Senior System Architect

7.2
Student Guide

Pega®
ACADEMY

CONTENTS

COURSE SUMMARY 365

DECISION DESIGN 369

COURSE INTRODUCTION

Before you begin

Senior System Architect 7.2 overview

The Senior System Architect course is an advanced course designed to help Pega 7 architects further their knowledge of Pega 7.

The lessons in this course focus on tasks a senior system architect performs to develop a Pega application.

This course is based on the Pega 7.2 platform. Pega recommends that students complete Pega Platform Fundamentals 7.2 and System Architect Essentials 7.2 prior to starting this course.

Objectives

After completing this course, you should be able to:

- Identify the tasks and responsibilities of the Senior System Architect on a Pega Implementation
- Create and extend a Pega application
- Manage rules and rulesets
- Leverage the Enterprise Class Structure (ECS) to promote rule reuse between case types and applications
- Configure roles, access groups and privileges
- Manage data access within an application
- Create wizards to configure a sequence of assignments
- Design applications for multiple device types and screen sizes, including mobile devices
- Manage integration settings
- Incorporate next-best-action decision-making into applications
- Test your application design to analyze rule behavior and identify configuration errors

Intended audience

This advanced course is designed to help System Architects further their knowledge of Pega 7 and improve their ability to implement Pega 7 solutions in an efficient manner. Upon completion of this course students should be ready to take the Certified Senior System Architect (CSSA) exam.

Prerequisites

To succeed in this course, students should have:

- Some familiarity with application development concepts
- Completed the System Architect Essentials 7.2 course

APPLICATION DESIGN

Creating a Pega application

Introduction to creating a Pega 7 application

Developing new applications from the ground up can be an overwhelming task for any organization and developer. Every application should adhere to an established set of organizational standards to simplify maintenance, customization, and reuse.

The New Application wizard enables you to create new applications in a few minutes, while ensuring you are following organizational best practices and standards.

After this lesson, you should be able to:

- Define the enterprise class structure (ECS)
- Differentiate between the implementation layer and framework layer
- Describe the purpose of the New Application wizard
- Create or extend an application with the New Application wizard

Enterprise Class Structure

An enterprise can have a complex organization structure with many locations. The enterprise may sell more than one product or service through multiple channels to different types of customers.

When you conduct business in different locations or countries, you need a way to manage the regulations of each jurisdiction, and the cultural differences in each region. When you sell multiple products through multiple channels, you need a way to manage the business rules for selling each product in each channel separately. Also, when you sell to different types of customers, you need a way to manage each customer's expectations and preferences.
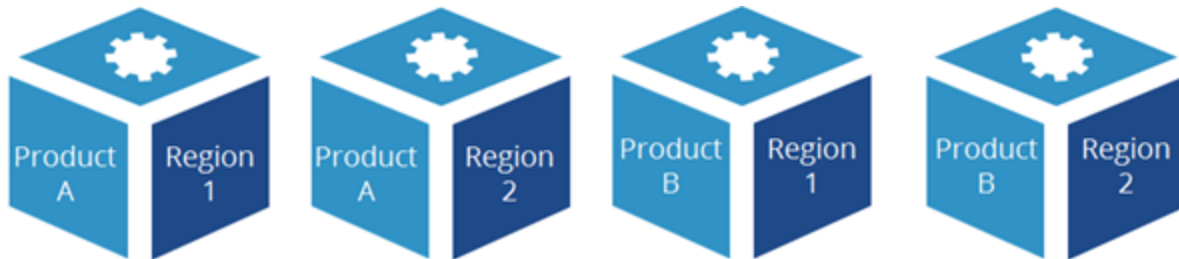
With some application development platforms, you must create separate copies of the application for each product, region, or channel. Or, you must create a single application that treats all business transactions the same, regardless of the business context. The result is enterprise applications that do not scale and are hard to maintain.

Pega's Situational Layer Cake allows you to organize your application using the same dimensions as your business. The Situational Layer Cake makes reusing common policies and procedures easy while allowing for differences between products, regions, channels, and customer segments.

The Situational Layer Cake is implemented in Pega 7 using a class hierarchy called an **Enterprise Class Structure (ECS)**.
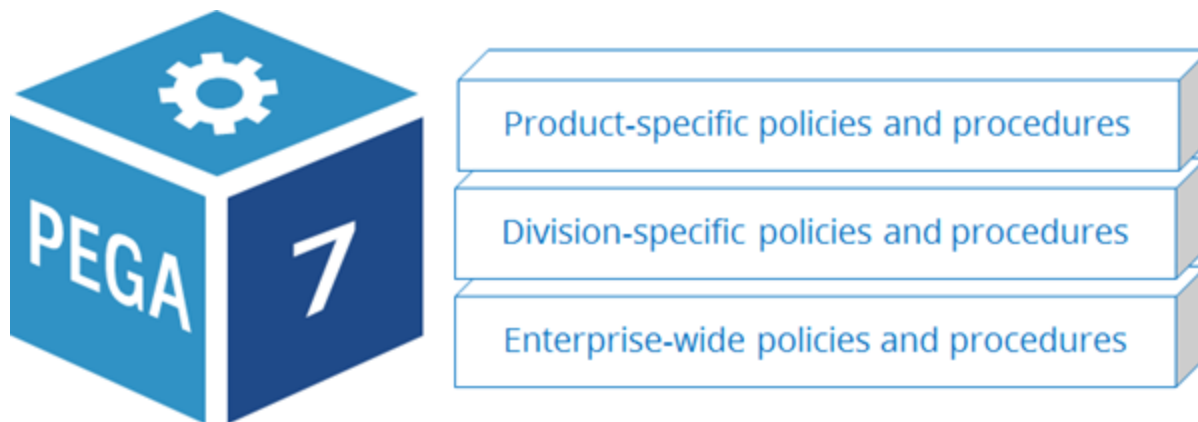
The ECS enables you to organize your application using the same dimensions as your business. The ECS makes reusing common policies and procedures easy while allowing for differences between divisions and products. When you place a rule in an ECS layer, it may be shared across multiple applications. As business operations change, existing layers of the ECS can be modified to address the changes.

Building an application on top of a well-designed class structure is vital for application scalability. A well-designed class structure also enforces best practices around reuse and standardization as the system expands to other lines of business.

State two advantages of organizing rules using an ECS.

Using an ECS helps you organize rules in a way that makes them scalable and encourages reuse.

Enterprise Class Structure layers

The ECS enables multiple Pega applications within the same system to coexist, and supports effective reuse among the applications.
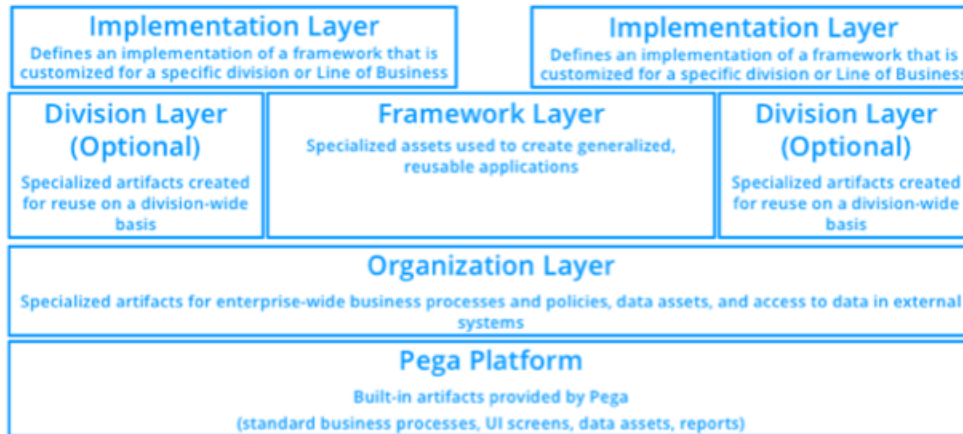
Note: For each release of Pega 7, the generated class structure reflects best practices available in that release.

Pega Platform layer

The **Pega 7** layer contains the built-in assets necessary for processing cases and other work in Pega applications. This layer also contains assets used by Pega 7.

Organization layer

The **Organization layer** contains the assets used on an enterprise-wide basis. Such assets are rules for enterprise-wide business logic such as standard properties, decision tables, and service level rules. For example, a property that holds a customer's account number would be reused on an enterprise-wide basis. As a result, the applications used across the enterprise can rely on the same account number property without having to make copies of the property in every application.

The Organization layer also contains enterprise-wide data assets such as classes and rules for data stored in the system, and classes and rules for access to data in external systems, via connectors. For example, access to an external customer database is an integration point that may be added to the Organization layer.

Division layer

The **Division layer** contains the assets used on a division-wide basis. The division layer is the middle level of the three-level organization hierarchy (Org-Div-FW) and is available for use in every application.

Assets in the Division layer may be applicable to a line of business, or for regions, brands, or departments. For example, a line of business wants to reuse a service level rule that defines the expected response time to a customer complaint in all of its applications. Placing the service level rule in the Division layer helps ensure all applications have access to the service level rule without having to create separate copies.

Note: The Division layer is an optional layer. When used, the Division layer is parallel to the Framework layer.

Framework layer

The **Framework layer** contains the assets used to create generalized, reusable applications. A framework application is then extended by specific implementations.

For example, a finance company provides auto loans. An auto loan framework is created that contains all of the assets needed for the standard auto loan process. Each line of business may extend the basic auto loan application to meet specific business needs. For example, the commercial business line division's auto loan application needs to handle loan requests that are distinct from that of the personal line division.

Implementation layer

The **Implementation layer** defines an implementation of a framework that is customized for a specific division, or line of business. For example, the commercial business line's auto loan application reuses assets from the commercial business line division layer and from the auto loan framework layer. The personal line's auto loan application reuses assets from the personal line division layer and the auto loan framework layer.

Note: While the assets in the Framework layer are designed for maximum reuse, the assets in the Implementation layer are specific and cannot be reused elsewhere.

KNOWLEDGE CHECK

In which layer of the ECS do you organize assets used to create generalized applications?
The Framework layer

KNOWLEDGE CHECK

When used, the Division layer is parallel to the _____ layer.
Framework

How to use the New Application wizard to create an application

Use the **New Application** wizard to build applications. The New Application wizard guides you through the basic configuration for your application. You use advanced options in the wizard to customize the settings to match your design requirements. Pega 7 generates the application and the application structure.

Start the New Application wizard

In the Designer Studio header, click **Application** > **New Application** to start the New Application wizard.

Select the type of application you want to create. Choose **Enterprise application** to create an application with a reusable class structure, case types, data types, and organizational rules. Choose **Express application** if you do not require a reusable class structure, or you are inviting another user to explore the Pega Express environment.

Click **Create new application** to start creating an application.

Specify basic application settings

In the **Application name** field, enter a unique name.

Note: The application name is limited to 49 characters. The application name must start with a letter and can only contain alphanumeric characters, ampersands, underscores, or hyphens. The application name is displayed in portals, menus, and forms.

Note: The advanced settings are the domain of an LSA. Focus on building an app at a time.

If the application name you enter exceeds 11 characters, the New Application wizard truncates the application name to an *application short name* of no more than 8 characters.

The *application short name* serves as a unique identifier the system uses for processing, and for naming the application's classes. To change the application short name, click the short name to open the *Application short name* dialog.

Note: The application short name is limited to 14 characters.

The **Built on application** option specifies the application you want to extend.

The **Application structure** option indicates how many class layers are created in your application.

Important: As a best practice, always use the **Implementation only** option. This option helps the project team focus on building one application at a time to help ensure project success.

Organization specifies the organization associated with the application. The organization value uniquely identifies organization records, classes, and the organization ruleset.

Note: The organization name is limited to 32 characters.

Organization ★
SHARKS
The organization short name was truncated to SHAR

If the organization name exceeds four characters, the New Application wizard truncates the name to create a short name consisting of 4 characters. To change the organization short name, click the short name to open the *Organization short name* dialog box.

Note: The Organization short name is limited to a maximum of 6 characters.

Specify business objectives

Business objectives define the expected business outcomes for the new application. Business architects work with the business stakeholders to define the business objectives.

To add business objectives to the application, click **Add business objective** to add a row and enter a new business objective.

Note: When using the New Application wizard, adding business objectives is optional. After you create your application, you can add or update business objectives on the Application Overview landing page.

Specify case types

Case types represent work in the application that follows a life cycle, or path, to completion. Cases traditionally have a life cycle, and through a set of actions, accomplish a meaningful business outcome.

To add case types to the application, click **Add case type** to add a row and enter a case type.

Note: When using the New Application wizard, adding case types is optional. After you create your application, you can add case types using the Case Designer.

Specify data types

Data types represent information that is necessary for a case to accomplish its actions or achieve its business outcome. This data is often shared across multiple cases and possibly applications, and could be stored local to this application or in an external system.

To add a data type to the application, click **Add data type** to add a row and enter a data type.

Note: When using the New Application wizard, adding data types is optional. After you create your application, you can add data types using Designer Studio.

Preview the application

To preview the class structure the New Application wizard creates, click **Preview**.

Name	Inherits from	Ruleset
▸ SHARKS Organization reuse layer		
▸ Loans Implementation layer		

The Application preview dialog box displays the names of the class layers that will be created for the new application.

▼ Other
- ▸ Access Group
- ▸ Access Role Name
- ▸ Operator ID
- ▸ Organization
- ▸ Work Group
- ▸ Workbasket

The Other section includes information such as access group and access role names that are created for the new application. To make changes to the application settings, close the preview dialog and return to the **Application settings** step.

Create the application

To create the application, click **Create**.
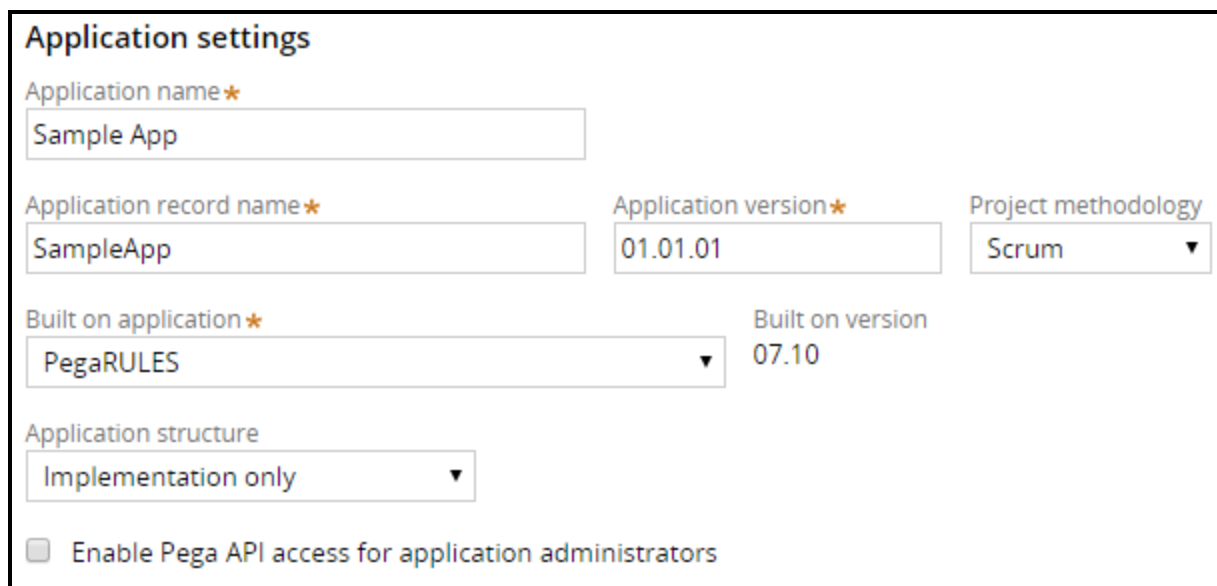
When the New Application wizard finishes creating the application, the application's Home page opens to the Application Overview landing page. You are now ready to develop your new application.

How to configure advanced settings in the New Application wizard

Use the advanced settings of the *New Application* wizard to customize your new application to match your design requirements.

Customize Application settings

The advanced Application settings provide access to details such as the application version and the project methodology. Some of the settings defined in the New Application wizard are also displayed in this dialog box. For example, the application short name is displayed in the *Application record name*.

The *Application record name* is also referred to as the application short name, and uniquely identifies application records across systems.

> **Note:** The system derives the application record name from the first 8 characters of the application name and is limited to 14 characters.

Follow your organization's naming conventions and best practices when customizing the application record name.
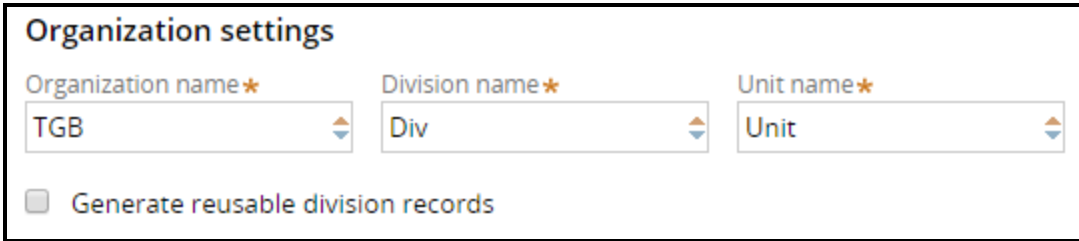
The *Application version* is set to 01.01.01 as the default version number. To customize the application version, enter a version number in the format [Major].[Minor].[Patch]. Follow your organization's application versioning conventions and best practices when customizing the application version.

The *Project methodology* is set to Scrum as the default. If your organization uses a different project methodology you can change this option to **PegaBPM** or **Other**. The project methodology can also be edited in the Application overview landing page.

Select the *Enable Pega API access for application administrators* option to add the *PegaRULES:PegaAPI* role to the administrators access group. This role enables you to power client and mobile applications using built-in Pega REST services. It also includes case, assignment, and data APIs that let you leverage your Pega 7 Platform Applications.

Customize Organization settings

The advanced Organization settings provide access to organizational details such as division and unit names.

The Organization name is a unique name used to create organization, division, and unit rules, and prefix class names within the organization layer. Use your organization's short name, ticket symbol, or internet domain name; for example: Pega, PEGA, or pega.com.

The **Division name** is a unique name for a division, or department within the organization. The division name is limited to 32 characters.

The *Unit name* is a unique name for a unit within a division. The unit name is limited to 32 characters.

Note: You can only define one new division and unit, or select an existing division and unit when using the New Application wizard. Use the *Organization and Security* landing page to add additional divisions and units to the application.

Select the *Generate reusable division records* to extend the organization class layer by creating a division class layer to the enterprise class structure.

Customize Class structure - Implementation layer settings

The fields in this section are visible when you select an application structure that includes an implementation layer.

The *Division layer* is used to create a class for the division. Use a name that clearly describes the name of the division within the organization.

The *Unit layer* is used to create a class for the unit. Use a name that clearly describes the name of the unit within the division.

Note: The division layer and unit layer fields are only available when you select the **Generate reusable division records** and the **Generate reusable unit records** options in the Organization settings section.

The *Application layer* is used to create an implementation class and ruleset. The application layer corresponds to the Implementation layer in the enterprise class structure.

Important: Consider using 4 to 6 characters for layer names to allow for more descriptive names when defining case and data types.

The **Class group name** is used to create a work pool. A **work pool** is a set of allowed cases a user can work on within an application. The standard naming convention is to use **Work** as the class group name. The context of the work pool is derived from the application name. For example: *TGB-Loans-Work* and *TGB-Insurance-Work*.

Creating a new application version

Introduction to creating a new application version

Pega provides the ability to preserve a version of an application. You can edit application functionality in a new version of the application without changing the initial version.

After this lesson, you should be able to:

- Identify the application components which versioning impacts
- Explain the role of versioning in application development
- Explain the purpose of ruleset skimming
- Create a new application version

Application versioning

Pega provides two methods for creating new versions of an application. Each method preserves prior application versions. **Application versioning** is a way to differentiate current and past application configurations. Rule resolution can look through all the minor and patch versions for the current major ruleset.

Application components include the application ruleset stack — this contains the rules and data types used by the application. To version an application, you must version the application's rulesets.

The versioning methods are **lock and roll** and **skimming**. The act of using a version method begins a release cycle. Every major version, minor version, and patch version represents a release cycle. Both methods list the highest version, and offers to roll the ruleset to a still-higher version by default.

The person performing the lock and roll or skim must understand the application structure. A senior system architect (SSA) or lead system architect (LSA) is able to understand application structures.

Your choice of method depends on the type of application change. Lock and roll is best for incrementing patch versions. Skimming is better for minor and major versions.

Lock and roll

Small changes or patches are ideal for lock and roll. Application patches and minor updates usually involve updating rules. When using lock and roll, you create a new empty ruleset version. To update the configuration, you copy the necessary rules into the new ruleset version.

The rule in the higher ruleset version overrides the rule in the lower version. You specify the new version number and whether to update the application record and access groups to reflect the ruleset version.

Note: Minor and major versions require application record and access group updates. Patches usually do not need the updates.

If you roll 01-01-01 to 02-03-01, Pega will start at 02-03-01 and look back to the previous minor version, 02-01-01, to find rules. As long as the rule is present in one of the versions, Pega can find it. If a rule is only in 01-03-05, rule resolution will not find it because it is in a different major version.

This graphic illustrates how a system architect would version an application. The lock and roll wizard creates an empty ruleset. The system architect adds the appropriate rules to configure the new version.

Original 01-01-01	First Patch 01-01-02	Second Patch 01-01-03	Third Patch 01-01-04
Rule A Rule B Rule C	Rule A Updated	Rule B Updated Rule C Updated	Rule B Updated
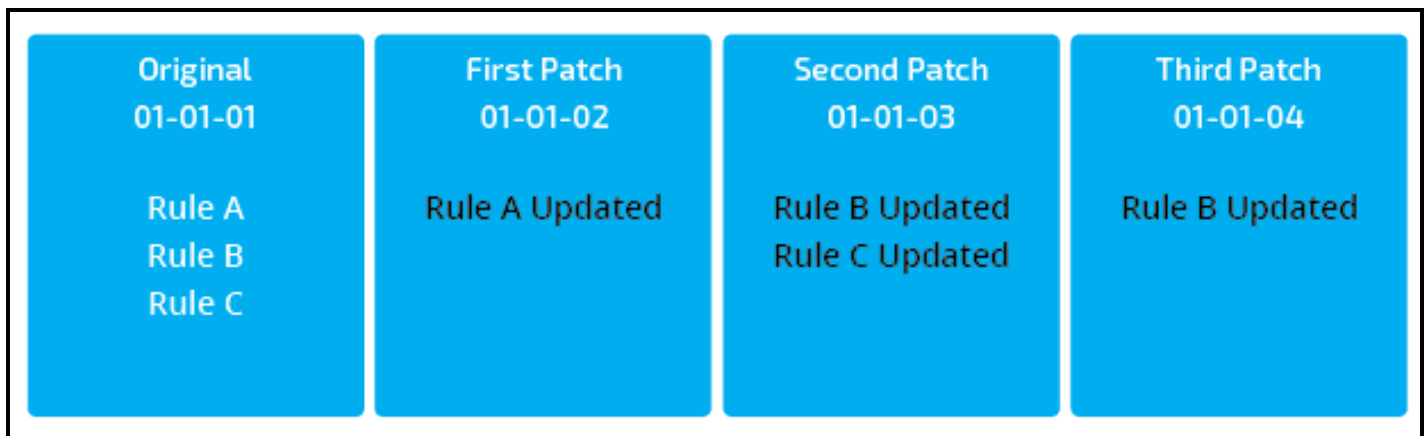
In the previous example, an SSA ran lock and roll to create an empty 01-01-02 version. Then, a system architect updated rule A in this version. When a user runs the application, they use the updated rule A and rules B and C from the 01-01-01 ruleset.

Then, an SSA ran lock and roll to create the 01-01-03 version. A system architect copied rules B and C to this version to update them. Now, when a user runs the application, they use the rule B and C from the 01-01-03 version and rule A from the 01-01-02 version. The copies of A, B, and C in 01-01-01 are all overridden.

Finally, the SSA ran lock and roll to create 01-01-04. A system architect copied rule B to this version to update that rule again. So, when a user runs the application now, they use rule B from the 01-01-04 version, rule C from the 01-01-03 version, and rule A from the 01-01-02 version. The copies of A, B, and C in 01-01-01 are all overridden.

KNOWLEDGE CHECK

ANSWER	In lock and roll, what occurs after you use the wizard to create a new, empty ruleset?
A system architect copies the required rules into the empty ruleset to create the new application version.	

Skimming

Skimming is the process of saving the highest version of a rule into a new, higher ruleset version. Skimming applies mainly to resolved rules. Skimming is useful when rule changes follow a logical sequence. The two types of skims are minor and major. The skim types correspond with the update types (major or minor/patch).

During a minor skim, rules are stored in a higher minor version, and during a major skim, rules are stored in a higher major version.

A rule's availability status determines if the rule is carried forward. This table defines the rules that are carried forward during a skim.

Rule Availability Status	Carried forward?
Yes	Yes
Blocked	Yes
Final	Yes
Withdrawn	No - major update; Yes - minor update
No (not available)	No

Blocked rules are carried forward because a blocked rule can block rules in other rulesets. You should maintain blocking relationships.

The key to skimming is you can start at a major version and skim all minor and patch numbers into a new version, or you can start at some minor version and work up from there.

Original 01-01-01 Rule A Rule B Rule C	First Revision 01-01-02 Rule B Updated	Minor Skim 01-02-01 Saved: Original 01-01-01 Rules A & C First Revision 01-01-02 Rule B
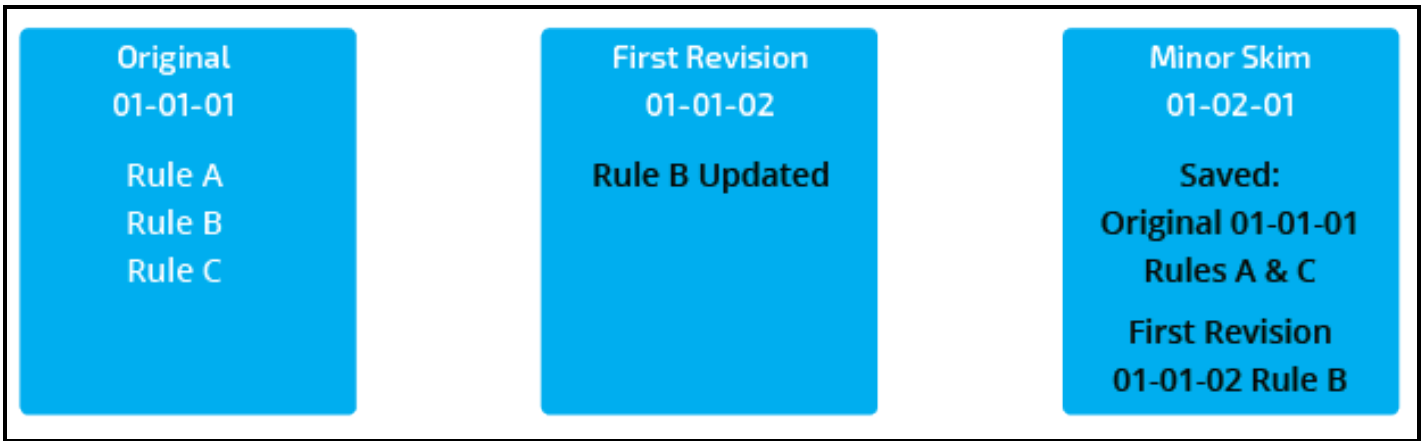
Pega provides a skimming wizard. For each rule instance in a specified ruleset, the wizard identifies the highest-numbered version and creates a copy with the number you specify.

For more information on the ruleset wizard, view the Help topic About the Ruleset Maintenance wizard.

KNOWLEDGE CHECK

Why would you use skimming instead of lock and roll when versioning an application?
Skimming is the process of saving the highest version of a rule into a new, higher ruleset version. Skimming streamlines application versions where rule changes follow a logical sequence.

Note: In the exercise, you version a patch update using lock and roll and create a new version of the application.

How to create a new application version

You determine the method to use to create a new application version. Your choice is based on the type of application change. Small bug fixes and incremental application enhancement patches are ideal for lock and roll. Skimming streamlines applications versions where rule changes follow a logical progression.

Preprocess best practice recommends confirming the rules for the new version are checked in. You can run a search for checked out rules from the Checked Out Rules page. An additional best practice is locking all but the highest ruleset versions.

Lock and roll

Within lock and roll, you have three choices for updating the application rule: no change, edit the current version, and create a new version.

You use **Do not update my application** when you update the patch version number of a ruleset without updating the application ruleset list. By default, the application rule only lists the major and minor version numbers for a ruleset, so incrementing the patch version number does not require a change to the application rule.

You use **Update my Application to include the new Ruleset Versions** when you are rolling out an application and updating the minor version or when the application rule lists the ruleset patch version number. You may enter a new application description. The default application description is current. If current application is locked, enter the application password.

You use **Create a new version of my application** when you want to lock and roll the version and create a new application rule. You may enter a new application version, if different than the default one increment higher. You may enter a new application description. The default application description is current. If current application is locked, enter the application password.

You can also use **Create a new version of my application** to allow people access to more than one version of the application (for example, during a phased roll-out or a test period).

You must select the appropriate ruleset versions for the lock and roll before proceeding. Most selections will be the most recent version. However, an earlier version of a ruleset might be appropriate. Application requirements dictate this decision.

You can view the rulesets in the current application version on the Ruleset Stack page. You can select the appropriate ruleset versions, enter the ruleset passwords, and select the update option in the Lock & Roll window.

KNOWLEDGE CHECK

When using lock and roll, what is the purpose of selecting the new version option?

You use create a new version when you want to lock and roll the version and create a new application rule. You can also create a new application version when you want to have people access more than one version of the application (for example, a phased roll-out or a test period).

Skimming

In Designer Studio, navigate to the Refactor > RuleSets page to access the link to the Skim a RuleSet page. Indicate whether the update is to be a major version (NN-01-01) or a minor version (NN-MM-01), the ruleset(s) to skim, and the version number to be created. Click **Skim** to begin the process.

The system creates a new ruleset version and begins copying rules. A status area shows progress and the results of the skim. The actual duration of the skim could be several minutes.

Tip: If errors occur, select the **Total Number of Errors** link in the lower right corner of the display form to view error messages. The error list is difficult to access after the results form closes. Print the list if you wish to research and address the errors after closing the form.

You must update application rules, the Requires RuleSets and Versions prerequisites array in RuleSet version rules, and access groups to reference the new major version after the skim completes. Log out and log in to access the new version.

Notes: Skimming only copies the rules in the major version you select. For example, if you skim 02-ZZ-ZZ into 03-01-01, rules in version 01-ZZ-ZZ are ignored.

You must have the zipMoveSkim privilege to perform the skim. Pega provides a default role for system architects which includes zipMoveSkim. SysAdm4 is the default system role for system architects. SysAdm4 includes the zipMoveSkim privilege. When an application is in production, the SysAdm4 role becomes the Administrator role.

For more information on skimming, view the Help topic <u>Skim to create a higher version</u>.

KNOWLEDGE CHECK

ANSWER	What tasks must you perform after the skim is complete?
Update the application rules, the Requires RuleSets and Versions prerequisites array in RuleSet version rules, and the access groups to reference the new major or minor version.	

Configuring application rulesets

Introduction to configuring application rulesets

In this lesson you learn how to configure an application ruleset to package for distribution. You also learn about the impact the configuration has during development and runtime.

After this lesson, you should be able to:

- Describe the purpose of a ruleset
- Describe the organization of rulesets in an application
- Compare the ruleset validation modes
- Identify how the ruleset list is built
- Add and remove application rulesets

Rulesets

Rules are the building blocks of a Pega application. The rule type determines the type of behavior modeled by the rule. In Pega, every instance of every rule type belongs to a **ruleset**. A ruleset is a container or an organizational construct used to identify, store, and manage a set of rules. The primary function of a ruleset is to group rules together for distribution. Understanding the relationship between an application and rulesets is essential in order to understand development and run-time behavior.

Each ruleset has versions. See the Help topic About ruleset versions for a primer on ruleset versioning.

Application rulesets

An application contains a set of rulesets. The New Application wizard creates the initial application rulesets. When the application is generated, the created rulesets include two rulesets for the application itself and two organizational rulesets. In the following example, *HRApps* and *HRAppsInt* contain application configuration. The two organizational rulesets — in this example, *TGB* and *TGBInt* — contain reusable organizational assets, such as data structures.

> **Note:** The rulesets ending in *Int* are used for rules related to integration.

Add additional rulesets for reusable functionality that you might want to migrate to other applications. For example, the integration wizards create separate rulesets for each integration. The *CreditCheck* ruleset holds the integration assets for a connector used to perform a background check.

Production rulesets

Production rulesets have at least one unlocked ruleset version in the production environment. Production rulesets include rules that are updated in the production environment. The most common use of production rulesets is for delegated rules. However, production rulesets can be used for any use case requiring rules to be updated in a production environment.

Production rulesets are configured in the Advanced tab on the application record. In addition, the production ruleset needs to be specified in the access group.
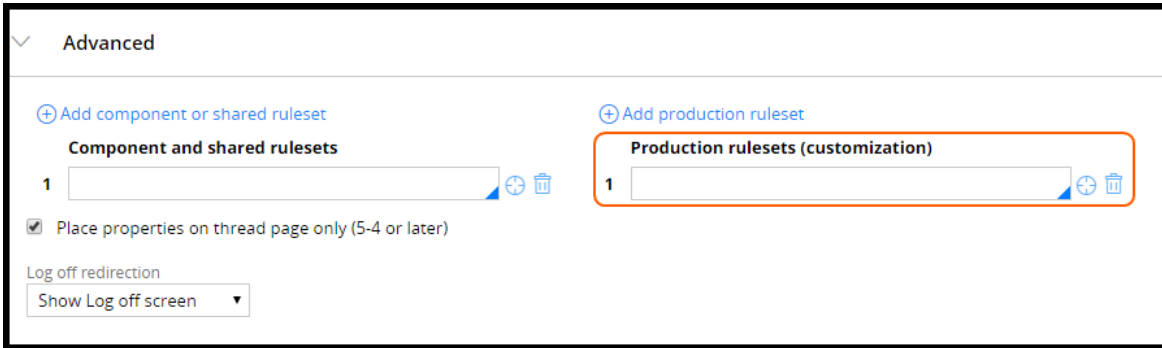
KNOWLEDGE CHECK

The default initial application created by the New Application wizard contains a set of ruleset for the application code and _____ assets.

organizational

Ruleset validation

Ruleset validation is performed every time a rule is saved. It guarantees that referenced rules are available on the target system when the ruleset is promoted.

Ruleset validation does not affect rule resolution at run time, but is only applied at design time.

There are two options for the validation mode:

- **Application Validation**
- **Ruleset Validation**

The selected validation mode applies to all versions of the ruleset.

The New Application wizard creates rulesets that are set to both Application Validation (AV) and Ruleset Validation (RV) modes. The rulesets containing the application rules are AV mode to reduce the difference between design and run time.

Conversely, the organizational rulesets created by the New Application wizard are RV mode. RV ensures strict validation on prerequisite rulesets when migrated.

Application validation mode

If the AV mode is used, rules in the ruleset can reference all rules in the rulesets defined in the:

- Same application
- Rulesets belonging to any built-on application

Rules in the ruleset cannot reference rules outside the current application stack or above the defining application.

