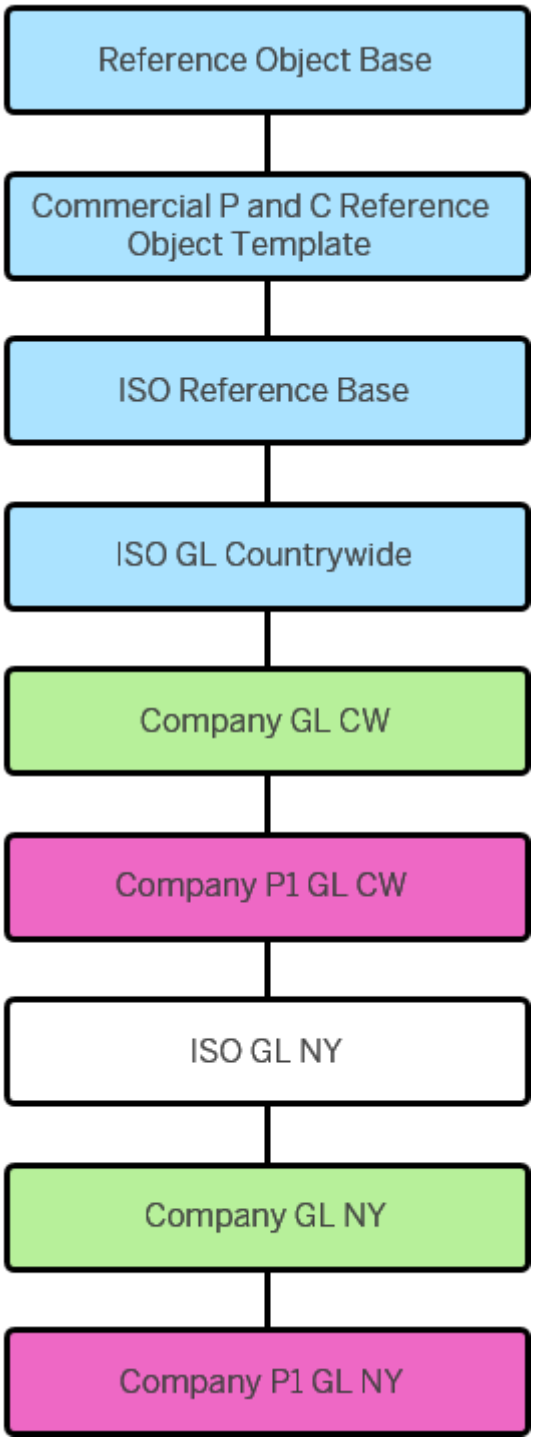
   For our example, use **`Company P1 GL NY`**.
4. Select Reference Object from the *Modifier Type* dropdown list.
5. Select the *Based On* icon.

   The *Object Picker* dialog displays.
6. Enter a name in the *Object Name* field that you want to base your deviation on.

   For our example, use **`Company GL NY`**.
7. Choose *Search*.
   The *Object Search Results* panel is populated.
8. Select Company GL NY from the *Object Search Result* panel.
   The *Object Attribute* panel is populated and the tree path is displayed at the bottom.
9. Choose *Select*.
   The *New Object* dialog is refreshed and the *Inheritance Path Substitution* field is populated with Company GL CW.
10. Select the *Change Substitution* icon to display the *Inheritance Path Substitution* dialog.
    The dialog is divided into two panels: *Substitution Nodes* on the left and *Inheritance Preview* on the right. The *Inheritance Preview* panel shows a preview of the inheritance based on the current selection in *Component Properties* dialog. The *Substitution Nodes* panel allows for inheritance substitution.
11. Select Company P1 GL CW from the *Substitution Nodes* panel to amend the inheritance path.
    The *Inheritance Preview* panel refreshes.
12. Choose *Select* to confirm and commit the change of inheritance path.
13. Choose *OK*.

Results

Our `Reference Object`, Company P1 GL NY, is added to the *Object List*. The System creates the product with the inheritance path substitution as illustrated below.

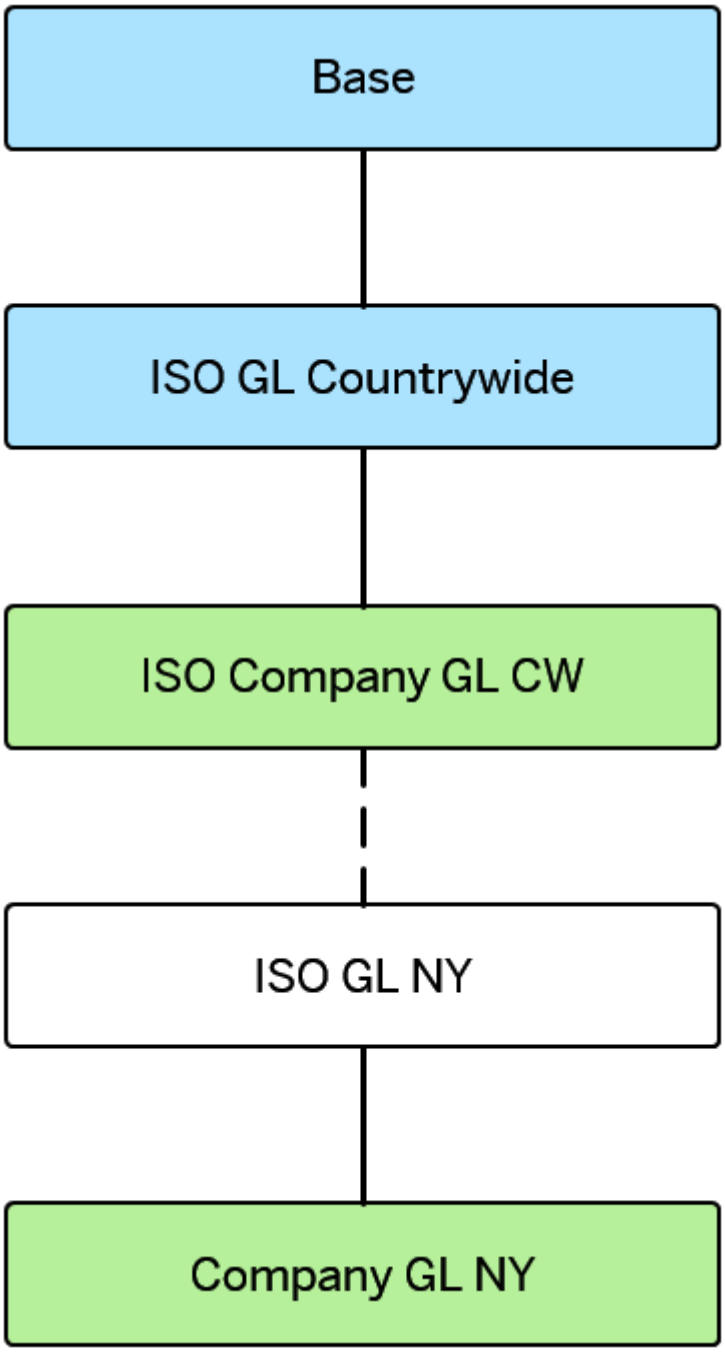

Reference Object Base
Commercial P and C Reference Object Template
ISO Reference Base
ISO GL Countrywide
Company GL CW
Company P1 GL CW
ISO GL NY
Company GL NY
Company P1 GL NY

23.6 Modifying a Reference Object with an Inheritance Path Substitution

A reference object with an inheritance path substitution can be modified by amending the Inheritance Path Substitution value in the *Properties* dialog.

Context

For this example, our company deviation is labeled "Company GL NY". It is based on "ISO GL NY", that is designated *Parent Can Be Substituted*. The parent node that can be substituted is ISO GL Countrywide. A tree view representation is shown below:



> ⚠ Caution
>
> You can't modify Inheritance Path Substitution if current product and inheritance have locked data value rows. An error message is displayed. You are unable to change the IPS value until all locked data value rows are committed/reverted.

For our example we want to modify the inheritance path from Company GL CW back to ISO GL Countrywide.

Procedure

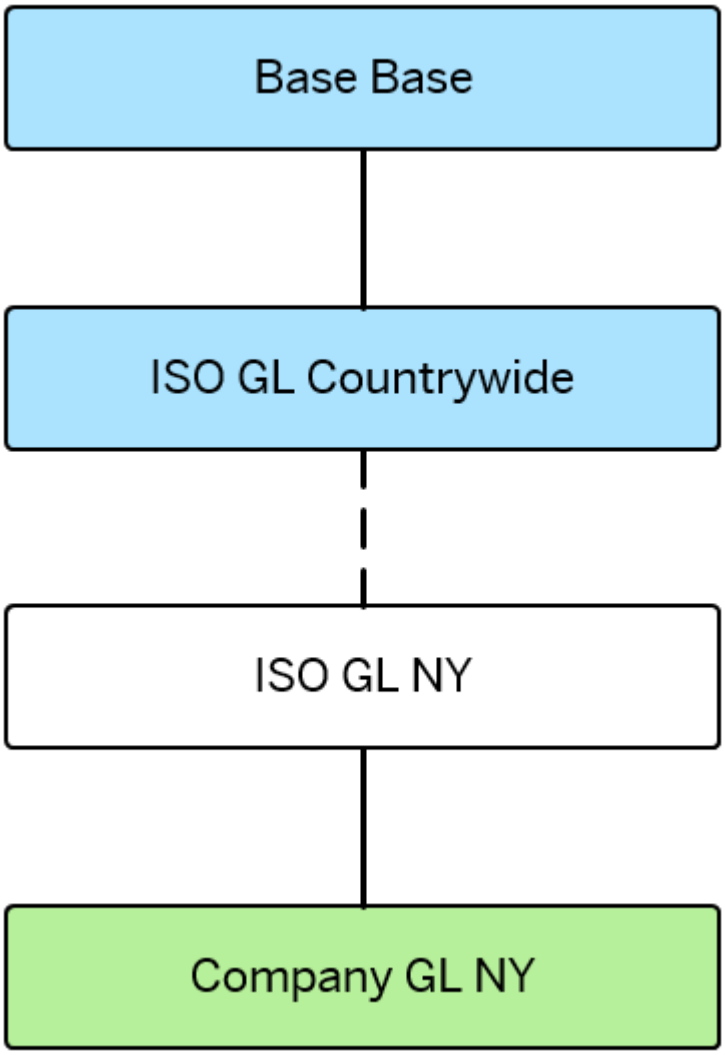
1. Navigate to the appropriate `Reference Object` folder in the *Studio Tree*.
2. Right-click the Company GL NY object in the *Object List* to display the pop up menu.
3. Choose *Properties* on the pop-up menu.
   The *Component Properties* dialog displays.
4. Select the  *Change Substitution* icon to display the *Inheritance Path Substitution* dialog.
5. Select ISO GL Countrywide from the *Substitution Nodes* panel to amend the inheritance path.
   The *Inheritance Preview* panel refreshes.
6. Choose *Select* to confirm and commit the change of inheritance path.

   > ⚠ Caution
   >
   > If the inheritance path can't be changed because the inheritance path is substituted in a child node; a *Warning* dialog displays indicating any nodes that must be modified first. Select *OK* to discontinue the inheritance substitution and make modifications to the listed nodes as required.

   > ℹ Note
   >
   > The system presents warning messages based on substitution rules when there is a conflict.

Results

A revised tree view representation is shown below:



Related Information

Creating a Reference Object Whose Parent can be Substituted [page 432]

23.7 Modifying the Based On Option in the *Properties* Dialog

When you modify the Base On option in the *Properties* dialog certain conditions apply.

You can perform change inheritance and inheritance path substitution actions together.

For example, when you change inheritance to a newer version of its parent, the current IPS value may not be valid anymore (the current IPS value doesn't exist on the new parent tree). In this case, a message is displayed asking you to select a new IPS value.

When you click on the IPS icon, selections are displayed with respect to the new parent. You are able to select a new IPS value.

When you click *OK* or *Apply* on the *Component Properties* dialog, both inheritance IPS value changes are applied together.

Related Information

Editing an Object's General Properties [page 42]
Changing Inheritance for a Product [page 36]

23.8 Modifying the *Parent Can Be Substituted* Option

When modifying the *Parent Can Be Substituted* option in the *Properties* dialog certain conditions apply.

Context

The following conditions apply:

- There can only be one node in the inheritance path that can be substituted.
- The *Parent Can Be Substituted* option can't be changed to Yes if it is already used in an inheritance path substitution. The option is hidden for children nodes.
- If any child of the node is used in substitution, the option can't be changed to Yes.

Procedure

1. Navigate to the appropriate `Reference Object` folder in the *Studio Tree*.
2. Right-click the Reference object in the *Object List* to display the pop up menu.
3. Select *Properties* from the pop-up menu.
   The *Component Properties* dialog displays.
4. Change the *Parent Can Be Substituted* radio button to Yes.
   A warning prompt displays if there is a conflict.

23.9 Deleting a Reference Object with a Parent that can be Substituted

You can delete a `Reference Object` with a parent that can be substituted.

Procedure

1. Navigate to the appropriate folder in the *Studio Tree*.
2. Right-click the object in the *Object List* tab and select *Delete*.

   A warning prompt displays if there is a conflict.

   > **i Note**
   >
   > The node can't be deleted if it's used directly or through substitution.

3. Choose Yes to confirm your choice when there are no conflicts.

23.10 Deleting a Reference Object that has Inheritance Path Substitution

You can delete a `Reference Object` with IPS.

Procedure

1. Navigate to the appropriate folder in the *Studio Tree*.
2. Right-click the object in the *Object List* tab and select *Delete*.

   A warning prompt displays if there is a conflict.

   > **i Note**
   >
   > A node can't be deleted if it's in any resolved inheritance path (including substitution).

3. Choose Yes to confirm your choice when there are no conflicts.

23.11 Publishing Products with an Inheritance Path Substitution

You are able to build and publish a product based on the resolved inheritance tree. With the introduction of Inheritance Path Substitution, logic for the Publish Family of Products Tree has been developed.

Related Information

Publishing [page 143]

23.12 Exporting and Importing Products with an Inheritance Path Substitution

You are able to export products with an inheritance path substitution. The system exports the resolved inheritance tree structure and data (not the full tree). The Export file can identify which node's parent can be substituted and which nodes are in the substituted path.

23.13 Copying Products with an Inheritance Path Substitution

You are able to copy products and family of products with an inheritance path substitution. The system will copy the resolved inheritance tree structure and data of a product. The inheritance path substitution is maintained in the copied product.

24 Product Web Services

Product Web Services are rules (created by the user) that are exposed as web services in a WSDL file.

Validation can be performed on the input and output of the web services by using XML Definition objects. Product Web Services can be tested using the *Product Web Services Manager* in the Administrative Console.

Related Information

Creating the XML Definition Objects [page 444]
Using the XML Schema Painter [page 445]
Importing the XML Schema Definition (XSD) File [page 449]
Assembling XML Definition Objects into Products [page 451]
Creating the Product Web Services Object [page 452]
Creating Product Web Services with XML Schema Manipulation [page 453]
Generating and Deploying the WSDL [page 454]
Generating and Downloading a Data Definition XSD [page 456]
Product Documentation for XML Definition Objects [page 456]
Audit Logging for XML Definition Objects [page 457]

24.1 Creating the XML Definition Objects

XML Definition objects are used to validate the input and output of a rule or web service. These objects must be created first so that they can be assigned to a rule's input and output.

Context

Notes:

- XML Definition objects can be versioned.
- The XML Definition object can't be copied.
- Updates to the *Name* of the XML Definition object from the *Component Properties* dialog also updates the *Schema Name* to match (while removing spaces). The new name is propagated to all rules using the XML Definition object, as follows:
  - You are prompted to confirm the update due to the impact on rules.
  - The *Input Argument* and *Return Type* XML Object Names are updated.

    - The Script Rule referencing the XML Definition object is updated.
- The XML definition object must be assembled in the product.

To create the XML definition objects, perform the following steps:

Procedure

1. In the Product Modeler *Studio Tree*, navigate outside the product to the standalone level.
2. Select the *New Object* icon .

   The *New Object* dialog appears.
3. Populate the object information by selecting *XML Definition* as the *Modifier Type*.
   The other fields are dynamically populated.
4. Finish populating the dialog and choose *OK* to finish.
   The object is created.

24.2 Using the XML *Schema Painter*

Context

The following functionality is provided on the toolbar within the XML *Schema Painter*:

Action Item	Description
Save	Saves all changes made to the schema. The schema is reloaded in the same view/structure as prior to saving.
Refresh Schema	Refreshes the schema. The standard flag icon should appear on the tab to indicate unsaved changes
Import XSD	Displays the *Import XSD* pop-up dialog.
View XSD	Displays the XML Schema Definition in a pop-up dialog, the *XML Definition Viewer*

XML Definition objects can only be modified at the standalone level.

To work with an XML Definition in the *Schema Painter*, perform the following steps:

Procedure

1. Select the XML Definition in the Product Studio (where the object was created), right-click it in the *Object List* tab and choose *Open*.

   The XML Definition opens in its own tab in the Product Studio.
2. Select the *Schema* tab to view the XML Definition in the *Schema Painter*.

   The *Schema Painter* appears.
3. The first attribute added must be a complex attribute. Right-click the blue row under *XML Tag Name* and select *Insert Child* > *Complex* from the menu.

   The element is added.

   > **i Note**
   >
   > The system supports only a single complex wrapper at the top level. All elements in the schema painter should only have a single complex wrapper at the top level.
4. The new element is named `ComplexElement1` by default. Change this to a meaningful name by either entering it in the name field (if the cursor is flashing there), or double-click the element name and enter the new element name.

   When naming an element, the following rules apply:
   - The name should be unique within the same complex type level
   - Spaces aren't allowed
   - The schema name appears as the root element by default
   - The XML Tag Name of the root element can't be edited
   - For new tag elements, the system pre-populates the field with the next sequential name
5. The *Min Occurs* and *Max Occurs* fields indicate the minimum or maximum number of occurrences of the element in the XML. Double-click these fields to enter values.

   The following rules apply for *Min Occurs*:
   - It must be defined as an integer
   - The default value is 1.
   - Message: `Invalid value for Min. Occurs. Must be an integer value.`

   The following rules apply for *Max Occurs*:
   - It must be defined as an integer greater than 0 or blank or unbounded
   - The default value is 1.
   - If this value is left blank, it implies that it is unbounded
   - Message: `Invalid value for Max. Occurs. Must be an integer value greater than zero, blank, or unbounded.`
6. Continue to add rows by right clicking and choosing *Add Row* (for the same level in the XML code) or *Add Child* (for a lower level). Choose *Complex* or *Simple*, as required.
7. For simple rows, select an *XML Data Type* from the dropdown list as follows:
   - Boolean

- Byte
- date
- DateTime
- double
- Float
- Integer
- Long
- Time
- Short
- String

Additional points:

- The *XML Data Type* for the root element is always schema
- The *XML Data Type* of the root element can't be edited
- For new tag elements, the system will pre-populate with string

8. Elements can be moved up or down within the same level in the schema by right-clicking and choosing *Move Up* or *Move Down*, as appropriate.
9. Elements can be copied, cut and pasted within the schema by right-clicking and selecting *Cut* or *Copy* as appropriate, then clicking on the desired point, right-clicking, and selecting *Paste*.
10. Delete elements in the schema by right-clicking and selecting *Delete*.
11. The above processes are summarized in the following table:

Action Name	Description
`Add Row`	Can't be inserted at the root ("schema") level
`Add Row - Complex`	Inserts an element of "complex" data type after the highlighted row at the same level
`Add Row - Simple`	Inserts a simple element (default "string" data type) after the highlighted row at the same level
`Copy`	Simple Element: • Stores a copy of the highlighted element in memory. Complex Element: • Stores a copy of the highlighted element and all child elements in memory. • Can't copy root ("schema") element

Action Name	Description
`Cut`	Simple Element: • Removes the highlighted element and stores it in memory. Complex Element: • Removes the highlighted element and all child elements and stores them in memory. • Can't cut root ("schema") element
`Delete`	Simple Element: • Deletes the highlighted element. Complex Element: • Deletes the highlighted element and all child elements. • Can't delete root ("schema") element
`Insert Child`	Can't be inserted if the highlighted row is a Simple Element
`Insert Child - Complex`	Inserts a child element of "complex" data type after the highlighted row
`Insert Child - Simple`	Inserts a simple element (default "string" data type) after the highlighted row
`Move Down`	Simple Element: • Moves the highlighted element down one, under the next element/element group of the same level • Can't move the last child element Complex Element: • Moves the highlighted element and all child elements down one, under the next element/element group of the same level • Can't move root ("schema") element • Can't move the last child element

Action Name	Description
`Move Up`	Simple Element: • Moves the highlighted element up one, above the next element/element group of the same level • Can't move an element above the root ("schema") element • Can't move the first child element above its parent element Complex Element: • Moves the highlighted element and all child elements up one, above the next element/element group of the same level • Can't move root ("schema") element • Can't move an element group above the root ("schema") element • Can't move the first child element above its parent element
`Paste`	Simple Element: • Pastes the stored element below the highlighted Simple element • Pastes the stored element below the highlighted Complex element (first child element) • If the element is a copy, append "CopyOf" to the element name Complex Element: • Pastes the stored elements below the highlighted Simple element • Pastes the stored elements below the highlighted Complex element (first child element) • If the element is a copy, append "CopyOf" to the element name

24.3 Importing the XML Schema Definition (XSD) File

Prerequisites

This functionality is only available at the standalone component level, not at the product level.

The XSD file must be valid. If the imported XSD file isn't valid, an error message appears and the import fails, and none of the elements are imported.

Context

Any XML Schema Definitions imported into the *Schema Painter* must have only a single complex element at the top level. If you attempt to import an XML Schema Definition with more than one complex element at the top level, only the first complex element will import. The second complex element will be ignored by the system.

> **i Note**
>
> If there are existing elements already defined within the schema, all will be deleted during import and replaced with the imported ones.

> **→ Tip**
>
> If you are importing an XSD file that had been previously exported and edited, be sure to remove the `<xs:sequence>` tag from Line 2 of the file and its end tag.

To import the XML schema definition file into the *Schema Painter*, perform the following steps:

Procedure

1. Select the *Import XSD* icon in the toolbar in the *Schema Painter*.

   The *Import Data* dialog displays.
2. Browse and select the file to upload, and choose *Upload*.

   The system validates the imported XSD file for proper formatting.

Results

The XML Schema Definitions are imported.

> **i Note**
>
> If `minOccurs` or `maxOccurs` attributes aren't provided, the default value of 1 is applied.

24.4 Assembling XML Definition Objects into Products

You can assemble a given XML Definition object into multiple product structures.

Context

The following guidelines apply:

- You can't modify XML Definition objects from within a product. Changes can only be made in the standalone object.
- Standard inheritance rules apply:
  - Modifications made to the standalone XML Definition object are applied to the product.
  - You can delete the standalone XML Definition object, if its assembled into a product.
  - Product associations are also deleted, following the current application's delete rules.
  - A list of impacted products is displayed.
- You can create versions of products with XML Definition objects assembled.
- You can copy and paste products with XML Definition objects assembled.
- You can import and export products with XML Definition objects assembled.

> ⚠ Caution
>
> Currently, there is no validation to enforce XML Definition object assembly into the product if its used by rules. Therefore, errors won't be caught until Runtime.

To assemble XML Definition objects into products, perform the following steps:

Procedure

1. In the Product Modeler, navigate to the product and create (if necessary) a folder for the XML Definitions.
2. Go to the XML Definitions folder, right-click the folder and choose *Add*.
3. From the *Object Type* dropdown list, select *XML Definition* and choose *Search*.
4. From the *Object Search Results*, select the desired XML definition and choose *Select*.

24.5 Creating the Product Web Services Object

Context

To create the Product Web Services Object, perform the following steps:

Procedure

1. Navigate to the *Components* folder outside the product at the standalone level.
2. Select the *New Object* icon .

   The *New Object* dialog appears.
3. Populate the object information by selecting *Product Web Services* as the *Modifier Type*.
   The other fields are dynamically populated.
4. Finish populating the dialog.
5. Choose *OK*.
6. Double-click your Product Web Services object, and it will open in a new tab in the Product Modeler.
7. Go to the *Product Web Services* tab.
8. Select the *Attributes* tab.

   The following default attributes should display:

   - `ServiceName` – Text, 50 characters
   - `Rule` – Rule, 16 characters
   - `SYSATTR_TABLE_NAME` – Text, 250 characters

Next Steps

The Product Web Services object must be assembled in the marketable product in order to use web services.

24.6 Creating Product Web Services with XML Schema Manipulation

Context

XML schema manipulation is supported in Product Web Services. When creating a rule, note the following:

- In the *Values* tab, each row created is an individual product web service.
- Only the selected product web services will be available in the WSDL of the published and deployed product.
- Product web service rules can only contain one script step. Anything else isn't supported.

  > ⚠ Caution
  >
  > The system allows the use of multiple script steps.

- Standard input and return type arguments are available for product web services, but only XML is supported for schema validation.

To create the web service, perform the following steps:

Procedure

1. Select the Product Web Services object in the *Product Tree*, and select the *Values* tab.
2. Each web service is represented by a row in the *Values* tab. Place the cursor in the left-most column, right-click and choose *Add*.
3. Double-click the *ServiceName* field in the new row and enter the web service name.
4. Select the *Rule* field in the new row and select *Create* to build the rule for the web service.
5. Specify the field entries as follows:
   - Specify a *Rule Name*
   - *Step Type* = *Script*
   - *Rule Return Type* = *XML*

   Choose *OK*. The *Rule Painter* screen opens.
6. Enter the plus symbol to specify the *Input Arguments*. Enter a *Name* and specify a *Type of XML* from the dropdown list.
7. When specifying an XML object name in the *Input Arguments*, choose the *Object Picker* icon  .

   The *Object Picker* dialog displays.
8. Choose *Search* to display a list of XML definitions. Select the desired object and choose *Select*. The XML object populates the *Input Arguments* line. Select the save icon when done.

9. Enter the script commands below the *Input Arguments* by choosing *Click to Edit*.
10. The product web service rule must loop through the request and the response in order for the schema elements to appear in the WSDL. Enter the script commands and save your changes when finished.
11. Under *Return Type*, select the *XML Object Name*. Save your changes when done.
12. The product web service rule must loop through the request and the response in order for the schema elements to appear in the WSDL.