In the following example, all rulesets are configured as AV. The shaded and unshaded cells represent applications. The loan application is built on a loan framework.
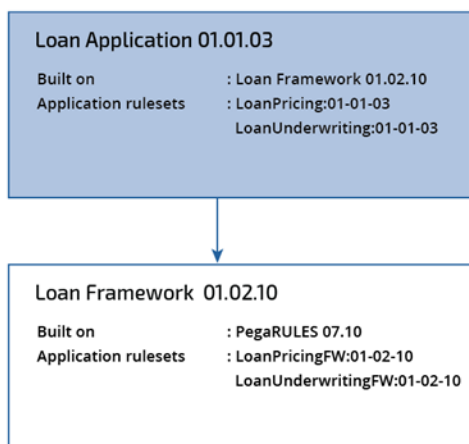
Ruleset	Mode
LoanPricing	Can call all other rulesets listed
LoanUnderwriting	Can call all other rulesets listed
LoanPricingFW	Can call LoanUnderwritingFW but not shaded rulesets
LoanUnderwritingFW	Can call LoanPricingFW but not shaded rulesets

AV allows for codependent rulesets within the same application. That is, rules in *LoanPricing* can reference rules in *LoanUnderwriting*, and rules in *LoanUnderwriting* can reference rules in *LoanPricing*.

Each ruleset in the application has a version. When AV mode is used, the application defines the ruleset versions accessible for a given ruleset. For example, ruleset *LoanPricing:01-01-03* can access ruleset version 01-01-01 to 01-01-03 of the ruleset *LoanUnderwriting* and 01-01-01 to 01-02-10 of the *LoanPricingFW* and *LoanUnderwritingFW* rulesets.

When AV mode is selected, if you change the application definition, the rules may become invalid. Invalid rules might cause serious errors at run time. Use the Validation tool (**DesignerStudio > Application >Tools > Validation**) to quickly identify invalid rules in the application.

Ruleset validation mode

When you use RV mode, each ruleset version defines one or more ruleset versions on which the ruleset version depends. For example, if you create a ruleset *Myco:01-01-01* that uses rules in *MyCoInt:01-01-01* and *Customer:01-01-01*, then *MyCoInt:01-01-01* and *Customer:01-01-01* ruleset versions must be specified as a prerequisite. Only rules in the ruleset versions that are specified as prerequisites (and their prerequisites) can be referenced from the ruleset.

For example, if *MyCo:01-01-01* specifies *Customer:01-01-03* as a prerequisite, rules in version 01-01-01 to 01-01-03 of the customer ruleset can be referenced.
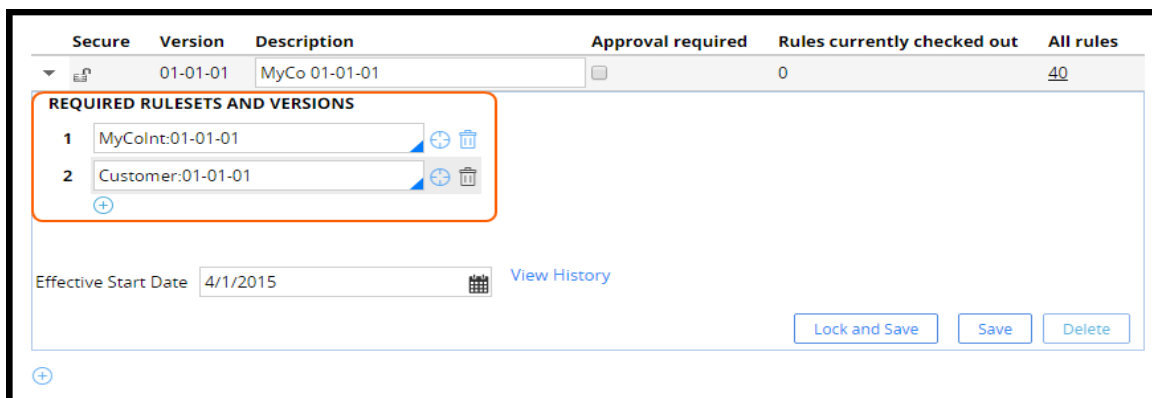
Ruleset prerequisites

If your ruleset version does not have any prerequisite ruleset versions, you need to specify the base product ruleset *Pega-ProcessCommander* as a prerequisite.

The *Pega-ProcessCommander* ruleset lists all product rulesets. You do not need to list any product rulesets below *Pega-ProcessCommander*. There is a 99 patch version of the *Pega-ProcessCommander* ruleset available in the product. Use that ruleset version as a prerequisite to avoid having to update the ruleset after product updates. For example, if the product is updated from 07-10-13 to 07-10-18, you do not need to update the rule prerequisites since the 99 version automatically picks the highest patch for the ruleset version.

Ruleset prerequisites cannot be cyclic. For example, if *Alpha:01-01-01* defines *Beta:01-01-01* as a prerequisite, then *Beta:01-01-01* cannot define *Alpha:01-01-01* as a prerequisite.

Mixing application ruleset validation modes

You can mix rulesets that use AV and RV.

- Rulesets with another ruleset in brackets — for example, MyCoPL [MyCo] — next to them use RV. The ruleset in brackets is the prerequisite ruleset.

- Rulesets without a ruleset with brackets next to them use AV.

With RV, you cannot call AV rulesets that are not in the prerequisites.

Loan Application

Built on	: Loan Framework
Application rulesets	: LoanPricing
	LoanUnderwriting
	MyCoPL [MyCo]

Loan Framework

Built on	: PegaRULES
Application rulesets	: LoanPricingFW
	LoanUnderwritingFW
	MyCoPL [PRPC]
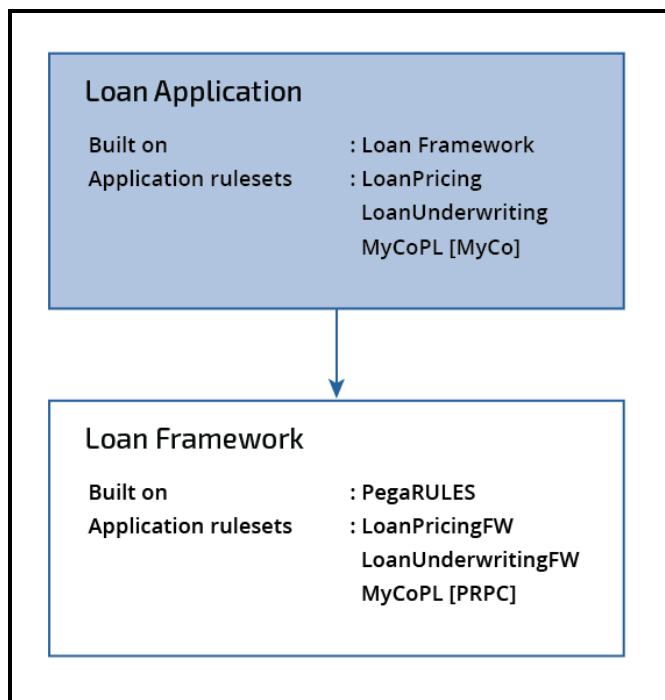
Name	Mode
LoanPricing	Can call all other rulesets listed
LoanUnderwriting	Can call rules in all other rulesets including LoanPricing
MyCoPL [MyCo]	Can call rules in MyCo but not in LoanUnderwriting, LoanPricing, LoanPricingFW, or LoanUnderwritingFW
LoanPricingFW	Cannot call rules in MyCoPL, LoanUnderwriting, or LoanPricing
LoanUnderwritingFW	Cannot call rules in MyCoPL, LoanUnderwriting, or LoanPricing
MyCo[PRPC]	Cannot call rules in MyCoPL, LoanUnderwriting, or LoanPricing

Ruleset validation best practices

Follow these best practices when configuring rulesets.

- Only use RV for rulesets that are designed to be used across multiple applications, such as organizational rulesets, to make them easily portable and prevent introduction of dependencies on a specific application.

- Create applications for common rulesets; use the built-on functionality to include common rulesets in the application.

- Include unlocked AV rulesets in one application only. This prevents AV rulesets from referring to rules that may not exist in applications that do not contain the ruleset.

- Run the Validation tool after critical changes/milestones have been implemented (for example, changes to the application ruleset list or built-on application as well as changes made before

lock/export).

Note: For more information about the Validation tool, see the Help topic About the Validation tool.

KNOWLEDGE CHECK

⌄ ANSWER	Which ruleset validation mode allows for codependent rulesets?
Application validation (AV)	

The ruleset list

While ruleset validation governs rule development and import, the **ruleset list**, sometimes referred to as the ruleset stack, governs rule execution at run time. Because ruleset validation is enforced during development, the system does not need to validate the rulesets during run-time processing.

The ruleset listindicates the rulesets that are available to the application for a given operator session. The ruleset list is available in the operator profile **Operator > Profile**.

Note: The order of the rulesets is important. The rule resolution algorithm refers to the order of the ruleset in the ruleset list. Rulesets at the top of the list have higher precedence.

The ruleset listis assembled by Pega when an operator logs in to the application. The process begins by locating the versioned application rule referenced on the access group of the operator.

Note: In rare configurations, the access group is provided as part of the requestor definition, organization, or division record.

The ruleset list consists of rulesets referenced on the application form. The built-on applications are processed repeatedly until the PegaRULES application is located.

The ruleset list is built by stepping through the built-on applications until the PegaRULES application is located. First the PegaRULES ruleset list is added to the ruleset list. Then the built-on applications are processed recursively adding each application's ruleset to the ruleset list on top of the previously added rulesets.

For example, the *PegaRULES* application is at the base, the *HRFW* application is built on top of *PegaRULES*, and the *HRApps* application is built on top of *HRFW*.

If you are allowed to check out rules, a **personal ruleset** is added to the top of the list. The personal ruleset has the name of the operator ID and contains the rules checked out by the operator.
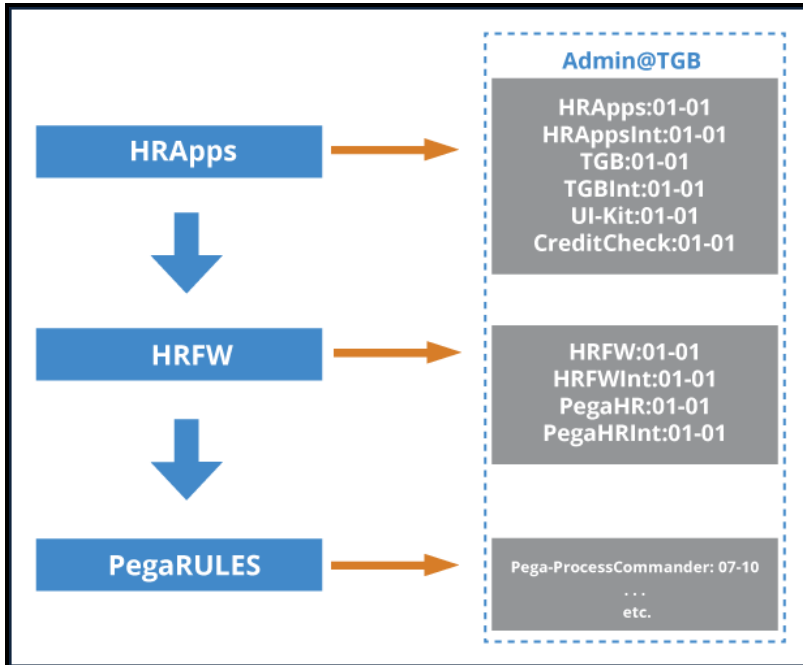
The ruleset list for the application with built-on applications can be viewed on the ruleset list landing page (**Designer Studio > Application > Structure > RuleSet Stack**).

KNOWLEDGE CHECK

The ruleset list indicates the rulesets that are available to the application for _____.

a given operator session

How to manage changes to rules in a ruleset

Rules are the source code of an application. During development, changes to rules need to be managed to ensure that:

- Rules belonging to a release that has been promoted are not updated
- Rules are configured in a coordinated manner to avoid rules being updated simultaneously

Ruleset locking

You can lock a ruleset to prevent changes using the **Lock and Save** button. Typically, you lock rulesets when development has reached a specific state and the application is ready to be promoted to testing. You cannot add or update rules in a locked ruleset.

Secure	Version	Description	Approval required	Rules currently checked out	All rules
▼ 🔒	01-01-01	Records for CreditCheck	☐	0	12

IN APPLICATION VALIDATION MODE 'REQUIRED RULESETS' ARE DISABLED

Effective Start Date 9/27/2016 📅   View History

Lock and Save   Save   Delete

⊕

Rule check-out and check-in

Applications are typically developed by a team. Multiple team members may check out rules to work on the same application in a coordinated way. The check-out feature ensures that one rule is not being edited by different team members at the same time.

The system enforces use of check-out and check-in operations when a ruleset has the check-out facility turned on. Before you can change that rule, you must perform either the standard check-out or private check-out operation.

Check out a rule

When you check out a rule, you are creating a private copy of the rule that you can modify and test.

On the Security tab on the ruleset, select **Use check-out?** to enable check-out.

On the Operator record Security tab, operators need to have the **Allow Rule Check out** selected in order to update rules in rulesets that require check-out.

The check-out button displays on rules that are in unlocked rulesets. If a developer checks out a rule, no one else may check the rule out until it is checked back in by the developer.

If a rule is not available for check-out, it is already checked out by someone else, or in a locked ruleset version.

When the ruleset is in a locked ruleset version, the private edit button is displayed instead of the check-out button. Private edit is a special case of the standard check-out that allows a developer to prototype or test changes to a rule that is not available for standard check-out. When an operator checks out or selects private edit for a rule, a copy of the rule is placed in the personal ruleset.

We can view our checkouts and private edits in the Private Explorer or by using the check mark icon in the header.
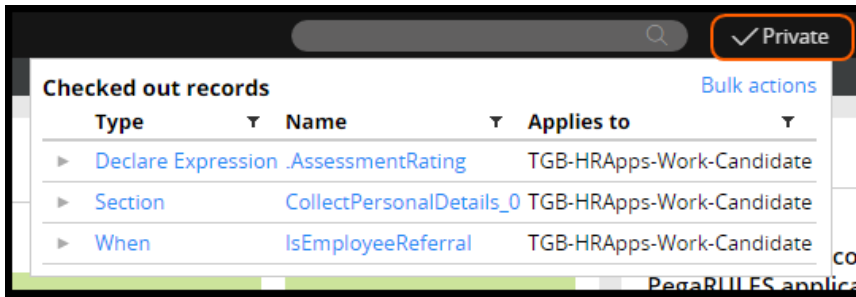
Checking in a rule

When a rule is checked in, the checked-in rule replaces the original base rule. Add a comment describing the changes to the rule. The check-in comments can be viewed on the rule History tab.

Use the bulk action feature to check-in, open, or delete several checked out rules at the same time. The bulk action feature is located in the Private Explorer menu or under the check mark icon in the header.
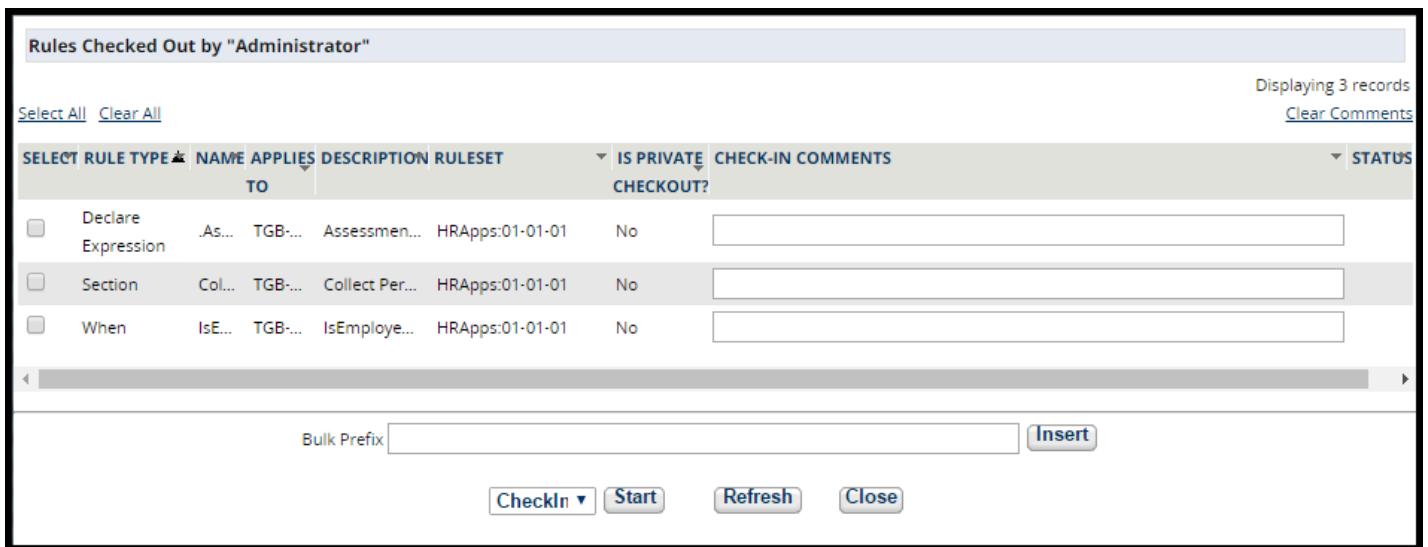
KNOWLEDGE CHECK

List the two reasons a rule is not available for checkout.

The rule is already checked out by someone else. The rule is in a locked ruleset

Branching rulesets for parallel development

Introduction to branching rulesets for parallel development

In this lesson, you learn how to use branches to enable parallel development work to take place within an isolated ruleset without impacting other development teams.

After this lesson, you should be able to:

- Explain the role of branched rulesets in development
- List the pros and cons of using branches
- Develop in parallel using branches
- Create a development branch
- Merge the contents of a branch into an application

Parallel development

When multiple teams are working in the same application rulesets, coordinating changes across teams is a challenge. When development teams configure an application in parallel, rule changes might result in conflicts. Resolving conflicts disrupts the overall project and may delay time to delivery.

Pega 7 uses branch rulesets to help teams manage parallel development in distributed environments. A **branch** is a container for rulesets with records that are undergoing rapid change and development. The rulesets associated with a branch are called **branch rulesets**.

A branch ruleset:

- Is based on (branched from) another ruleset
- Contains rules that are in active development in the associated branch

In Pega, you create a branch ruleset for each team. These branches allow each team to create and update rules without impacting other teams.

The following scenario describes how branches allow multiple teams to make rule update changes in parallel.

Team Alpha and Team Beta are developing an HR Onboarding application. Team Alpha is assigned to develop a new UI feature. Team Beta is assigned to work on candidate profile information. While each team works independently, both features involve changes to rules in the same rulesets: HR, HRApps, and HRAppsInt.
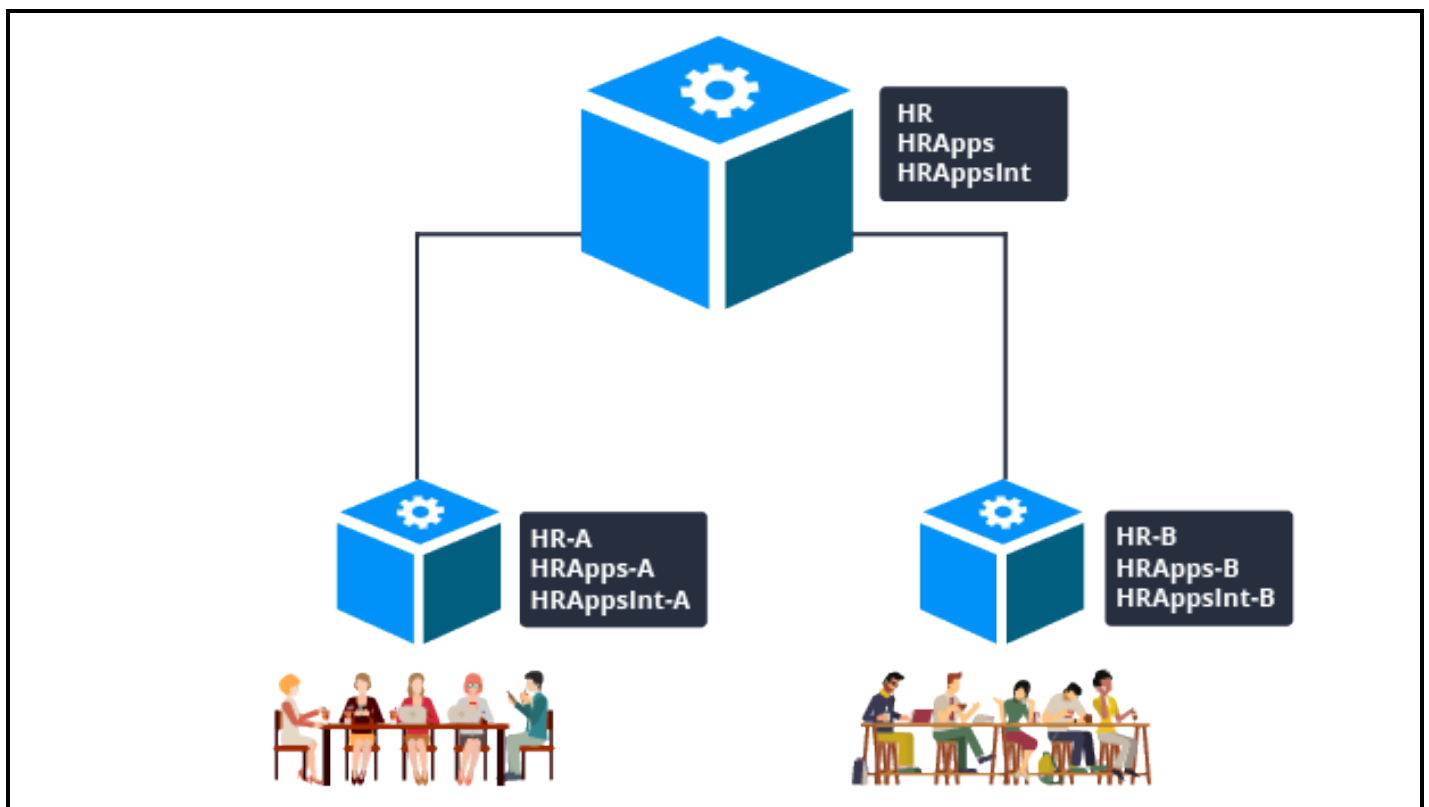
Using branches and branch rulesets, each team creates its own development branch. When the rule updates are complete, each team resolves conflicts the system detects between the branch rules and

©2017 Pegasystems Inc.

other instances of the rules. After the team resolves rule conflicts, the system merges the updated branch rules into the original application.

Branch rulesets allow each team to work within an isolated space (the branch) without affecting functionality other teams are developing. They are especially useful on large development projects, where multiple teams work simultaneously on the rules in an application. Each team works on their changes independently. All members of a team can see each other's work, while isolating the development changes from other teams. Changes do not affect other teams until the changes are stable, conflicts are resolved, and approval is granted to make the changes available to all development teams.

KNOWLEDGE CHECK

ANSWER	Describe two advantages of branching rulesets.
Two or more development teams can configure an application in parallel. Rule development takes place within an isolated space (the branch) without affecting functionality in the source rules.	

How to develop in parallel by branching rulesets

In this lesson, you learn about an approach Pega development teams follow to develop features and rules in parallel. To manage the independent development of features that share rulesets, each team creates its own branch and branch rulesets.

For example, two development teams are building features for a new application. Changes made by each team impact the same application rulesets. To develop rules in parallel using branched rulesets, each team follows these steps:

1. Creates a team application built on the main application

2. Creates one or more development branches in the team application

3. Creates branch rulesets for each branch listed in the application rule

4. Grant access to the team application

5. Creates or configures rules using the branch rulesets

6. Merges each branch into the application rulesets when development in branches is complete and stable

When the work is done, the team resolves conflicts the system detects between its branch and the application. After resolving any conflicts, the team merges the contents of the branch into the main application rulesets.

Create a team application

First, Team A and Team B each create a development **team application** built on the main application. Team application names typically reflect the base application, team name, or the focus of the team. For example, for a base application named **NewHireOnApp**, the development team working on the layout of the user portal might name the team application **OnBCoolPortalFeature**.

Use the New Application wizard to create a team application. From the **Built on application(s)** list, enter a **Name** and **Version** for the team application.

Applications are associated with rulesets. The Application rulesets for the team application are drawn from the main application. To learn more about how to associate rulesets with a team application, refer to the Using branches for parallel development online tutorial.

The following image displays a team application rule with branched rulesets.
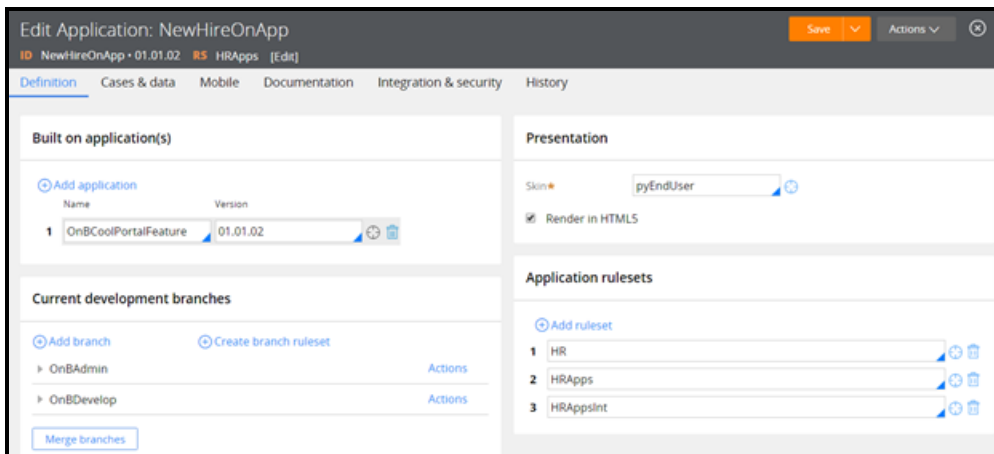
KNOWLEDGE CHECK

> (ANSWER) Why do you create a team application?
>
> You create a team application to allow your team to manage the branches they use without impacting other teams.

Create branches in the team application

After you create the team application, you define branches in the application and add them to the application record for the team application. Specify a unique name for the branch that identifies the feature being configured. Pega limits the name of each branch to 16 characters.

> **Important:** Branch names are visible across all of the applications on a system. When creating a branch, use a name that identifies the appropriate application, such as `Expense[Feature]`. This avoids confusion when development teams attempt to add branches to their applications.

> **Note:** The branch list establishes a dependency order from top to bottom. If a branch is dependent upon other branches, it should be listed above the branches it depends upon.

Create branch rulesets for the branches

The development application must have at least one branch before you create a branch ruleset. A branch ruleset is branched from a ruleset in the main (base) application, or any of its built-on applications. The team determines which rulesets contain the rules they expect to modify for a planned enhancement. The branch ruleset is based on those rulesets. For more information refer to the Help topic Creating branch rulesets.

Grant access to the team application

Create an access group that references the team application. To name the access group, include the application name plus the standard user type, such as **ExpenseCPFAdmn**. When you create the access group:

- Verify that the access roles are sufficient to develop and test the application
- Reference the work pool for the main application

- Specify the correct portal, such as the Developer portal for application developers

Tip: When creating access groups for each team, start by copying the access groups for the original application.

Configure rules using the branch rulesets

You have created:

- A team development application, built on the main application

- Branches defined in that team development application

- Branch rulesets branched off of the appropriate base rulesets, and associated with the appropriate branches

The development team members can now implement the planned enhancements in the branches, using the branch rulesets.

To update an existing rule, save a copy of the base rule into the associated branch ruleset, and work on that copy.

Tip: If a ruleset is configured to support check out, you can check out a record directly to the branch. Checking out a record to a branch copies the rule in its current state to the branch and checks out the new record for configuration.

To create new rules, save them directly into the branch ruleset that is branched from the main ruleset.

Merge branches when development is complete and stable

Move the new and updated records from the branch to the base rulesets when development activity in the branch has reached a stable point. This makes the newest updates available to the wider development team. When you merge a branch into the application, you can either delete the branch if development is complete, or maintain the branch to support additional development of the feature.
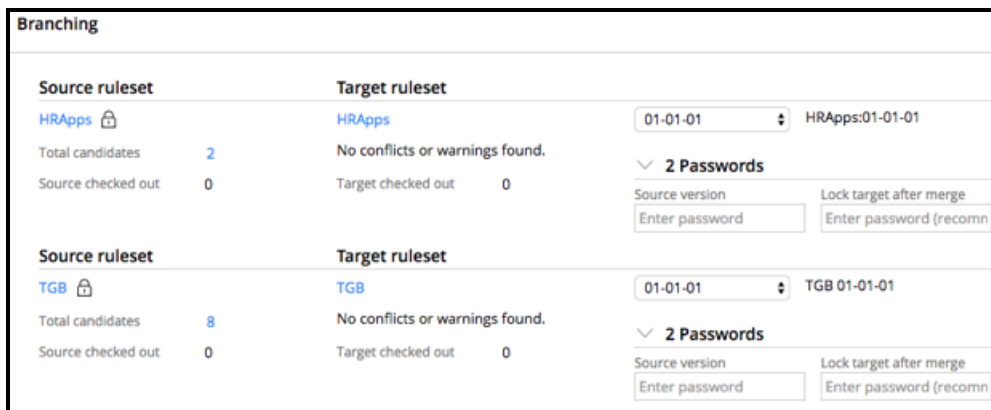
Important: You must check in all rules to the branch ruleset before merging.

For more information, refer to How to support parallel development and deployment of two separate projects for one application.

Refer to the online tutorial to practice creating branches for parallel development.

How to merge changes from a branched ruleset

After development in a branch has reached a stable state, use the **Merge Branch Rulesets wizard** to move branch contents into the base rulesets. The Merge Branch ruleset wizard helps identify potential conflicts so that you can address them before moving your changes.

Before moving the updated rules into the base rulesets, you first identify any conflicts between your changes. Other teams might have branched the same base rulesets to enhance the same rules, or modified the core rules at the same time as work was done in your branch.

Refer to the Merging Branches with the Merge Branches wizard procedure and Merging branches tutorial for examples.

> **Note:** Changes to rulesets must be closely coordinated. If a developer tries to check in a rule that already exists in a higher-numbered ruleset version, a warning appears in the rule form. Developers seeing the warning must communicate the changes they have made to the owner of the higher-numbered ruleset Version.

KNOWLEDGE CHECK

⌄ ANSWER	Before moving the updated rules into the base rulesets, you first _____.
identify any conflicts between your changes	

For additional information, refer to:

Merge Branch Rulesets wizard

About the Merge Branch RuleSets wizard

Conflicts and warnings in the Merge Branches wizard

Resolving conflicts in the Merge Branches wizard

Rule resolution

Introduction to rule resolution

In this lesson, you learn about rule resolution. Rule resolution finds the best or most appropriate rule instance to apply in a situation.

After this lesson, you should be able to:

- State the rule resolution process
- Influence rule resolution results by adjusting rule availability

Rule resolution

Rule resolution is a search algorithm used to find the most appropriate instance of a rule to execute in any situation.

Rule resolution occurs whenever a rule is needed to accomplish processing of a case. As you create applications, the choices you make when defining the values for the key parts of a rule determine how Pega finds the rule through rule resolution.

Rule resolution applies to most rules that are instances of classes derived from the abstract *Rule-* base class. The following are examples of instances of rules derived from the abstract Rule-base class:

- Case types (Rule-Obj-CaseType)
- Properties (Rule-Obj-Property)
- UI rules such as Sections (Rule-HTML-Section) and Harnesses (Rule-HTML-Harness)
- Ruleset names (Rule-Ruleset-Name)
- Ruleset versions (Rule-RuleSet-Version)

Rule resolution does not apply to rules that are instances of classes derived from any other abstract base class such as Data-, System-, or Work-. The following are examples of instances of rules derived from the abstract System- base class;

- Operator IDs (Data-Admin-Operator-ID)
- Email listeners (Data-Admin-Connect-EmailListener)
- Operator's favorites (System-User-MyRules)
- The rule check-in process (Work-RuleCheckIn)

Tip: A rules' type is defined by the class from which the rule is derived. For example, the rule type of a rule derived from Rule-HTML-Section is called a section rule. The rule type of a rule derived from Data-Admin-Operator-ID is called an operator ID rule.

KNOWLEDGE CHECK

ANSWER Rule resolution applies to most rules that are instances of classes derived from the _____ base class.

abstract Rule-

Key inputs to the rule resolution algorithm

The key inputs to the rule resolution algorithm are the:

- Key parts of a rule, such as the *Apply to:* class, rule name, and rule type
- User's ruleset list
- Class hierarchy of the rule in question
- Circumstances such as the value of a property, or time and date restrictions

- Availability of the rule
- User's access roles and privileges

The output of the resolution process is the first rule found that matches all of the key input criteria.

How the rule resolution process works

Rule Resolution is the process Pega uses to determine the most appropriate rule to execute.
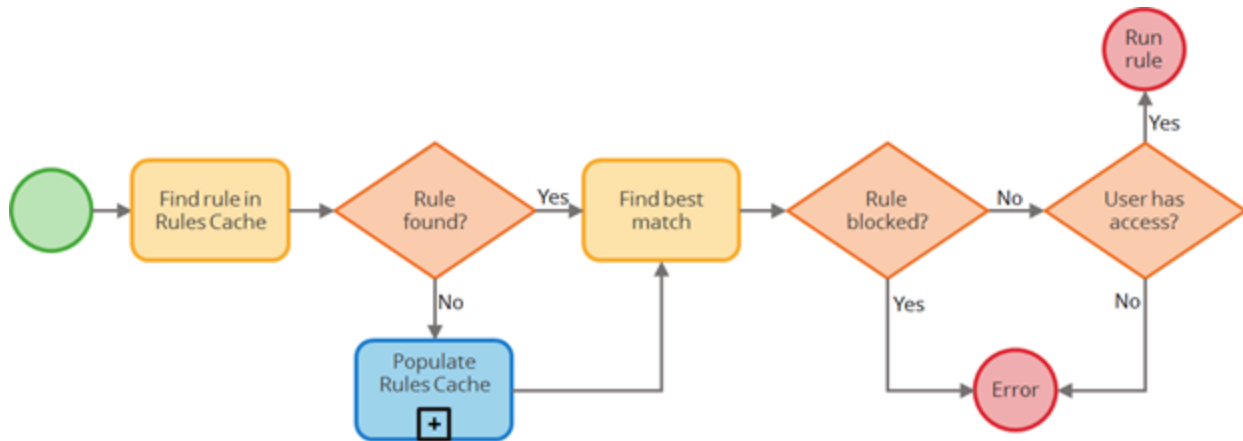
When a rule is referenced in a Pega application, rule resolution attempts to locate instances of the rule in the rules cache. If instances of the referenced rule are found, rule resolution finds the best instance of the rule and checks for duplicates. Then Pega confirms the rule is available for use. Finally, Pega verifies the user is authorized to use the rule.

If instances of the rule are not found in the rules cache, Pega runs a special sub-process to populate the rules cache.

The point of the rule resolution is to return the most appropriate rule to satisfy the need of a specific user for a specific purpose.

Find the rule in the rules cache

If rule resolution was already run for the rule, the rules cache has a list of all possible rule candidates.

For example, the referenced rule is a section rule named *CreateRequest* for a Purchase Request case. The requestor (the user working on the case) has the Purchasing:02-01-05 ruleset in their ruleset stack.

Pega 7 searches the Rules Cache for a list of all possible rule candidates for the rule in question.

The rules cache contains three rule candidates.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier	Privilege
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	Supplier=Restricted	
TGB-Purchasing-Work	Rule-HTML-Sectio	CreateRequest	Available	Purchasing	02-01-05	> 01 June 2000	

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier	Privilege
	n						
TGB-Purchasing-Work	Rule-HTML-Section	CreateReque st	Available	Purchasing	02-01-05		

Find the best instance and check for duplicates

In this step, the rule resolution algorithm determines which rule candidate is a first, best match. A first, best match is determined by an exact match of a property or date circumstance, or a default rule if no exact circumstance matches are found.

When a rule that matches any of these conditions is found, the rule resolution algorithm checks whether the next rule in the list is equally correct. If a subsequent match is found, Pega sends a message that there are duplicate rules and stops processing. If no other matches are found, Pega prepares to use the rule that matched the listed conditions.