24.7 Generating and Deploying the WSDL

Product Web Services are deployable as true web services. The system generates the WSDL.

Context

> **i Note**
>
> The first two steps of this task must be performed prior to generating and deploying the WSDL. Refer to the Installation Guide for details.

To generate and deploy the WSDL, perform the following steps:

Procedure

1. Pre/Post Publishing must be enabled from the product. .
   a. Open your web browser 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. Navigate to *Admin Console* > *System* > *Edit Configuration Settings* > *PC* > *Env* > *PC Environment*
   c. change `PrePostPublishingEnabled` to **`Yes`**
   d. Save your changes.
   e. Select the Reload Config icon .
2. The repository and Runtime environments must be configured in the Design Time Administrative Console:
   a. Navigate to *Edit Configuration Settings* > *PC* > *Env* > *PWS Manager Setting*
   b. Set the applicable parameters as follows:
      - `ProductRepositoryUrl` – location of the PWS Manager - Repository `http://<pro_designtime_app_host>:<pro_designtime_app_port>/csiroot/psrepository`
      - `ProductWSRuntimeUrl` – location of the PWS Manager – Runtime `http://<pro_designtime_app_host>:<pro_designtime_app_port>/csiroot/psruntime`

c. Save your changes.

d. Select the Reload Config icon .

3. The `Custom Publishing` object provides two out-of-the-box post-publishing services which are run after publishing the JAR. Two new Post-Publish services are available to upload the product, generate and deploy the WSDL. This service uploads the published product JAR to the Repository of the *Product Web Services Manager*.

> **i Note**
>
> These Post-Publish services should not be selected at the same time. If they are selected together, the product JAR will be uploaded twice in the Repository and assigned two different build numbers.

Attribute Name	Attribute Value
`Service Name`	Upload to Repository
`Service Class Name`	com.camilion.pspublish.UploadToRepositoryService
`Required`	N
`Pre or Post`	Post
`Execution Sequence`	1
`Params`	
`Continue on Failure`	Y

This service uploads the published product JAR to the Repository and then deploys the published product JAR to the Runtime environment of the *Product Web Services Manager*. The WSDL is generated with the deployment.

Attribute Name	Attribute Value
`Service Name`	Upload and Deploy to Runtime
`Service Class Name`	com.camilion.pspublish.DeployToRuntimeService
`Required`	N
`Pre or Post`	Post
`Execution Sequence`	1
`Params`	
`Continue on Failure`	Y

These services will now be available for selection when publishing.

24.8 Generating and Downloading a Data Definition XSD

You can generate and download a product's data definition XSD.

Prerequisites

Before performing this task, all Product data definitions must be published. Reference products containing data definitions must be built and published at the standalone level. If this isn't done, an error message appears.

Context

All assembled data definitions, including those referenced in the reference product should be in the generated in a single XSD file. This logic is equivalent to the information displayed in *Column Picker* in the questionnaire model for the product.

To generate and download a data definition XSD, perform the following steps:

Procedure

1. Select *File* > *Generate & Download XSD*, transforming the Product data definition (and any referenced data definitions in the `Config` object) into an XSD file.

   A dialog opens indicating the XSD has generated successfully.
2. Choose *Download XSD*.

   A dialog opens for you to specify a location to save the downloaded XSD file on your local computer.

24.9 Product Documentation for XML Definition Objects

You can select all product content defined to support Web Services when creating a view or generating a Product Specification Document.

The following applies to product documentation for XML Definition objects:

- You can select XML Definition objects in the Product Specification Documents.
- If an XML Definition object is selected, the XSD is included in the Product Specification Document, formatted the same as an *XML Definition Viewer* dialog.
- You can include Product Web Services rules (scripting rules) in the Product Specification Document by selecting *Create Product Specification Document* > *Values Tab* > *Include Values Tab Information*

Include Rules when creating the Product Specification Document. Product Web Services rules are included in the Product Specification Document when this option is selected.

Related Information

Generating Product Reports [page 66]

24.10 Audit Logging for XML Definition Objects

You can view all creation, update, and deletion activities to product content defined to support Web Services using the Product Modeler Audit Log.

The audit log for XML Definition object activity is in the Product Modeler in *Tools* *Audit Trail*. Activities are recorded as follows in the Audit Log.

Related Information

Viewing Object Histories and Audit Trails [page 61]

25 User-Defined Product Services

A user-defined product service enables you to encapsulate business logic in a rule that becomes a product service, callable from within an application. All user-defined services for a product are contained in a special purpose component named `Service API`. From within this component you create the rule and access the *Rule Painter*, where you define the rule in the standard way. A service rule has the option of accessing a data definition within the product.

After the rule is published, Product Services makes it available to direct calls from the application, or using web services, calls from other platforms.

In products, user-defined product services are commonly used for implementing forms attachments, referral rules, and rating rules.

Related Information

Viewing the User-Defined Product Services [page 458]
Overriding the User-Defined Product Services [page 459]
Renaming User-Defined Product Services [page 459]
Editing User-Defined Product Services [page 460]
Deleting User-Defined Product Services [page 461]
Adding Child Data Value Rows to User-Defined Product Services [page 461]
Making User-Defined Product Services Available [page 462]
Calling User-Defined Product Services from an Application [page 462]

25.1 Viewing the User-Defined Product Services

Procedure

1. Open the product .
2. Go to *Configuration* > *Product Services* in the *Product Tree*.
3. Select the `Service API` component .
4. Select the *Values* tab.
5. Locate the row of the product service.

6. Right-click the row header and select *Usage Report*.

Results

The *Usage Information* dialog appears.

25.2 Overriding the User-Defined Product Services

You can override the row of a user-defined product service to change the content of the API's rule, or the selection of the API's row.

Procedure

1. Open the product .
2. Go to *Configuration* > *Product Services* in the *Product Tree*.
3. Select the `Service API` component .
4. Select the *Values* tab.
5. Locate the row of the product service.
6. Right-click the row header and select *Override*.
7. Choose one of the following values:
   - Override association only
   - Override association and value
8. Choose *Select*.
9. Save your changes.

25.3 Renaming User-Defined Product Services

Procedure

1. Open the product .

2. Go to *Configuration* > *Product Services* in the *Product Tree*.
3. Select the `Service API` component .
4. Select the *Values* tab.
5. Locate the row of the product service.
6. Select the *Service Name* cell and make the edit.

25.4 Editing User-Defined Product Services

Prerequisites

Service rules of user-defined product services can be edited only if a data value row is local or is overridden in the current product.

Procedure

1. Open the product .
2. Go to *Configuration* > *Product Services* in the *Product Tree*.
3. Select the `Service API` component .
4. Select the *Values* tab.
5. Locate the row of the product service.
6. Left-click the icon in the *Service Rule* column and select *Edit*.
   The *Rule Painter* dialog appears.
7. Edit your rule.
8. Build the rule.
9. Close the *Rule Painter*.
10. Save your changes.

25.5 Deleting User-Defined Product Services

Prerequisites

Service rules of user-defined product services can be deleted only if a data value row is local or is overridden in the current product.

Procedure

1. Open the product .
2. Go to *Configuration* > *Product Services* in the *Product Tree*.
3. Select the `Service API` component .
4. Select the *Values* tab.
5. Locate the row of the product service.
6. Left-click the icon in the *Service Rule* column and select *Delete*.

   A confirmation message appears.
7. Choose *Yes* to confirm your decision.
8. If you also want to remove the rule's connection to the application, choose *Yes*.
9. Save your changes.

25.6 Adding Child Data Value Rows to User-Defined Product Services

You add child data value rows to a product service data value row in the usual way. However, all columns and `Service API` arguments must be the same in the child as in the parent, with the exception of the rule, the `EFFECTIVE_DATE` and the `EXPIRY_DATE`.

Related Information

Working with Subobject Data Value Rows [page 128]

25.7 Making User-Defined Product Services Available

To make a product service available from Product Services, you must select the data value row and publish the product that contains the product service.

Related Information

Publishing a Product [page 147]

25.8 Calling User-Defined Product Services from an Application

To invoke a user-defined product service from a rule within an application, use the following stem: `ProductService.`

Combine this stem with any of the user-defined product services methods.

25.9 Methods for User-Defined Product Services

25.9.1 runProductService()

Use this function to call a product service rule.

```
runProductService(<ServiceName>,<RuleAttributes>):double
```

Parameters

ServiceName

The name of the user-defined service, as it appears in the `Service API` component's `Service Name` attribute.

RuleAttributes

The names of any attributes used with the input. stem in the product service rule in FS-PRO.

This parameter takes the names of the attributes used with the `input.` stem in the rule. If there are multiple attributes, separate them with commas. For example: `State=CA,Limit=1000`

Returns

Double, otherwise an exception is generated

25.9.2 runProductServiceWithName()

Use this function to call a product service rule that returns.

```
runProductServiceWithName(<ProductName>,<ProductVersion>,
<ContextDate>,<ServiceName>,<RuleAttributes>):double
```

Parameters

ProductName

The name of the product.

ProductVersion

The product version number.

ContextDate

The context date of the product service API.

ServiceName

The name of the user-defined service, as it appears in the `Service API` component's `Service Name` attribute.

RuleAttributes

The names of any attributes used with the `input.` stem in the product service rule in FS-PRO.

This parameter takes the names of the attributes used with the `input.` stem in the rule. If there are multiple attributes, separate them with commas. For example: `State=CA,Limit=1000`

Returns

Double, otherwise an exception is generated

25.9.3 runNumberProductService()

Use this function to call a product service rule that returns a number.

```
runNumberProductService(<ServiceName>,<RuleAttributes>):double
```

Parameters

ServiceName

The name of the user-defined service, as it appears in the `Service API` component's `Service Name` attribute.

RuleAttributes

The names of any attributes used with the `input.` stem in the product service rule in FS-PRO.

This parameter takes the names of the attributes used with the `input.` stem in the rule. If there are multiple attributes, separate them with commas. For example: `State=CA,Limit=1000`

Returns

Double, otherwise an exception is generated

25.9.4 runNumberProductServiceWithName()

Use this function to call a product service rule.

```
runNumberProductServiceWithName(<ProductName>,<ProductVersion>,
<ContextDate>,<ServiceName>,<RuleAttributes>):double
```

Parameters

ProductName

The name of the product.

ProductVersion

The product version number.

ContextDate

The context date of the product service API.

ServiceName

The name of the user-defined service, as it appears in the `Service API` component's `Service Name` attribute.

RuleAttributes

The names of any attributes used with the `input.` stem in the product service rule in FS-PRO.

This parameter takes the names of the attributes used with the `input.` stem in the rule. If there are multiple attributes, separate them with commas. For example: `State=CA,Limit=1000`

Returns

Double, otherwise an exception is generated

25.9.5 runTextProductService()

Use this function to call a product service rule that returns text.

```
runTextProductService(<ServiceName>,<RuleAttributes>):CsiString
```

Parameters

ServiceName

The name of the user-defined service, as it appears in the `Service API` component's `Service Name` attribute.

RuleAttributes

The names of any attributes used with the `input.` stem in the product service rule in FS-PRO.

This parameter takes the names of the attributes used with the `input.` stem in the rule. If there are multiple attributes, separate them with commas. For example: `State=CA,Limit=1000`

Returns

CsiString, otherwise an exception is generated.

25.9.6 runTextProductServiceWithName()

Use this function to call a product service rule that returns a string.

```
runTextProductServiceWithName(<ProductName>,<ProductVersion>,
<ContextDate>,<ServiceName>,<RuleAttributes>):CsiString
```

Parameters

ProductName

The name of the product.

ProductVersion

The product version number.

ContextDate

The context date of the product service API.

ServiceName

The name of the user-defined service, as it appears in the `Service API` component's `Service Name` attribute.

RuleAttributes

The names of any attributes used with the `input.` stem in the product service rule in FS-PRO.

This parameter takes the names of the attributes used with the `input.` stem in the rule. If there are multiple attributes, separate them with commas. For example: `State=CA,Limit=1000`

Returns

CsiString, otherwise an exception is generated

25.9.7 runDataTableProductService()

Use this function to call a product service rule that returns a data table.

```
runDataTableProductService(<ServiceName>,<RuleAttributes>):CsiDataTable
```

Parameters

ServiceName

The name of the user-defined service, as it appears in the `Service API` component's `Service Name` attribute.

RuleAttributes

The names of any attributes used with the `input.` stem in the product service rule in FS-PRO.

This parameter takes the names of the attributes used with the `input.` stem in the rule. If there are multiple attributes, separate them with commas. For example: `State=CA,Limit=1000`

Returns

CsiDataTable, otherwise an exception is generated.

25.9.8 runDataTableProductServiceWithName()

Use this function to call product service rule that returns a data table.

```
runDataTableProductServiceWithName(<ProductName>,<ProductVersion>,
<ContextDate>,<ServiceName>,<RuleAttributes>):CSIDataTable
```

Parameters

ProductName

The name of the product.

ProductVersion

The product version number.

ContextDate

The context date of the product service API.

ServiceName

The name of the user-defined service, as it appears in the `Service API` component's `Service Name` attribute.

RuleAttributes

The names of any attributes used with the `input.` stem in the product service rule in FS-PRO.

This parameter takes the names of the attributes used with the `input.` stem in the rule. If there are multiple attributes, separate them with commas. For example: `State=CA,Limit=1000`

Returns

CSIDataTable, otherwise an exception is generated

26 *Rule Painter* Functions

This section contains a complete listing of the functions used in the *Rule Painter*. The functions are divided into several categories. You access the complete list of functions in a category by typing its prefix in the expression field in the *Rule Painter*.

> → Tip
>
> Clicking beside the expression field causes a list of the function prefixes to appear.

Related Information

Date Functions [page 469]
Flow Functions [page 504]
Number Functions [page 509]
String Functions [page 540]
System Functions [page 573]
UAS Functions [page 586]

26.1 Date Functions

To access a complete list of date functions, enter **`date.`** or **`datefn.`** in the expression field.

Related Information

compareDatestring() [page 471]
convertToJdbcTimeStamp() [page 471]
convertToJdbctimestamp() [page 472]
createDate() [page 473]
date() [page 473]
dateToText() [page 474]
dayNameConverter() [page 474]
daysInFebruary() [page 475]
formatDate() [page 476]
formatDateFromTo() [page 476]

isValidDay() [page 503]
isYear() [page 504]

26.1.1 compareDatestring()

Use this function to compare two date strings and converts both strings to timestamps.

Syntax

```
compareDatestring(string1,string2)
```

Parameters

string1
: Date 1 or Timestamp object 1.

string2
: Date 2 or Timestamp object 2.

Response

Returns the value 0 if `string1` and `string2` are equal, a value less than 0 if `string1` is before `string2`, and a value greater than 0 if `string1` is later than `string2`.

26.1.2 convertToJdbcTimeStamp()

Given an SQL timestamp, returns a string that represents the SQL version of the timestamp.

Syntax

```
convertToJdbcTimeStamp(java.sql.Timestamp)
```

Parameters

java.sql.Timestamp
: The date to convert to a string.

Response

Returns a string that represents the SQL version of the timestamp.

26.1.3 convertToJdbctimestamp()

Use this function to convert a timestamp to a database date representation.

Syntax

```
convertToJdbctimestamp(timestamp)
```

Parameters

timestamp
: A timestamp, such as March 23, 2021.

Response

Returns a database representation of the date, such as: `{ts '2021-03-23 00:00:00'}`

26.1.4 createDate()

Use this function to return a valid date/time object.

Syntax

```
createDate(string)
```

Parameters

string
: Parameters are year, month, day.

Response

Returns `java.sql.Timestamp`

26.1.5 date()

Use this function to convert date passed as a string to the date object.

Syntax

```
date(string1,string2)
```

Parameters

string1
: string date to be converted to Date.

string2
: One of the three formats: DMY-UK, MDY-US, or YMD-JDBC.

Response

Returns the date object that represents the input string argument.

26.1.6 dateToText()

Use this function to return a fixed string representation for a specified timestamp.

Syntax

```
dateToText(timestamp)
```

Parameters

timestamp
: The timestamp, such as Dec 31, 2020.

Response

Returns the fixed string representation for the timestamp, such as `Thurs Dec 31 12:00:00 EST 2020`.

26.1.7 dayNameConverter()

Use this function to convert a number to a day of the week.

Syntax

```
dayNameConverter(double)
```

Parameters

double
: A number representing a day of the week.

Response

Returns a day of the week.

26.1.8 daysInFebruary()

Use this function to determine the number of days in February for a specified year.

Syntax

```
daysInFebruary(integer)
```

Parameters

integer
: The format of the year should be a 4 digit number. For example: 2021

Response

Returns the number of days in February for the specified year.

26.1.9 formatDate()

Use this function to format a date using a specified format string.

Syntax

```
formatDate(string1,string2)
```

Parameters

string1

The date in its native format.

Usually similar to DD/MM/YY, MM-DD-YY, or YYYYMMDD.

string2

This is a number indicating how you would like the date string formatted.

Response

Returns the formatted date.

26.1.10 formatDateFromTo()

Use this function to convert the specified from-date string based on the fromFormat, to an alternate date string based on the toFormat.

Syntax

```
formatDateFromTo(string1,string2,string3)
```

Parameters

string1

The dateTime string to convert.

string2

The string for the fromFormat: the original string format of the specified date.

string3

The string for the toFormat: the format that you would like to convert the date string to.

Response

Returns a date string.

26.1.11 formatDateMediumShort()

Use this function to format a string representation of a date to a new string representation of a date.

Syntax

```
formatDateMediumShort(string1,string2)
```

Parameters

string1

Input date format: YYYY-MM-DD HH:MM:SS 00.

string2

Output date format. The form of return value is dictated by the requested format type, which is a string.

Permitted format types:

MDY	Month-Day-Year
DMY	Day-Month-Year
MDY_AMPM	Month-Day-Year hour:minute:second
DMY_AMPM	Day-Month-Year hour:minute:second

Response

Returns the formatted date. For example: Input 2021-03-03 00:00:00 00, Format Request: DMY, Output: 03-03-2021.

26.1.12 getAge()

Given a birth date as a timestamp, returns the age of the person relative to today's date.

Syntax

```
getAge(timestamp)
```

Parameters

timestamp
: A date value that is the date of birth for a person.

Response

Returns the age of the person, relative to today's date

26.1.13 getBusinessDays()

Use this function to determine the number of business days between the two input days (including the first day).

Syntax

```
getBusinessDays(timestamp1,timestamp2)
```

Parameters

timestamp1
: A date value that is the start date of the interval being measured.

timestamp2
: A date value that is the end date of the interval.

Response

Returns the total number of business days.

26.1.14 getCurrentDate()

Use this function to retrieve the current date.

Syntax

```
getCurrentDate()
```

Parameters

None

Response

Returns today's date in mm/dd/yyyy format.

26.1.15 getCurrentday()

Use this function to retrieve the current system date.

Syntax

```
getCurrentday()
```

Parameters

None

Response

Returns the current system date.

26.1.16 getDay()

Use this function to obtain the day of the month in a date value.

Syntax

```
getDay(timestamp)
```

Parameters

timestamp

A date value from which you want the day.

Response

Returns an integer (1 to 31) representing the day of the month in the date.

26.1.17 getDayName()

Use this function to determine the day of the week in a date value.

Syntax

```
getDayName(timestamp)
```

Parameters

timestamp
: A date value for which you want the name of the day.

Response

Returns the name of the weekday.

26.1.18 getDayNumber()

Use this function to determine the day of the week of a date value.

Syntax

```
getDayNumber(timestamp)
```

Parameters

timestamp
: The date value from which you want the number of the day of the week.

Response

Returns an integer (1-7) representing the day of the week of the date. Sunday is day 1, Monday is day 2, and so on.

26.1.19 getDaysAfter()

Use this function to determine the number of days one date occurs after another.

Syntax

```
getDaysAfter(timestamp1,timestamp2)
```

Parameters

timestamp1
: A date value that is the start date of the interval being measured.

timestamp2
: A date value that is the end date of the interval.

Response

Returns a value representing the number of days `timestamp2` occurs after `timestamp1`.

26.1.20 getDaysBetween()

Use this function to return the number of days between two dates.

Syntax

```
getDaysBetween(timestamp1,timestamp2)
```

Parameters

timestamp1
: The "from" date.

timestamp2
: The "to" date.

Response

Returns the number of days between the "from" date and the "to" date.

26.1.21 getDifferentInDays()

Use this function to return the difference in days between date1 and date2. Returns a negative number of days if date1 occurs after date2.

Syntax

```
getDifferentInDays(java.util.Date1,java.util.Date2)
```

Parameters

java.util.Date1
: The "from" date.

java.util.Date2
: The "to" date.

Response

Returns the difference in days between two dates, or a negative number of days if the "from" date occurs after the "to" date.

26.1.22 getHour(timestamp)

Use this function to obtain the hour in a time value. The hour is based on a 24-hour clock.

Syntax

```
getHour(timestamp)
```

Parameters

timestamp
: The time from which you want to obtain the hour.

Response

Returns an integer (00 to 23) whose value is the hour portion of the time.

26.1.23 getJDBCCurrentDateTime

Use this function to return the current date in JDBC format.

Syntax

```
getJDBCCurrentDateTime
```

Parameters

None

Response

Returns the format: `{ts '2021-03-14 10:08:32'} {ts 'yyyy-mm-dd hh:mm:ss'}`.

26.1.24 getLastDayOff()

Use this function to retrieve the last day off.

Syntax

```
getLastDayOff(timestamp,double)
```

Parameters

timestamp
: The value of the first date.

double
: The value of the vacation days.

Response

Returns the value of the last day off.

26.1.25 getLastDayOfMonth()

Use this function to return the last date of the month.

Syntax

```
getLastDayOfMonth(timestamp)
```

Parameters

timestamp
: A date value.

Response

Returns a date object representing the last day of the month in a given date.

26.1.26 getLastDeadLineDay()

Use this function to return the value of the last deadline day.

Syntax

```
getLastDeadLineDay(timestamp,double)
```

Parameters

timestamp
: A date value.

double
: The value of the offset.

Response

Returns the date value of the last deadline day.

26.1.27 getLastWorkingDayOfMonth()

Use this function to return the last working day of the month.

Syntax

```
getLastWorkingDayOfMonth(timestamp)
```

Parameters

timestamp
: A date value.

Response

Returns the last working day of the month.

26.1.28 getMaximumDaysInMonth()

Use this function to return the maximum number of days in the specified month.

Syntax

```
getMaximumDaysInMonth(double1,double2)
```

Parameters

double1
: The month.

double2
: The year.

Response

Returns the maximum number of days in the specified month.

26.1.29 getMiddleOfMonth()

Use this function to return the value of the middle of the month.

Syntax

```
getMiddleOfMonth(timestamp)
```

Parameters

timestamp
: The date value.

Response

Returns the `timestamp` value of the date of the middle of the month.

26.1.30 getMinute()

Use this function to return the number of minutes past the hour represented by the date. The value returned is between 0 and 59.

Syntax

```
getMinute(timestamp)
```

Parameters

timestamp
: The time value from which you want the minutes.

Response

Returns the number of minutes past the hour represented by this date.

26.1.31 getMonth()

Use this function to determine the month of a date value.

Syntax

```
getMonth(timestamp)
```

Parameters

timestamp
: The date from which you want the month.

Response

Returns an integer (1 to 12) whose value is the month portion of the date.

26.1.32 getNextBusinessDay()

Use this function to return the value of the next business day.

Syntax

```
getNextBusinessDay(string)
```

Parameters

string
: The current date.

Response

Returns the value of the next business day.

26.1.33 getNextFriday()

Use this function to retrieve the string value of next Friday.

Syntax

```
getNextFriday(string)
```

Parameters

string
: The current date.

Response

Returns the value of next Friday.

26.1.34 getNextHoliday()

Use this function to return a date based on a set of parameters.

Syntax

```
getNextHoliday(double1,double2,string,double3)
```

Parameters

double1
: The year number.

double2
: The month number.

string
: The day of the week (Monday-Sunday) as a string.

double3
: A rank as a number (between 1-5).

Response

Returns the date that satisfies all parameters.

For example, `getNextHoliday(2021,2,"Monday",1)` searches for a date matching the first Monday in the month of February for the year 2021. The result is February 5, 2021.

26.1.35 getNextMonthlyPayDate()

Use this function to retrieve the value of the next monthly pay date.

Syntax

```
getNextMonthlyPayDate(timestamp)
```

Parameters

timestamp
: The value of the current date.

Response

Returns a timestamp value of the next monthly pay date.

26.1.36 getNextSemiMonthlyPayDate()

Use this function to return the value of the next semi-monthly pay date.

Syntax

```
getNextSemiMonthlyPayDate(timestamp)
```

Parameters

timestamp
: The value of the current date.

Response

Returns the timestamp value of the next semi-monthly pay date.

26.1.37 getNow()

Use this function to retrieve the current date and time.

Syntax

```
getNow()
```

Parameters

None

Response

Retrieves the current date and time in Greenwich time.

26.1.38 getRelativeDate()

Use this function to obtain the date that occurs a specified number of days after or before another date.

Syntax

```
getRelativeDate(timestamp,double)
```

Parameters

timestamp

A value of type date.

double

An integer indicating the number of days.

Response

Returns the date that occurs <n> days after the date if <n> is greater than 0.

26.1.39 getRelativeDateInText()

Use this function to return a date in a specified format that is incremented by a specified number of days from a specified date.

Syntax

```
getRelativeDateInText(string1,string2,double)
```

Parameters

string1

The date ('mm-dd-yyyy').

string2

One of the following output formats:

DMY-UK	DD-MM-YYYY
MDY-US	MM-DD-YYYY
YMD-JDBC	YYYY-MM-DD

double

The number of days to increment the date.

Response

Returns the output date. For example, `getRelativeDateInText("4-2-2021","DMY-UK", 2 )` returns the output: `04-02-2021`

26.1.40 getRelativeMonth()

Use this function to return the timestamp of the specified date with the specified months added to it.

Syntax

```
getRelativeMonth(timestamp,double)
```

Parameters

timestamp
: The date.

double
: The number of months to increment the date by.

Response

Returns the `timestamp` of the date with the specified number of months added to it.

26.1.41 getRelativeTime()

Use this function to obtain a time that occurs a specified number of seconds after or before another time within a 24-hour period.