Continuing with the example, the value of *.Supplier* in the Purchase Request case is set to *Open*. The first condition — *.Supplier=Restricted* — is not met, so the rule candidate is skipped and Pega moves to the next rule in the list.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier	Privilege
TGB-Purchasing-Work	Rule-HTML-Section	CreateReque st	Available	Purchasing	02-01-05	Supplier=Restricted	
TGB-Purchasing-Work	Rule-HTML-Section	CreateReque st	Available	Purchasing	02-01-05	> 01 June 2000	
TGB-Purchasing-Work	Rule-HTML-Section	CreateReque st	Available	Purchasing	02-01-05		

The next rule candidate has a date circumstance defined, so Pega compares the date circumstance against the current system date.

Continuing with the example, the next condition is a date range specified as *Before 01 June, 2000*. Assume the current system date is 15 August 2000.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier	Privilege
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	> 01 June 2000	
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05		

The date range circumstance is not met, so the rule candidate is skipped and Pega moves to the next rule in the list. The next rule candidate does not have a qualifier, so the system selects this rule.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier	Privilege
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05		

KNOWLEDGE CHECK

> During rule resolution, a first, best match is determined by _____.

an exact match of a property or date circumstance

KNOWLEDGE CHECK

> During rule resolution, the _____ rule is selected if no exact circumstance matches are found.

default

Confirm the rule is available for use

In this step, the rule resolution algorithm checks to see if the Availability of the rule is set to Blocked. If the rule is blocked, the system sends a message that it could not find an appropriate rule to use.

Continuing with the example, the availability of the rule candidate is set to *Available*, so the rule is considered available to run.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier	Privilege
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05		

Verify the user is authorized to use the rule

In this final step, the rule resolution algorithm verifies that the user has authorization to access the selected rule. If the user has all of the privileges required by the selected rule, the rule is executed. If the user does not have any of the privileges required by the rule, Pega sends a message that it could not find an appropriate rule to execute.

The rule does not list a required privilege, so the rule is selected and executed.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier	Privilege
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05		

How the rules cache is populated

Pega uses a caching mechanism called the **Rules Cache** to help ensure rule resolution operates efficiently.

When your application references a rule, Pega checks the rules cache for the referenced rule. If the referenced rule is not available in the rules cache, Pega uses a multiple-step process to populate the rules cache.
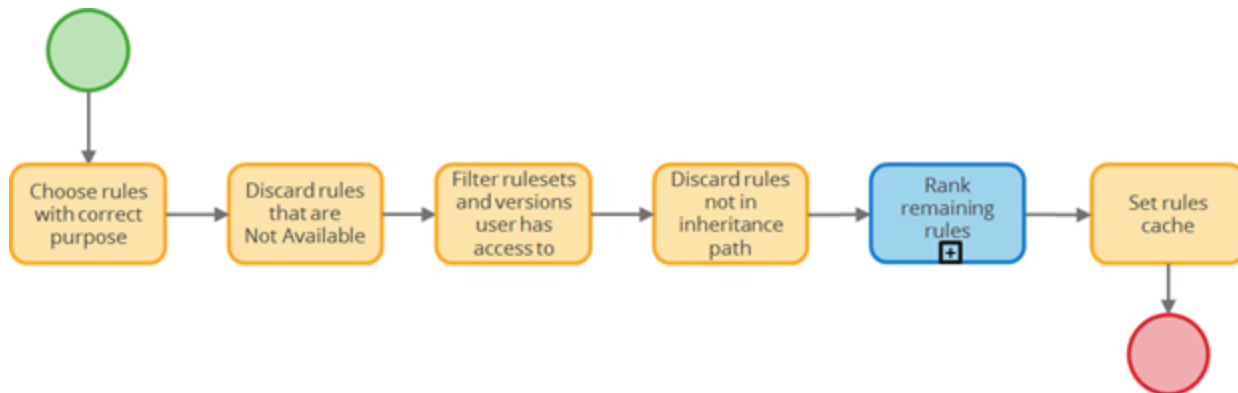
To populate the rules cache, the rule resolution algorithm first creates a list of all rules that match the purpose of the rule in question. Next, rule candidates marked as *Not Available* are removed from the list. Then the rule resolution algorithm uses the operator's *Ruleset list* to determine which rule candidates the operator can access. The rule resolution algorithm then removes all rule candidates not defined in a class in the ancestor tree. The rule resolution algorithm then ranks the remaining rule candidates. Finally, the rule resolution algorithm adds the remaining rule candidates to the Rules Cache.

The referenced rule used in the following examples is a section rule named *CreateRequest* for a Purchase Request case. This example helps illustrate the rule resolution process.

Choose all instances with the correct purpose

In this step, the rule resolution algorithm creates a list of all rules that match the purpose of the referenced rule.

Note: Remember, the purpose of a rule is defined by a combination of all the key properties of a rule, except the *Apply to:* class on which the rule is defined. For example, the key properties of a section rule include the *Apply to:* class, the rule type, and the rule name.

In the example, all HTML section rules named *CreateRequest* are collected into a list. There can be many instances of the rule in the initial list of rule candidates.

Note: The *Apply to:* class is displayed in this list to help provide context where each instance of the rule in question is found.

Apply to: class	Rule type	Rule name
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest

Apply to: class	Rule type	Rule name
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest
TGB	Rule-HTML-Section	CreateRequest
TGB	Rule-HTML-Section	CreateRequest
TGB	Rule-HTML-Section	CreateRequest
TGB	Rule-HTML-Section	CreateRequest
TGB	Rule-HTML-Section	CreateRequest
TGB	Rule-HTML-Section	CreateRequest
SAE-Quoting-Work-EnterQuote	Rule-HTML-Section	CreateRequest
SAE-Quoting-Work-EnterQuote	Rule-HTML-Section	CreateRequest
SAE-Quoting-Work	Rule-HTML-Section	CreateRequest
SAE	Rule-HTML-Section	CreateRequest

Discard rules where Availability = Not Available

In this step, the rule resolution algorithm filters the list of rule candidates and removes any rules where the Availability of a rule is set to Not Available.

Continuing with the example, the rule resolution algorithm finds three rules where the Availability of a rule is set to *Not Available*. The rules are removed from the list of rule candidates.

Apply to: class	Rule type	Rule name	Availability
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Not Available
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available

Apply to: class	Rule type	Rule name	Availability
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Blocked
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest	Available
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest	Available
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Not Available
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available
TGB	Rule-HTML-Section	CreateRequest	Available
TGB	Rule-HTML-Section	CreateRequest	Available
TGB	Rule-HTML-Section	CreateRequest	Available
TGB	Rule-HTML-Section	CreateRequest	Available
TGB	Rule-HTML-Section	CreateRequest	Available
TGB	Rule-HTML-Section	CreateRequest	Available
SAE-Quoting-Work-EnterQuote	Rule-HTML-Section	CreateRequest	Available
SAE-Quoting-Work-EnterQuote	Rule-HTML-Section	CreateRequest	Not Available
SAE-Quoting-Work	Rule-HTML-Section	CreateRequest	Available
SAE	Rule-HTML-Section	CreateRequest	Available

Discard inapplicable rulesets and versions

In this step, the rule resolution algorithm uses the operator's *Ruleset list* to determine which candidate rules the operator can access.

Tip: The *Ruleset list* is a combination of the ruleset name and a Major-Minor version number.

To be included in the results, each rule candidate must belong to a ruleset listed in the operator's ruleset list. Each rule must have the same Major version number, and a Minor version number less than or equal to the specified Minor version number listed in the operator's ruleset list.

Continuing with the example, assume the operator's ruleset list includes *Puchasing:02-01* and *TGB:03-01*.

The three rules in the *Purchasing:01-01-01* ruleset are eliminated because they do not match the major version as defined in the operator's ruleset list (Purchasing:**02**-01).

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-02-01

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn	Purchasing	02-01-10
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Blocked	Purchasing	01-01-01
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest	Available	Purchasing	01-01-01
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	Purchasing	01-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-10-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-10-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	01-01-01
SAE-Quoting-Work-EnterQuote	Rule-HTML-Section	CreateRequest	Available	Quoting	01-01-06
SAE-Quoting-Work	Rule-HTML-Section	CreateRequest	Available	Quoting	01-01-01
SAE	Rule-HTML-Section	CreateRequest	Available	SAE	01-01-01

The rule in the *Purchasing:02-02-01* ruleset is also eliminated because the minor version number (Purchasing:02-**02**) is higher than the minor version number defined in the operator's ruleset list (Purchasing:02-**01**).

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-02-01
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn	Purchasing	02-01-10
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-10-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-10-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	01-01-01
SAE-Quoting-Work-EnterQuote	Rule-HTML-Section	CreateRequest	Available	Quoting	01-01-06
SAE-Quoting-Work	Rule-HTML-Section	CreateRequest	Available	Quoting	01-01-01
SAE	Rule-HTML-Section	CreateRequest	Available	SAE	01-01-01

Of the rules listed in the TGB rulesets, only one rule matches the current major version listed in the operator's ruleset list (TGB:**03**-01). All other rules in the TGB rulesets are eliminated.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn	Purchasing	02-01-10
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-10-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-10-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	01-01-01
SAE-Quoting-Work-EnterQuote	Rule-HTML-Section	CreateRequest	Available	Quoting	01-01-06
SAE-Quoting-Work	Rule-HTML-Section	CreateRequest	Available	Quoting	01-01-01
SAE	Rule-HTML-Section	CreateRequest	Available	SAE	01-01-01

The rules listed in the *Quoting* and *SAE* rulesets are eliminated because the rulesets are not listed in the operator's ruleset list.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn	Purchasing	02-01-10
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01
SAE-Quoting-Work-EnterQuote	Rule-HTML-Section	CreateRequest	Available	Quoting	01-01-06
SAE-Quoting-Work	Rule-HTML-Section	CreateRequest	Available	Quoting	01-01-01
SAE	Rule-HTML-Section	CreateRequest	Available	SAE	01-01-01

KNOWLEDGE CHECK

Assume an operator's ruleset list includes *Loans:01-01*. Why would a candidate rule found in Loans:01-02 not be included in the results of the rule resolution process?

To be included in the results, each candidate rule must have the same Major version number, and a Minor version number less than or equal to the specified Minor version number listed in the operator's ruleset list.

Discard all candidates not defined in a class in the ancestor tree

In this step, the rule resolution algorithm examines the *Apply to:* class in the list of candidate rules to determine if the remaining candidate rules are in the inheritance hierarchy of the referenced rule.

Note: The **ancestor tree** refers to a rule's inheritance.

To be included in the results, the *Apply to:* class of the remaining candidate rules must match the ancestor tree of the referenced rule. Only rules found in the ancestor tree of the referenced rule — by either pattern or direct inheritance — will be retained in the list.

Important: A class must have the *Use class-based inheritance to arrive at the correct rule to execute* check box selected in the class definition to be considered in this step.

Continuing with the example, the referenced rule has an *Apply to:* class of *TGB-Purchasing-Work-PurchaseRequest*. There is one candidate rule not in the ancestor tree, so it is removed from the list.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn	Purchasing	02-01-10
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work-PurchaseOrder	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01

KNOWLEDGE CHECK

ANSWER   Assume the referenced rule has an *Apply to:* class of *TGB-HRApps-Work-Onboarding*. Why would a candidate rule with an *Apply to:* class of TGB-HRApps-Work be retained in the list of rule candidates?

Rules found in the ancestor tree of the rule in question — by either pattern or direct inheritance — are retained in the list of rule candidates.

Rank remaining rule candidates

In this step, the rule resolution algorithm uses a three-step sub-process to rank the remaining candidate rules. First, the list of rule candidates is sorted. Then, all rule candidates marked as Withdrawn are removed from the list. Finally, a default rule candidate is defined.

Sort the remaining rule candidates

The rule resolution algorithm sorts the remaining rule candidates according to this specific order: Class, Ruleset, Circumstance, Circumstance Date, Date/Time Range, Ruleset, then Version.

The first two criteria — Class and Ruleset — provide the basics of rule resolution. The closer a rule candidate is to the *Apply to:* class of the referenced rule, the higher the rule candidate is ranked. Within each class, the rule candidates are sorted according to the operators' ruleset list.

Important: Rules which do not have the *Use class-based inheritance to arrive at the correct rule to execute* check box selected in their class definition are not ranked by class.

The next three criteria — Circumstance (Property or Template), Circumstance Date, and Date/Time Range — are used as qualifiers to the basics of rule resolution, and are used to further refine, or specialize, rule candidates.

The last criteria — Version — ranks the remaining candidate rules by the ruleset version that contains them. This ensures that circumstanced rules are not automatically overridden if the base rule is updated in a more recent ruleset version.

Continuing with the example, the list of rule candidates is ranked according to class and ruleset order, including any circumstanced rules, so no rules are removed from the list of rule candidates.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn	Purchasing	02-01-10	
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01	
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	Yes (Circumstance)
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	Yes (Date)
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01	

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01	

KNOWLEDGE CHECK

As candidate rules are ranked during rule resolution, which criteria is considered last? Why?

The version of the ruleset is considered last. This ensures that circumstanced rules are not automatically overridden if the base rule is updated in a more recent ruleset version.

Remove rule candidates with an Availability set to Withdrawn

After the rule candidates are ranked, the rule resolution algorithm removes any rule candidates where the Availability of the rule is set to Withdrawn.

When the Availability of a rule is set to Withdrawn, the rule is removed from the list of rule candidates. Finally, all other rule candidates that match the *Apply to:* class, the ruleset name and major version number, the rule purpose, and any qualifiers of the rule set to Withdrawn are removed from the list as well.

Continuing with the example, there is one rule candidate with an Availability set to Withdrawn.

Important: Notice all rules in the same *Apply to* class as the rule with the Availability set to Withdrawn are removed from the list of candidate rules.

In this sub-step, three rules are removed from the list of candidate rules.

The rule candidate in TGB-Purchasing-Work-PurchaseRequest with the Availability set to Withdrawn is removed. There are two rule candidates that match the *Apply to:* class, the ruleset name and major version number, and the rule purpose of the Withdrawn rule, so they are also removed.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn	Purchasing	02-01-10	
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01	
TGB-Purchasing-Work	Rule-	CreateRequest	Available	Purchasing	02-01-05	Yes (Circumstance)

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier
	HTML-Section					
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	Yes (Date)
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01	
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01	

Determine default rule candidate

The last sub-step in the ranking phase of the rule resolution algorithm determines the *default* rule candidate.

A default rule candidate is the first rule candidate (highest ranked) that has no qualifiers. This default rule candidate is the last possible rule to be executed as it always matches any additional requests for this rule.

Additional rule candidates ranked below the default rule candidate are discarded.

Continuing with the example, the default rule candidate is *TGB-Purchasing-Work-CreateRequest* in the *Purchasing:02-01-05* ruleset.

The two rules below the default rule candidate are removed from the list of rule candidates.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	Yes (Circumstance)
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	Yes (Date)
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01	
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01	

Set the rules cache

The rule resolution algorithm adds the remaining rule candidates to the rules cache.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version	Qualifier
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	Yes (Circumstance)
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	Yes (Date)
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05	

How to influence rule resolution through rule availability

Rules that are subject to the rule resolution process have an Availability setting. A rule's current Availability is visible on the rule form next to the rule's name or description.
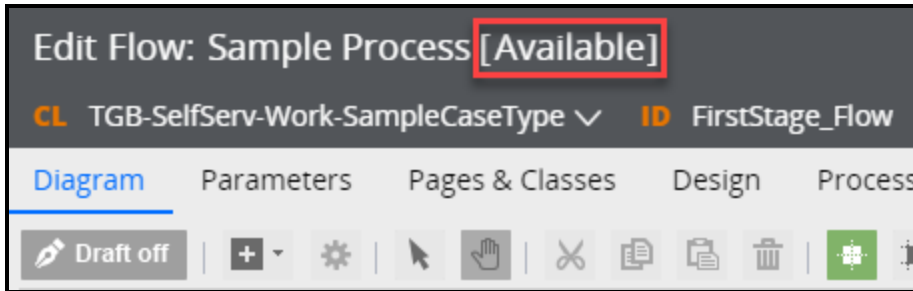
The Availability setting is used to determine if a rule is available for use during rule resolution. The availability of a rule is also used to determine if you can view, copy, or edit a rule in Designer Studio.

You can set the availability of a rule to one of five values.

Availability = Available

An availability of *Available* indicates the rule may be used during the rule resolution process.

Tip: When you create a rule, the default availability of the rule is set to *Available*.

You can view, copy, edit, and execute rules in Designer Studio when the availability is set to *Available*.

Availability = Final

An availability of *Final* indicates the rule may be used during the rule resolution process.

Rules marked as *Final* can be viewed and executed in Designer Studio, but cannot be edited or copied into another ruleset.

Note: The *Final* setting is used by Pega to indicate Pega-provided rules that may be changed in subsequent releases.

Availability = Not Available

An availability of *Not Available* indicates the rule may not be used during the rule resolution process. When a rule set to *Not Available* is found during the rule resolution process, the rule in the next-highest version is considered for rule resolution.

In the table below, the availability of the rule named *CreateRequest* in the *Purchasing:02-01-10* ruleset is set to *Not Available*. The rule in the next-highest version — *Purchasing:02-01-05* — is considered for rule resolution.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Not Available	Purchasing	02-01-10
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01

Rules marked as *Not Available* can be viewed, copied, or edited in Designer Studio, but will not execute.

Tip: Set the availability of a rule to *Not Available* during initial development. This allows you to save a rule without validation.

Availability = Blocked

An availability of *Blocked* indicates the rule may be used during the rule resolution process. If a blocked rule is selected during rule resolution, execution is halted and an error message is displayed.

In the table below, the availability of the rule named *CreateRequest* in the *Purchasing:02-01-10* ruleset is set to *Blocked*. This version of the rule is available during rule resolution.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Blocked	Purchasing	02-01-10
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01

Block a rule when access to the rule must be blocked immediately and you need more time to develop and release an updated rule.

You can view, copy, edit, or execute rules in Designer Studio when the availability is set to *Blocked*.

Availability = Withdrawn

An availability of *Withdrawn* indicates all rules with the same purpose, *Apply to:* class, and in the same ruleset are not considered during the rule resolution process.

When a rule marked as *Withdrawn* is found during rule resolution, the system looks for an instance of the rule in the next highest class or ruleset.

In the table below, the availability of the rule named *CreateRequest* in the *Purchasing:02-01-10* ruleset is set to *Withdrawn*. All rules with same purpose, the same *Apply to:* class, and in the same ruleset are not considered during rule resolution.

Apply to: class	Rule type	Rule name	Availability	Ruleset	Version
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Withdrawn	Purchasing	02-01-10
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work-PurchaseRequest	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-05
TGB-Purchasing-Work	Rule-HTML-Section	CreateRequest	Available	Purchasing	02-01-01
TGB	Rule-HTML-Section	CreateRequest	Available	TGB	03-01-01

You can view, copy, edit, or execute rules in Designer Studio when the availability is set to Withdrawn.

Parameterizing rules for reuse

Introduction to parameterizing rules for reuse

In this lesson, you learn how parameters can help promote rule reuse and make your application easier to maintain.

After this lesson, you should be able to:

- Describe the importance of creating modular, reusable rules
- Explain how parameters make a rule more reusable
- Add parameters to a rule to improve reusability

Parameters

Parameters enable you to create generic rules that can be reused in more than one context.

For example, a list of available healthcare plans are stored in a single database table.

PlanName	Type	Description
Plan One	Medical	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc eleifend orci urna, vitae dapibus mi placerat quis.
Plan Two	Medical	Ut quis nulla eu neque sollicitudin molestie. Mauris semper tortor non libero euismod, ac volutpat dui rutrum
Plan Three	Medical	Duis sit amet nisl sed dui molestie cursus eu id odio. Suspendisse scelerisque nunc vel risus feugiat.
Plan Four	Medical	Nullam in posuere nibh. Nullam sit amet convallis eros, eu ultrices sapien.
Plan Alpha	Dental	Maecenas gravida est purus, eget tincidunt enim imperdiet at. Pellentesque a malesuada lorem, vel vulputate nunc.
Plan Bravo	Dental	Curabitur sollicitudin, justo a gravida fringilla, lectus leo tempor lectus, a aliquet arcu lectus finibus magna.
20/20	Vision	Pellentesque eget fermentum diam, nec convallis nisl. Ut quis nulla eu neque sollicitudin molestie.
20/100	Vision	Fusce non nisi egestas, pulvinar felis a, aliquet arcu. Maecenas at lectus posuere, auctor odio sed, dictum arcu.

Retrieving a list of available plans for a specific plan type requires a report definition with a filter condition set to the specific plan type.
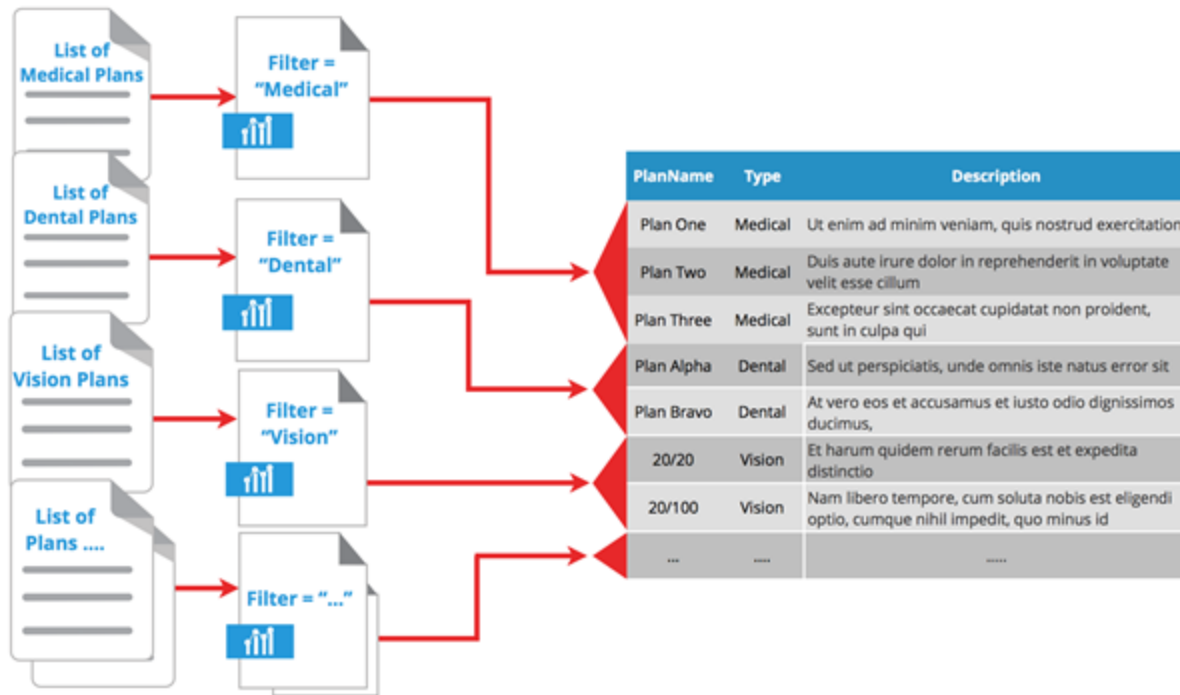
If you hard code the value of the filter condition in a report definition, you must create a separate report definition for each additional plan type. If additional plan types are added, you must create additional report definitions for each new plan type. As the number of report definitions increases, implementing changes becomes more time consuming and the risk of introducing an error increases.

KNOWLEDGE CHECK

	State two disadvantages of creating a unique rule for each business context.
	Implementing changes takes more time and the risk of introducing errors increases.

Create reusable rules with parameters

Parameters eliminate the need for a unique rule for each condition, or context. A **parameter** is a value passed into a rule to make it more reusable.

PlanName	Type	Description
Plan One	Medical	Ut enim ad minim veniam, quis nostrud exercitation
Plan Two	Medical	Duis aute irure dolor in reprehenderit in voluptate velit esse cillum
Plan Three	Medical	Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
Plan Alpha	Dental	Sed ut perspiciatis, unde omnis iste natus error sit voluptatem
Plan Bravo	Dental	At vero eos et accusamus et iusto odio dignissimos ducimus,
20/20	Vision	Et harum quidem rerum facilis est et expedita distinctio
20/100	Vision	Nam libero tempore, cum soluta nobis est eligendi optio, cumque nihil impedit, quo minus id

In the healthcare plans example, using a parameter to set the filter condition enables you to use a single report definition to retrieve a separate list for each specific plan type.

Parameters enable you to separate the functional behavior of a rule from specific business contexts, and therefore, maximize the reuse of the rule. Implementing changes takes less time and, because only one report definition must be maintained, the risk of introducing errors is diminished.

KNOWLEDGE CHECK

State two advantages of separating the functional behavior of a rule from the business context.

Implementing changes takes less time and the risk of introducing errors is diminished.

How to make rules reusable with parameters

Rules that accept parameters have a Parameters tab where you specify the parameters the system uses when the rule is executed. After the parameters are added, you can reference them in a rule, including the rule in which the parameter is defined.

Define a parameter for a rule

Common fields

Each parameter consists of three common fields — Name, Description, and Data type.

> **Note:** The Name and Data type fields are always required.

Name

Enter a unique name that clearly identifies the parameter. For example, to add a parameter for a tax rate, use **TaxRate**. Do not use **Rate**, as it may be ambiguous to other architects.

The name should begin with a letter and can include letters, numbers, hyphens, and underscore characters. When defining parameter names, follow your organization's naming convention for properties.

Description

Provide a short description that describes how the parameter is used. For example, use **Regional tax rate used to calculate total order price**.

Data type

Select the expected data type of the parameter's value.

Pega 7 performs a level of type-checking when you save a rule that references the rule on which the parameter is defined, not at run time.

Additional fields

Additional fields — Required, In/Out, Default value, Smartprompt type, and Validate as — are available for certain rule types.

Required

Use this field to indicate the parameter must have a value when the rule is executed.

You must select an option when the Required field is displayed. When Required is set to Yes, the restriction is enforced when you save a rule that references the rule on which the parameter is defined, not at run time.

In/Out

Use this field to indicate whether the parameter is used for input to the rule, or returned as output from the rule.

The In/Out field is required when displayed. The In option is selected by default.

Default value

Use this field to define a literal constant value that appears as the default parameter value when the system presents a parameter prompt dialog for this rule.

Smartprompt type

Use this field to configure a list of values to choose from when setting the parameter's value in other rules.

Validate as

Use this field to identify a property for the Smartprompt Type operation.

Parameter pages

Each parameter you add to a rule is added to a parameter page. The parameter page stores values for each parameter defined for a rule. Pega maintains a parameter page in memory for each parametrized rule. The parameter page for a rule cannot be viewed using the clipboard. To view the contents of the parameter page, you use the Tracer to record execution of the rule, then view the contents of the parameter page for the appropriate step in the Tracer results.

How to pass a parameter value to a rule

Parameters can be referenced from any rule, including the rule in which they are defined. Depending on where a parameter is defined — in the same rule, or in another rule — determines how it is referenced.

Reference a parameter in a rule

To reference a parameter in the rule in which the parameter is defined, use the keyword *Param* in front of the parameter name. For example, to add a filter to a report definition using a parameter named *PlanType*, enter **Param.PlanType** in the **Value** field for the filter condition. When running the report, Pega looks for the value of the *PlanType* parameter on the parameter page for the report definition, and applies the value to the filter condition.
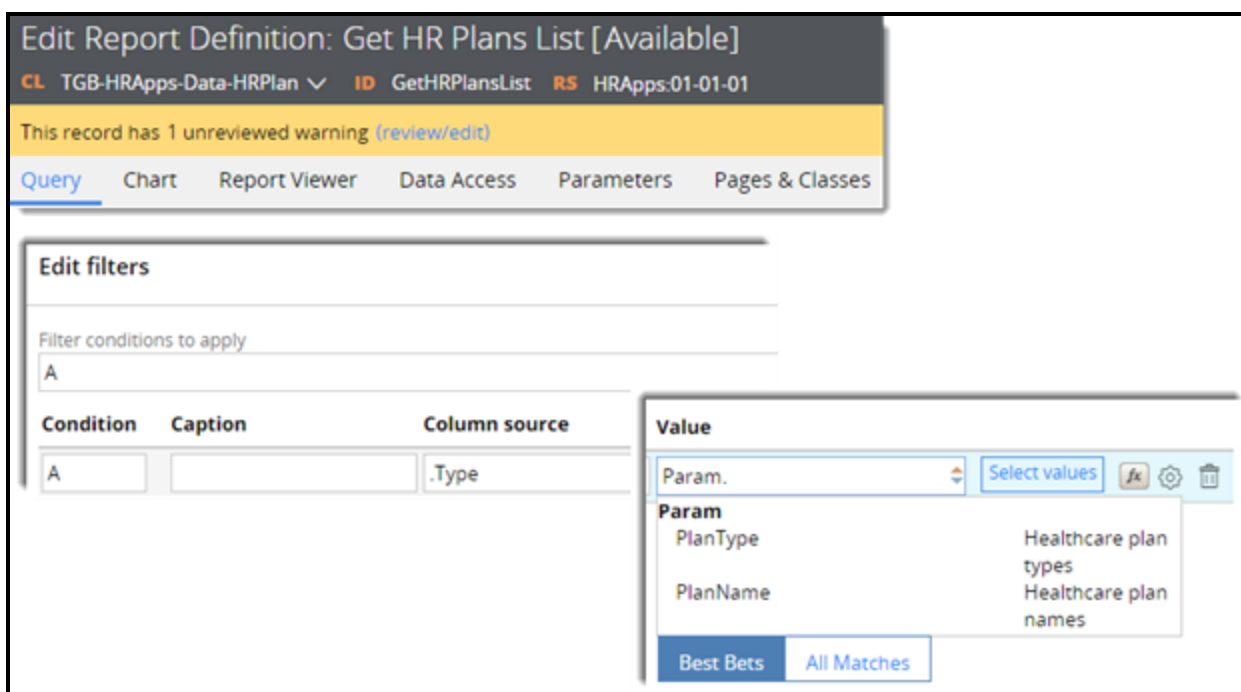
In the example above, two parameters (*PlanType* and *PlanName*) are defined on the **Parameters** tab of the report definition. Use the keyword *Param* to access a list of available parameters, then select the desired parameter.

Provide a value for the parameter

When you reference a parametrized rule, Pega prompts you with the name of each parameter accepted by the rule. For each parameter, Pega provides a field in which to enter a value. This value can be a constant, a property reference, or another parameter. When an application uses a parametrized rule, Pega passes the value to the rule, and the parametrized rule returns the appropriate result.
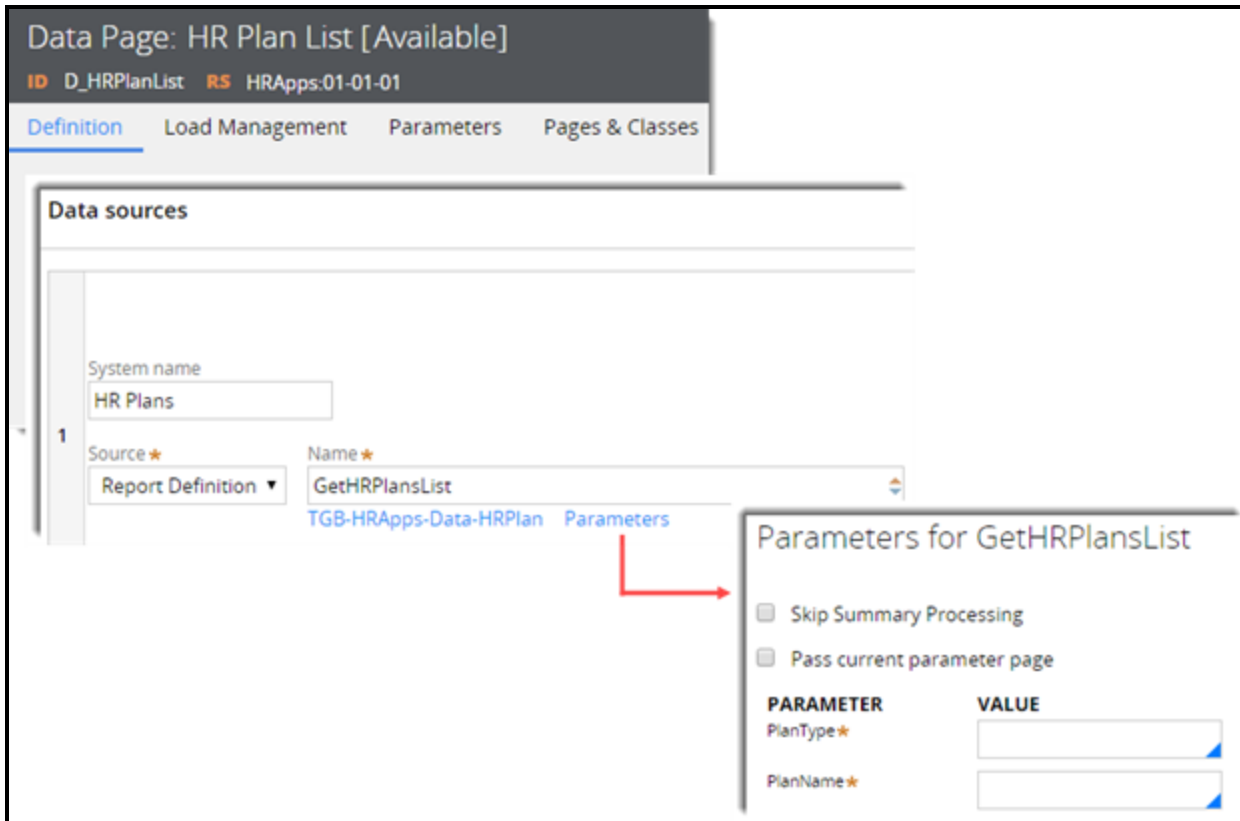
In the example above, the data page references a report definition that has defined two required parameters. The **Parameters** link on the data page rule opens a modal dialog that displays a list of parameters defined in the report definition.

Note: If a referenced rule requires a parameter, the referencing rule indicates the required parameter with an asterisk (*).

Pass a parameter page

Pega provides two options for passing parameters to a rule. By default, when you select a parametrized rule, Pega lists each parameter accepted by the rule and provides a field for you to enter a value for each parameter. You can also pass the entire parameter page for the current rule to the parametrized rule by selecting the **Pass current parameter page** option. In the previous example, you could select **Pass current parameter page** to pass a value for the *PlanType* parameter from the data page to the report definition.

When you pass a parameter page to a rule, the rule reads parameter values from the parameter page associated with the calling rule. Updates to parametrized values are made on the parameter page, which may affect the behavior of the calling rule.

KNOWLEDGE CHECK

What keyword do you use to reference parameters in the rule on which the parameters are defined?

Param

CASE DESIGN

Creating temporary cases

Introduction to Creating Temporary Cases

When starting a business process, an organization may want to defer recording the case in the database. Starting a case as a temporary case allows you to defer saving the case until one or more threshold conditions are met. If the temporary case is closed or resolved, no record of the case will exist.

In this lesson, you learn how to configure a temporary case for processing. You also learn how to use a SmartShape to persist data to the database.

After this lesson, you should be able to:

- Describe the purpose of a temporary case
- Explain when to create a temporary case
- Configure a case for temporary processing
- Persist a temporary case

Temporary Cases

Temporary cases store data that is not stored in the Pega database. For example, consider a change of address case for a customer service application. The customer service representative (CSR) checks the user's current address in the database. If the address is up-to-date, no changes are necessary. This address check is temporary.
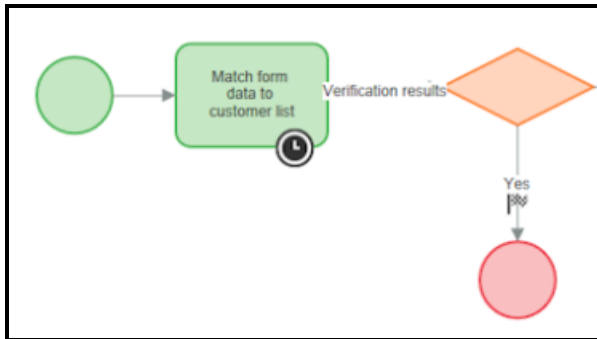
Temporary cases are modeled in screen flows and regular flows. In this diagram, the temporary case is resolved.

Persist the temporary case

Once a case meets the qualifying condition specified by the organization, the case is ready to be recorded in the database for future reference. The recording of a case in the database is referred to as **persisting** the case. For example, when the customer has a new address that does not match the existing address, the CSR continues the update process. This is the point in the workflow where you as a developer indicate where to persist the case in the database.

To make the case permanent, and continue to the next stage of the Update Address process, add a *Persist Case* SmartShape to the workflow.
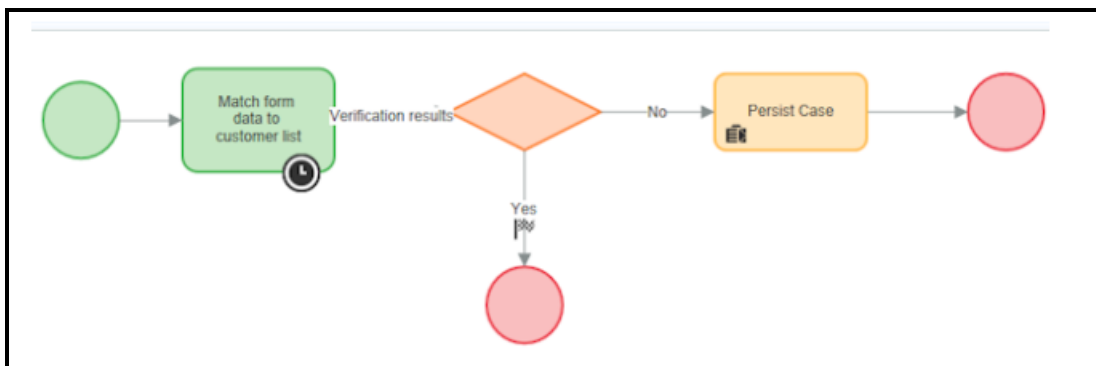
KNOWLEDGE CHECK

Temporary cases capture information that is _____.

not stored in the database

KNOWLEDGE CHECK

$\vee$ **ANSWER**   What happens when the case is persisted?
When the case is persisted, updated data is stored in the database

Configuring temporary case processing

When a business user creates a temporary case, the system does not create a case instance in the database (an object ID is not created). As the case is processed, data remains in memory but is not committed to the database.

In your process, decide when to persist the case. To persist the case, add a Persist Case SmartShape to the flow. When Pega encounters the Persist Case shape, Pega creates a case instance and commits the case data to the database.

When adding the Persist Case shape to the case life cycle, consider the information that must be input to qualify the case. Remember to add the Persist Case shape only after this information has been added to the case. For example, if a case type provides the user with a wizard to select the type of bank account to open, you only persist the case after the user selects the account type and is ready to open an account.

Create a temporary case

Configure the starting flow for a case (*pyStartCase*) so that the case is created as a temporary case.

1. From the Cases Explorer, open the case type rule.

2. On the **Processes** tab, next to the **pyStartCase** field in the **Starting processes** section, click the **Crosshair** icon to open the *pyStartCase* flow rule.

©2017 Pegasystems Inc.
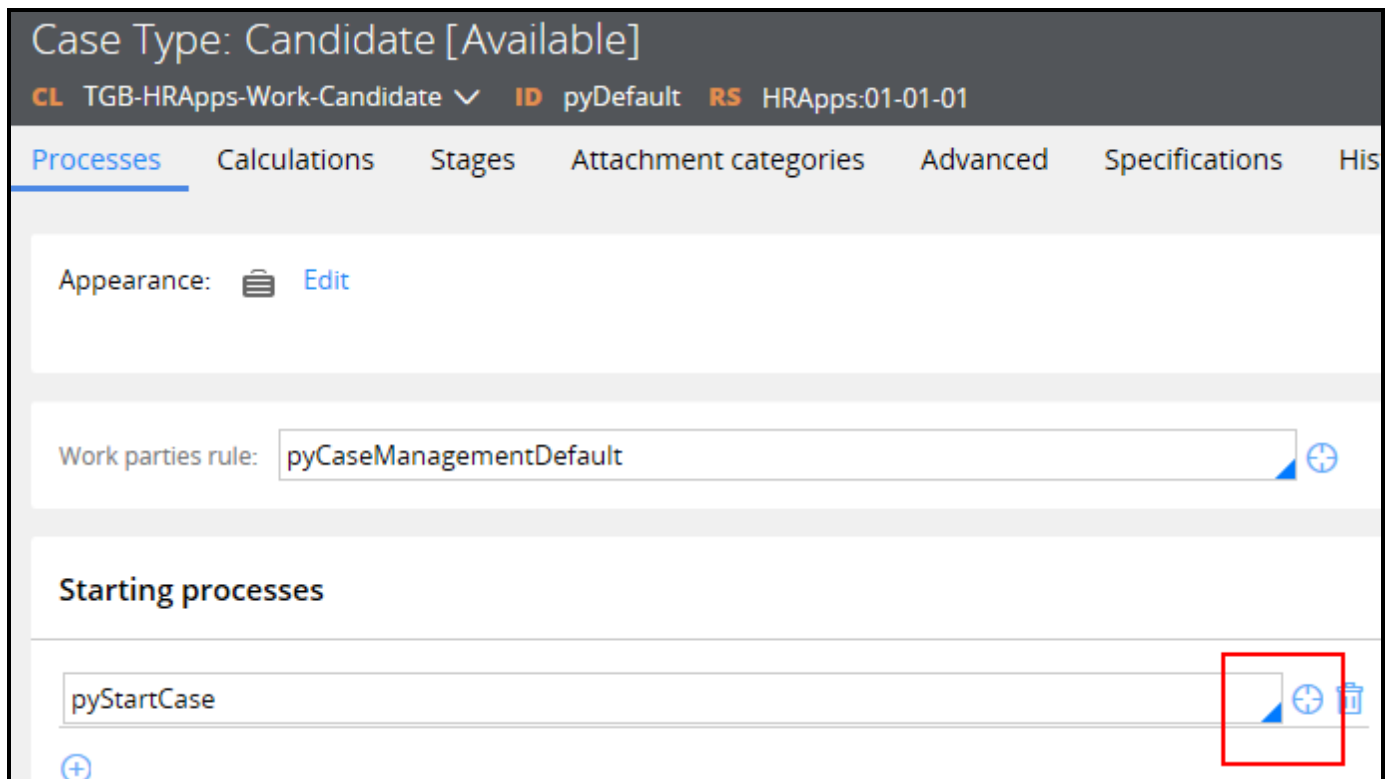
> **Note:** When you create a case type, Pega automatically creates a starting flow for you, named pyStartCase. This flow initializes the case, and then runs the first process in the first stage of the case life cycle.

3. On the **Process** tab, select the **Temporary object** check box.
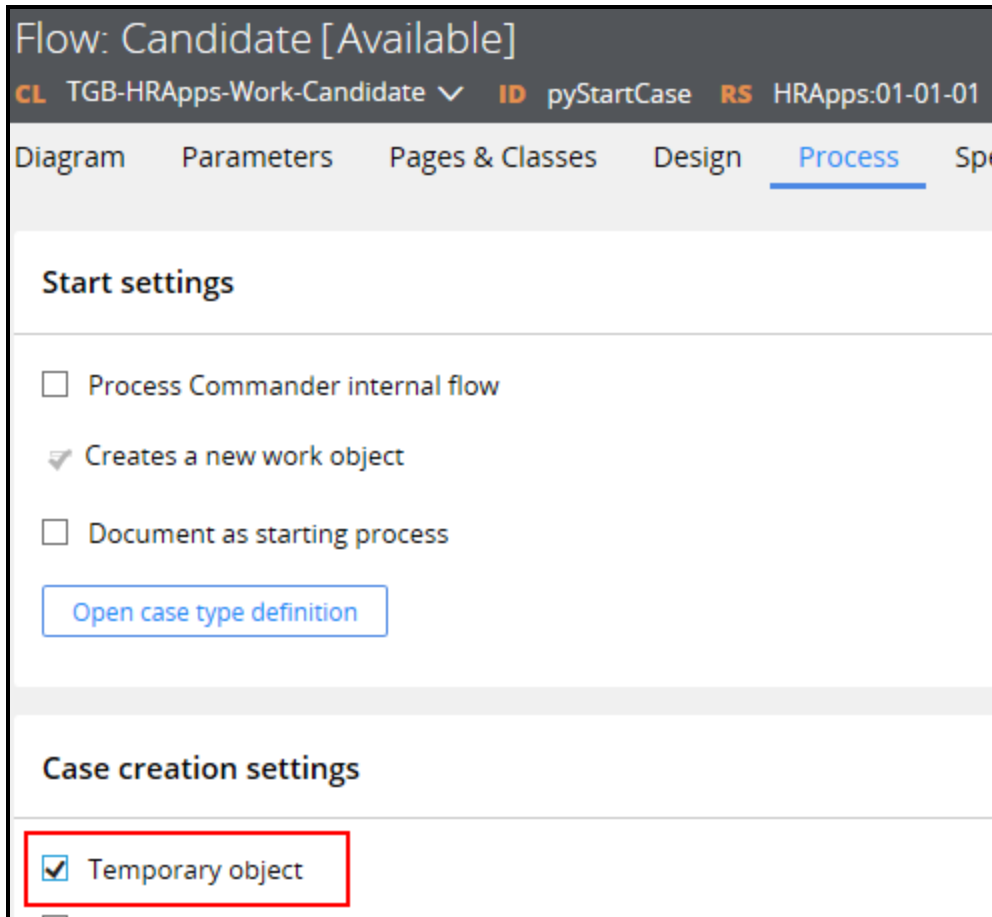
4. Click **Save**. When the user creates a case, it will be temporary.

Persist a temporary case

1. In the Case Designer, open the case type.

2. Identify the step in the starting workflow where the case will be persisted to the database.

3. After the **+Step** you have identified, add a new step and select **More**.
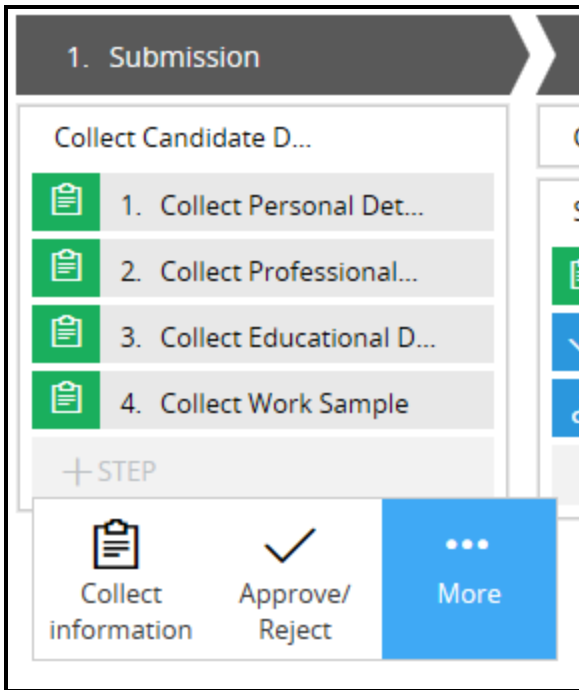
4. In the dialog that appears beneath the new step, expand the Utilities section.

5. Select **Persist Case**. The Persist Case panel is displayed.

6. In the panel, click **Select**.
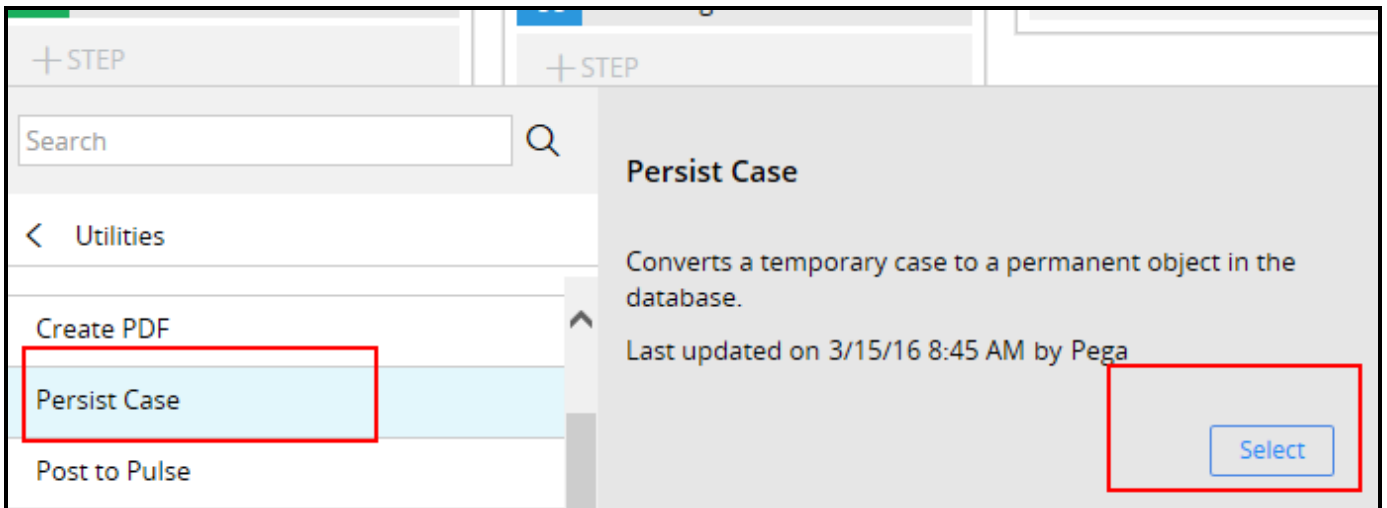
7. On the case life cycle header, click **Save**. The process now includes the Persist Case SmartShape. In this example, it follows the Collect Work Sample assignment shape.

When a case is submitted after the Collect Work Sample step, the case is persisted to the database.

Searching for duplicate cases

Introduction to searching for duplicate cases

In this lesson, you learn how to configure the duplicate case search feature to identify duplicate cases in the system. This feature allows users to decide which case is redundant and should be removed from the process.

After this lesson, you should be able to:

- Describe situations in which users might submit duplicate cases
- Describe the duplicate case search logic and how it identifies duplicate cases
- Configure a duplicate case search

Duplicate cases

In many situations, a user may enter a case that has many of the same data values as another case already in the system. Usually, matching data is not an issue. For instance, two purchase requests may have the same request date, the same items, or the same customer name. However, if a specific combination of data values match, the new case is possibly a **duplicate case**.

For example, a health care services application processes requests to authorize insurance coverage for surgical procedures. A user enters a request for a specific surgeon to perform inpatient surgery on May 1 to repair a torn tendon. Two days later, the patient's health care provider discovers that the surgeon is not available on the specified date. The provider submits a new request for the same type of surgery, the same surgeon, and the same procedure but for a procedure date of May 7. Because three of the four data values — the surgeon, type of surgery, and procedure — match in both requests, the second request is likely a duplicate. To avoid double-booking the procedure, the user should not process the request with the incorrect date. In these situations, users need to identify duplicate cases so that users can process the correct case.

Pega provides the **duplicate case search** process to help users identify and resolve duplicate cases. This process is implemented in your case life cycle as a Duplicate Search step. When a case enters the step, the system uses **weighted conditions** to compare specific properties or values with cases already in the system. Each condition has a weight (between 1 and 100) to determine the condition's relative importance when making the comparisons. The system adds up the weights of all the conditions that evaluate to true. If the sum exceeds a specified threshold value, the system flags the current case as a duplicate. The duplicate case search process then displays to the user the current case and the matching case in the system. The user may select the unwanted duplicate and resolve it. The case is not processed. Alternatively, the user may decide that the current case is not a duplicate and choose to continue processing the case.

In the previous example, assume that you have given the surgery type, procedure, surgeon, and date a weighted condition of 25 each. The system displays the second request as a duplicate because the surgeon name, surgery type, and procedure match the values in an existing case. The total condition value is 75, exceeding a matching threshold of 50. The user decides that the second request is valid and resolves the original request as a duplicate.

KNOWLEDGE CHECK

What does duplicate case search use to compare specific properties or values with cases already in the system?

Duplicate case search uses weighted conditions. Each condition has a relative importance defined by a weighting value.

©2017 Pegasystems Inc.

How to identify duplicate cases

There are two main steps to implementing duplicate case search. First, you configure the duplicate case search logic by adding weighted conditions. You also add conditions that must match in order for a case to be evaluated a possible duplicate. Then, you add a Duplicate Search step to your case lifecycle. The Duplicate Search step runs the duplicate case search process when a case enters the step. The associated process uses the duplicate case search logic to flag duplicates. The process also provides a user interface allowing users to either resolve or process duplicates.

Configure the condition logic

On the Case Designer Settings tab, use the **Track duplicates** panel to configure the weighted and must match conditions. When you create the conditions, the system creates the case match rule named *pyDefaultCaseMatch* rule. The logic for tracking duplicate cases is implemented in the duplicate case search process.

Add weighted conditions

Enter weighted conditions that the duplicate search case process uses to identify duplicates. In the two **Conditions** fields, enter the property names or values in the current case you want to compare with existing cases. You can enter the current case and existing case values in either column. In the field containing the value for the current case, you must add the *Primary* keyword to the value. For example, in the first field, enter **Primary.FirstName**, and in the first field, enter **.FirstName**. Select the appropriate relational operator between the fields.

Specify a relative weight for each condition. In the **Weight** field, enter an integer value between 1 and 100. The higher the value is, the greater the effect this value has on flagging a duplicate case.

In the **Case is a duplicate when sum>=** field, specify a numerical value to set the duplicate case threshold. The duplicate case logic sums the weights of matching conditions. The sum is tested against the threshold value. When the total weights of matching conditions equals or exceeds this threshold, the system flags the current case as a duplicate case.

The following image shows an example of weighted conditions in the **Track duplicates** settings panel.

Track duplicates
Automatically track duplicate cases using the following conditions

Weighted conditions

25	Primary.SurgeonName = .SurgeonName
25	Primary.ProcedureDate = .ProcedureDate
25	Primary.ProcedureType = .ProcedureType
25	Primary.Procedure = .Procedure

+ Add condition

Case is duplicate when sum >= * 50

The weights and thresholds depend upon the specific requirements of your organization. If weights are incorrectly assigned, a large number of cases may unnecessarily be flagged as duplicates.

For example, assume an insurance authorization for a surgical procedure uses procedure type, surgeon name, procedure date, and procedure as your conditions. You have set the threshold to 50. Because procedure types and procedures frequently match with existing cases, you do not give the conditions high weights. In combination, they should not exceed the threshold. In contrast, you give surgeon name and procedure date higher weights. A case is more likely to be a duplicate if either of these conditions evaluate to true.

The following table is an example of how you might weight the conditions.

Condition values	Condition weights
.SurgeonName	30
.ProcedureDate	30
.ProcedureType	20
.Procedure	20

Also consider adjusting the threshold value if too many or too few cases are being identified as duplicates.

KNOWLEDGE CHECK

How do you identify a weighed condition value that is in the current case?

You add the keyword Primary to the value.

Add must match conditions

As a best practice, include one or more **must match conditions**. When the duplicate case search process begins, it first evaluates the must match condition. If the condition evaluates to true, the current

case is considered a potential duplicate. The process then evaluates the weighted conditions. To create a must match condition, you enter a value to evaluate for potential duplicates. You then specify an operation. In some cases, you also enter a filter value. For example, assume you only want to evaluate cases where the patient ID in existing cases match the patient ID in the current case. You enter the patient ID as the value and an operation of **is same**. If the patient IDs match, the process then evaluates the weighted conditions. In another example, you can filter out resolved cases by entering .pyStatusWork as a potential duplicate value, **does not contain** as the operation, and **"Resolved"** as the filter value.

Exact match conditions limit the number of weighted condition evaluations. This can help reduce the number of incorrectly tagged duplicates and can help enhance performance.

> **Important:** Properties that are referenced by a must match condition must be exposed as columns in the database.

KNOWLEDGE CHECK

> ⌄
> ANSWER
>
> How does the duplicate case search process use the weights in weighting conditions to flag a duplicate case?

> The process sums the weights of all the conditions that evaluate to true. If the sum is greater than a threshold value, the process flags the current case as a duplicate.

Add the Duplicate Search step

You add the duplicate case search process by adding a Duplicate Search step in the case lifecycle. You can add the step anywhere in the case lifecycle. Typically, you check for duplicates after initial information about a case has been collected. For example, when processing a loan request, users first collect the customer's personal information and loan information. You add a Duplicate Search step after this information is collected. Duplicate Search can evaluate conditions such as customer name, loan type, and loan amount.

> **Tip:** On the case lifecycle, click **+STEP > More > Utilities > Duplicate Search** to add the step.

When Duplicate Search identifies a duplicate case, the system displays a user form that lists the current case and the matching cases. The system prompts the user to either select a duplicate case and resolve it, or to continue processing the current case. This standard flow action *pyDuplicateSearchCases* uses properties such as case status, creation date, and match score to identify each case. You can customize *pyDuplicateSearchCases* by adding property values that help users identify and resolve the correct duplicate. For example, you may add work party and email address.

KNOWLEDGE CHECK

> ⌄
> ANSWER
>
> Where in a process are you most likely to place a Duplicate Cases step?

> Place the step after basic case information is collected.

Combining temporary cases with duplicate cases

In some situations, duplicate cases can commonly occur due to high intake volume. For example, assume that your application has a process for updating customer information. Because multiple users enter

updates, duplicate cases often occur. To prevent large numbers of unprocessed cases from accumulating in the system, consider using temporary cases in the process. In this scenario, you would set up the address update case type to instantiate temporary cases. You would then add the Persist Case step after the Duplicate Search step. New cases that are not duplicates are persisted as they advance through the process. If a new case is flagged as a potential duplicate, the case is not written to the database if the user resolves it as a duplicate.

Configuring duplicate case search

For instructions on how to configure the duplicate case search feature, see the Help topic Tracking duplicate cases.

DATA MODEL DESIGN

Configuring a localizable list of values

Introduction to configuring a localized list of data values

In this lesson, you learn how to use Field Value rules to define items in a selection list presented to end users. This enables you to restrict the values of a property to one of a fixed list of choices.

After this lesson, you should be able to:

- Describe how field values are used to define allowed data for properties
- Configure a field value rule
- Add a list of field values to a property rule

Field values

When building an application, you often need to use a list of allowed values for a specific property. If the list of allowed values is short, mostly static, and common, for all case types in the application, the list of allowed values may be defined in a local list on the property record.

If the list of allowed values is large, expected to change frequently, or may be specific for each case type, you can use a field value.

Field Values provide an alternate method for defining allowed values for properties. Field values enable you to manage the list of allowed values separately from the property. Managing the allowed values separately from the property enables you to reuse a single property, and customize the allowed values based on the context of the property.

For example, in a Pega 7 application, every case instance has a status, which changes as the case progresses through the case life cycle. The status of a case is set using the property named *.pyStatusWork*. The list of allowed values for setting *.pyStatusWork* is defined using field values.

You can add different field values for a single property in the same context, or in separate contexts using the *Apply to:* class setting for each value. You can also use rulesets to maintain different versions of each field value in each context.

Applies To	Field Name	Field Value	Ruleset:Version
Work-	pyStatusWork	Open	Pega-ProCom:07-10-01
Work-	pyStatusWork	Pending	Pega-ProCom:07-10-25
Work-	pyStatusWork	Resolved-Completed	Pega-ProCom:07-10-25
TGB-HRApps-Work-Candidate	pyStatusWork	Open-Scheduling	HRApps:01-01-01
TGB-HRApps-Work-Onboarding	pyStatusWork	Open-Enrollment	HRApps:01-01-01
TGB-HRApps-Work-Onboarding	pyStatusWork	Pending-Enrollment	HRApps:01-01-02

In the table above, the Pega-provided property used to set the status of a case — *.pyStatusWork* — uses a common set of allowed values as defined in the Work- context. This common set of allowed values is available for all applications built on the Pega platform.

Additional custom values are defined for the HRApps application. These values are usable in the *TGB-HRApps-Work-Candidate* and the *TGB-HRApps-Work-Onboarding* contexts (case types).

Field values also support localization of words, phrases, and sentences that appear on portal displays, reports, and user forms.

KNOWLEDGE CHECK

> A field value enables you to manage the _____ separately from the _____.

list of allowed values, property.

KNOWLEDGE CHECK

> True or False: You can add different field values for a single property in the same context.

True

How to configure field values

You can create field values to restrict the values of a property to a list of allowed values.

First, organize a list of allowed values you want to display in the list for the property. Next, create a *Field Value* record for each allowed value. In the record, enter the value you want to display. Then, identify the appropriate *Apply to:* class of the property where you want to display the list. Finally, associate each Field Value record to the property where you want to display the allowed value.

Identify the values you want use

Use values that are meaningful to the user so the purpose of each value in the list is clear. For example, to prepare a list of values used to select a state or territory, use standard abbreviations or the full name of each state or territory (TX or Texas; KA or Karnataka; AA or Alsace).

Create a Field Value record for each allowed value

Create a new *Field Value* record for each allowed value.

In the **Label** field, enter one of the allowed values you organized.

In the **Apply to:** field, select the class where you want to apply the allowed value.

In the **Field Name** field, select the property on which you want to display the allowed value.

About Field Values provides details on how to create a field value record.

Configuring data access patterns

Introduction to configuring data access patterns

Pega includes powerful data capabilities to improve the speed and quality of development.

This lesson introduces data access patterns — these are used when creating the data model and integration that support the Pega application. Patterns provide a common taxonomy for Pega solutions promoting scalability and reuse. This lesson covers five data access patterns.

After this lesson, you should be able to:

- Describe the data access patterns available in Pega and their uses
- Configure a property to refer to a data page
- Configure a property to copy a data page
- Configure a data page with keyed access
- Reference a data page from a UI control
- Configure a reference property

Data access patterns

Data access patterns provide simple mechanisms to help technical staff effectively manage data in a Pega application. This section introduces five data access patterns that are available in Pega. These patterns may be used as a starting place to create solutions for specific situations.

This section provides an overview of how each pattern may be tailored for a Pega application. The best way to master these patterns is to experiment with them in different situations.

System of record pattern

The System of Record (SoR) pattern is used when a case needs access to data that is stored in another system or application. The case does not own the reference data. Data is referenced as needed by the application for context or to populate rules. For example, a loan application or credit card dispute accesses customer account information and history.

Data loaded from the SoR pattern usually comes from an external data source. The SoR pattern helps to ensure that data displayed in the Pega application is current. For example, if the account holder has a new phone number, the updated number is displayed when the data is accessed from the case.
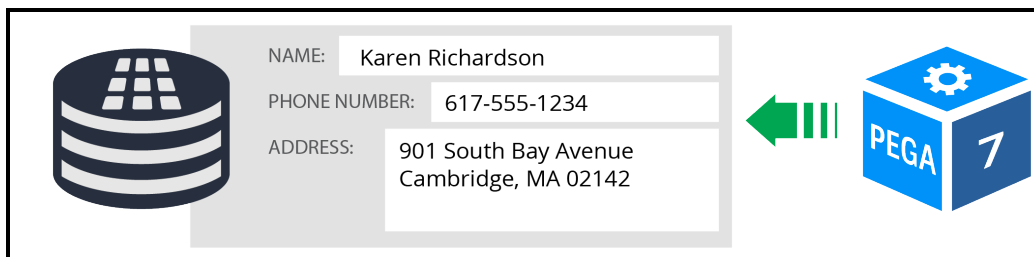
Snapshot pattern

The snapshot pattern copies data into the case. Once the data is copied into the case, the data is not retrieved from the source again unless the request parameters change.

The snapshot pattern may be used to provide a copy of the data at a specific point in time. For example, for an insurance claim, the snapshot pattern provides a copy of the policy data at the time the claim is filed. If the policy changes after the claim, you do not want to update the policy data. This is the opposite of the SoR pattern discussed earlier.
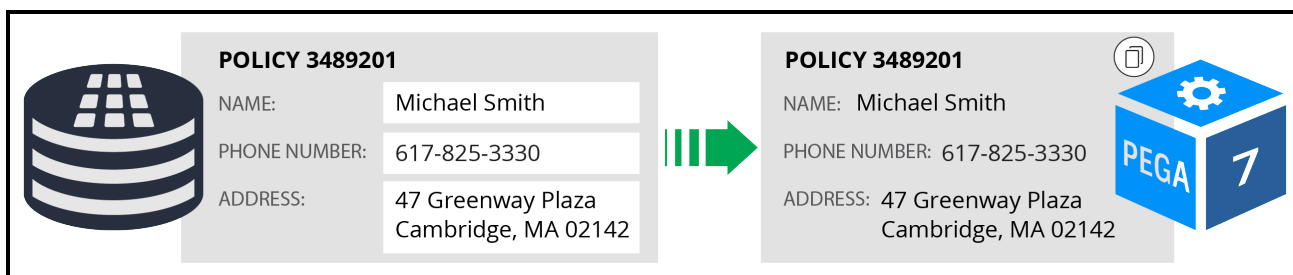
Reference pattern

Reference data is a simple pattern used frequently in Pega applications. This pattern references data that is usually not directly connected to a given case. Therefore, the data is not part of the data model.

Examples of data in a reference list include:

- Products
- Automobile makes and models
- Drop-down values (for example, countries and states)

The data can be used by multiple cases or applications. In many cases, the data is used to populate user interface controls.
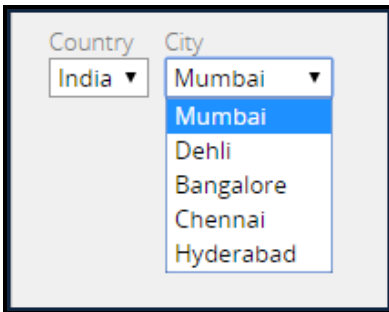
Keyed access pattern

The keyed access pattern is used to populate a list of objects and access information about a specific object in the list. When retrieving a list of objects, the details for each object are retrieved as part of the list. This eliminates the need for subsequent calls to retrieve object details.
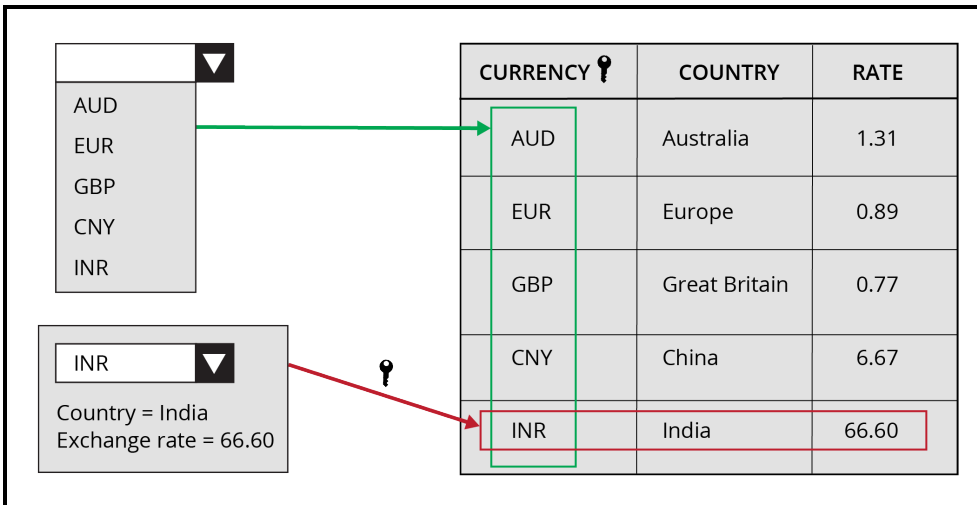
The keyed access pattern is appropriate when users need to frequently switch back and forth between objects in a large list, such as when loading a list of currencies and exchange rates.

When appropriately applied, the keyed access pattern significantly improves application performance and maintainability.

Alias pattern

The alias pattern links one property to another — it is an alias for a property.

For example, in an auto insurance quoting application is a data structure with two lists. One list includes drivers on the auto insurance policy. The other list includes vehicles covered by the policy. Each vehicle has a primary driver. Use the alias pattern to link each vehicle to a driver in the driver list.
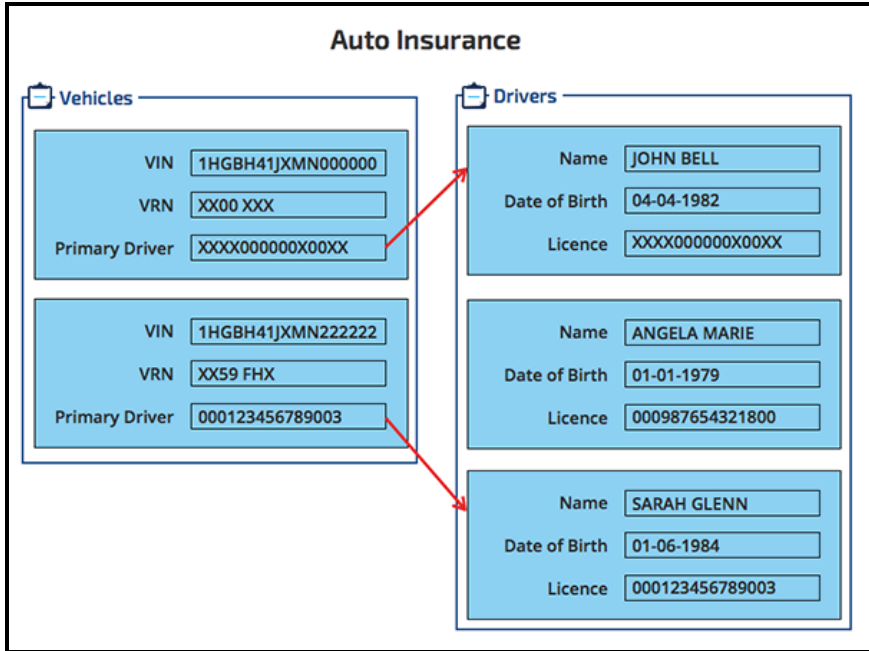
The alias pattern prevents duplication of data, streamlines data management, reduces memory footprint, and improves performance.

How to configure the reference pattern

To implement the reference pattern, reference a data page directly in your application. For example, a data page can be referenced in a drop-down control.

The data pages are used directly in the drop-down control and are not part of the data model. When implementing the reference pattern, data pages are often node level since the data is not linked to a specific case and can be shared.

KNOWLEDGE CHECK

To implement the reference pattern, use a data page in your application without defining it as part of the _____.

data model

How to configure the SoR pattern by referencing a data page from a property

If you need to reference the most current data in an external system, use the System of Record (SoR) pattern. For example, use the SoR pattern if you are building a seating request application. Every time the case is accessed, the user must see up-to-date information for the seating assignment transaction.

To implement the SoR pattern, you reference a data page from a property rule. The type of property (page or page list) must correspond to the structure of the data page (page or list). A new data page is created on the first reference to the property.

Data is not stored in the property that refers to the data page. The property only contains a reference to a data page, as illustrated in the following screenshot of the clipboard.

Reloading the data

The data reloads according to the refresh strategy specified on the data page. The property always points to the current version of the data page.

Whenever a data page parameter is updated, a new data page is created. The property then points to the new page.

KNOWLEDGE CHECK

⌄ ANSWER	Data is, or is not, stored in the property that refers to the data page?
A: is not	

©2017 Pegasystems Inc.

How to configure the snapshot pattern by copying data from a data page

If you want to populate a property with data from a specific point in time, use the snapshot data access pattern. For example, you may opt to use a snapshot pattern for an insurance claim case. When the business user creates a new insurance claim, the customer's insurance coverage information is copied into the claim from an external system. Each time the case is opened, the insurance coverage information recorded in the case remains unchanged.

To implement the snapshot pattern, use the **Copy data from a data page** option on a property. The type of property (page or page list) must correspond to the structure of the data page (page or list).

The first time the property is referenced, the data page is created and the data is copied to the property. You may also choose to specify a data transform for data mapping in **Optional data mapping**.