Syntax

```
getRelativeTime(timestamp,double)
```

Parameters

timestamp
: A value of type time.

double

A long number of seconds.

Response

Returns the time that occurs <n> seconds after `timestamp` if <n> is greater than 0. Returns the time that occurs n seconds before `timestamp` if <n> is less than 0. The maximum return value is 23:59:59.

26.1.42 getRelativeYear()

Use this function to return the timestamp of the specified date with the specified years added to it.

Syntax

```
getRelativeYear(timestamp,double)
```

Parameters

timestamp

The specified date.

double

The number of years to increment the specified date by.

Response

Returns the `timestamp` of the specified date with the specified number of years added to it.

26.1.43 getSecond()

Use this function to obtain the number of seconds in the seconds portion of a time value.

Syntax

```
getSecond(timestamp)
```

Parameters

timestamp
: The time value from which you want the seconds.

Response

Returns the seconds portion of the time value (00 to 59).

26.1.44 getSecondsAfter()

Use this function to determine the number of seconds one time occurs after another.

Syntax

```
getSecondsAfter(timestamp1,timestamp2)
```

Parameters

timestamp1
: A time value that is the start time of the interval being measured.

timestamp2
: A time value that is the end time of the interval.

Response

Returns the number of seconds `timestamp2` occurs after `timestamp1`. If `timestamp2` occurs before `timestamp1`, this function returns a negative number.

26.1.45 getToday()

Use this function to return the current system date and time.

Syntax

```
getToday()
```

Parameters

None

Response

Returns the current system date and time

26.1.46 getYear()

Use this function to determine the year of a date value.

Syntax

```
getYear(timestamp)
```

Parameters

timestamp
: The date that you want to derive the year from.

Response

Returns an integer whose value is a 4-digit year derived from `timestamp`.

If an error occurs, returns 1900.

26.1.47 hasDate()

Use this function to determine whether the timestamp variable contains a date value.

Syntax

```
hasDate(timestamp[])
```

Parameters

timestamp[]
: The timestamp specified.

Response

Returns a Boolean indicating whether a date has been set for `timestamp`.

26.1.48 isAfter()

Use this function to return true if date2 occurs after date1.

Syntax

```
isAfter(timestamp,timestamp)
```

Parameters

timestamp
: The date1 value.

timestamp
: The date2 value.

Response

Returns true if date2 occurs after date1.

26.1.49 isBefore()

Use this function to determines whether date2 occurs before date1.

Syntax

```
isBefore(timestamp1,timestamp2)
```

Parameters

timestamp1
: The date1 value.

timestamp2

The date2 value.

Response

Returns true if date2 occurs before date1.

26.1.50 isDate()

Use this function to determine whether string arguments form a valid date.

Syntax

```
isDate(string1,string2,string3)
```

Parameters

string1
: The year.

string2
: The month.

string3
: The day.

Response

Returns true if the string arguments for year, month, and day form a valid date.

26.1.51 isDay()

Use this function to return true if the day number in the string is between 1 and 31; it returns false otherwise.

Syntax

```
isDay(string)
```

Parameters

string
: The day number.

Response

Returns true if `string` is a valid day value, (a number between 1 and 31); otherwise returns false.

26.1.52 isLeapYear()

Use this function to return true if the year in the date is a leap year, otherwise returns false.

Syntax

```
isLeapYear(timestamp)
```

Parameters

timestamp
: The date.

Response

Returns true if the year in `timestamp` is a leap year, otherwise returns false.

26.1.53 isMonth()

Use this function to determine whether a string is a month name or abbreviation.

Syntax

```
isMonth(string)
```

Parameters

string
: A string representing a possible month name or abbreviation.

Response

Returns true if the string is either a full month name or a month abbreviation.

26.1.54 isValidDay()

Use this function to determine whether a day is valid. Both parameters are represented in number format.

Syntax

```
isValidDay(double1,double2)
```

Parameters

double1
: The month.

double2
: The day.

Response

Returns true if the specified day is valid, else false. For example, `month = 2` and `day = 35` would return false.

26.1.55 isYear()

Use this function to return true if the string is a valid year number; that is, the string must be either two or four digits; otherwise returns false.

Syntax

```
isYear(string)
```

Parameters

string
: The date to evaluate.

Response

Returns true if `string` is a valid year number (either two or four digits); otherwise returns false.

26.2 Flow Functions

To access a complete list of process-flow functions, enter **`flow.`** or **`flowfn.`** in the expression field.

Related Information

activityHasError() [page 505]
getChildColumnValue() [page 505]
getDoubleSet() [page 506]
removeAllProcessLocks() [page 507]
removeProcessLock() [page 507]
getSQLName() [page 508]
getSQLName() [page 508]

26.2.1 activityHasError()

Syntax

```
activityHasError()
```

Parameters

None

Response

Returns a Boolean value.

26.2.2 getChildColumnValue()

Use this function to return the first string value in the specified column, from the specified child table, that matches the specified value, based on the current flowstore being updated or used in Runtime.

Syntax

```
getChildColumnValue(string1,string2,string3,string4)
```

Parameters

string1
: The name of the child table.

string2
: The column to search.

string3
: The value to find in the column specified by `string2`.

string4
: The column that stores the required value.

Response

Returns a string.

26.2.3 getDoubleSet()

Use this function to return, from the current Runtime resultset, an array of numbers specified by the column name in the string. This column must store a number.

Syntax

```
getDoubleSet(string)
```

Parameters

string
: Contains the required value from the column.

Response

Returns from the current Runtime result set, an array of numbers specified by the column name.

26.2.4 removeAllProcessLocks()

Use this function to remove all process lock records locked by the current user.

Syntax

```
removeAllProcessLocks()
```

Parameters

None

Response

Returns a zero.

26.2.5 removeProcessLock()

Use this function to remove the process lock record locked by the current user and the specified process instance ID.

Syntax

```
removeProcessLock(double)
```

Parameters

double

The process instance ID to unlock.

Response

Returns a zero.

26.2.6 getSQLName()

Use this function to return the master table name for the given process definition ID, assuming that the language is English.

Syntax

```
getSQLName(double,string)
```

Parameters

double
: The process definition ID.

string
: The table display name.

  This may be left empty.

Response

Returns the master table name for the given process definition ID, assuming the language is English.

26.2.7 getSQLName()

Use this function to return the master table name for the given process definition ID.

Syntax

```
getSQLName(double,string1,string2)
```

Parameters

double
: The process definition ID.

string1
: The table display name.

  This value may be left empty.

string2
: The language code.

  This value may be left empty.

Response

Returns the master table name for the given process definition ID and a language code.

26.3 Number Functions

To access a complete list of number functions, enter **`num.`** or **`numfn.`** in the expression field.

Related Information

abs() [page 511]
average() [page 511]
buildNumberSet() [page 512]
calcProduct() [page 512]
ceiling() [page 513]
convertNumToBigDecimal() [page 514]
convertToDouble() [page 514]
convertToLong() [page 515]
cos() [page 515]
count() [page 516]
exp() [page 517]
fact() [page 517]
format() [page 518]
formatNumber() [page 519]
formatNumberDecimal() [page 519]

26.3.1 abs()

Use this function to return the absolute value of a Double value. If the argument isn't negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Syntax

```
abs(double)
```

Parameters

double
: A "double" value.

Response

Returns the absolute value of the argument.

26.3.2 average()

Use this function to return the average value from an array of numbers.

Syntax

```
average(double[])
```

Parameters

double[]
: The array of numbers to average.

Response

Returns the average value from the input array of numbers.

26.3.3 buildNumberSet()

Use this function to take a comma-separated string and converts it to an array of Double values.

Syntax

```
buildNumberSet(string)
```

Parameters

String
: A comma-separated string of numbers.

Response

Returns an array of numbers.

26.3.4 calcProduct()

Use this function to multiply an array of Double values (similar to the `Sum()` function).

Syntax

```
calcProduct(double[])
```

Parameters

double[]
: The array of numbers to multiply.

Response

Returns the product of all the numbers given as arguments.

26.3.5 ceiling()

Use this function to determine the smallest whole number that is greater than or equal to a specified limit.

Syntax

```
ceiling(double)
```

Parameters

double
: The number for which you want the smallest whole number that is greater than or equal to the parameter.

Response

Returns the smallest whole number that is greater than or equal to the parameter.

26.3.6 convertNumToBigDecimal()

Use this function to convert a BigDecimal from a Double.

Syntax

```
convertNumToBigDecimal(double)
```

Parameters

double
: The number to convert.

Response

Returns a BigDecimal.

26.3.7 convertToDouble()

Use this function to transform a string value into a number.

Syntax

```
convertToDouble(string)
```

Parameters

string
: The string to convert to a number.

Response

Returns the converted string. For example, `convertToDouble("23")` returns number 23. However, `convertToDouble("Hello")` throws an error: `Invalid String`.

26.3.8 convertToLong()

Given a double, returns a string.

Syntax

```
convertToLong(double)
```

Parameters

double
: The value to convert to a string.

Response

Returns a string.

26.3.9 cos()

Use this function to return the trigonometric cosine of a value.

Syntax

```
cos(double)
```

Parameters

double
: The value.

Response

Returns the cosine of the parameter.

26.3.10 count()

Use this function to return the number of items in a given array.

Syntax

```
count(double[])
```

Parameters

double[]
: The array to count.

Response

Returns the number of items in a given array.

26.3.11 exp()

Use this function to return <e> raised to the power of the parameter.

Syntax

```
exp(double)
```

Parameters

double
: A parameter.

Response

Returns the value $<e>^n$, where e is the base of the natural logarithms.

26.3.12 fact()

Use this function to determine the factorial of a number.

Syntax

```
fact(double)
```

Parameters

double
: The number that you want the factorial for.

Response

Returns the factorial of the parameter.

26.3.13 format()

Use this function to format the input string according to the format pattern supplied.

Syntax

```
format(string1,string2,string3,string4)
```

Parameters

string1
: The string representation of the input value.

string2
: The type of the variable stored in `<string1>`; for example, NUMBER.

string3
: Generally the pattern to be applied, except if `string2` = NUMBER, in which case string3 is the code for the sub-variable type represented by the variable `<string1>`, for example, CURRENCY.

string4
: The format string to convert the variable `<string1>` to, if the variable type is a NUMBER.

Response

Returns the string, with the format applied.

26.3.14 formatNumber()

Use this function to return a formatted number string based on the `java.text.NumberFormat`, the specified format, and a specified number.

Syntax

```
formatNumber(double,string)
```

Parameters

double
: The number to format.

string
: The format string as specified by the `java.text.NumberFormat` format.

Response

Returns a formatted number string.

26.3.15 formatNumberDecimal()

Use this function to format a number with zeros for decimal and removes the decimal. For example, the number 1234.43, with a format of 0000000000 and two decimals, becomes 00000123443

Syntax

```
formatNumberDecimal(double1,string,double2)
```

Parameters

double1

The number to format.

string

The format string, that is, 0000; padded with zeroes to a limited width of 4.

double2

The decimal places to save and to pad for the number.

Response

Returns a formatted number based on a simple format pattern.

26.3.16 getFilteredData()

Use this function to return the filtered number array for a given table and column, given a filter rule.

Syntax

```
getFilteredData(string1,string2,double,string3,string4)
```

Parameters

string1

The table name of the result set to filter.

string2

The column name of the values that you would like to filter.

double

The filter rule ID.

string3

The source of the rule.

string4

The arguments for the rule.

Response

Returns the filtered number array for a given table and column, given a filter rule.

26.3.17 greatest()

Use this function to return the largest number in a number array.

Syntax

```
greatest(double[])
```

Parameters

double[]
: A number array.

Response

Returns the largest number in the number array.

26.3.18 integer()

Use this function to determine the largest whole number less than or equal to a number.

Syntax

```
integer(double)
```

Parameters

double
: The number for which you want the largest whole number that is less than or equal to it.

Response

Returns the largest whole number less than or equal to the parameter. If the parameter is too small or too large to be represented as an integer, integer returns 0.

26.3.19 isFloat()

Use this function to return true if the string is an unsigned floating point (real) number.

Syntax

```
isFloat(string)
```

Parameters

string
: The string to search.

Response

Returns true if `<string>` is an unsigned floating point (real) number. Also returns true for unsigned integers. If you want to distinguish between integers and floating point numbers, first call `isInteger`, then call `isFloat()`.

26.3.20 isInteger()

Use this function to determine whether all characters in a string are digits.

Syntax

```
isInteger(string)
```

Parameters

string
: The string to be verified.

Response

Returns false if some characters aren't digits.

26.3.21 isIntegerInRange()

Use this function to determine whether a string is a number within a numerical range.

Syntax

```
isIntegerInRange(string,double1,double2)
```

Parameters

string
: The string.

double1
: The start of the range.

double2
: The end of the range.

Response

Returns true if the string is between the two numbers, else returns false.

26.3.22 isNegativeInteger()

Use this function to determine whether an integer is negative or positive.

Syntax

```
isNegativeInteger(string)
```

Parameters

string
: The integer.

Response

Returns true if the string is an integer less than 0.

26.3.23 isNonNegativeInteger()

Use this function to determine whether an integer is equal to zero or positive.

Syntax

```
isNonNegativeInteger(string)
```

Parameters

string
: The integer.

Response

Returns true if the string is an integer greater than or equal to 0.