When you copy data from a data page, the data is stored in the property. The data page is not accessed again unless it has a parameter that changes. When the parameter changes, a new data page is created. The impacted data is copied to the property and overwrites the existing data.

KNOWLEDGE CHECK

When you copy data from a data page, the data is refreshed only when _____.

How to configure the keyed access pattern using keyed data pages

The keyed access pattern serves as an alternative to having two separate data pages. Follow these steps to configure keyed data pages:

1. Define the data page **Structure** as a List.
2. Select **Access pages with user defined keys**.
3. Select **Allow multiple pages per key** to filter a large list to create a smaller list.
4. Specify the **Page list keys** used to access the list entry.

In the following example, the currency code is defined as the key for a data page of currency conversion rates.

If you use the data page without a key, all of the exchange rates are displayed, for example, in a drop-down.

Note: There is no option to specify keys.

Using the data page with a property allows you to specify keys. In this example, the application displays the exchange rate for a specific currency if you provide a key.

KNOWLEDGE CHECK

ANSWER
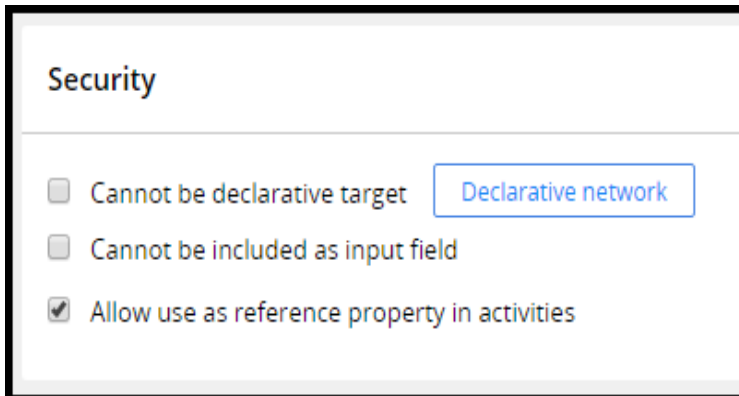
A keyed data page serves as an alternative to having _____.

two separate data pages

How to configure the alias pattern using reference properties

Use reference properties to implement the alias patterns. On the property, select the **Allow use as reference property in activities** option. Reference properties can reduce the need for data duplication. They can also simplify lengthy property references. The setting can be found on the Advanced tab of the property rule.

You need an activity to set up the property reference. Use the *Property-Ref* method to create the link between the source and target properties. Once linked, the references are maintained until the link is explicitly broken or changed using the *Property-Ref* method.

 Note: Property references cannot be circular.

In this example, two embedded page lists are defined in the case type. One list includes drivers on the policy. The other list includes vehicles covered by the policy. Each vehicle has a primary driver that links to an entry in the list of drivers on the policy.
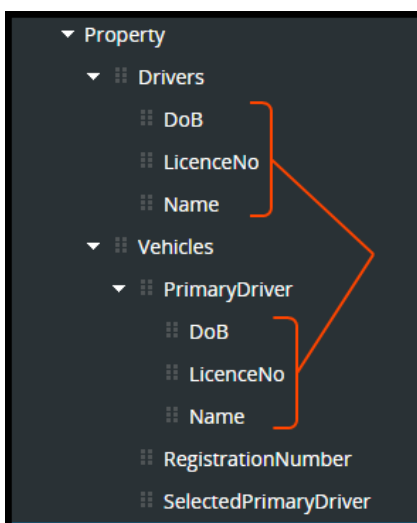
Insurance representatives need to see primary-driver details such as name and license number together with vehicle data. Use reference properties to link the primary driver to an entry in the list of drivers on the policy. Using reference properties, you can access driver details from the vehicle page since the driver page appears as if it is part of the vehicle page.
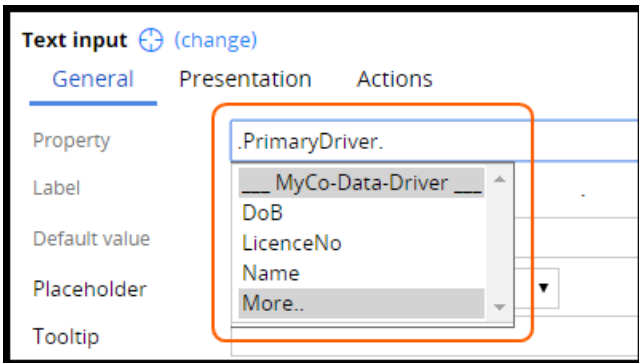
Use the **Property-Ref** activity method to set up the reference property.
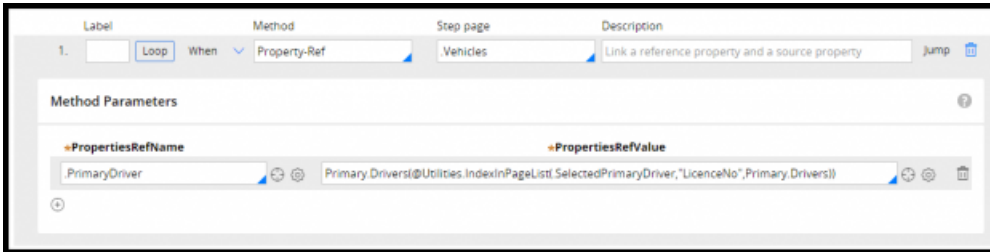
On the clipboard is a reference from the *PrimaryDriver* page property to the driver page in the *Drivers* page list property.
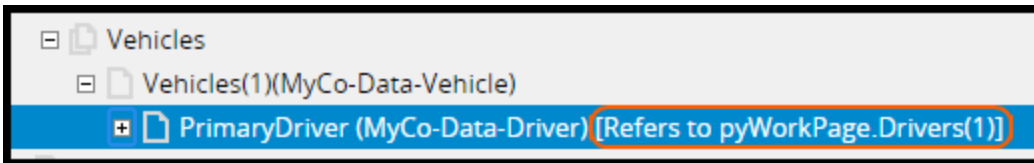
Reference properties are not common, but they can be powerful in more advanced data structures that require linking of embedded entities. Reference properties can help improve run-time performance, and make design time easier by making property references simpler and more intuitive.

KNOWLEDGE CHECK

A reference property links one property to _____.

another property

PROCESS DESIGN

Creating organization records

Introduction to creating organization records

Pega 7 supports a three-level organizational structure. In this lesson, you learn how to use the Organization Chart and organization records to modify the organizational structure. You also learn how to use Dynamic Class Referencing (DCR) to reference objects in different classes.

After this lesson, you should be able to:

- Describe the relationships between parts of the organizational structure
- Differentiate between a work group and an organizational structure
- Modify an existing organizational structure
- Create a work group and a workbasket
- Describe Dynamic Class Referencing (DCR)
- Update a workbasket name using DCR

Organization records

A Pega application uses an organizational structure to direct assignments to the right operator or workbaskets, determine the access rights of operators, and report on activity in various company departments. For example, in order to approve a home loan, the request has to move through different levels in the organization, starting at the customer representative all the way to the approving manager. The structure is built with standard data instances called **organization records**. You use these rules to support your requirements, and you can find these rules in the Records Explorer in the Organization category.

The Pega organizational structure is a three-level hierarchy. The top level is known as the **organization**, the middle level contains **divisions**, and the lowest level contains organization **units**.

A hierarchy has only one organization, and this represents the entire enterprise. A company such as Apple or IBM would map to an organization in the hierarchy. An organization can have one or more divisions. A division can be defined as a layer for categorizing business units such as a region, a brand, or a department. Divisions can have one or more units. A unit contains operators who perform work specific to their organization. These operators can include caseworkers, agents, and customer service representatives. A unit can have child units. For example, two units (such as Hardware and Software) may report to a single unit (Internal) which belongs to the IT division. The Internal unit manages IT requests from inside the organization. The child units handle the requests and inventories and ships the items.

 Note: A division cannot have subdivisions.

The following chart shows a sample organization and its hierarchy. The OurCo organization has three divisions — Sales, IT, and HR. Each division has its own units. In this example, IT has two units named Internal and Customer. The Internal unit has two units that report to it — Hardware and Software.
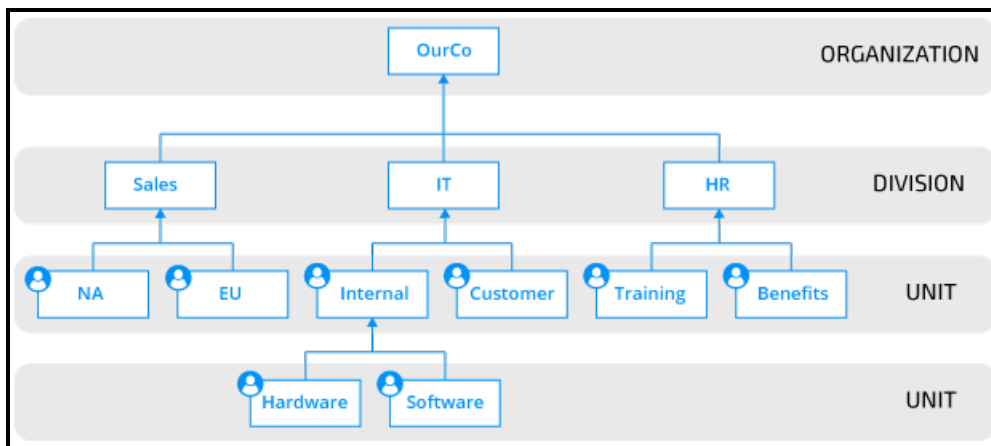
 Note: The Pega designations of organization, division, or unit do not necessarily coincide with the company's official organizational chart. Instead, organization rules refer to the work that users do and the levels of access they have in Pega applications.

An operator is associated with a unit, division, and organization. The operator ID record (also an organization rule) stores the operator's organizational structure. You can update the operator's organization structure, if necessary.

A workbasket is an organization rule, but is not part of the organizational hierarchy. A workbasket belongs to a unit, a division, and an organization. By default, an operator can access work in the workbasket to which the operator belongs.

> **Note:** A workbasket is sometimes referred to as **work queue** because it provides a queue of open assignments available to multiple operators. For more information, see the Help topic [Routing an assignment to a work queue](#).

You can use the organization structure in various ways:

- You can define access groups available to operators at the organization or division levels. These access groups apply only if operators do not have an access group defined in their operator ID record.

- You can configure routers that send assignments to operators or workbaskets defined within the organizational structure.

- You can use manager data associated with the organization records in routing configurations, approval processes, and SLAs. For example, an operator ID lists a manager the operator reports to, and a unit record specifies a unit manager. The operator's manager can report to the unit manager. For example, you can route a purchase request to a manager if the amount is up to USD10,000. If the amount exceeds USD10,000, you can route the request to a unit manager.

- You can configure SLA configurations. Since each assignment contains the organizational data of the assigned operator, you can use the data to determine escalation actions to specific managers. For instance, if a purchase request is past goal, the case is escalated to the operator's manager. If the case is past deadline, the case is escalated to the unit manager.

KNOWLEDGE CHECK

ANSWER   When you associate an operator with an organization hierarchy, which rules do you use?
Organization, Division, and Unit

Work groups

A **work group** identifies a cross-functional team that contains a manager, and a set of operators, a set of workbaskets, or both. You create work groups so that resources can be shared among units, divisions, or the entire organization. Even though operators and workbaskets belong to a unit, associating an operator with a work group allows the operator to share work with another operator in other units. In the following example, the Onboarding work group is associated with the Human Resources (HR) unit and Training unit. Because each unit has its own default workbasket, the work group contains both workbaskets. Operators in the HR unit and Training unit are associated with the work group and share work. One operator in the Accounting work group is also associated with the Onboarding work group. This allows the operator to share work with the other operators in the Onboarding work group.
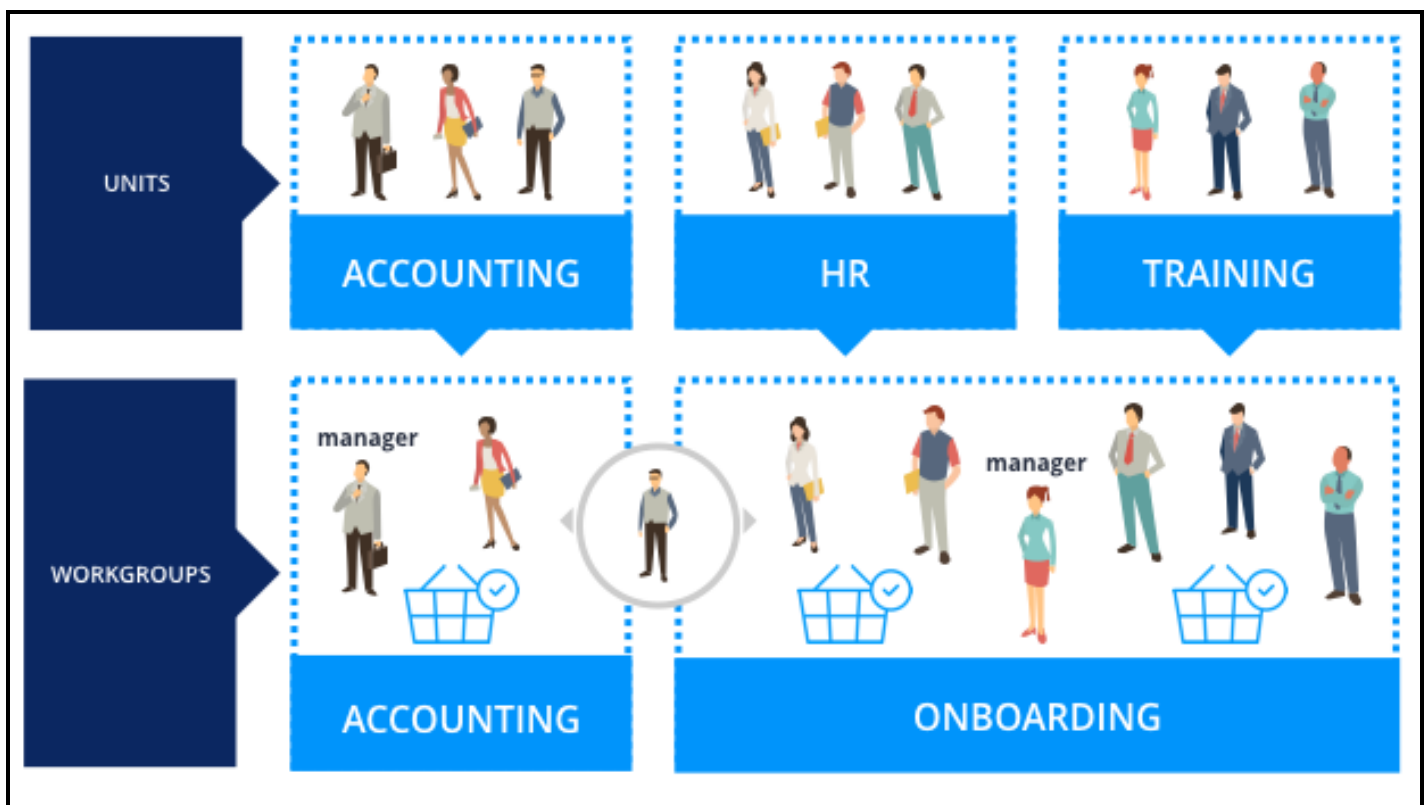
A work group is an organization rule but not a level of the three-level hierarchy. An operator must be associated with at least one work group and can belong to more than one work group. A work group contains one workbasket.

A work group instance identifies one user who is the work group manager. The system can use manager information in a work group for notification tasks and routing tasks. Work groups give managers the flexibility to assign, monitor, and report on work performed in each work group. Managers use the Case Manager portal to access the work being performed in work groups. The Case Manager portal refers to work groups as **teams**. The following image shows work groups displayed as Teams in the Case Manager portal.

Teams contain operators. In the Case manager portal and in work queues, operators are referred to as Members. The portal allows managers to drill down and monitor work for each team member and workbaskets in a team. For instance, a manager can select a member's icon to open the member's worklist. If the member belongs to more than one team, the manager can see the items the member is working on in all the teams. The manager can also add or delete operators and workbaskets in the **Members** and **Work queues** sections, respectively.

KNOWLEDGE CHECK

Which of the following is not part of the organizational structure: Division, unit, work group, or organization?

A work group is not part of the organizational structure. Although it is an organization rule, a work group is used to allow managers to monitor, report, and assign work among operators and workbaskets across the organization.

How to update an organizational structure

As companies grow, system architects may extend the existing organizational structure to reflect organizational change. You may have to add new organization levels, work groups, and workbaskets. You associate operators with an organization structure in order to define how work is routed to the operator.

When you create a new application using the New Application wizard, you specify an organization. Pega provides a default division and unit for that organization. For example, if you enter TGB as the organization, the system creates a division named Div and a unit named Unit. You have the option to name the division and unit you want in your new organization.

The example in this topic assumes that an organization has a Human Resources (HR) unit. Due to business expansion, the company intends to add a unit named Benefits that reports to an HR unit. Operators belonging to the unit will also be in a new work group that contains a new workbasket.

Follow these steps to update the organization in this example:

1. Add a new unit to the hierarchy.

2. Create a new work group.

3. Create a new workbasket.

4. Associate the workbasket with the work group.

5. Associate the unit and work group with an operator.

Important: Because of mutual dependencies, a workbasket or a work group must already exist before either instance can be created. You cannot create both the workbasket and the work group at the same time. For example, when you create a new work group, you must use an existing workbasket — this is required value. After you create a new workbasket, go back to the work group and update the workbasket.

The following image shows the sequence of steps using the previous example.

Adding a level to the organization hierarchy

Use the chart on the Organizational Chart landing page to add levels to an organization. The chart helps you see where in the hierarchy you are adding a level. From the Designer Studio menu, select **Org & Security > Organization > Organizational Chart** to open the landing page. You can also access organizational charts on the **Chart** tab on Organization, Division, and Unit records.

To add a layer, select an entry in the organizational hierarchy. Right-click to display a menu that lets you add organizations, divisions, and units. Organization names are unique and cannot be used in more than one hierarchy. For example, a division cannot belong to more than one organization, or a unit cannot belong to more than one division.
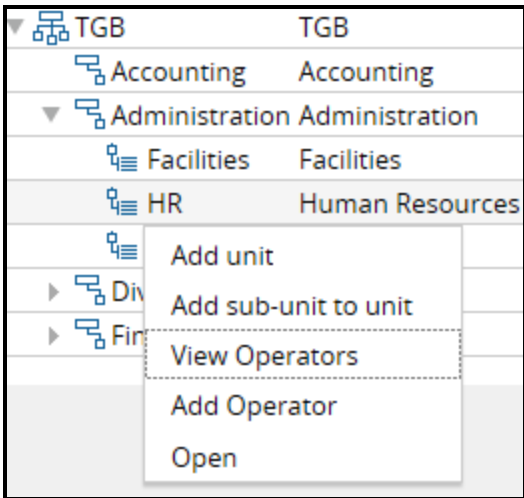
In this example, a subunit named Benefits has been added to its parent unit, HR.
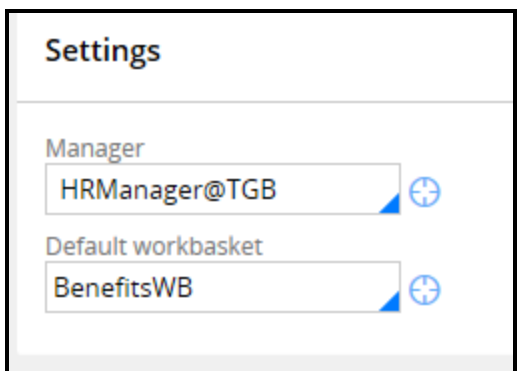
For more information on how to use the chart, see the Help topic Organizational Chart.

Adding a work group

From the Records Explorer, create a work group record. By convention, end the work group name with an at sign (@) and an organization name, such as Benefits@TGB. In the **Manager** field, enter a work group manager for reporting and routing purposes. In the **Default workbasket** field, select a workbasket. The standard router activity *ToDefaultWorkbasket* uses this value to locate a workbasket, based on the work group associated with the current operator. For example, you can enter the default organization workbasket such as *default@TGB*. After you create a new workbasket for the work group, you update this default workbasket with the new one.
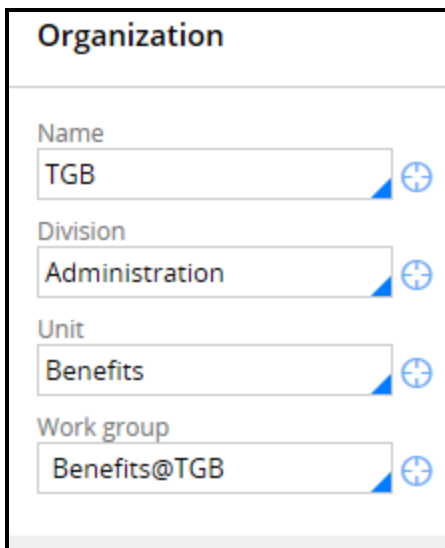
Save the work group record but leave it open. Next, enter the name of the new workbasket in the **Default workbasket** field. To help users and developers easily distinguish workbasket and operator identifiers, choose a naming convention specifically for workbaskets. For example, include WB as a part of the name as shown in the following example. Then, click the **Crosshair** icon to open a Create Workbasket form.

Adding a workbasket

Complete the New Workbasket form and create the record. In the Organization section on the workbasket form, first select the organization in the **Name** field, and then complete the **Division** and **Unit** fields. In the **Work group** field, enter the name of a work group that uses this workbasket. This field determines which workbaskets appear in a Team's Work Queues list on the Manager portal. Save the record to associate the workbasket with the work group.

Return to the open work group record and save it. The workbasket you created is now the work group's default workbasket.

Update the operator record

Open an operator record that you want to associate with the new level. On the **Work** tab, update the **Organization unit** values. Then, in the **Work group** field, select the work group you want to associate with the operator. Finally, add the work group to the **Workbasket** field.
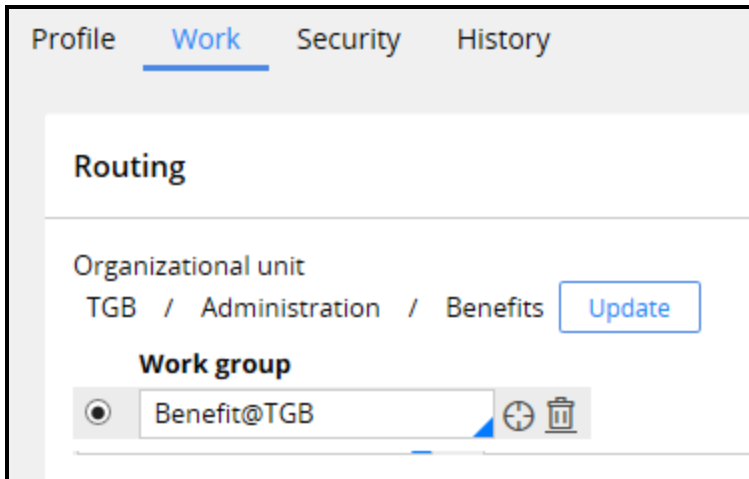
KNOWLEDGE CHECK
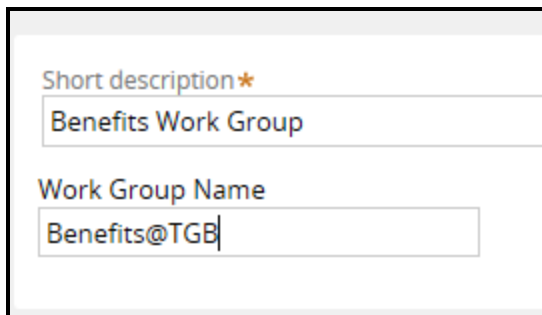
A requirement states that you must add a new division to your organization. What approach allows you to see where in the organization hierarchy you are adding the division?

Select the organization in the organization chart and use the chart menu to add the division.

Creating a work group and a workbasket
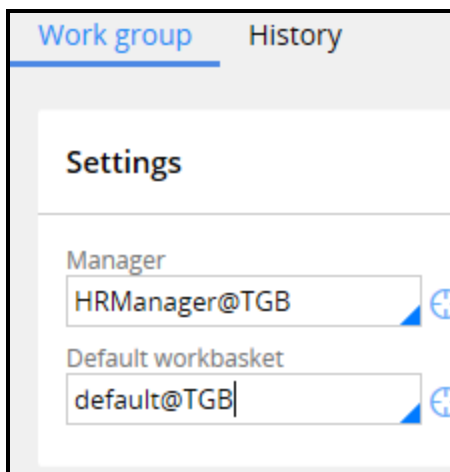
A workbasket must be associated with a work group. Similarly, a work group must be associated with a workbasket. In some situations, you must create both a new workbasket and work group that are associated with each other. Follow these steps to create a work group and a workbasket, and then associate the workbasket with the work group.

1. In the Records Explorer, select **Organization > Work Group** and click **+Create** to display the Create Work Group form.

2. In the **Short description** field, enter a description.

3. In the **Work Group Name** field, enter a work group name. Remember to use the standard naming convention by adding an at symbol (@) and the organization name. In the following example, the Work Group name is **Benefits@TGB**.
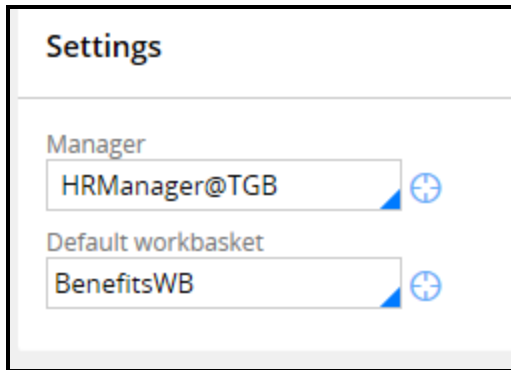
4. Click **Create and Open**.

5. In the **Manager** field, on the **Work group** tab, in the Work Group form, select a work group manager.

6. In the **Default workbasket** field, select a default workbasket. In the following example, the workbasket is the default for the organization, **default@TGB**.
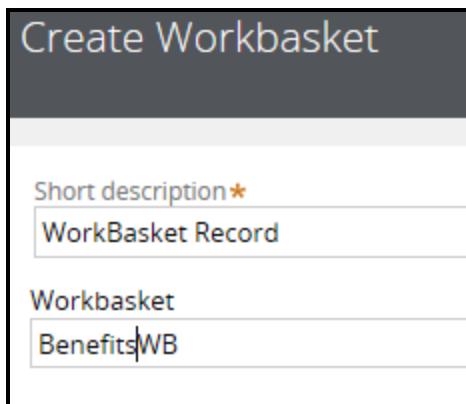
7. Click **Save** to create the new work group.

8. In the **Default workbasket** field, in the Work Group form, overwrite the value with the name of the new workbasket. Use a naming convention that indicates the record is a workbasket. In the following example, the new workbasket is **BenefitsWB**. Later, you will enter this name in the Create Workbasket
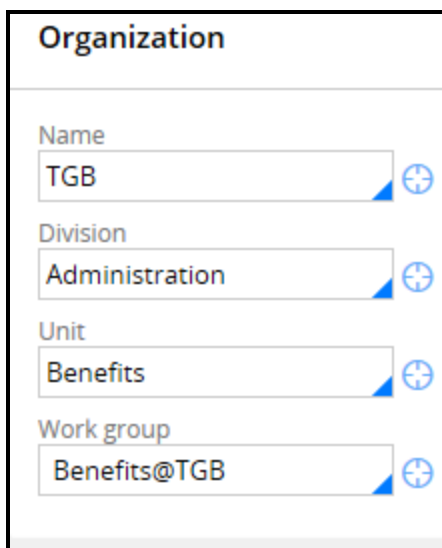
©2017 Pegasystems Inc.

form.

9. Click the **Crosshair** icon to open the Create Workbasket form.

10. In the **Short description** field, enter a description.

11. In the **Workbasket** field, enter the name you entered in the Work Group form.

12. Click **Create and Open**.

13. On the Workbasket form, on the **Workbasket** tab, select the required organization levels in the **Name**, **Division**, and **Unit** fields.

14. In the **Work group** field, select the work group you created.

15. Click **Save** to create the new workbasket.

16. Return to the Work Group record and click **Save** to associate the new workbasket with the work group.

How to customize reusable processes with Dynamic Class Referencing

Many framework properties and data instances are meant to be reused without modification in the implementation layer. For example, a process for approving a loan request in the framework layer may be reused as-is without having to copy the process into your implementation layer.

However, you may have to make minor modifications. For example, you may want to use a workbasket defined in your implementation, not in the framework, for routing assignments. If the framework is locked, you have to copy the entire process to make the update.

Copying from the framework to the implementation layer can increase rule maintenance costs. Whenever framework rules are updated, you must update the copy in your implementation. This process can be especially time-consuming with multiple implementations.

Dynamic class referencing (DCR) is a Pega design pattern that enables you to easily reuse framework rules without having to copy the rules into implementation layers. DCR uses a data page in the framework layer that holds properties typically sourced from the data transform. In your framework, you configure rules that reference the data page property. In your implementation, you specify the data page and the property you want.

DCR is especially useful when you make small modifications to implementation rules you are reusing. In the following example, the only differences between the processes is that the assignments use different workbaskets in their routers. You can use DCR to reference the workbasket property in the data page. This means that you do not have to copy the flow rule and manually update the workbasket value in the router. Using DCR, the system runs the framework flow and references the workbasket used in the implementation class.
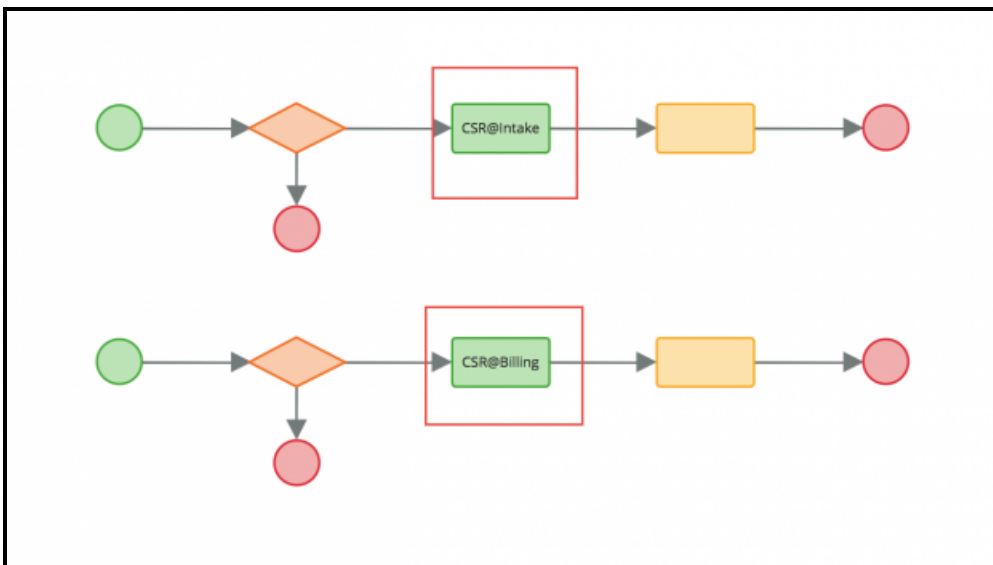
The use of dynamic referencing is not limited to frameworks. You can configure the DCR data page and data transform in the implementation layer. For instance, a claims application can process both home and auto insurance claims. Some of the flows can be reused as-is for both claim types. The flows are located in the class group.

However, each claim type may require minor customization. For example, you may have a step in which cases are routed to a workbasket. The workbaskets in each case type differ depending on the claim type or other parameters, such as the organizational unit. You can configure DCR in each case type to handle the differences. When you use this approach, you do not have to specialize the flows in each case type.

KNOWLEDGE CHECK

You want to promote reuse of framework assets using DCR. Where would you put the required extension properties?

In the framework layer

Configuring DCR

When you configure a framework DCR, you first create a data class in the framework that contains the properties used as the dynamic reference variables. This approach lets you easily find the extension points that need to be configured when extending the framework. In the following example, the Ins-FW-ClaimsFW-Data-AppExtension data class contains a property (.ManagerWB) that is used as a variable.
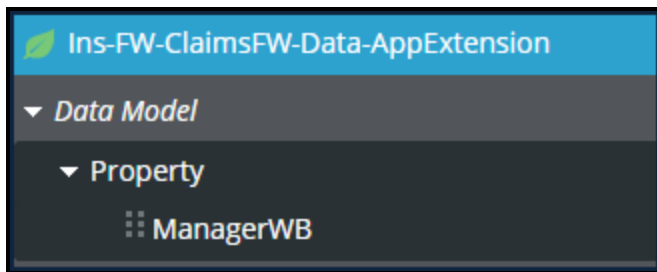
You then create a page property in the framework class group. This approach makes the property available to all application layers. In the following illustration, the page property, *AppExtension*, is in class group *Ins-FW-ClaimFW-Work*. The value in the **Page Definition** field is the data class holding the property. The **Data access** section refers to a data page holding the extension values. In this example, the data page is named *D_AppExtension*.

Property: AppExtension [Available]

CL Ins-FW-ClaimsFW-Work | **ID** AppExtension | **RS** ClaimsFW:01-01-01

General   Advanced   History

Property type

Single Page

Page definition *
Ins-FW-ClaimsFW-Data-AppExtension

Data access

○ Manual
● Refer to a data page
○ Copy data from a data page

When data page parameters change and a reference to a non-parameter value is made, the system loads a new data page. Reads from and writes to the property occur on the new data page.

Data Page: *
D_AppExtension   Ins-FW-ClaimsFW-Data-AppExtension

Parameters
No parameters to set.

☑ Save parameters with this property for access on reopen

In the data page holding the extension values, you specify the data source. You typically use a data transform as the data source. You can alternately use other data sources such as the results of decision rules or declare expressions. For example, you can use a decision table to determine the correct property based on an organization unit. In the following example, the data page is sourced from a data transform.

Data Page: D_AppExtension [Available]
ID D_AppExtension | RS ClaimsFW:01-01-01

| Definition | Load Management | Parameters | Pages & Classes | Usage | History |

Data page definition

Structure

Page ▼

Object type *

Ins-FW-ClaimsFW-Data-AppExtension

Edit mode

Read Only ▼

Scope

Thread ▼

Data sources

1 Source *
   Data Transform ▼

   Data transform name *
   AppExtension 🔍
   Parameters

Add New Source

The data transform is in the framework data class but is overridden in each application ruleset. You set the target to the variable property defined in the framework data class. In this example, the target is .ManagerWB.
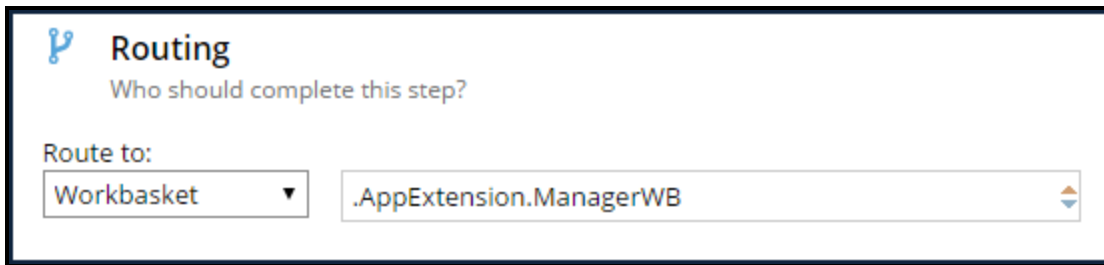
Data Transform: AppExtension [Available]   🔒 Check out   Save as ▼
CL Ins-FW-ClaimsFW-Data-AppExtension   ID AppExtension | RS Claims:01-01-01

| Definition | Parameters | Pages & Classes | History |

	Action	target	relation	source
• 1	Set ▼	.ManagerWB 🔍	equal to	"AppManagerWB"

When you reference the target property, you first specify the framework data class and then the property. In the following example, the assignment is routed to .ManagerWB in the framework data class .AppExtension.

At run time, the process uses the data page value on the clipboard and routes the assignment to the workbasket specified in the implementation layer instead of the workbasket specified in the framework layer.

Configuring a cascading approval process

Introduction to configuring a cascading approval process

Pega provides the ability to include single and cascading approvals when designing case types. This lesson focuses on how to convert a single approval to a cascading approval.

After this lesson, you should be able to:

- Differentiate a cascading approval from a single approval
- Identify the appropriate cascading approval model
- Configure a cascading approval process

Cascading approval

Approvals vary depending on the case type. An application for auto insurance may need one approval from the company underwriter. Some case types, such as a purchase request or an expense report, may need a series of approvals. A **cascading approval** process configures a series of approvals.

The two cascading approval models are **reporting structure** and **authority matrix**.
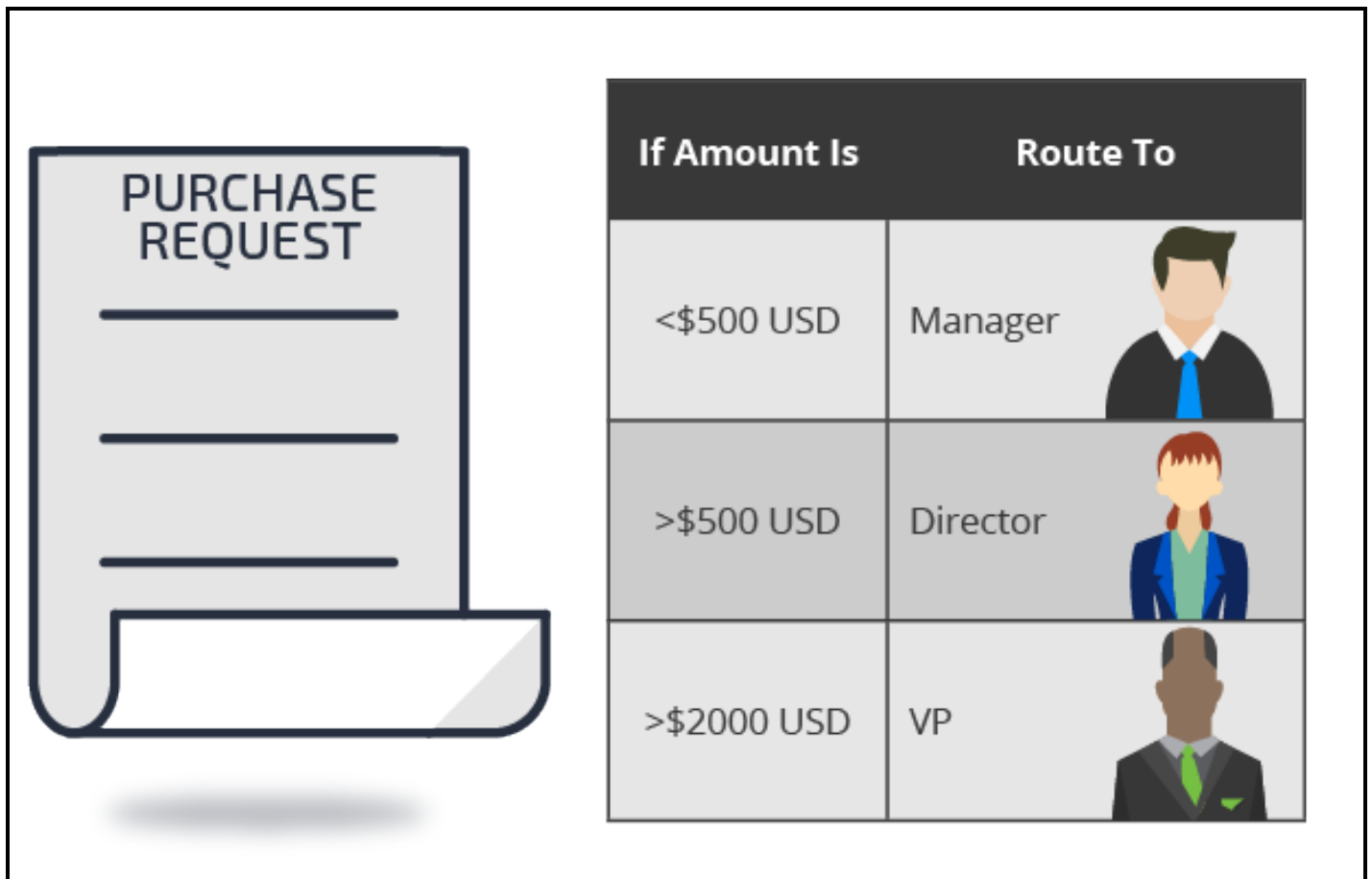
The reporting structure model works when approvals always move up the submitter's reporting structure or another defined list.

The authority matrix model works when the approval chain is directed by a set of rules to individuals both in and out of the submitter's organization.

To determine which model to use, write the requirement in a simple sentence, beginning the sentence with "When". Following are examples for each approval model.

Reporting structure example

When an employee submits a purchase request, the employee's direct manager must approve the request. Amounts over USD500 require more approvals from people above the manager.

If Amount Is	Route To	
<$500 USD	Manager	
>$500 USD	Director	
>$2000 USD	VP	

The purchase request example contains no conditions. Every time an employee submits a purchase request, the request must be approved by a defined list of approvers. When there are no conditions, use the reporting structure model.

Authority matrix example

When an instructor submits an expense report, if a line item is billable to a customer, then accounts payable must approve the line item.

The expense report example contains a condition. If an expense item is billable, then accounts payable must approve the expense item. When conditions exist, use the authority matrix model.

KNOWLEDGE CHECK

What is the difference between the authority matrix and reporting structure cascading approval models?

The authority matrix model works best when the approval chain is directed by a set of rules to individuals both in and out of the submitter's organization. The reporting structure model works best when approvals always move up a defined list.

How to configure cascading approval

When an application requires multiple approvals of a case based upon specific criteria, you add a cascading approval step in your case life cycle.

Consider a requirement for approvals of a purchase order based on the value of the order.

Purchase request value up to	Approval requirement
USD25,000	Cost Center Manager
USD75,000	Vice President
USD100,000	Vice President of Finance
USD500,000	Senior Vice President
over USD500,000	Chief Financial Officer

You use the criteria in the table to configure a process for routing a case through multiple approvals. For example, the Cost Center Manager, the Department VP, VP of Finance, and Senior VP must approve a USD110,000 purchase request.

The approvals proceed sequentially through the hierarchy. For instance, after the Cost Center Manager approves the request, the case is routed to the Vice President.

To configure a cascading approval step, you first add an approval step to a stage. Then, you specify Cascading as the approval type. Finally, you choose the approval model: a reporting structure or an authority matrix.

Note: You can also configure a cascading approval in a process flow by selecting the Cascading Approval Smart Shape from the **Smart Shapes** menu in the process modeler.

Reporting structure

The Reporting structure has two default single-level modes for routing an approval — either the submitter's reporting manager or work group manager. The manager information is defined in each submitter's operator ID and associated work group. If the operator ID lists more than one work group, Pega uses the default work group to determine the work group manager.

You can assign levels of approval to route cases up the organization's management hierarchy:

- One — This is either the reporting manager or the work group manager you specified.

- All — This is the submitter's entire manager hierarchy.

- Custom — This is a specified number of levels as determined by the evaluation of when rules. Specify a manager at each approval level and define a cut off with a when rule for each manager.

The following example shows a cascading approval configuration based on the reporting structure, with two when rules used to determine the number of approvals needed.

Authority matrix

The authority matrix model determines the approvers using a list of operators stored in a Page List and a single value property that identifies the approver. In most situations, you use a decision table to define conditions for populating the list. When a request is made, the system populates the approver list with the operators who evaluate to true in the table. The following image shows an example of a cascading approval based on an authority matrix.

KNOWLEDGE CHECK

What is the purpose of a cascading approval step?

A cascading approval step simplifies the configuration of a process that requires multiple approvals. You can base the cascading approval on a reporting structure or on an authority matrix.

Configuring cascading approval

When configuring a process that requires multiple approvals, you use a cascading approval step.

Prerequisites to configuring a cascading approval

Prior to configuring a cascading approval, determine the cascading approval model (reporting structure or authority matrix) to implement and identify the approval criteria.

If you choose the reporting structure model, determine the required number of approval levels. If the number of levels varies based on case data, such as the amount of a purchase request, configure when rules to test the threshold for each approval level.

If you choose the authority matrix model:

- Configure a page list property to hold the list of approvers.

- Configure a single-value property as an element of the page list to identify each approver in the list.

- Optionally, configure a decision table to determine the conditions for populating the page list. For example, you can use the property *.ApproverID* to identify the approver for each cost level.

Conditions		Actions
○ **TotalPRCost >=**		○ **Approver ID**
○ **when** 0	→	"CSM@Pco.com"
○ **when** 25000	→	"VP@Pco.com"
○ **when** 75000	→	"VPFinance@Pco.
○ **when** 100000	→	"SVP@Pco.com"
○ **when** 500000	→	"CFO@Pco.com"
○ **when** 500001	→	"CEO@Pco.com"
otherwise	→	""

> **Important:** When using a decision table with an authority matrix, set the decision table to **Evaluate all rows** to return a list of results. Otherwise, the decision table returns only one result.

- If you do not use a decision table, configure a data transform or activity for populating the list if the approvers are the same under all conditions.

> **Tip:** Configure the authority matrix using a decision table if you intend to delegate control of the authority matrix to business users.

See the PDN article Route cases for approval with the Cascading Approval Smart Shape for instructions on configuring properties and a decision table for an authority matrix, and configuring levels of approval for a reporting structure. Note that some descriptions of the Case Designer are out-of-date.

Configure the cascading approval

To add a cascading approval step to your case life cycle, do the following:

1. On the Life cycle tab of Case Designer, hover over a process and click **+ Step**.

2. In the palette that is displayed, click **Approve/Reject**.

> **Caution:** The system adds an alternate stage named Approval Rejection to your case type. Do not delete or rename this stage because users will not be able to reject cases.

3. In the text field that is displayed, enter a unique name that describes the step.

4. On the step panel's General tab, select **Cascading** in the **Approval flow type** field.

5. Refer to the help topic [Configuring approval shape options](#) for instructions on how to configure the approval models. For reporting structure, see the topic section: To use a reporting structure. For authority matrix, see the topic section: To use a list of reviewers.

Prioritizing user assignments

Introduction to prioritizing user assignments
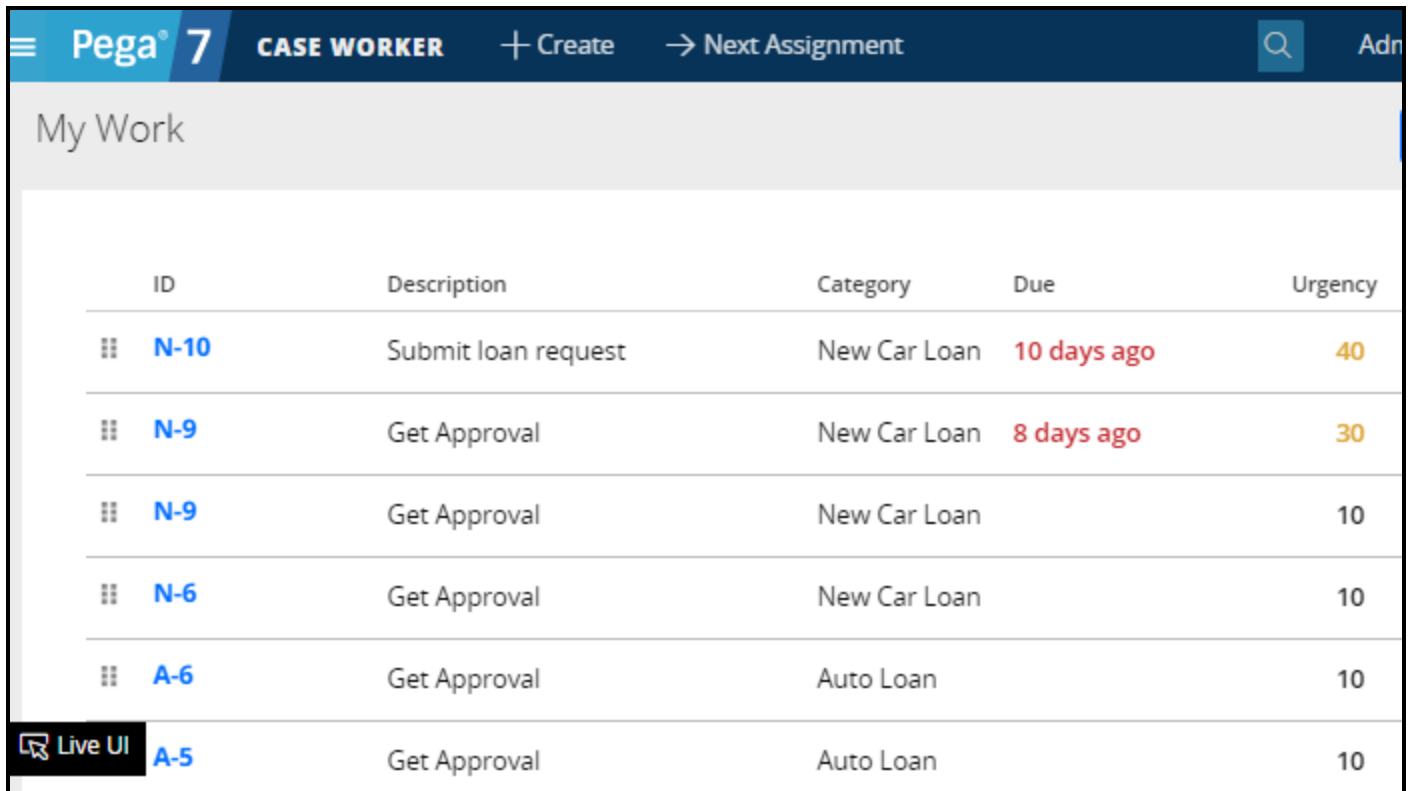
Cases consist of a series of tasks, often assigned to users. Users must perform the most important work first, and only perform assignments appropriate for their skill set. System architects can configure applications to select user assignments, matching each user with an appropriate assignment.

After this lesson, you should be able to:

- Explain the user-selected assignment model
- Explain the system-selected assignment model
- Prioritize assignments using Get Next Work

Assignment models: user- and system-selected

With Pega, end users can choose what to work on next by selecting an item from a worklist. For example, a case worker at a bank may choose to get approval for a new car loan or submit an overdue loan request.

The ability to select work assignments at the user's discretion comprises the **user-selected assignment model**.

Cases often consist of a series of tasks, assigned to users.

Completion of each assignment advances the case through its life cycle.

Assignments can go to a specific user, or to a general queue for a group. In the user-selected assignment model, users choose from many pending assignments. Users must balance two goals: performing the most important work first, and performing work appropriate for the user's skill set.

In the system-selected assignment model, you configure applications to choose user assignments according to importance and required skills.

Get Next Work is the Pega term for system-selected assignment model. Get Next Work matches each user with the appropriate assignments and prevents users from selecting preferred assignments rather than high-priority assignments. For a financial-services firm, you can assign mortgage application reviews to a general queue. Assignments for complex mortgages can be assigned to underwriters with the appropriate expertise.

System architects can configure applications to identify and present the assignment with the most immediate need. The system can prioritize assignments from either the user's worklist or the workbaskets a user can access. Allowing the application to prioritize the most important assignment is the **system-selected assignment model**.

For example, a loan underwriting group pulls work from a common workbasket. The members of the group specialize by loan type. You want to ensure that only group members, who are trained underwriters, process mortgages. System architects use the system-selected model to determine how the system chooses a user's next assignment. The system-selected model ensures that the right user performs the right assignment.

For more information, refer to the help topic [Maximizing user productivity with Get Next Work](#).

KNOWLEDGE CHECK

⌄ ANSWER	What concern does the system-selected assignment model resolve?
System architects can configure applications to identify and present the assignment with the most immediate need. Allowing the application to locate the most important assignment is the system-selected assignment model.	

How to manage assignment selection

Many business processes require storing tasks in a centralized workbasket. Users can then select a task when ready for new work. The concern is that users only select desirable tasks, leaving the less desirable tasks in the workbasket.

You can configure the application so that each time a user completes an assignment, the application automatically selects the most appropriate assignment for the user to work on next.

Pega 7 provides a set of facilities called *GetNextWork* that helps match users to the next most appropriate assignment. The *GetNextWork* functionality considers all eligible assignments to which a user has access.

To control the order in which users complete work, answer the following questions:

- Will the user's worklist be checked for assignments first?
- Will the workbaskets a user is assigned to be checked for assignments first?
- If assignments come from one or more workbaskets, should the list of assignments be consolidated into a single list before being filtered and sorted?

Checking the user's worklist for assignments first

The default behavior of the *GetNextWork* functionality in Pega 7 is to check the user's worklist for open assignments.
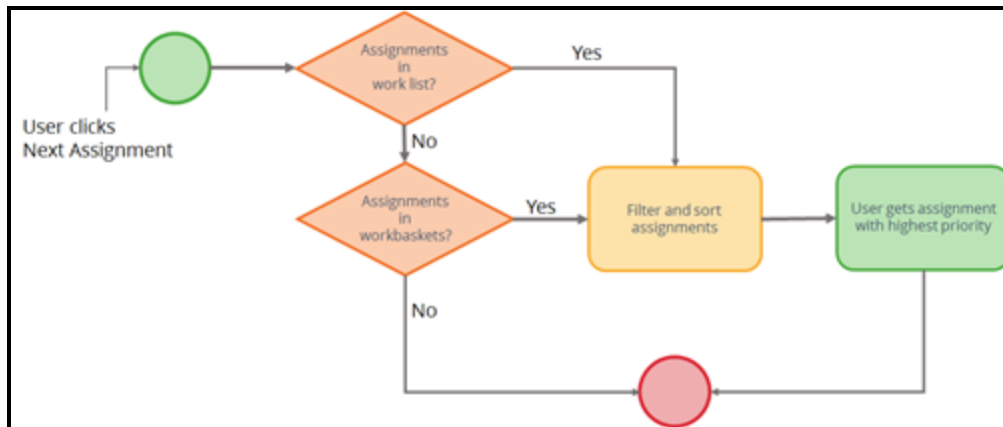
When a user clicks the **Next Assignment** link, the system checks the user's work list. If assignments are available in the user's work list, the assignments are filtered and sorted. The assignment ranked with the highest priority is selected.

If no assignments are found in the user's worklist, the workbaskets associated with the user are checked. If assignments are available in the workbaskets, the assignments are filtered and sorted. The assignment ranked with the highest priority is selected.

Getting assignments from workbaskets first

You can adjust the default settings to adapt how the system assigns work. For example, in some organizations, users must work on the next available, highest priority task in a centralized work queue.
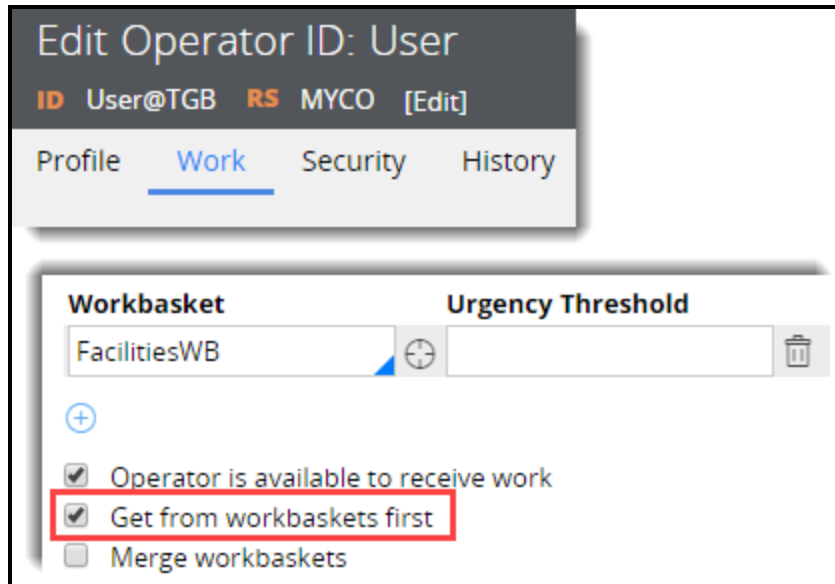
In the user's operator ID record, select the **Get from workbaskets first?** option to configure the application to check the workbaskets a user is associated with before checking the user's work list.
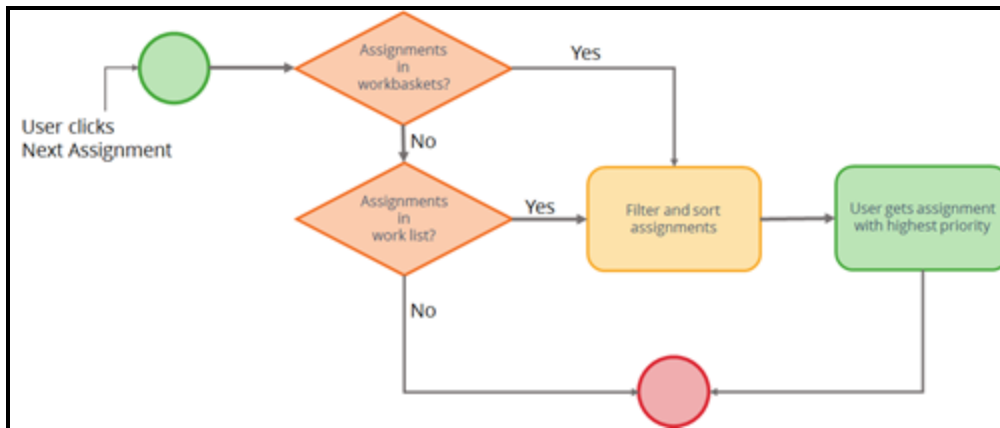
When the **Get from workbaskets first?** option is selected, the workbaskets associated with the user are checked first. If assignments are available in the workbaskets, the assignments are filtered and sorted. The assignment ranked with the highest priority is selected.

If no assignments are found in the user's workbaskets, the user's work list is checked. If assignments are available in the user's worklist, the assignments are filtered and sorted. The assignment ranked with the highest priority is selected.

Note: By design, workbaskets are searched in the order they appear in the user's operator ID record. The highest priority assignment in each workbasket is added to a final list of candidate assignments. This final list of candidate assignments is then filtered and sorted again to find the assignment with the highest priority.

KNOWLEDGE CHECK

KNOWLEDGE CHECK

Consolidating assignments from multiple workbaskets into a single list

If a user is assigned to multiple workbaskets, the task assigned could possibly come from the first workbasket listed in the user's operator ID record regardless of the priority of tasks in other workbaskets.

Edit Operator ID: User
ID User@TGB RS MYCO [Edit]

Profile   Work   Security   History

Workbasket
FacilitiesWB

☑ Operator is available to receive work
☑ Get from workbaskets first
☑ Merge workbaskets
☐ Use all workbasket assignments in user's workgroup
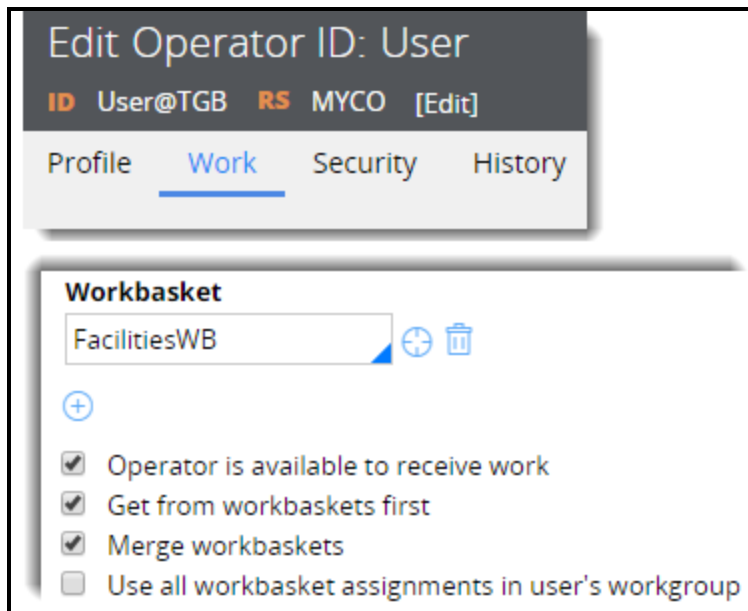
Use the **Merge workbaskets?** option to organize all assignments in all of the workbaskets the user is associated with into a single list before the list is filtered and sorted. In some applications, a user may be associated to additional workbaskets other than the workbasket associated to the user's work group.

When the **Merge workbaskets?** option is selected, use the **Use all workbasket assignments in the user's work group** option to limit the assignments available to the user to only those assignments listed in the workbasket of the user's work group.

KNOWLEDGE CHECK

Delegating rules to business users

Introduction to delegating rules to business users

When building a Pega 7 application, remember that it is designed to meet business needs. Business needs reflect the operating environment of the business itself, and can change frequently and unexpectedly.

Using Pega 7, you can delegate responsibility for updating selected parts of an application to business users. Delegating business rules can help promote an agile response to changing business needs, and empower those closest to the day-to-day business operations. Delegating business rules can also help focus the workload for architects by transferring to the business the responsibility for maintaining low-risk items.

After this lesson, you should be able to:

- Describe the use of rule delegation in an application
- Explain the benefits of rule delegation
- Delegate a rule for maintenance by business users

Rule delegation

Every business application comprises a series of business processes that describe the steps a business takes to achieve a specific business outcome.
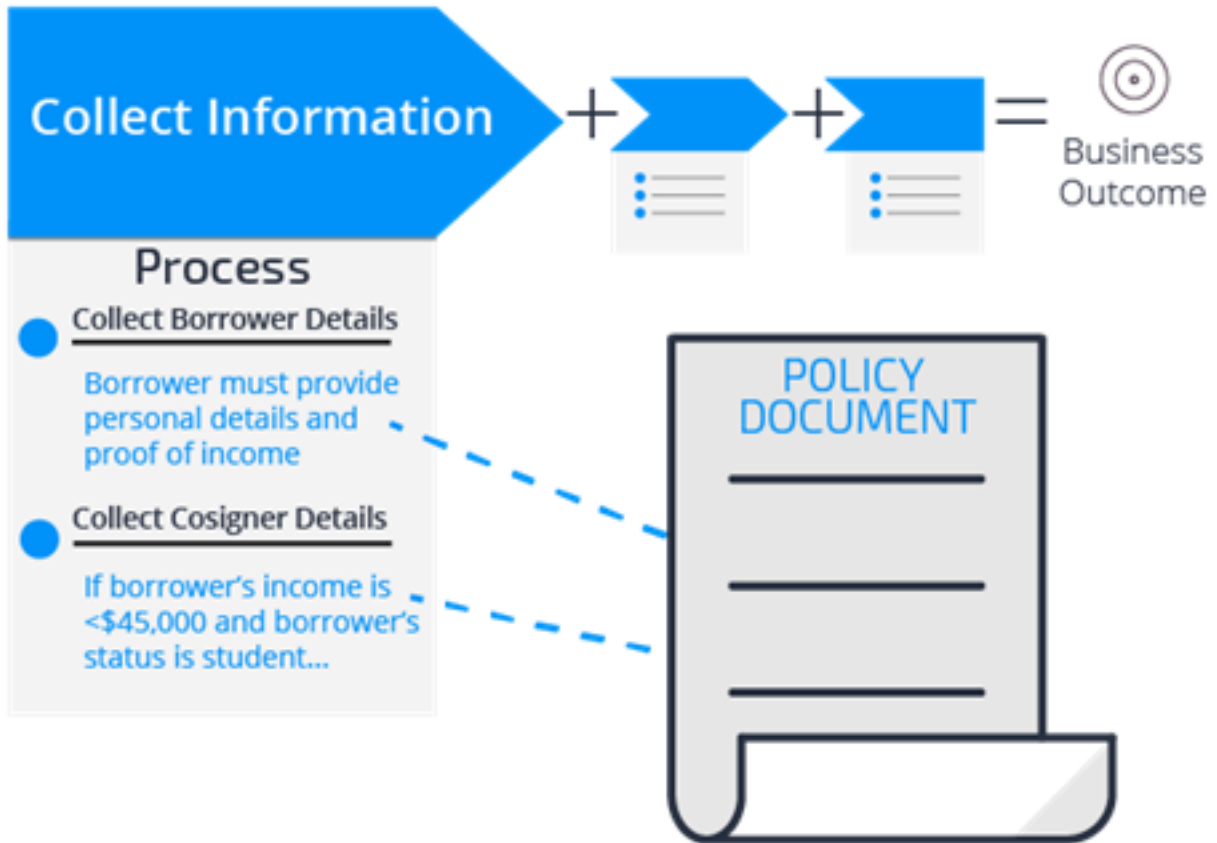
Business processes are driven by policies that define what tasks must be completed, and when those tasks must be completed. Business policies may even specify the order in which tasks must be completed, or if certain tasks are necessary.
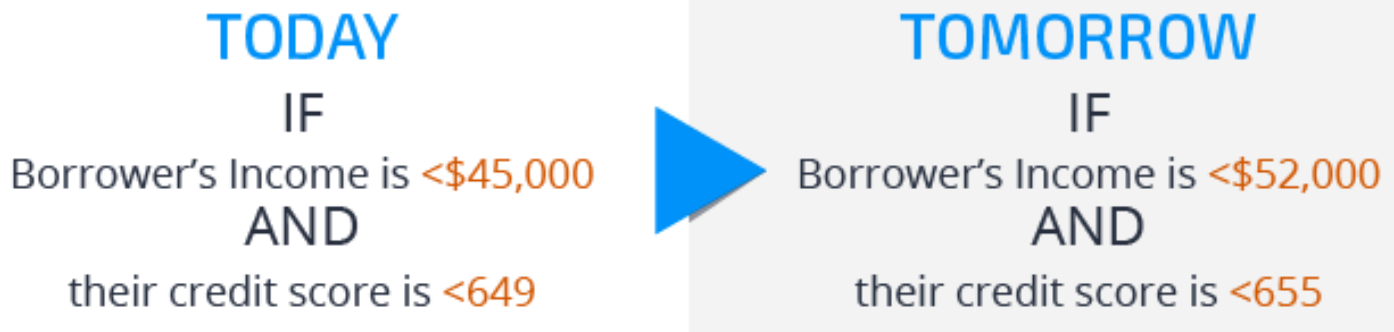
These business policies reflect the operating environment of the business itself, and can change frequently — often unexpectedly. Changes to business polices can be caused by updates to internal procedures, evolving industry guidelines, or changes in governmental regulations.

When you model business policies in an application, you need to adopt a strategy to update these models as business conditions change. Doing so enables the business to be more agile.

This approach also helps focus the workload for system architects. Rule delegation transfers the responsibility for maintaining low-risk business policies to business users. This allows system architects to concentrate on tasks suited to a longer, more rigorous development cycle. For example, a business can delegate to the Human Resources (HR) department a service level agreement for processing benefits enrollments. The HR department can adjust the details of the service level agreement, allowing system architects to focus on other development tasks.

Keeping pace with change

Most changes to a business application occur within the scope of a development cycle.
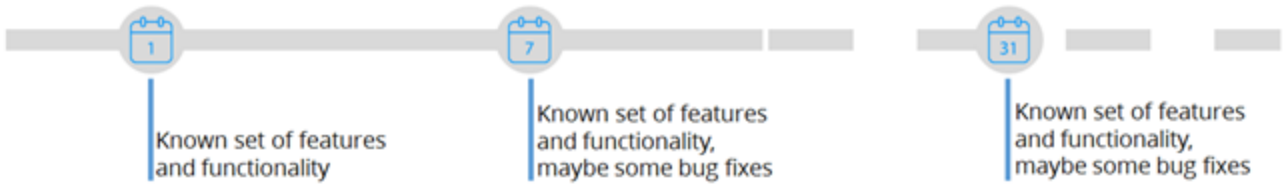
A known set of business requirements are used to define the features and functionality for a specific version of a business application. Each specific version of the business application is scheduled for release on a certain date.

Business policies tend to change more frequently than any other part of a business application. If you treat each business policy change as a change to the application, the result is either an endless series of development projects, or the inability to keep up with the policy changes at the pace required by the business.
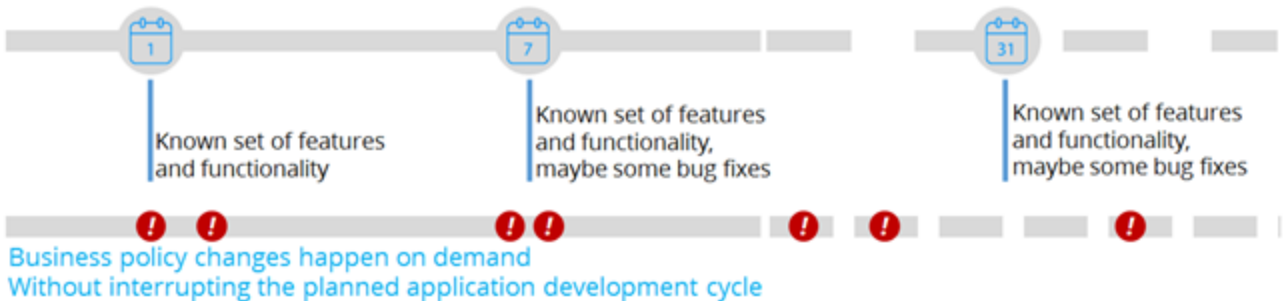
If you allow business users to update business policies, the business application can be adapted to changes as they occur, rather than during the next update cycle. Designing the application so business policies are owned and modified by business users can help promote an agile response to ever changing business conditions. This design also empowers business users to make changes to business policies as those changes occur.

Empowering business users with rule delegation

In Pega 7, **rule delegation** allows business users to make changes to business policies without involvement from IT.

Using rule delegation, you put a set of policies under business control even while the application is in production. Business users view the policies in a familiar environment, and can make updates to the policies without knowing all the related, technical details.
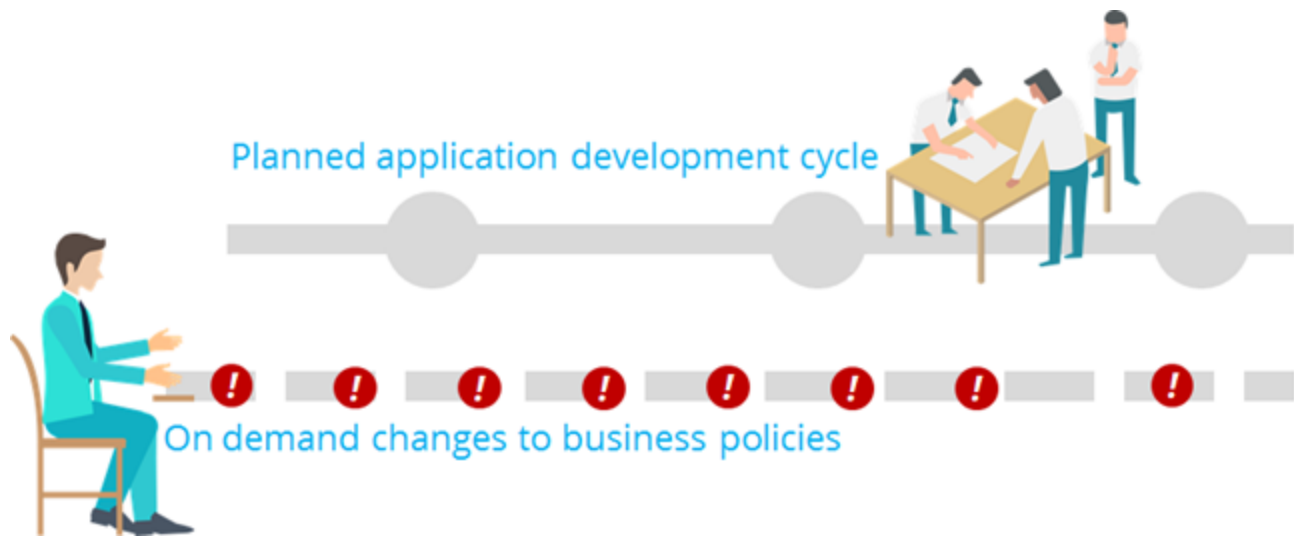
Enabling business users to manage changes to business policies on their own helps the business be more agile, and rely less on people outside of their control. This in turn allows system architects and other technical members to spend more time on architectural decisions and implementation strategies.