26.3.24 isNonPositiveInteger()

Use this function to determine whether an integer is negative or equal to zero.

Syntax

```
isNonPositiveInteger(string)
```

Parameters

string
: The integer.

Response

Returns true if the string is an integer less than or equal to 0.

26.3.25 isNumber()

Use this function to determine whether the parameter has a decimal point. If so, returns the string without a decimal point.

Syntax

```
isNumber(string)
```

Parameters

string
: The string to remove the decimal point from.

Response

Returns an integer without the decimal point.

26.3.26 isNumberEmpty()

Returns true if the input value is NaN.

Syntax

```
isNumberEmpty(double)
```

Parameters

double
: The number to evaluate.

Response

Returns true if the number is NaN.

26.3.27 isPositiveInteger()

Use this function to determine whether an integer is greater than 0.

Syntax

```
isPositiveInteger(string)
```

Parameters

string
: The integer.

Response

Returns true if the string is an integer greater than 0.

26.3.28 isSignedFloat()

Use this function to determine whether a string is a floating point number.

Syntax

```
isSignedFloat(string)
```

Parameters

string
: A string representing a floating point number.

Response

Returns true, if the string is a floating point number. A leading plus sign or minus sign is permitted.

26.3.29 isSignedInteger()

Use this function to determine whether the characters in a string are numbers.

Syntax

```
isSignedInteger(string)
```

Parameters

string
: The string to be searched.

Response

Returns true if all characters in string are numbers; a leading plus sign or minus sign is allowed.

26.3.30 least()

Use this function to return the smallest number in a number array.

Syntax

```
least(double[])
```

Parameters

double[]
: number

Response

Returns the smallest number in `double[]`.

26.3.31 logTen()

Use this function to determine the base 10 logarithm of a number.

Syntax

```
logTen(double)
```

Parameters

double
: The number that you want the base 10 logarithm for. The value of the parameter must not be negative.

Response

Returns the base 10 logarithm of the parameter.

26.3.32 max()

Use this function to determine the larger of two double values.

Syntax

```
max(double1,double2)
```

Parameters

double1
: The first double value to compare.

double2
: The second double value to compare.

Response

Returns the larger of the two double values.

26.3.33 min()

Use this function to determine the smaller of two double values.

Syntax

```
min(double1,double2)
```

Parameters

double1
: The first Double value to compare.

double2

The second Double value to compare.

Response

Returns a Double value that is the smaller value.

26.3.34 mod()

Use this function to obtain the remainder (modulus) of a division operation.

Syntax

```
mod(double1,double2)
```

Parameters

double1

The number to divide by `double2`.

double2

The number to divide `double1` by.

Response

Returns the result of the division operation.

26.3.35 pi()

Use this function to multiply the specified parameter by pi.

Syntax

```
pi(double)
```

Parameters

double
: The number you want to multiply by pi.

Response

Returns the result of multiplying the parameter by pi.

26.3.36 power()

Use this function to raise the first parameter to the power of the second parameter and returns the result.

Syntax

```
power(double1,double2)
```

Parameters

double1
: The base value.

double2
: The exponent value.

Response

Returns a double value.

26.3.37 rand()

Use this function to obtain a random whole number between 1 and a specified upper limit.

Syntax

```
rand(double)
```

Parameters

double
: The upper limit of the range of random numbers you want returned.

Response

Returns a random whole number between 1 and the parameter, inclusive.

26.3.38 replaceEmptyNumber()

Use this function to return the replacing value if the original value is invalid, otherwise returns the original value.

Syntax

```
replaceEmptyNumber(double1,double2)
```

Parameters

double1
: The original Double value.

double2
: The replacing value if the original value is invalid.

Response

Returns a Double value.

26.3.39 round()

Use this function to round a number to the specified number of decimal places.

Syntax

```
round(double1,double2)
```

Parameters

double1
: The number that you want to round.

double2
: The number of decimal places to which you want to round `double1` to.

  Valid values are 0 through 18.

Response

Returns `double1` rounded to the specified number of decimal places, or null if the function fails or if any argument's value is null.

26.3.40 sign()

Reports whether a number is negative, zero, or positive.

Syntax

```
sign(double)
```

Parameters

double
: The number that you want to find out the sign for.

Response

Returns an integer (-1, 0, or 1) that indicates the sign of the parameter. If the parameter is 0, returns 0.

26.3.41 sin()

Use this function to return the sine of a Double in radians.

Syntax

```
sin(double)
```

Parameters

double
: The angle (in radians) for which you want the sine.

Response

Returns the sine of the parameter.

26.3.42 sqrt()

Use this function to return the square root of the number.

Syntax

```
sqrt(double)
```

Parameters

Double
: The number that you want the square root for.

Response

Returns the square root of the parameter.

26.3.43 stripNonNumericCharacters()

Use this function to strip a string to a digital string, and then convert it to a Double.

Syntax

```
stripNonNumericCharacters(string)
```

Parameters

string
: The string to strip.

Response

Returns a Double.

26.3.44 sum()

Use this function to sum a double array.

Syntax

```
sum(double[])
```

Parameters

Double[]
: A Double array.

Response

Returns a Double.

26.3.45 tan()

Use this function to return the tangent of an angle.

Syntax

```
tan(double)
```

Parameters

double
: The angle (in radians) for which you want the tangent.

Response

Returns the tangent of the parameter.

26.3.46 textToNumber()

Use this function to convert a String to a Double value.

Syntax

```
textToNumber(string)
```

Parameters

string
: A String object.

Response

Returns a Double.

26.3.47 toDouble()

Use this function to convert a Double to a Double Object.

Syntax

```
toDouble(double)
```

Parameters

double
: A Double value.

Response

Returns a Double object.

26.3.48 truncate()

Use this function to truncate a number to the specified number of decimal places.

Syntax

```
truncate(double1,double2)
```

Parameters

double1

The number you want to truncate.

double2

The number of decimal places to which you want to truncate `double1` (valid values are 0 through 18).

Response

Returns the result of the truncation, or null if the function fails or if any argument is null.

26.4 String Functions

To access a complete list of string functions, enter **`string.`** or **`strfn.`** in the expression field.

Related Information

26.4.1 asc()

Use this function to return the integer ASCII value of the first character of a string. You can use asc to determine the case of a character by testing whether its ASCII value is within the appropriate range.

Syntax

```
asc(string)
```

Parameters

string
: The string for which you want the ASCII value of the first character.

Response

Returns the ASCII value of the first character in the string. If the string is null, returns 0.

26.4.2 buildTextSet()

Use this function to take a comma-separated string and converts it to an array of strings.

Syntax

```
buildTextSet(string)
```

Parameters

string
: A comma-separated input string.

Response

Returns an array of strings.

26.4.3 charInString()

Use this function to determine whether a given character occurs in a string. Returns true if the single character (actually a string) is contained within the string.

Syntax

```
charInString(char,string)
```

Parameters

char
: The character to search for in the string.

string
: The string to be searched.

Response

Returns true if the single character (actually a string) is contained within `string`.

26.4.4 charInString()

Use this function to determine whether a character exists within a specified string.

Syntax

```
charInString(char,string)
```

Parameters

char
: The character to search for.

string
: The string within which to search for the character.

Response

Returns true if the character does exist; otherwise returns false.

26.4.5 checkInvalidChar()

Use this function to determine whether the string contains a digit.

Syntax

```
checkInvalidChar(String)
```

Parameters

string
: The string to search for a digit.

Response

Returns true if the string contains a digit; otherwise returns false.

26.4.6 checkMinimumLength()

Use this function to determine whether the string has a minimum length of `int`.

Syntax

```
checkMinimumLength(string,int)
```

Parameters

string
: The string to evaluate.

int
: The minimum length to check for.

Response

Returns 0 if `string` is less than the minimum length; otherwise returns 1.

26.4.7 checkMinimumLength()

Use this function to determine whether the length of the string is less than the specified number.

Syntax

```
checkMinimumLength(string,double)
```

Parameters

string
: The string.

double
: The number of characters in the string.

Response

Returns true if the string contains fewer characters than the specified length; otherwise returns false.

26.4.8 chr()

Use this function to extract the first character of a string.

Syntax

```
chr(string)
```

Parameters

string
: A string from which you want the first character extracted.

Response

Returns the first character of the string. If string is null, returns null.

26.4.9 collectionsEqual()

Use this function to determine whether two string arrays contain the same strings.

Syntax

```
collectionsEqual(string1[],string2[])
```

Parameters

string1[]
: String array collection1.

string2[]
: String array collection2.

Response

Returns true if `string1[]` contains all the items in `string2[]` and vice versa.

26.4.10 collectionsIn()

Use this function to determine whether all the strings of the first array are in the second array.

Syntax

```
collectionsIn(string1[],string2[])
```

Parameters

string1[]
: String array collection 1.

string2[]
: String array collection 2.

Response

Returns true if all items in `string1[]` are contained in `string2[]`.

26.4.11 collectionsIntersect()

Use this function to determine whether any string items in the first array are present in the second array.

Syntax

```
collectionsIntersect(string1[],string2[])
```

Parameters

string1[]
: String array collection1.

string2[]
: String array collection2.

Response

Returns true if any string item in `string1[]` is contained in `string2[]`.

26.4.12 concat()

Use this function to concatenate two strings into one string.

Syntax

```
concat(string1,string2)
```

Parameters

string1
: The first string.

string2
: The second string.

Response

Returns the concatenated result of two strings.

26.4.13 concatTextSet()

Use this function to return the concatenated result of a text array delimited by the string.

Syntax

```
concatTextSet(string[],string)
```

Parameters

string[]
: The string array to concatenate.

string
: A delimiting string.

Response

Returns the concatenated result of `string[]` delimited by `string`.

26.4.14 contains()

Use this function to determine whether one of the values in the comma-separated list of strings is contained in the string array.

Syntax

```
contains(string[],string)
```

Parameters

string[]
: An array of strings.

string

A comma-separated list of strings.

Response

Returns true if one of the values in the comma-separated list of strings is contained in the string array; otherwise returns false.

26.4.15 fill()

Use this function to build a string of the specified length by repeating the specified characters until the result string is long enough.

Syntax

```
fill(string,double)
```

Parameters

string

A string whose value will be repeated to fill the return string.

double

An integer that represents the length of the string you want returned.

Response

Returns a string filled with the specified number of characters.

If the string has more than the specified number of characters, only the specified number of characters is returned.

If the string has fewer than the specified number of characters, the characters specified in the `Double` are repeated until the return string has the specified number of characters.

If any argument's value is null or 0, the function returns null.

26.4.16 format()

Use this function to format the input string based on the format pattern supplied.

Syntax

```
format(string1,string2,string3,string4)
```

Parameters

string1

The string to be formatted.

string2

TEXT, NUMBER, or DATE.

> **i Note**
>
> To use a COBOL-style pattern, this parameter must be TEXT and the number should be passed as a string.

string3

The format pattern to apply.

When `string1` is a number as a string, and `string2` is TEXT, and the format is a COBOL-style pattern, the function truncates the string value if it is longer than the format.

The function pads with 0 if the place was empty and the mask had 0 for that position. Padding isn't applied if the mask has # for the position and there was no 0 before #. Digit 0 shows as absent depending upon position (leading if before v, trailing if after v) 0 digit v line up the decimal with this s if present in the pattern then replace with a specific character. For example, -34.23000, with the mask 0000v####s, should return 0034& because the last digit was 3 represented by &.

string4

This parameter is only used when `string2` contains NUMBER and `String3` is DECIMAL only; otherwise, takes an empty string.

Response

Returns the formatted string.

26.4.17 getDataTableValue()

Use this function to return the first occurrence of a value stored in the data table in the column specified. If there are zero rows in the data table, a null string is returned.

Syntax

```
getDataTableValue(dataTable,string)
```

Parameters

dataTable
: The source data table.

string
: The column name for the required value.

Response

Returns the first occurrence of a value stored in the data table in the column specified.

26.4.18 getStringInPOS()

Given a comma-separated string, returns the position of the specified string.

Syntax

```
getStringInPOS(string1,double,string2)
```

Parameters

string1
: A comma-separated list.

double
: The position of the value required from the string.

string2
: The default value to return if the value in the specified position doesn't exist.

Response

Returns the value stored in the position specified of a comma-separated list.

26.4.19 getValueByDate()

Use this function to return the first value for a specified column in a data table, where a specified date falls between the values of two dates.

Syntax

```
getValueByDate(dataTable,string1,string2,timestamp,string3)
```

Parameters

dataTable
: The specified data table.

string1
: The from date.

string2
: The to date.

timestamp
: The lookup date.

string3
: The name of the column.

Response

Returns a string.

26.4.20 isAlphabetic()

Use this function to return true if the string consists of English letters.

Syntax

```
isAlphabetic(string)
```

Parameters

string
: The string.

Response

Returns true if the string consists of English letters (A .. Z, a..z).

26.4.21 isAlphanumeric()

Use this function to determine whether a string is alphanumeric. Returns true if the string contains only English letters (A to Z/a to z) and/or numbers; otherwise returns false.

Syntax

```
isAlphanumeric(string)
```

Parameters

string
: String to validate

Response

Returns true if `string` contains only English letters and/or numbers, otherwise returns false.

26.4.22 isDigit()

Use this function to determine whether the passed character is a digit.

Syntax

```
isDigit(int)
```

Parameters

int
: An integer.