Adopting a rule delegation strategy significantly improves the business' ability to adapt to changing business conditions, and allows business users and architects to focus on what they do well.

KNOWLEDGE CHECK

ANSWER	Delegating business rules can _____ and _____.
promote an agile response to constantly changing business conditions provide a level of empowerment to business users	

How to delegate rules to business users

Rule delegation enables business users to change simple application logic without involvement from IT. For example, you can delegate correspondence or service level agreement rules to a business line manager. The business line manager views the delegated rules in the Case Manager portal, and can make changes to the delegated rules without knowing all of the related, technical details.

Several parties collaborate to perform the tasks necessary to delegate rules:

- A business architect (BA) identifies the business users who manage the delegated rules.

- A senior system architect (SSA) or lead system architect (LSA) creates an access group for business users responsible for managing the delegated rules.

- A system administrator organizes the business users who manage the delegated rules in the access group.

  Note: During development, an SSA may perform this task with test operators to test the configuration.

- A BA identifies the rules to delegate.

- An SSA organizes the rules to delegate in a production ruleset. This ruleset must remain unlocked in the production environment as changes cannot be made to rules in a locked ruleset.

  Note: Depending on the customer, creating the rulesets used for delegating rules may fall to a system administrator, an LSA, or others with detailed knowledge of the application structure.

- An SSA delegates the rules for availability in a user portal.

Note: Although the tasks listed in this process are not considered sequential, they are listed in a logical order that makes them easier to complete.

Create an access group for users who manage delegated rules

Create an *Access Group* to organize the business users responsible for managing the delegated rules. A separate access group for delegated rules simplifies the management of the delegated rules. The access group includes the rights to view and modify the delegated rule while preventing other users from making unauthorized changes.

Note: In some organizations, creating an access group may be the responsibility of a system administrator or others who understand the application structure and security implementation.

Identify the rules to delegate

In Pega 7, there are no restrictions on which rules can be delegated. However, to be successful, your rule delegation strategy must consider which rule types are best to delegate. In general, BAs work with the business users to determine what components of the business logic — which rule types — they want to maintain.

Important: The business user'sfamiliarity with managing rules is a key factor when deciding which rules to delegate.

The best candidates for delegation are the rules most affected by frequently changing business conditions such as correspondence, paragraphs, decision tables, service level agreements, or when rules.

Start by delegating a small, focused set of decision rules and correspondence templates. As the business users become more comfortable with delegation, you can expand the number and types of delegated rules.

Organize the rules to delegate in an unlocked, production ruleset

To update any rule on a production system, the rule must be contained in an unlocked ruleset. An **unlocked ruleset** is a ruleset in which rules can be modified and saved.

An unlocked ruleset is a potential security risk, so it should be dedicated solely to delegated rules. This allows a system administrator to prevent unauthorized access to critical process flows or case design elements. Changes to these elements can have drastic effects on case processing.

Add the ruleset to the application rule as a production ruleset. A **production ruleset** is a ruleset containing rules that can be modified after the application is deployed.

Using a production ruleset to organize and contain delegated rules insulates the rest of the application from frequent changes. Because the original version of the rule still exists in a locked ruleset, a system administrator can revert to the original copy if something happens to the delegated version of the rule.

Important: The ruleset containing the delegated rules must remain unlocked in production so business users can make changes to the delegated rules. Changes cannot be made to a rule in a locked ruleset.

Delegate the rule

Create a new rule, or open an existing one.

Important: Remember to add the rule you want to delegate to the ruleset created specifically for delegated rules.

Delegate the rule to the access group used to organize the business users who will manage the delegated rule. For example, a Human Resources (HR) department may send an email to a new employee prior to the employee's first day of work. You can delegate the correspondence rule that contains the email content to allow the HR department to update the email to reflect a different new hire orientation procedure.

Delegate Correspondence ✕

What will the end user be able to edit?
Select one:

◉ Manage content

Delegate to access group *
Managers for the HRApps application ⇕

Title *
New customer welcome email

Detailed description
This will be displayed to the end-user to help
them understand what they are changing

Provide a meaningful title and detailed information about how the delegated rule affects the application. For example, if delegating a decision table, include a detailed description such as: **The logic in this decision table determines whether a travel request requires additional manager approval. Changing the logic in this decision table may affect all subsequent travel requests submitted in this application.**

Note: To delegate a rule, your operator ID must be assigned to an available role that has the *pxCanDelegateRules* privilege. By default, this privilege is extended to users assigned to the Administrator or SysAdm4 access role. If you cannot delegate rules, contact your Pega 7 system administrator to confirm that your operator ID has this privilege.

KNOWLEDGE CHECK

⌄
ANSWER   Why is organizing delegated rules in an unlocked ruleset necessary?

This is necessary so that business users can make changes to the rule. Rules in a locked ruleset cannot be edited.

KNOWLEDGE CHECK

⌄
ANSWER   What two prerequisite tasks should you complete before attempting to delegate a rule?

1. Prepare an access group used only for granting access to delegated rules.
2. Organize the rules you want to delegate in an unlocked, production ruleset.

Delegating a rule to business users

Delegating a rule enables business users to make changes to simple application logic without involvement from architects or other technical team members.

Prerequisites to delegating a rule

Prior to delegating a rule, you must:

- Identify an access group responsible for managing delegated rules

  Using a separate access group for delegated rules simplifies the management of the delegated rules. The access group includes the rights to view and modify the delegated rule while preventing other users from making unauthorized changes.

  For help with access groups, see the Access Group Help topic.

- Copy the rule to a production ruleset

  Organizing the rule in a production ruleset insulates the rest of the application from frequent changes.

  For help with making a copy of a rule, see the Copying a rule or data instance Help topic.

  Important: The ruleset used to organize delegated rules must remain unlocked in the production environment so business users can make changes to the delegated rules. Remember, changes cannot be made to a rule in a locked ruleset.

Delegate the rule to users

1. Open the rule you want to delegate to users.
2. From the **Actions** menu, click **Delegate**.
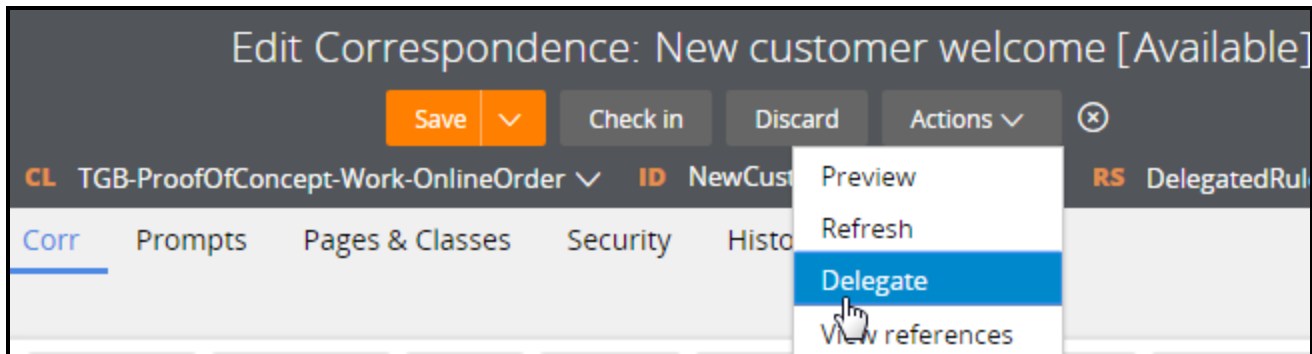
3. Select how the end user will interact with the delegated rule. The options that are displayed depend on the rule or data type being delegated.

   Note: See the Delegating a rule or data type help topic for detailed information on the available options.

4. In the **Delegate to access group** drop-down list, select the access group to which to delegate the rule or data type.

5. In the **Title** field, enter a title for the delegated rule. Use a title that is meaningful in a business context. For example, use **New customer welcome email**. Do not use **Welcome email**.

6. In the **Detailed Description** field, enter detailed information about the delegated rule. This text is displayed to the business users to help them understand the rule, and the impact changes they make to the rule may have on the application logic.

   Provide information about how the delegated rule or data type affects the application. For example, write: **The logic in this delegated decision table determines whether a travel request requires**

©2017 Pegasystems Inc.

additional manager approval. Changing the logic in this decision table may affect all subsequent travel requests submitted in this application.

7. Click **Delegate**.
8. Click **Save** to save your changes.

Configuring parallel processing

Introduction to configuring parallel processing

In this lesson, you learn how to configure processing that allows two or more processes to proceed in parallel as a case is processed. You also learn how to configure case locking.

After this lesson, you should be able to:

- Describe parallel processing
- Configure parallel processing for cases
- Describe case locking strategies
- Configure case locking

Parallel processing in Pega applications

You can configure a stage to run multiple processes in parallel. This configuration allows users to perform tasks independently in order to complete the work in a stage. For example, in the recruitment stage, you can include a process for interviewing a candidate. In the same stage, you can include a process for verifying a candidate's job history. Both processes can be started and completed independently. When the interview and job verification are completed, the case moves to the next stage.

For more complex parallel processing requirements, Pega provides the Split Join shape, the Split For Each shape, and the spinoff option in the Subprocess shape.

The process to which you add the shape is called the main process. The shapes call one or more subprocesses that proceed in parallel.

Split Join

You use the Split Join shape to call multiple independent processes that operate in parallel and then later rejoin. For example, a mortgage application process may require that a user validates the home buyer's credit history. At the same time, another user must perform a title search. Both of these processes are unrelated and can be performed in subprocesses that proceed independently and in parallel. When the subprocesses are complete, the main mortgage application process can continue. This is similar to a parallel process in the case lifecycle — when all the processes in a stage are completed, the case enters the next stage or is resolved.

However, the Split Join shape gives you the flexibility to use join conditions that determine when the primary process can continue. The join condition may iterate over a when condition or a count to determine when to resume the flow. For instance, a Split Join may include three separate approval subprocesses. You can specify that only two of the three approvals must be completed before resuming the main flow.
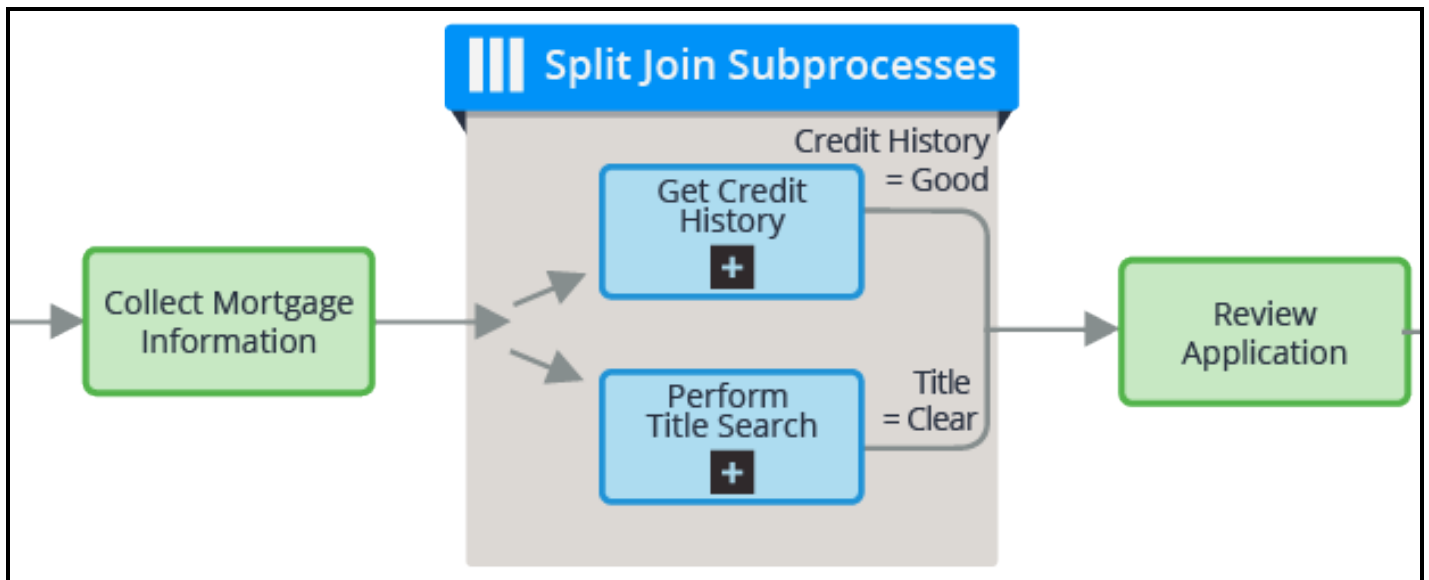
Split For Each

A Split For Each shape allows you to run one subprocess multiple times by iterating through a set of records stored in a page list or page group. When the items on the list have been processed, the main flow continues. For example, you can use a Split For Each to iterate over a list of vendors and send a quote request from each vendor on the list. Like the Split Join, you can use a join condition to control when the primary process resumes. If you use an iterate join condition, you can start flows for elements of the Page Group or Page List property one by one, and configure testing conditions to determine whether to continue.
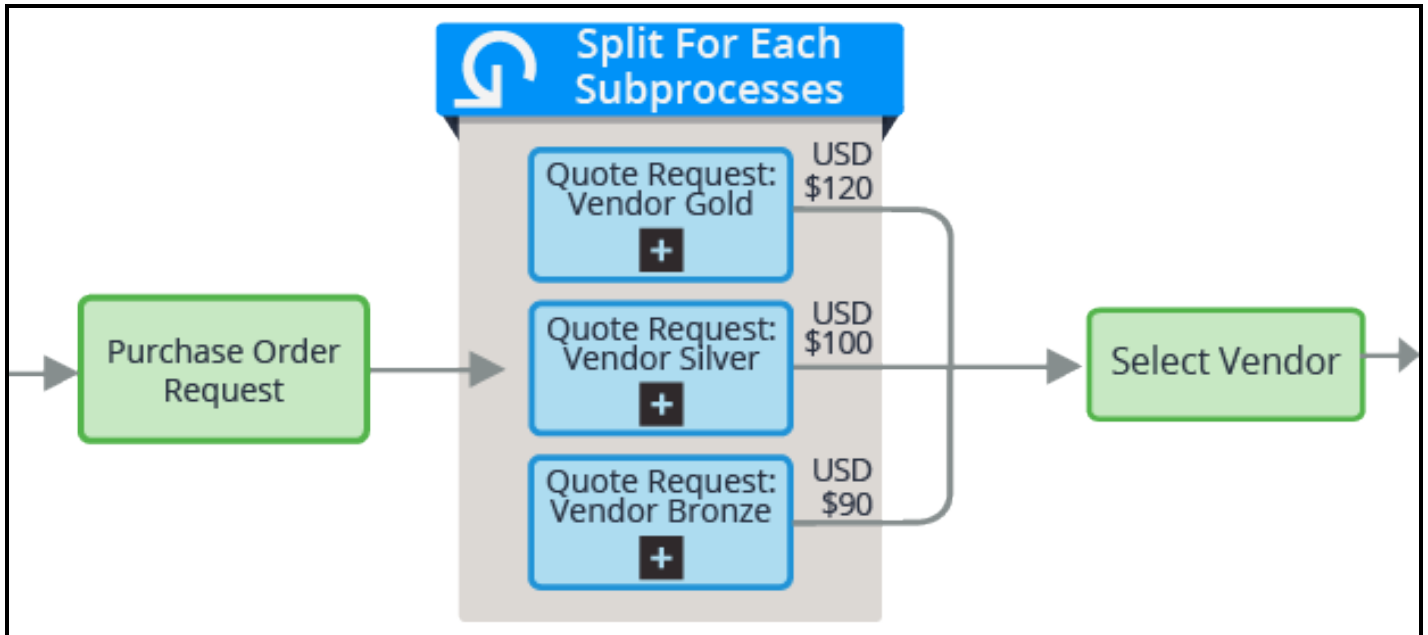
Spinoff

The spinoff option in the Subprocess shape allows you to run the subprocess in parallel with the main flow. The main process does not wait for the subprocess to complete before proceeding. The spinoff option is an advanced feature in the Subprocess shape and is accessed in the flow diagram.

KNOWLEDGE CHECK

ANSWER
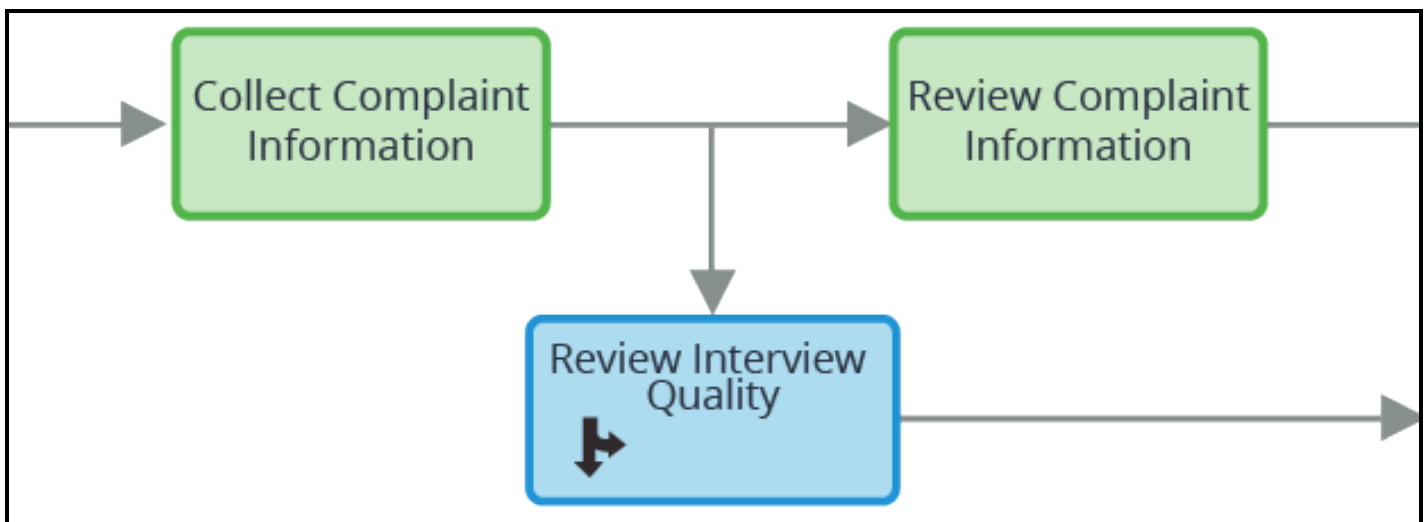
Which type of Smart Shape would you use to start an interview processes based on a list of employees who must interview a job candidate?

A Split For Each allows you to iterate over a list of employees to start an interview process for each employee.

How to configure parallel processing

You configure Split For Each, Split Join, or Spinoff parallel processes in a flow diagram. To add a shape, right-click and select a shape. For a Spinoff, select the Subprocess shape.
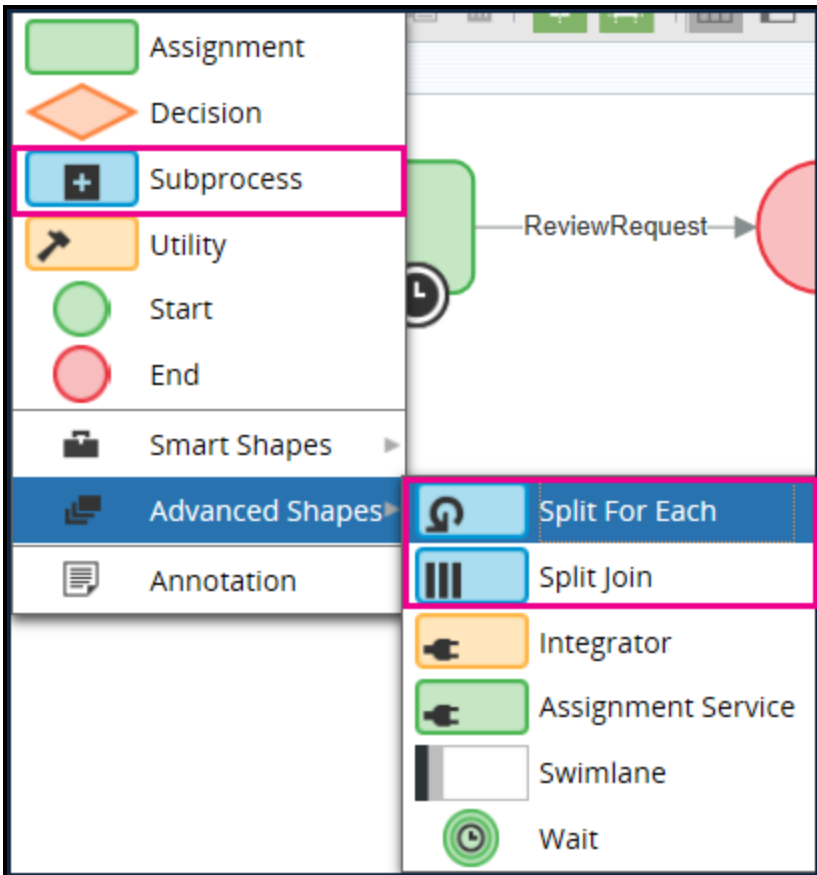
First, decide where in the flow you want to place the shape. After you place the shape, you open the property panel and configure the properties. These properties vary depending upon the shape. For example, in an interview flow, job candidates are scheduled for interviews. Each candidate is interviewed by multiple employees. As each employee completes the interview, the employee submits a candidate assessment in a subprocess. In this example, you place the Split For Each shape after an assignment for scheduling an interview.
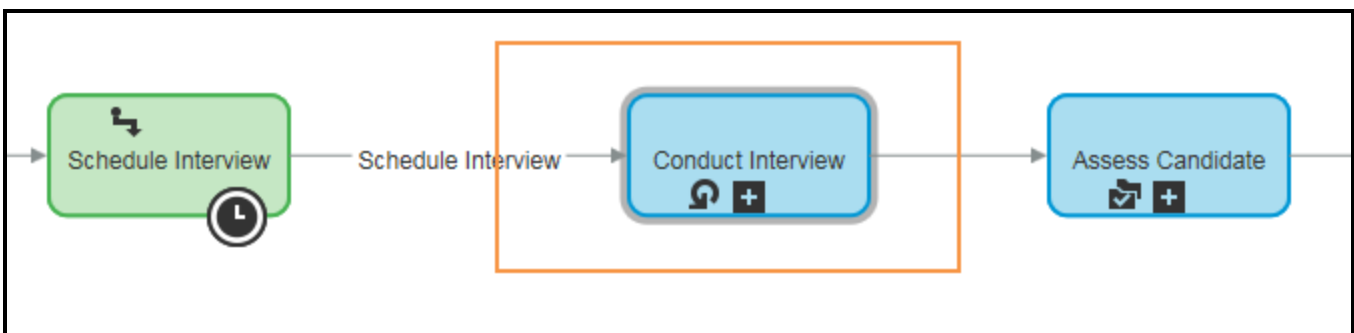
Then, open the shape and configure the properties. For a Split For Each, you enter the page property you want the process to iterate. The Split For Each functionality starts a new flow for each item on the list. In

©2017 Pegasystems Inc.

this example, the property contains a list of interviewers. An InterviewCandidate flow is created for each interviewer.

Note: When you use a Split For Each shape, make sure that the flow and the page list used in the iteration are in the same class.

A Split Join shape allows you to call multiple subprocesses from the main process. In the following example, the main process calls a background check and collects candidate details subprocesses, which run in parallel with each other.

To use the Spinoff parallel processing, you must open the Subprocess shape on the flow diagram and select the **Spinoff** option. You can add subprocesses in the case life cycle by adding process steps. However, to use the spinoff feature, you must configure the shape in the flow rule.

Embedded classes

Split Join shapes allow you to call a flow on an embedded page such as a data class. For example, assume the Purchase Request case has an embedded page named .ShippingAddress and the data class is ADV-Purchasing-Data-Address. Based on the specified address location, you can call a flow in that data class. The flow is configured to check whether a Saturday delivery is possible for that address. The subprocess returns a Yes or No result to the main process.

Join conditions

On Split For Each and Split Join shapes property forms, you specify a join condition. The join condition setting controls when the main flow resumes.

- Select **Any** if you want the main flow to resume after any one of the subprocesses completes. At that time, processing of the other subprocesses that have not completed is stopped. Open assignments

©2017 Pegasystems Inc.

for these subprocesses are cancelled.

- Select **All** if you want the main flow to resume after all of the subprocesses complete.
- Select **Some** if you want to use a when rule or a count to determine when the main process can resume.

The Split for Each shape also contains an **Iterate** join condition. This starts flows for items on the Page Group or Page List property one by one, testing a when condition to determine whether to continue.

A Spinoff does not have a join condition because the subprocess never rejoins the main process.

KNOWLEDGE CHECK

In an equipment selection process, you want to start a Facilities Setup and IT Setup processes to run in parallel using a smart shape. Which smart shape would you choose?

You would choose a Split Join because it lets you call multiple subprocess that run in parallel.

Configuring parallel processing

The following resources provide information to help you configure the parallel processes.

- For a description of the settings you use to configure a subprocess for parallel processing, see the Help topic Advanced settings in Process Modeler for the Subprocess shape.
- For a description of how to configure a Split For Each shape, see the Help topic Configuring a Split For Each shape.
- For a description of how to configure a Split Join shape, see the Help topic Configuring the Split Join shape.

Case locking

In most situations, when a user is working on a case, other users are locked out of the case until the first user submits their case. This locking strategy prevents potential conflicts when a user attempts to submit cases that have already been updated. **Default locking** in Pega locks the case when an operator opens the case. If a second operator tries to open the same case, they get a message that the case is locked by the first operator. The second operator is able to open the case in review mode only. The lock is acquired when the case was opened by the first operator. The second operator cannot access and update the case. By default, the system locks the case for 30 minutes or until the user submits or closes the case, whichever comes first.
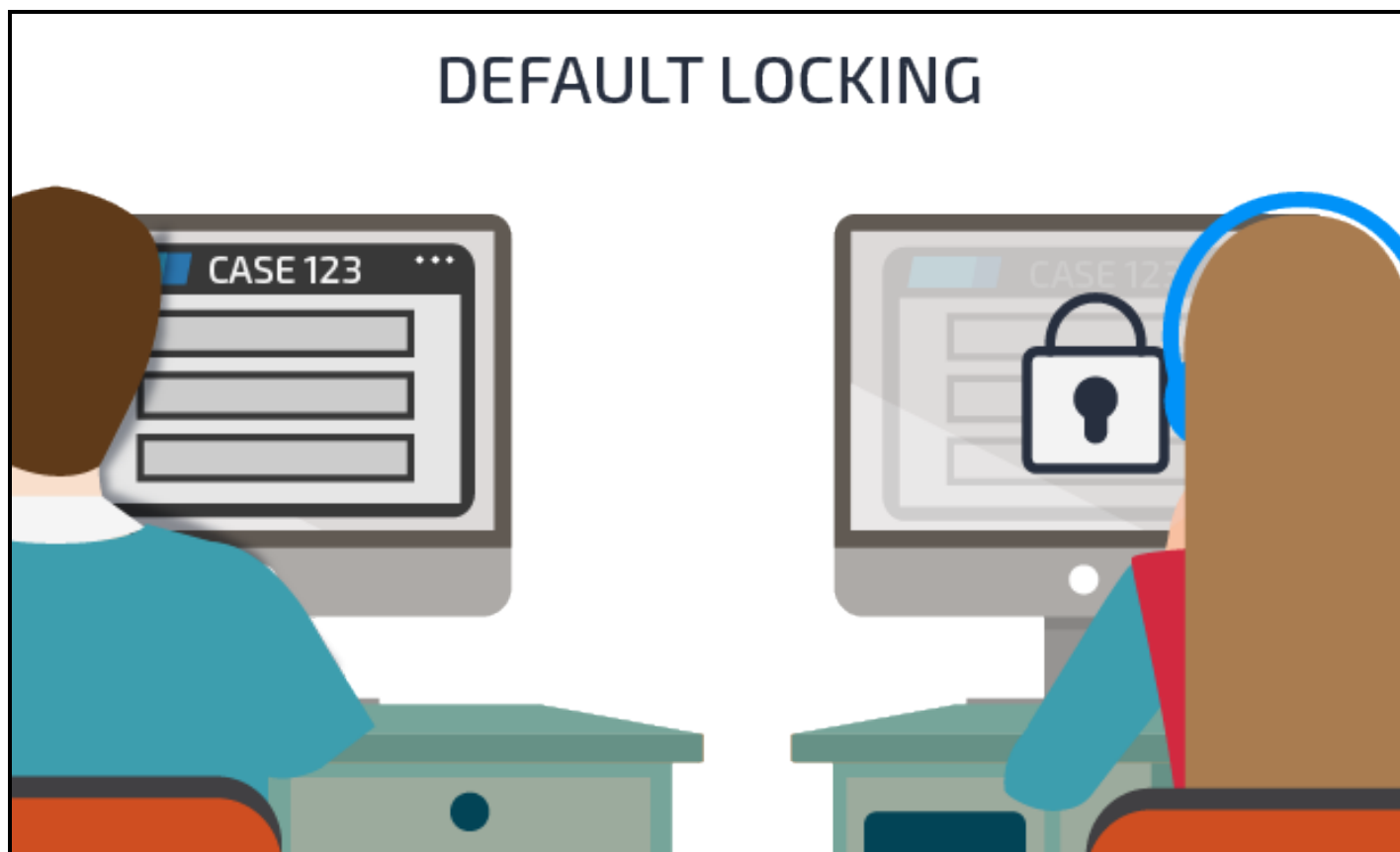
In certain situations, a business may want to allow more than one user to open a case at the same time. For example, an insurance company may want to allow claims adjustors to make real-time updates to a claim request. These updates could be adding a new item or adjusting the estimated cost. The company may also want customer service representatives (CSRs) to have access to the request in case the CSR needs to update the customer's personal information. The rationale is that the adjustor and CSR will likely be updating different data and not risk overwriting the same data.

In those situations, **optimistic locking** enables both the operators to open the case. No lock is obtained when the case is opened. The lock occurs when the user submits the case. When two operators are working on the same case, the changes to the case are made by whoever submits the case first. When the second operator submits the form, the operator receives a message with the first operator's name and the time of the change. The second operator clicks a refresh button in the message to get the new changes made by the first operator. The second operator's changes are not applied until they submit their action after the refresh.

OPTIMISTIC LOCKING
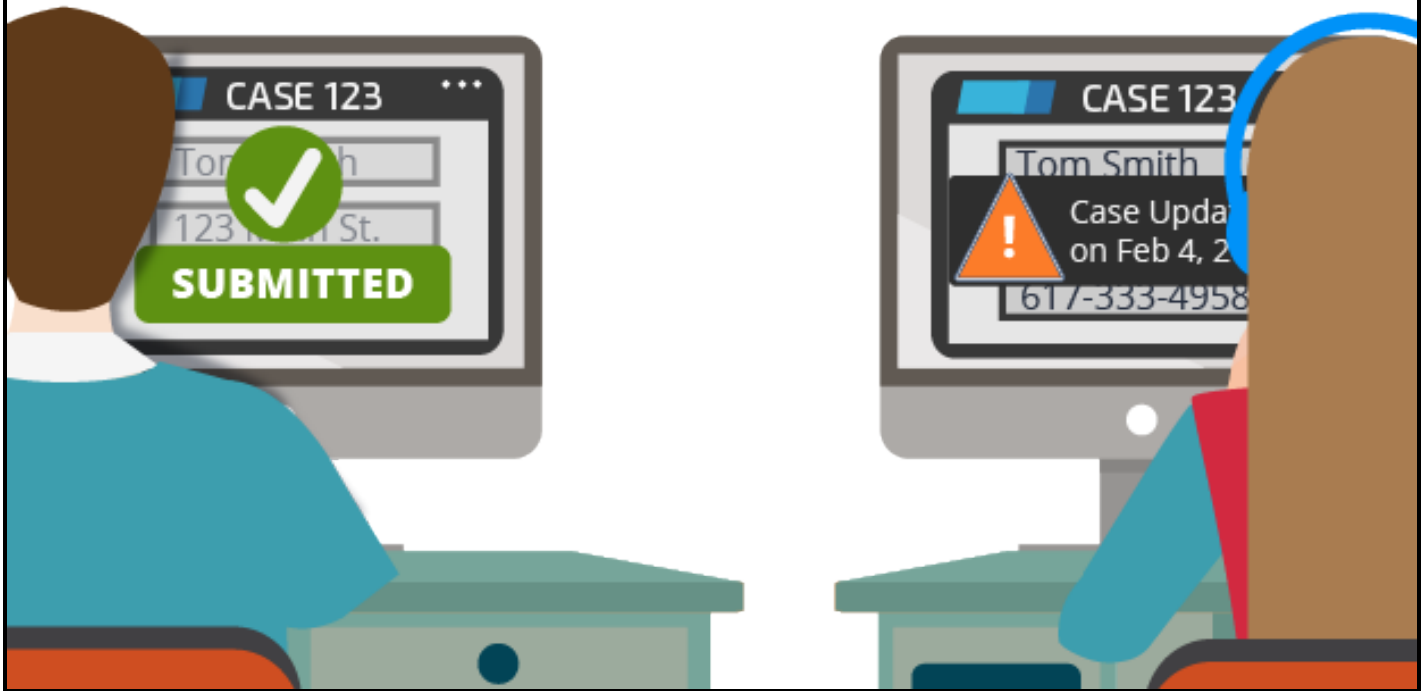
Locking in the case type hierarchy

In a case type hierarchy, you configure case locking on the top-level parent. When child cases are created, the parent's lock settings cascade down through all the child-case types. At run time, the lock settings are applied to the child case you created. The standard setting is default locking. Assume the parent case type, Onboarding, has a child case type, Benefits Enrollment. Both use default locking. When a user opens a Benefits Enrollment case, it and its parent Onboarding case are locked. If default locking hampers user throughput in your application, you can override the default locking setting at the child-case level. This lets users concurrently make updates to parent cases and their child cases without conflict.

The recommended approach for most situations is to lock the parent when the child case is being worked on. Default locking helps preserve transaction integrity among cases. For example, the Onboarding case may contain properties such as the total cost of benefits that are totaled from values in the Benefits Enrollment case.

If you set the top-level parent case type to optimistic locking, all of the parent's child case also use optimistic locking. You cannot override the setting at the child-case level.

Choosing a locking strategy

Choosing the right locking strategy depends on your organization's business requirements. If multiple users will likely attempt updates to cases in one or more case types, you should use default locking.

Optimistic locking may be called for where multiple users need only to open and review cases without having to perform updates.

Also consider creating a child case type as an alternative to allowing multiple users perform tasks on an open case. Using the previous insurance example, assume that you make Account Updates a child case type of the Claim Request case type. This allows you to use default locking for both case types. Users in both case types can make updates without causing conflicts.

KNOWLEDGE CHECK

Assume you want users to have the ability to work on top-level cases while their child cases are open. How would you configure your locking settings?

For each child case, you override the default setting that locks the parent case.