Response

Returns true if the parameter is a letter or digit.

26.4.23 isEmail()

Use this function to determine whether a string is a valid email address.

Syntax

```
isEmail(string)
```

Parameters

string
: A string representing an email address.

Response

Returns true if the string is a valid email address.

26.4.24 isEmpty()

Use this function to return true if a string is empty or contains space characters only.

Syntax

```
isEmpty(string)
```

Parameters

string
: The string to test.

Response

Returns true if the string is empty or contains space characters only.

26.4.25 isLetter()

Use this function to determine whether a character is a letter.

Syntax

```
isLetter(string)
```

Parameters

string
: The string to be tested for letters.

Response

Returns an array containing logical true (1) where the string contains letters of the alphabet and logical false (0) where it doesn't.

26.4.26 isLetterCSI()

Use this function to determine whether the first character is a character or a digit.

Syntax

```
isLetterCSI(string)
```

Parameters

string
: A string.

Response

Returns a character or a digit.

26.4.27 isLetterOrDigit()

Use this function to determine whether the specified character is a letter or a digit.

Syntax

```
isLetterOrDigit(string)
```

Parameters

string
: The character to check.

Response

Returns true if the character is a letter or a digit; false if the character isn't a letter or a digit.

26.4.28 isWhitespace()

Use this function to test a string to determine whether the string is empty or contains only space characters.

Syntax

```
isWhitespace(string)
```

Parameters

string
: The string to test.

Response

Returns true if the string is empty or contains space characters only.

26.4.29 left()

Use this function to obtain a specified number of characters from the beginning of a string.

Syntax

```
left(String,double)
```

Parameters

string
: The string containing the characters you want.

double
: An integer specifying the number of characters you want.

Response

Returns the leftmost number of characters in the string if it succeeds and null if an error occurs. If any argument's value is null, returns null.

26.4.30 leftTrim()

Use this function to remove spaces from the beginning of a string.

Syntax

```
leftTrim(string)
```

Parameters

string
: The string you want returned with leading spaces deleted.

Response

Returns a copy of the string with leading spaces deleted if the function succeeds and null if an error occurs. If string is null, returns null.

26.4.31 length()

Reports the length of a string.

Syntax

```
length(string)
```

Parameters

string
: The string that you want to determine the length of.

Response

Returns an integer the value of which is the length of the string if it succeeds and -1 if an error occurs. If the string is null, the function returns 0.

26.4.32 lower()

Use this function to convert all the characters in a string to lowercase.

Syntax

```
lower(string)
```

Parameters

string
: The string you want to convert to lowercase letters.

Response

Returns the string with uppercase letters changed to lowercase.

26.4.33 match()

Use this function to determine whether a string's value contains a particular pattern of characters.

Syntax

```
match(string1,string2)
```

Parameters

string1
: The string that you want to look for a pattern of characters in.

string2
: The string whose value is the text pattern.

Response

Returns true if string matches the text pattern and false if it doesn't. Also returns false if either argument hasn't been assigned a value or the pattern is invalid. If any argument's value is null, returns false.

26.4.34 matchPattern()

Use this function to determine whether the string contains the matched pattern.

Syntax

```
matchPattern(string1,string2)
```

Parameters

string1
: The string to search within.

string2
: A regular expression pattern.

Response

Returns true if the match is made; otherwise returns false.

26.4.35 mid()

Use this function to obtain characters starting from a specified position in a string to the end of the string.

Syntax

```
mid(string,double1,double2)
```

Parameters

string

The string from which you want characters returned.

double1

An integer specifying the position of the first character that you want returned.

The position of the first character in the string is 1.

double2

Specifies the number of characters that you want returned.

If you don't enter a value or if the value is greater than the number of characters to the right of start, the function returns the remaining characters in the string.

Response

Returns the characters specified, starting from the position in `double1` up to the number specified in `double2`.

If the starting position is greater than the number of characters in the string, this function returns null. If the value in `double2` is greater than the number of characters remaining after the start position, returns the remaining characters.

The returned string isn't filled with spaces to make it the specified length.

If any argument's value is null, null is returned.

26.4.36 numberToText()

Use this function to convert a number to a string value.

Syntax

```
numberToText(double)
```

Parameters

double
: A number to convert to a string.

Response

Returns the number as a string.

26.4.37 pos()

Use this function to search for one string within another string from a specified position.

Syntax

```
pos(string1,string2,double)
```

Parameters

string1
: The string you want to find `string2` in.

string2
: The string you want to find in `string1`.

double

An integer indicating where the search will begin in `string1`. The default is 1.

Response

Returns an integer whose value is the starting position of the first occurrence of `string2` in `string1` after the specified position. If `string2` isn't found in `string1` or if the starting position isn't within `string1`, returns 0. If any argument's value is null, returns 0.

26.4.38 pos()

Use this function to search for one string within another string and returns the position of the found string.

Syntax

```
pos(string1,string2)
```

Parameters

string1
: The string in which you want to find `string2`.

string2
: The string you want to find in `string1`.

Response

Returns a Double whose value is the starting position of the first occurrence of `string2` in `string1`. If `<string2>` isn't found in `string1`, returns 0. If any argument's value is null, returns 0.

26.4.39 replace()

Use this function to replace a portion of one string with another.

Syntax

```
replace(string1,double1,double2,string2)
```

Parameters

string1
: The string in which you want to replace characters with `string2`.

double1
: An integer whose value is the number of the first character you want replaced. The first character in the string is number 1.

double2
: An integer whose value is the number of characters in `string1` you want to replace.

string2
: The replacement string.

Response

Returns the string that replaces characters in `string1`. The number of characters in `string2` can be greater than, equal to, or less than the number of characters you are replacing.

26.4.40 replacePattern()

Use this function to execute a regex pattern replacement (Perl-like); for example: `s/xx/yy/ig`

Syntax

```
replacePattern(string1,string2)
```

Parameters

string1
: The string that you want to apply regex to.

string2
: The search and replace pattern to apply.

Response

Returns the replaced string.

26.4.41 reverse()

Use this function to reverse the order or characters in a string.

Syntax

```
reverse(string)
```

Parameters

string
: A string whose characters you want to reorder so that the last character is first, the second-last character is second, and so on.

Response

Returns the original `string` with the characters in reverse order. Returns null if the argument string is null or if the function fails.

26.4.42 right()

Use this function to obtain a specified number of characters from the end of a string.

Syntax

```
right(string,double)
```

Parameters

string
: The string from which you want characters returned.

double
: An integer whose value is the number of characters you want returned from the right end of the string.

Response

Returns the rightmost <n> characters in the string or null if an error occurs. If any argument's value is null, this function returns null.

26.4.43 rightJustifyFill()

Use this function to fill the first <n> characters of the formatting string to the left side of the original string (<n> = formatting string length - original string length).

Syntax

```
rightJustifyFill(string1,string2)
```

Parameters

string1

The string to format.

string2

The format string.

The format string has to be longer than `string1` or else the function returns the original string.

Response

Returns the formatted string.

26.4.44 rightTrim()

Use this function to remove spaces from the end of a string.

Syntax

```
rightTrim(string,double)
```

Parameters

string

The string that you want the trailing blanks deleted from.

double

An integer whose value is the number of characters you want returned.

Response

Returns the string with trailing blanks deleted and null if an error occurs. If any argument's value is null, the function returns null.

26.4.45 searchAndReplace()

Use this function to replace one string with another string inside a given string object.

Syntax

```
searchAndReplace(string1,string2,string3)
```

Parameters

string1
: The string to replace.

string2
: The search string, which is part of `string1`.

string3
: The string that replaces the search string in `string1`.

Response

Returns a string.

26.4.46 space()

Use this function to build a string of the specified length consisting of spaces.

Syntax

```
space(double)
```

Parameters

double

An integer whose value is the length of the string you want filled with spaces.

Returns

Returns a string filled with <n> spaces if the function succeeds and null if an error occurs. If the integer is null, this function returns null.

26.4.47 titleCase()

Use this function to return a new string in title case: the first character of every word is uppercase and the remaining characters are lower case.

Syntax

```
titleCase(string)
```

Parameters

string
: The input string to convert to title case.

Response

Returns a new string object in title case.

26.4.48 toText()

Use this function to convert a number to a string value.

Syntax

```
toText(double)
```

Parameters

double
: The number to convert to a string value.

Response

Returns the number converted to a string value.

26.4.49 trim()

Use this function to remove leading and trailing spaces from a string.

Syntax

```
trim(string)
```

Parameters

string
: The string that you want the leading and trailing spaces deleted from.

Response

Returns the string with all leading and trailing spaces deleted.

26.4.50 upper()

Use this function to convert all the characters in a string to uppercase.

Syntax

```
upper(string)
```

Parameters

string
: The string you want to convert to uppercase letters.

Returns a string with lowercase letters changed to uppercase; otherwise returns null if an error occurs. If the string is null, returns null.

26.5 System Functions

To access a complete list of system functions, type **`system.`** or **`sysfn.`** in the expression field.

Related Information

getBooleanValue() [page 574]
getCounter() [page 575]
getDataTableValue() [page 575]
getDateValue() [page 576]
getFilteredTextData() [page 576]
getNextKey() [page 577]
getNumberValue() [page 578]
getTextValue() [page 578]
log() [page 579]

26.5.1 getBooleanValue()

Use in a library function to access a rule-generated constant value of the specified type from system memory (rules place these values into memory using a reciprocal function (that is, `system.put<data_type>()`).

Syntax

```
getBooleanValue(aoServiceCommand,string)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

Response

Returns a Boolean value.

26.5.2 getCounter()

Use this function to return a numeric field by the name specified.

Syntax

```
getCounter(string)
```

Parameters

string
: The name of the field that stores the required numeric field.

Response

Returns a numeric field stored in the service command by the name specified.

26.5.3 getDataTableValue()

Use in a library function to access a rule-generated constant value of the specified type from system memory (rules place these values into memory using a reciprocal function (that is, `system.put<data_type>()`).

Syntax

```
getDataTableValue(aoServiceCommand,string)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

Response

Returns a DataTable value.

26.5.4 getDateValue()

Use in a library function to access a rule-generated constant value of the specified type from system memory (rules place these values into memory using a reciprocal function (that is, `system.put<data_type>()`).

Syntax

```
getDateValue(aoServiceCommand,string)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

Response

Returns a TimeStamp.

26.5.5 getFilteredTextData()

Use this function to return the filtered string array for a table and column, given a filter rule.

Syntax

```
getFilteredTextData(string1,string2,string3,string4,double,string5,string6)
```

Parameters

string1
: The product name.

string2
: The product version.

string3
: The table name.

string4
: The column name of the values that you would like to filter.

double
: The filter rule.

string5
: The source of the rule.

string6
: The arguments for the rule.

Response

Returns the filtered number array for a given table and column, given a filter rule.

26.5.6 getNextKey()

Use this function to return the next primary key available for a given table name.

Syntax

```
getNextKey(string)
```

Parameters

string
: The table name specified.

Response

Returns the next primary key available for a given table name.

26.5.7 getNumberValue()

Use in a library function to access a rule-generated constant value of the specified type from system memory (rules place these values into memory using a reciprocal function (that is, `system.put<data_type>()`).

Syntax

```
getNumberValue(aoServiceCommand,string)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

Response

Returns a Double.

26.5.8 getTextValue()

Use in a library function to access a rule-generated constant value of the specified type from system memory (rules place these values into memory using a reciprocal function (that is, `system.put<data_type>()`).

Syntax

```
getTextValue(aoServiceCommand,string)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

Response

Returns a string.

26.5.9 log()

Use this function to return the natural logarithm of a number.

Syntax

```
log(double)
```

Parameters

double
: The number that you want the natural logarithm (base e) for. The value of the parameter must be greater than 0.

Response

Returns the natural logarithm of the parameter.

26.5.10 logMessage()

Use this function to log a message in all functions in the utility Logger, for debugging purposes.

Syntax

```
logMessage(string)
```

Parameters

string
: The message to log.

Response

Returns the specified log message that was passed in.

26.5.11 logResult()

Use this function to determine whether the RuleResult object is present and logs to the Logger, otherwise logs a `ResultRule is null` message.

Syntax

```
logResult(string)
```

Parameters

string
: The RuleResult object.

Response

Returns a string.

26.5.12 putBooleanValue()

In a rule, places a constant value of the specified type into system memory for access by library functions using the reciprocal function (that is, `system.get<data_type>()`).

Syntax

```
putBooleanValue(aoServiceCommand,string,boolean)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

boolean
: A Boolean value.

Response

26.5.13 putCounter()

Use this function to put a counter with an initial value.

Syntax

```
putCounter(string,double)
```

Parameters

string
: The counter name.

double
: The counter's initial value.

Response

Returns the counter's initial value.

26.5.14 putDataTable()

In a rule, places a constant value of the specified type into system memory for access by library functions using the reciprocal function (that is, `system.get<data_type>()`).

Syntax

```
putDataTable(aoServiceCommand,string,dataTable)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

dataTable
: A DataTable value.

Response

26.5.15 putDateValue()

In a rule, places a constant value of the specified type into system memory for access by library functions using the reciprocal function (that is, `system.get<data_type>()`).

Syntax

```
putDateValue(aoServiceCommand,string,timestamp)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

timestamp
: A timestamp value.

Response

26.5.16 putNumberValue()

In a rule, places a constant value of the specified type into system memory for access by library functions using the reciprocal function (that is, system.get`<data_type>`()).

Syntax

```
putNumberValue(aoServiceCommand,string,double)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string
: The name of the system variable.

double
: A Double value.

Response

26.5.17 putTextValue()

In a rule, places a constant value of the specified type into system memory for access by library functions using the reciprocal function (that is, `system.get<data_type>()`).

Syntax

```
putTextValue(aoServiceCommand,string1,string2)
```

Parameters

aoServiceCommand
: `ServiceCommand`

string1
: The name of the system variable.

string2
: A string value.

Response

26.5.18 runSQLScript()

Use this function to execute an SQL stored procedure from a rule.

Syntax

```
runSQLScript(sqlScript)
```

Parameters

sqlScript
: The name of the SQL script (package) in the database.

Response

Returns Integer 1 if the function executes successfully.

26.5.19 sortDataTable()

Use this function to sort a data table based on the sort column, the column type, and the sort order.

Syntax

```
sortDataTable(dataTable,string1,string2,string3,string4)
```

Parameters

dataTable
: The data table to sort.

string1

The name of the sort column.

string2

The type of the sort column.

string3

The sort order.

string4

The name of the new data table object.

Response

Returns the sorted DataTable.

26.6 UAS Functions

To access a complete list of User Authorization Service functions, enter **`uas.`** or **`uasfn.`** in the expression field.

Related Information

UASBooleanRule() [page 586]
UASContains() [page 587]
UASDataTableRule() [page 588]
UASDateRule() [page 588]
UASGetList() [page 589]
UASGetValue() [page 590]
UASNumberRule() [page 590]
UASTextRule() [page 591]

26.6.1 UASBooleanRule()

Use this function to execute a `User Authorization Service` rule and returns a Boolean value.

```
UASBooleanRule(string1,string2,string3,string4,string5)
```

Executes a `User Authorization Service` rule and returns a Boolean value.

Parameters

string1
: The name of the profile.

string2
: The XPath of the component.

string3
: The qualifiers.

string4
: The attributes.

string5
: The input values.

Response

Returns a Boolean value from a `User Authorization Service` rule.

26.6.2 UASContains()

Use this function to check to see whether the input parameter is contained in the `User Authorization Service`.

```
UASContains(string1, string2,string3, string4,string5)
```

Checks to see whether the input parameter is contained in the `User Authorization Service`.

Parameters

string1
: The name of the profile.

string2
: The XPath of the component.

string3
: The qualifiers.

string4
: The attributes.

string5
: The name of the value to check for.

Response

Returns a Boolean value indicating whether this value is listed in the `User Authorization Service`.

26.6.3 UASDataTableRule()

Use this function to execute a `User Authorization Service` rule that returns a data table.

```
UASDataTableRule(string1,string2,string3,string4,string5)
```

Executes a `User Authorization Service` rule that returns a data table.

Parameters

string1
: The name of the profile.

string2
: The XPath of the component.

string3
: The qualifiers.

string4
: The attributes.

string5
: The name of the value to check for.

Response

Returns a data table from a `User Authorization Service` rule.

26.6.4 UASDateRule()

Use this function to execute a `User Authorization Service` rule that returns a date.

```
UASDateRule(string1,string2,string3,string4,string5)
```

Executes a `User Authorization Service` rule that returns a date.

Parameters

string1
: The name of the profile.

string2
: The XPath of the component.

string3
: The qualifiers.

string4
: The attributes.

string5
: The name of the value to check for.

Response

Returns a date value from a `User Authorization Service` rule.

26.6.5 UASGetList()

Use this function to return a List from the `User Authorization Service` for a profile.

```
UASGetList(string1, string2,string3, string4)
```

Returns a List from the `User Authorization Service` for a profile.

Parameters

string1
: The name of the profile.

string2
: The XPath of the component.

string3
: The qualifiers.

string4
: The attributes.

Response

Returns a List from the `User Authorization Service`.

26.6.6 UASGetValue()

Use this function to return a value from the `User Authorization Service` based on the profile, qualifiers, and attribute.

Syntax

```
UASGetValue(string1,string2,string3,string4)
```

Parameters

string1
: The name of the profile.

string2
: The XPath of the component.

string3
: The qualifiers.

string4
: The attributes.

Response

Returns a value from the `User Authorization Service`.

26.6.7 UASNumberRule()

Use this function to execute a `User Authorization Service` rule that returns a number.

```
UASNumberRule(string1,string2,string3,string4,string5)
```

Executes a `User Authorization Service` rule that returns a number.

Parameters

string1
: The name of the profile.

string2
: The XPath of the component.

string3
: The qualifiers.

string4
: The attributes.

string5
: The rule arguments.

Response

Returns a number from a `User Authorization Service` rule.

26.6.8 UASTextRule()

Use this function to execute a `User Authorization Service` rule that returns a string.

```
UASTextRule(string1,string2,string3, string4,string5)
```

Executes a `User Authorization Service` rule that returns a string.

Parameters

string1
: The name of the profile.

string2
: The XPath of the component.

string3
: The qualifiers.

string4
: The attributes.

string5
: The rule arguments.

Response

Returns a string from a `User Authorization Service` rule.

27 User-Defined Product OData Services

27.1 Product Model Prerequisites

This configuration is used to generate the Product Runtime OData EDMX.

It consists of the following components:

OData Object Layer

Used to configure OData Entity and Complex Types.

The `ODATA_USAGE` attribute value determines if model is intended to be Entity or Complex type.

If the model is to be Complex Type, the `TABLE_NAME` attribute value must be empty to ensure that the data and model are transient.

OData Service API

Allows configuration of OData Function Imports.

OData Function Imports are defined by setting values for `Service Rule`, `Return Type`, `Return Type Multiplicity`, `Is Entity Return Type`, `Method` and `Odata Usage` attributes.

Service Rule

Function Import logic is configured using the Script Rule editor.

The Function Import parameter list is based on the input arguments configured for the script rule.

Return Type

The OData object type the function import will be returning.

This is the value contained within the `DESCRIPTION`

Return Type Multiplicity

Determines if the function import is to return a single object or a collection.

Is Entity Return Type

A Boolean flag indicating whether or not the object is an Entity or a Complex type.

Method

The HTTP method of the function import.

For example, GET or POST.

OData Usage

Determines if the Service API is to be modeled and exposed as a function import in the Product Runtime OData EDMX.

Assembled Components of

Any such assembled components are automatically modeled as an Entity Type in the Product Runtime OData EDMX.

modifier type
"Reference"

OData Service uses a variety of configured data from FS-PRO to support various operations. To enable this, a bootstrap called `SAP PWP Bootstrap` is provided. The contents of this bootstrap are described below.

SAP Product Base Template (Modifier Type - Product Base)

We suggest that consumers of this feature develop and configure new products in FS-PRO by inheriting from this template since it has the product model prerequisites configurations available ready to use.

The attributes of the `OData Service API` component in the `Product Services` folder are listed below:

Attribute Name	Internal Name	Data Type	Length	Precision	Data List Source	Description
Service Name	ServiceName	TEXT	255	0		Specifies the name of the service.
Service Rule	ServiceRule	RULE	16	0		Specifies the rule to execute the service.
Return Type	ReturnType	TEXT	255	0		Specifies the `<TABLE_NAME>` that is expected as a return to the service call.
Return Type Multiplicity	ReturnTypeMultiplicity	DATA_LIST	250	0	getAPIReturnTypeMultiplicity	Configured using the data list to define how many entities are to be returned.
Is Entity Return Type	IsEntityReturnType	BOOLEAN	1	0		A Boolean Flag to determine if the service call has a return type of Entity.
Method	Method	DATA_LIST	250	0	getAPIHttpMethod	Specifies the OData Method configured using Data list.
Odata Usage	OdataUsage	DATA_LIST	250	0	APIOdataUsage	Defines whether the service call is Function import or not. Defined using data list.
SYSATTR_TABLE_NAME	SYSATTR_TABLE_NAME	TEXT	255	0		Specifies the System Attribute Table Name.

The `OData Service API` component in the `Product Services` folder holds the following APIs, provided out of the box:

- `createDataObject`
- `deleteDataObject`
- `readDataObject`
- `updateDataObject`
- `searchDataObject`

- `readAssociation`

The `Config` component in the `Configuration` folder holds the flag to set EDMX Generation. The `GenerateEDMX` key needs to have a value of `TRUE`.

OData Object Layer (Modifier Type - Data Definition)

The OData Object Layer is created by inheriting Data Definition Base and contains OData-specific local attributes for both the `Table` component and `Column` component, as described below.

We suggest that consumers of this feature develop and configure new data definitions in FS-PRO by inheriting from this template since it has the product model prerequisites configurations available ready to use.

The local attributes for the `Table` component are located in *OData Object Layer* > *Table* > *Attributes* > *Local Attributes*:

Attribute Name	Internal Name	Data Type	Length	Precision	Data List Source	Description
ODATA_ANNOTATIONS	ODATA_ANNOTATIONS	CLOB	0	0		Used to define whether the table is used as 1) Entity Type or 2) Complex Type by way of configured Data List. (data list configuration is mentioned in detail below). If the table isn't used for either, this attribute can be left empty.
ODATA_USAGE	ODATA_USAGE	DATA_LIST	250	0	TableOdataUsage	Used to define OData Annotations for this table to drive the UI behavior.

The local attributes for the `Table` component are located in the *OData Object Layer* > *Column* > *Attributes* > *Local Attributes*:

Attribute Name	Internal Name	Data Type	Length	Precision	Data List Source	Description
ODATA_ANNOTATIONS	ODATA_ANNOTATIONS	CLOB	0	0		Used to define OData Annotations for column level to drive UI behavior. SAP Annotations and FS-PRO Annotations are provided as standard. Customers may create their own namespace Annotations.
ODATA_USAGE	ODATA_USAGE	DATA_LIST	250	0	ColumnODataUsage	Used to define whether the column is used as 1) Property or 2) Navigation Property by way of configured Data List. (data list configuration is mentioned in detail below). If the column isn't used for either, this attribute can be left empty.

Attribute Name	Internal Name	Data Type	Length	Precision	Data List Source	Description
ODATA_ASSOCIATION	ODATA_ASSOCIATION	CLOB	0	0		Used to define the relationships between entities by way of navigation properties.

The acceptable values are `Dependent=<TABLE_NAME>`, `Multiplicity=<One>` or `Multiplicity=<Many>`, `PropertyRef=<COLUMN_NAME>` in the child table.

Data List Base

Data List Base was enhanced with OData-specific data lists, as described below:

Data List Base > *API* > *Data List Group* > *Odata Usage*

`Data List API`

DATA_LIST_SOURCE_NAME	DATA_LIST_SOURCE_RULE
TableOdataUsage	Product.queryProduct('//DLTableOdataUsage', 'Key,Value')
getAPIReturnTypeMultiplicity	Product.queryProduct('//DLOdataAPIReturnTypeMultiplicity', 'Key,Value')
getAPIHttpMethod	Product.queryProduct('//DLOdataAPIHttpMethod', 'Key,Value')
ColumnOdataUsage	Product.queryProduct('//DLColumnOdataUsage', 'Key,Value')
APIOdataUsage	Product.queryProduct('//DLAPIOdataUsage', 'Key,Value')

Five new domain tables were added to *Data List Base* > *Static Data*:

DLAPIOdataUsage	Key=1 Value=Function Import
DLColumnOdataUsage	Key=1 Value=Property Key=2 Value=NavigationProperty
DLTableOdataUsage	Key=1 Value=Entity Key=2 Value=ComplexType
DLgetAPIHttpMethod	Key=1 Value=GET Key=2 Value=POST
DLgetAPIReturnTypeMultiplicity	Key=1 Value=One Key=2 Value=Many

27.2 Entity Type Association Configuration

The `Product Runtime OData Service` supports configuration of relationships between different Entity Types modeled using data definitions. Configuring Entity Type relationships between various Entity Types enables expand and navigation OData queries to be performed in the service.

The `Product Runtime OData Service` currently supports **one-to-one** and **one-to-many** relationship configurations. These configurations are entered in the `ODATA_USAGE` and `ODATA_ASSOCIATION` attributes of the data definition `Column` component.

ODATA_USAGE
: Determines if the column is intended to be modeled as a OData `Property` or `Navigation Property`.

  The default value is `Property`.

  To configure relationship between Entity Types, this value must be set to `Navigation Property`.

ODATA_ASSOCIATION
: Specifies the details of the Entity Type relationship.

  The details are modeled as comma-delimited list of key-value pairs.

  The following keys are supported:

  Dependent
  : Specifies the name of Entity Type the principal (implicit) Entity Type has a relationship with.

  Multiplicity
  : Specifies the multiplicity of the dependent Entity Type.

    The supported values are either `One` or `Many`.

  PropertyRef
  : Specifies the name of the Property common to both the principal and dependent Entity Types to perform the necessary JOIN operation in the service back-end.

27.3 Data Object Access Stem

The Data Object Access Stem is used by the Product Runtime OData Service to access data in FS-PRO Runtime.

The Data Object Access Stem (`com.sap.fs.pro.stem.DataObjectAccessStem`) performs CRUD and Search operations for DataObjects using the "Native" DataObject implementation (`PersistableDataObjectService`).

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