How to configure case locking

When working with a case type hierarchy, you set locking on the top parent case. The settings cascade down to each child case when it is instantiated. If the child cases are instantiated as part of the parent case, they have the same locking settings as the parent. In the Case Explorer, select the parent case type to set locking. Then, in the Case Designer, on the Settings tab, select the **Locking** option. If you select default locking, you can modify the default locking timeout of 30 minutes in the **Release lock after ____ __ mins** field. Consider the business context when setting the timeout duration. For instance, if a case will likely be opened frequently, you may want to shorten the timeout so that users can more quickly access the case.
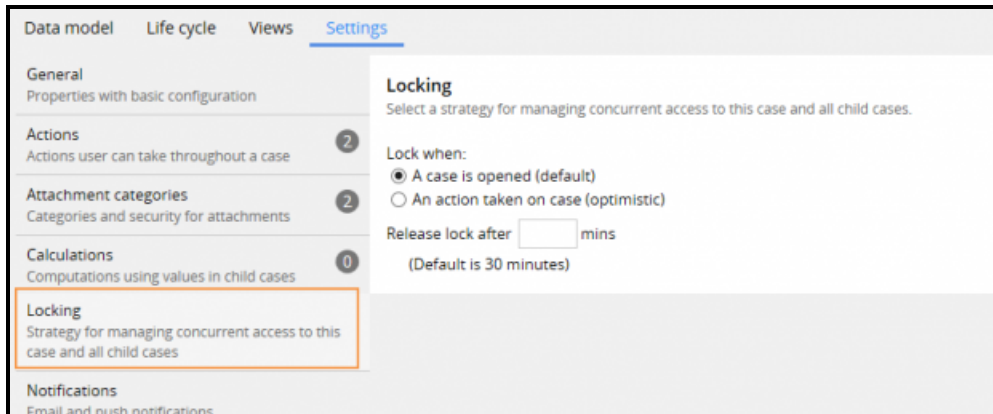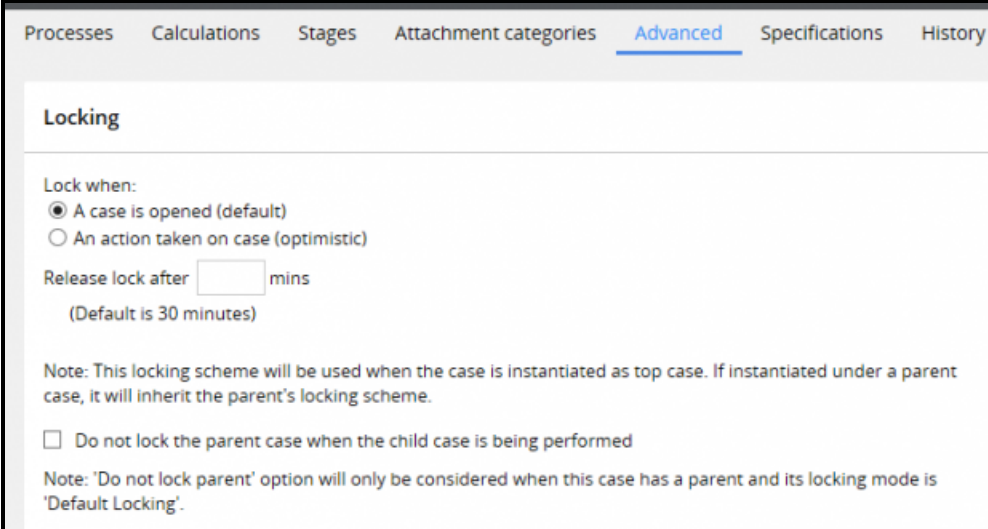
If you select default locking, you can update the lock timeout for any child cases. If you do not want to lock the parent case if the child case is open, select the **Do not lock the parent case when the child case is opened** check box.
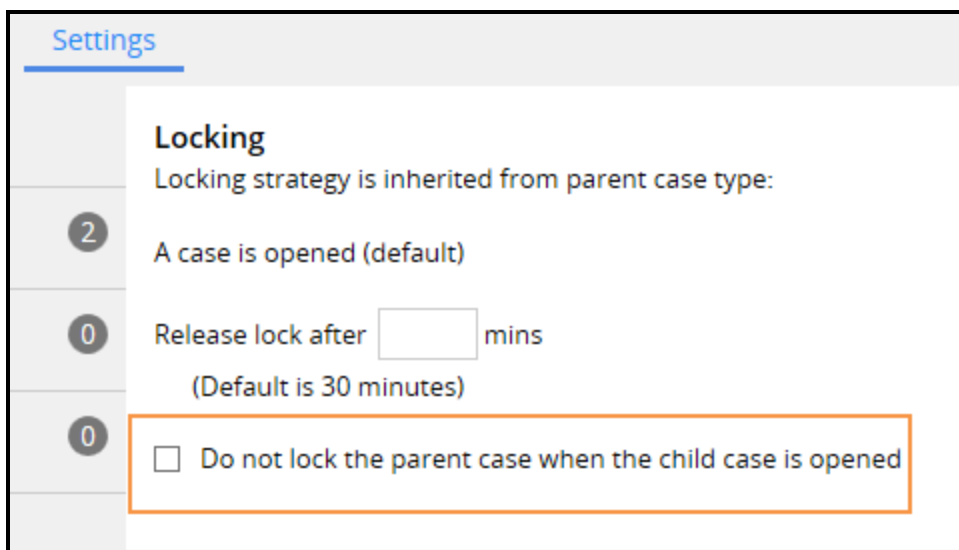
For more information about the Locking option, see the Help topic [Setting the locking strategy for a case type.](#)

Locking standalone cases

Child cases may be instantiated independent of a parent case. For example, you may want to create a shipping case as a standalone case that is not a child case of a purchase request parent case. If a child case is instantiated as a standalone case, it does not inherit its lock settings. You can configure case locking for this case type on the **Advanced** tab of the Case Type rule.

©2017 Pegasystems Inc.

Improving the user experience with screen flows

Introduction to improving the user experience with screen flows

Screen flows are an efficient way to capture a lot of data at once from an end user. Screen flows divide complex tasks into a series of steps. By splitting the task into a series of steps, you effectively simplify the task.

After this lesson, you should be able to:

- Describe the purpose of a screen flow.

- Configure navigation and persistence options for a screen flow.

Screen flows

Long, complex online forms can be difficult and frustrating for users of the application to navigate.

For example, online order forms require a lot of data. Users must enter their contact information, the items they want to buy, and the payment method. They may also have to select a shipping method that impacts the total cost of the order.

To help users complete complicated tasks, UX designers often design a guided, linear workflow using simple UI screens. Each screen captures specific and related data such as order details or payment information.
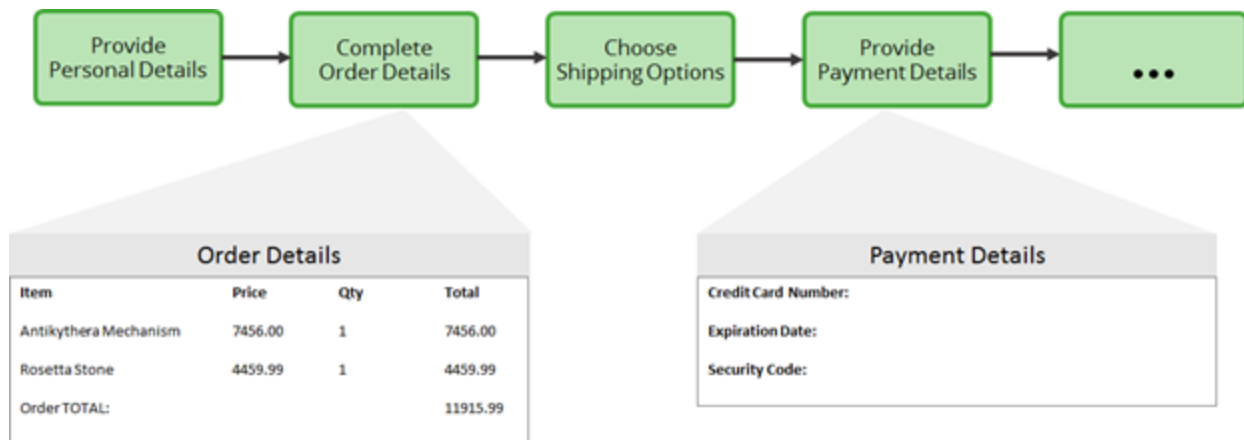
Users easily navigate these screens to enter data and complete tasks without frustration.

In Pega, a **screen flow** is used to create a linear series of UI screens. The screen flow allows users to navigate multiple screens to enter data and complete tasks. Users can return to a prior screen to change or review the input on each screen.

©2017 Pegasystems Inc.
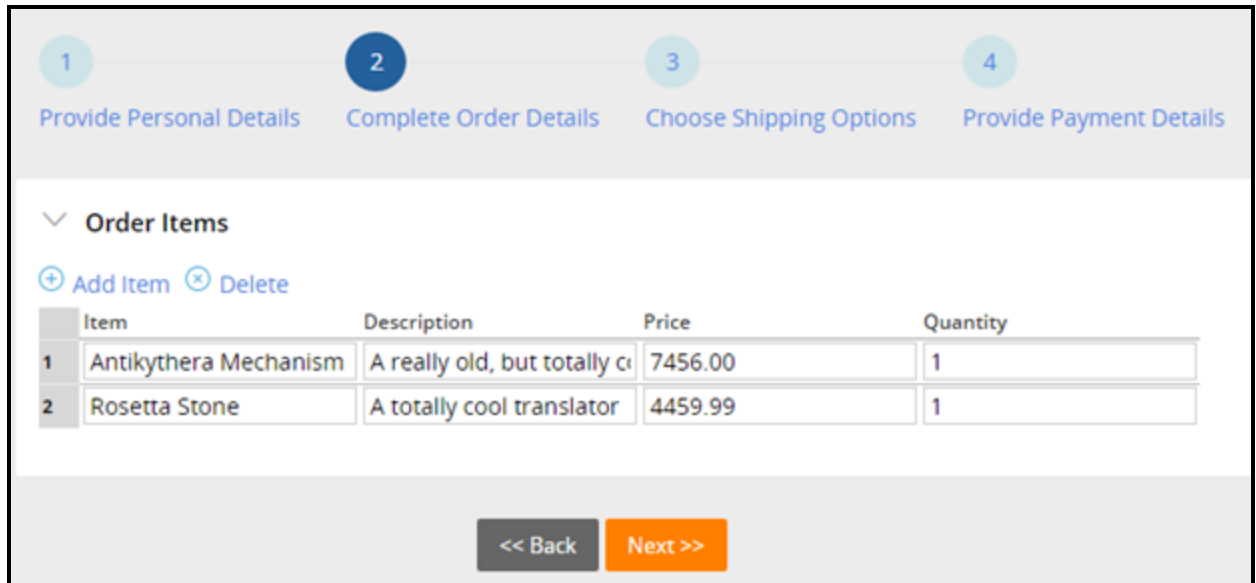
The individual UI screens are part of a single form presented in a logical sequence. Users can complete the UI screens in any order. At any point in the flow, users can review or change information on previous screens.

Note: A screen flow represents a single assignment completed by a single user. Individual UI screens in a screen flow cannot be routed to different users.

Screen flows present users with a road map to complete the task. The linear structure ensures users focus on each step eliminating frustrations.

KNOWLEDGE CHECK

Screen flows can help users complete complicated tasks by _____ .

presenting a linear workflow using simple UI screens

How to configure a screen flow

A screen flow allows users to navigate multiple screens to enter data and complete tasks. Users can return to a prior screen to change or review the input on each screen.

To configure a screen flow, determine the following:

- What type of navigation style will the screen flow use?
- What is the sequence of steps? Must users complete each screen in sequence, or can they complete the screens in any order?
- What post-processing actions must execute? Must data be validated on each step, or only when the screen flow is completed?
- What data, if any, needs to be persisted? Will data be saved on each screen, or only when the user has completed all screens?

Configure the navigation style for the screen flow

Use a harness to define the navigation style for the screen flow. Pega 7 provides three harnesses you can use to define the navigation style:

- TabbedScreenFlow7
- TreeNavigation7
- PerformScreenFlow

Note: The harness is configured on the Start shape when using the *ScreenFlow* process template.

TabbedScreenFlow7 harness

Use the **TabbedScreenFlow7** harness to display the steps of the screen flow in a tabbed format.
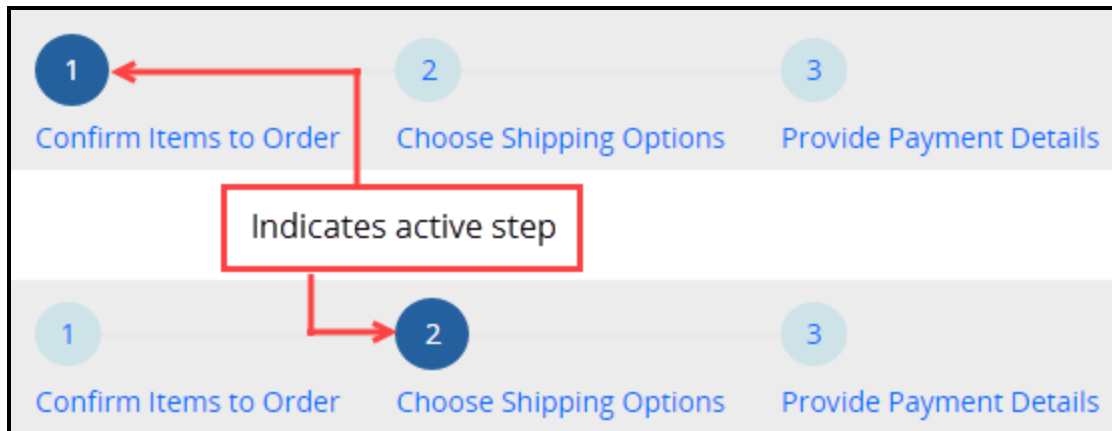
This navigation option displays the current step of the screen flow highlighted in blue and shows all the steps in a tabbed format. Users navigate using the Back and Next buttons, or by clicking on a tab. Users can move forward or backward by clicking on a tab in the tab navigation.

TreeNavigation7 harness

Use the *TreeNavigation7* harness to display the steps of the screen flow in a tree structure format.
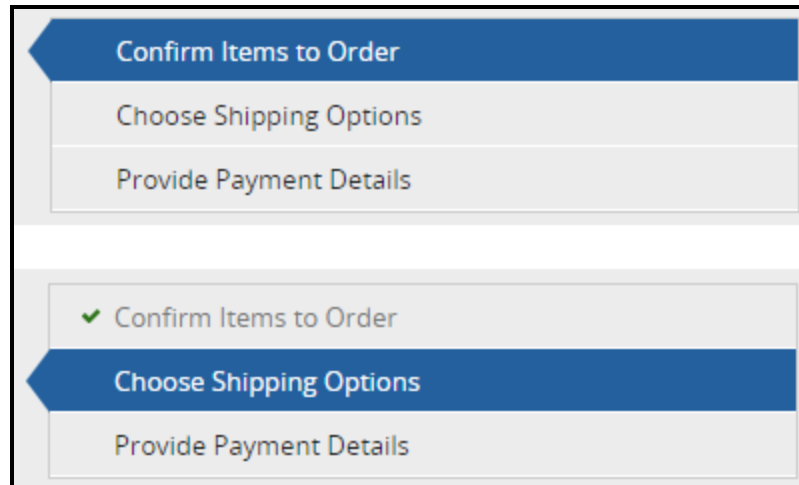
This navigation option displays the current step of the screen flow highlighted in blue and shows all the steps in the navigation tree. Users navigate using the Back and Next buttons, or by clicking on the menu item in the tree structure. Users can move forward or backward by clicking on a step in the tree structure.

PerformScreenFlow harness

Use the *PerformScreenFlow* harness to display the steps of the screen flow as a trail of breadcrumbs.

Note: The *PerformScreenFlow* harness will only display the current and completed steps of the screen flow.

Users navigate using the Back and Next buttons. Users can only move multiple steps backwards.

Configure the sequence of steps

Each assignment shape in a screen flow represents a step. Configure each assignment shape to control the order in which the steps are executed.

©2017 Pegasystems Inc.
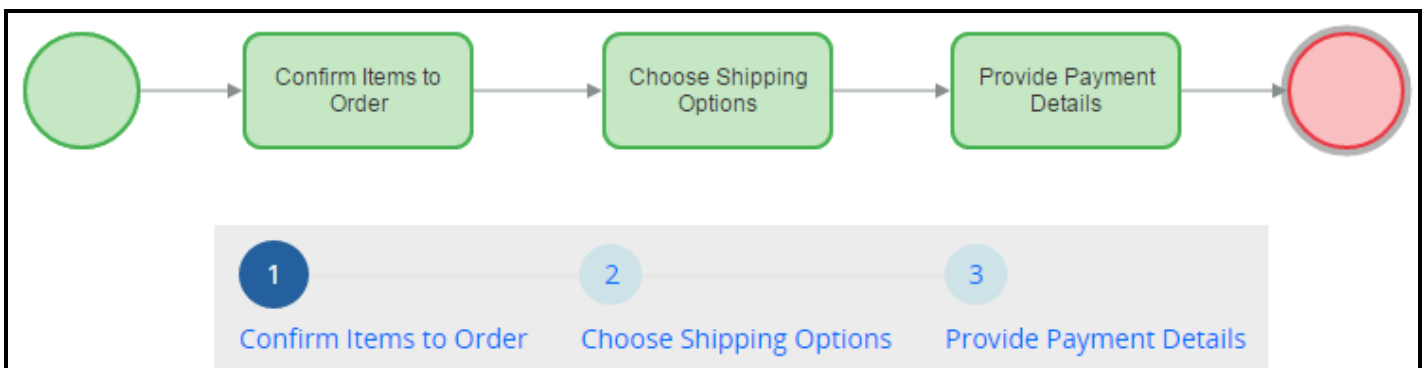
You can configure a screen flow so that:

- steps are run in a strictly enforced sequence; that is, step 2 can only be executed after step 1, and step 3 can only be executed after step 2.

- steps are run in any order; users can move from step 1 to step 5, and then back to step 3.

To strictly enforce the sequence in which steps can be executed, select the **Enable navigation link** and the **Only allow navigating back to this step** options on each assignment.

Note: One way to enforce a strict sequence is to use the *PerformScreenFlow* harness. This harness displays a breadcrumb trail that does not include future steps. Users must use the **Next** button to move to the next screen in the sequence.

To allow steps to be run in any order, select only the **Enable navigation link** option on each assignment.

Caution: If you clear the **Enable navigation link** option on an assignment, the step will not display in the navigation sequence, but will display to the end user when the step is encountered in the screen flow.
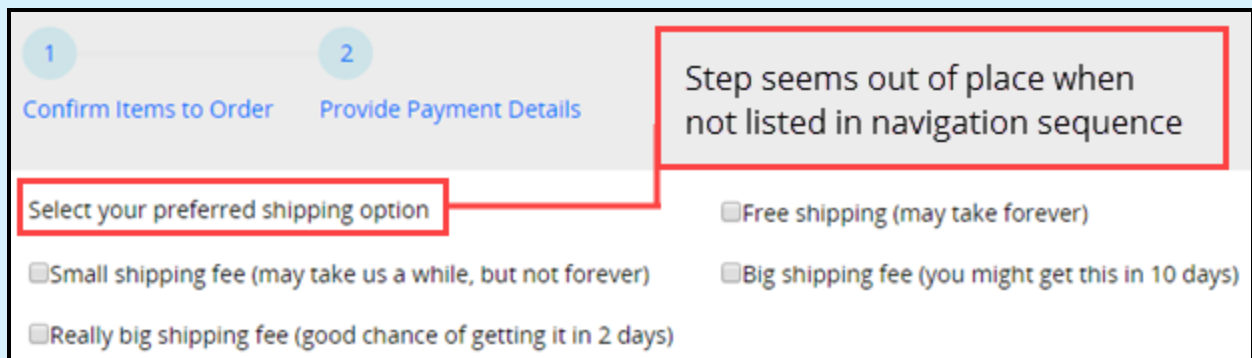
This may confuse end users. As a best practice, use the **Enable navigation link** on all steps.

Configure post-processing actions

When using the screen flow template, post-processing actions listed on the flow actions are not automatically executed. You can force post-processing actions to execute by selecting the **Perform post-processing when navigating away from this step** option on each assignment. When the process moves from the step, any validation rules and post processing rules listed in the flow action are executed.

Examples of leaving a shape at run time include clicking another entry point in the navigation sequence or automatically moving to the next step in the sequence when the current step is completed.

Configure the persistence options

When using the screen flow template, each step in the screen flow is committed to the database as it is executed.

Use the **Save on last step** option on the Start shape to prevent commits to the database as each step is executed. When the option is selected, the data collected on each of the assignments is not saved until the last step.

©2017 Pegasystems Inc.

Use the **Allow errors** option to allow users to progress the screen flow to a different step even if the current step fails validation.

Note: The **Allow errors** option is often used with the **Save on last step** option.

Tip: As a best practice, select the **Save on last step** option to reduce the number of case and history commit operations.

Configuring a screen flow

Configure a screen flow to allow users to navigate multiple screens to enter data and complete tasks.

To configure a screen flow, create a flow record using the standard template for screen flows. Next, configure the screen flow navigation style and the sequence of steps. Then, configure post-processing options. Finally, configure persistence.

Create a new flow record using the screen flow template

A screen flow is a special type of flow. To configure a screen flow, first create a flow record and specify the screen flow template.

Note: By design, flow records are created using the standard flow template, and cannot be converted to a different template.

1. Create a new flow record.

2. Click **View additional configuration options**.

3. Select the **Standard template for screen flows** option.

4. Click **Create and open** to create and open a screen flow that contains a Start shape, a single Assignment shape, and an End shape.

Configure the screen flow navigation style

Use a harness to define the navigation options for the screen flow. The harness is configured on the Start shape.

Note: When using a screen flow, the harness rule is configured on the Start shape. Because of this, a screen flow can only use one type of navigation style.

1. Double-click the Start shape to open the properties panel.

2. In the **Harness** field, press the down arrow key and select a harness that defines the presentation of the screen flow.

3. Click **Submit** to close the properties panel.

Configure the sequence of steps

By design, all steps in a screen flow are presented in the navigation sequence. To make changes to how steps are presented in the navigation sequence:

1. Double-click an Assignment shape to open the properties panel.

2. Clear the check box for the **Enable navigation link** option to prevent users from advancing to the step.

3. Select the **Only allow navigating back to this step** option to prevent users from skipping ahead to this step before the preceding steps are completed.

4. Click **Submit** to close the properties panel.

Configure post-processing actions

In a screen flow, post-processing actions listed on the flow action are not automatically executed. Follow these steps to force post-processing actions to execute:

1. Double-click an Assignment shape to open the properties panel.

2. Select the **Perform post-processing when navigating away from step** option to run validation and post-processing actions each time a user leaves the shape.

3. Click **Submit** to close the properties panel.

Configure persistence options

In a screen flow, Pega commits each step to the database as the step executes. To prevent commits to the database when each step is executed:

1. Double-click the Start shape to open the properties panel.

2. Select the **Save on Last Step** option to prevent commits to the database when each step is executed.

3. Click **Submit** to close the properties panel.

Remember to save your changes to the screen flow record.

Adding attachments

Introduction to adding attachments

Many cases need associated assets such as files, screen captures, and scanned documents to substantiate the case. You can also use Pega's internal security precautions to provide users access to search, view, add, and edit the attachments.

After this lesson, you should be able to:

- Describe the purpose of attachments
- Configure the ability to add attachments manually to a case
- Configure the ability to add attachments automatically to a case
- Control user access to attachments

Attachments

During case processing, end users may need to attach documentation containing additional information. For example, an insurance claim for a car accident might need the police report to identify who is at fault. A case attachment can be a file, screen shot capture, URL, or text note.

This video illustrates attachment options.

Many cases require associated documentation and assets such as files, screenshots, and scanned documents.

You use out-of-the-box tools to attach files to a case for further reference. You can use security precautions to ensure that only people who are allowed access to a specific case file can edit the attachments.

For example, a legal secretary can attach a scanned legal document, making it into an editable PDF.

When another member of the legal team needs to review the case documents, the team member can access the secured document.

When the team needs to look for a specific type of attachment within the case, the search is based on filter criteria, such as all file attachments.

Pega 7 supports configuring a case type to add attachments manually and automatically, as required by the specific application.

Pega 7 supports security and access restrictions for certain attachment options. For example, any member of the claims processing group may add attachments, while only claims managers have the right to delete attachments. You use the **Attachment Category** rule type to enable security and restrict access for certain attachment operations and organize and classify attachments. You can differentiate between a police report and a repair estimate and assign each a different access restriction.

Attachment file conversion and editing

When a user designates a file as an attachment, the system uploads a copy of the file and links it to the history of the work item. The system saves attachments for correspondence in HTML format (.htm file type) even if a user edited the correspondence with Microsoft Word. If the correspondence includes embedded images or other embedded objects, the attachment is saved as a .zip archive containing the objects and the .htm file. The system converts the .zip file internally into characters using Base64 encoding.

File attachments are normally a permanent read-only part of the history of a work item. However, in some applications, you can edit and update the attached file (using Windows workstation software), creating a higher-numbered version of the file attachment. The application retains all versions, in sequence. You can edit the description of a new version to add a number or other distinguishing information.

For example, a photograph stored as a .jpg file might be difficult to interpret or might contain extraneous material. When rules have the appropriate configuration, you can open and manipulate the .jpg file (using Windows imaging software) to crop, filter, or annotate the image. Pega 7 Platform retains the original unaltered file as the first edition and the updated file as the second edition.

Similarly, you can edit a project plan file with Microsoft Project, or revise a Microsoft Word document.

The standard flow action **Work-.EditAttachment** supports opening and editing file attachments.

Attachment classes

An attachment is an object of a class that inherits from the **Data-WorkAttach** class. Pega provides several standard classes to describe attachments.

Data-WorkAttach-File — Holds files of any type and format, including PDF files generated by an application

Data-WorkAttach-Note — Contains text that is pasted into, or typed directly into, a work item

Data-WorkAttach-URL — Records an Internet Uniform Resource Locator (URL) or Uniform Resource Identifier (URI)

Data-WorkAttach-ScreenShot — Holds screen shot attachments that usually record facts about the work item that were obtained from another system

Data-WorkAttach-ScanDocument — Contains a TIFF image file created by a scanner

Data-WorkAttach-ECM — File attachments saved in an external enterprise content management (ECM) system, accessed through a Connect CMIS rule

Attachments are stored in a different database table than cases. By default, attachments are stored as rows in the **pc_data_workattach** table.

KNOWLEDGE CHECK

Pega supports attaching documentation to cases. Name the documentation feature that must be specifically configured.

Security and access restrictions for certain attachment options
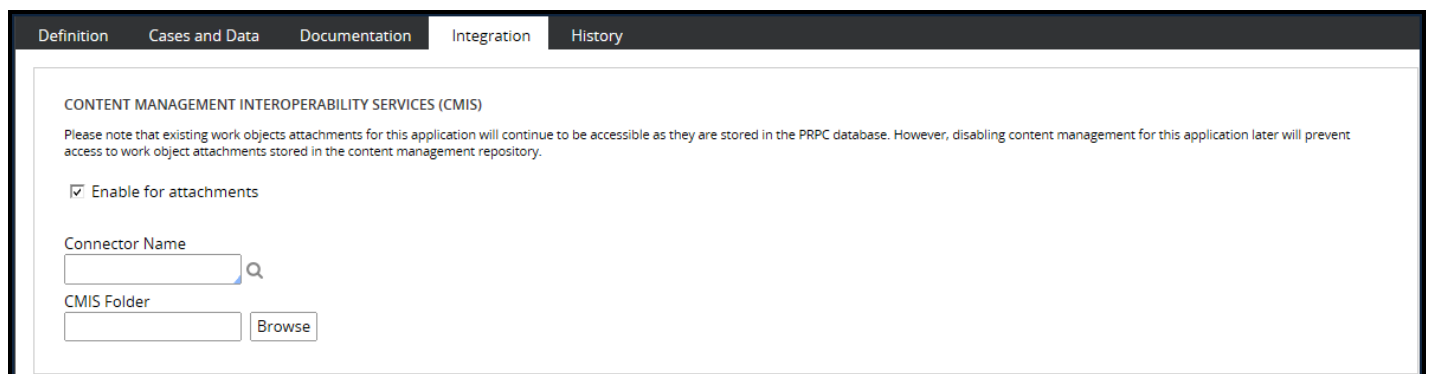
How to configure a case to accept attachments

You use standard perform and review harnesses to enable users to add attachments to a case. The add attachment function is built into the standard user interface (UI). End users access the Case Attachments section to add attachments.

The ability to add attachments may be unavailable with custom perform or review harnesses. System architects may configure a local add attachment action or select one of the available standard flow actions. You can also configure local actions for UI events. End users select local actions from the Other actions menu.

Enterprise Content Management attachments

Pega provides the ability to configure external attachment storage. Pega uses a Content Management Interoperability Services (CMIS) protocol to connect with an external enterprise content management system (ECM). Compatible CMIS protocols include Microsoft SharePoint and IBM FileNet.

On the application rule **Integration** tab, select the **Enable for attachments** check box to enable external storage.

Next, in the **Connector Name** field, specify the Connect CMIS rule used to connect to the external CMS system. If necessary, click the **Magnifying glass** icon to create a new Connect CMIS rule.

In the **CMIS Folder** field, use the Browse button to select the content management system storage directory.

Anyone who has access to a particular case can view the list and content of the attachments. Case access is subject to the security configuration for the category.

The Advanced attachments window groups attachments by type. You use the attachments section in the harnesses for quick viewing.

Attach Content smart shape

You use the Attach Content smart shape to add a specific file, note, or URL automatically to a case (for example, a standard Terms and Conditions form for user review). You can configure the Attach Content

shape after you add it to a flow. By customizing the shape, you can control the kind of information that is attached to the case at run time.

Double clicking in the Attach Content smart shape displays the attachment type selection list. You can select the format of the expected attachment. You also enter data specific to the attachment file format, including a description, the external file storage location (if relevant), and any relevant rule name. The Attachment Category field is optional.

KNOWLEDGE CHECK

The ability to add attachments to a case is available out-of-the-box. Which attachment function must be configured by system architects?

External attachment storage

©2017 Pegasystems Inc.

How to configure attachment access

Pega 7 enables case attachment access and edit control through attachment categories.

An **attachment category** rule represents a business classification indicating the attachment content or significance. The attachment category controls operations on attachments including create, edit, review, and delete. Attachment categories are part of the Security category and are instances of the *Rule-Obj-AttachmentCategory* rule type. Users can select an attachment category when adding attachments to work items.

This screen capture shows the standard File Attachment Category rule.
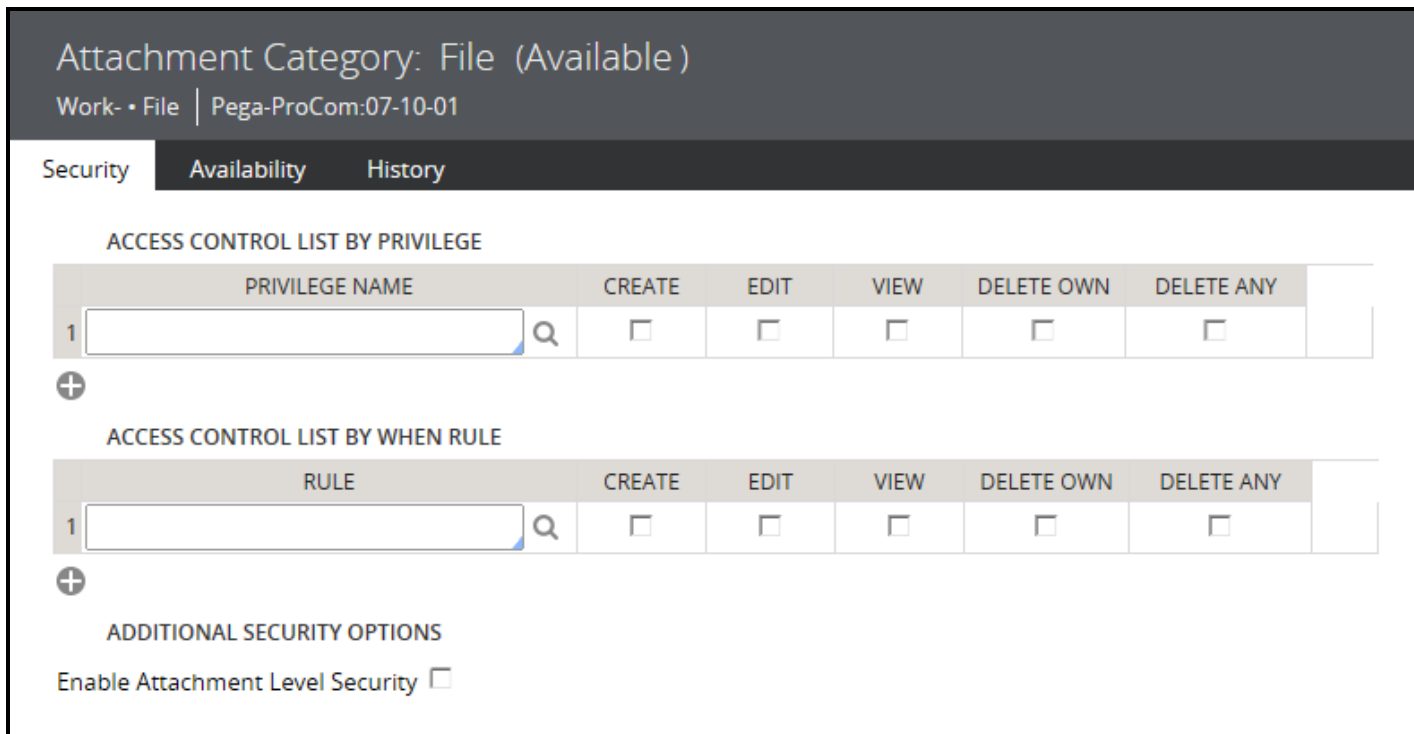
You can specialize the file attachment category rule by copying it to a different ruleset and Applies To class.

You can reuse the standard categories or create new attachment category rule instances, such as invoice and packing slip.

You use the Security tab to control access to the attachments through privileges and when rules. In the ACCESS CONTROL LIST BY PRIVILEGE section, in the PRIVILEGE NAME field, you select a privilege rule. The system uses the Applies to class of the attachment category to validate the privilege name.

Select any of the following check boxes that apply:

- CREATE — Add category attachments
- EDIT — Edit category attachments
- VIEW — View category attachments
- DELETE OWN — Delete category attachments that they added earlier
- DELETE ANY — Delete any category attachments

You can use the **Add a row** icon to add multiple privileges. The order of rows in this section is not significant. In categories with multiple rows, users must hold at least one privilege to gain access.

You can define a list of when rules to control access to the attachments. All when rules must evaluate to true for a qualified user to be granted access.

In the ACCESS CONTROL LIST BY WHEN RULE section, in the RULE field, you select a when rule. The system uses the Applies to class of the attachment category rule to find the when rule. Next, select any of the operation check boxes that apply.

The Enable Attachment Level Security option allows the operator who attaches a work attachment of this category to identify one or more work groups that have access to the attachment.

When enabled, the attachment-level restriction operates in addition to and independently of any restrictions defined on the tab for the category.

The Availability tab provides a list of the attachment types.

You select the check boxes for the attachment types that are valid for the category and the work type identified by the Applies to class of this attachment category rule. In this example, only File attachments are valid for the category.

If no types are selected, a default category rule in Work- class is used for its respective type. The default category has no security restrictions.

KNOWLEDGE CHECK

What is the purpose of attachment categories?

The attachment category controls operations on attachments including create, edit, review, and delete. Attachment categories are part of the Security category and are instances of the *Rule-Obj-AttachmentCategory* rule type. Users can select an attachment category when adding attachments to work items.

Configuring flow action pre- and post-processing

Introduction to configuring flow action pre- and post-processing

In Pega, you can add pre- and post-processing actions to a flow action to manipulate data. These actions enable you to add related tasks to the flow action. For example, you can concatenate a person's first name and last name to create their full name.

After this lesson, you should be able to:

- Explain how pre- and post-processing configurations affect flow actions
- Configure pre- and post-processing actions for a flow action

Pre- and post-processing in flow actions

Sometimes you need to perform a set-up or wrap-up action in conjunction with a flow action. For example, you may need to initialize items in a list or copy data from one property to another. To satisfy these needs, you can add pre-processing and post-processing actions to a flow action.

Consider an example of a trip case type. TGB hosts an annual meeting for employees and vendors. TGB's employees use the trip case type to finalize travel arrangements for all business trips. Approximately 60% of the trip cases processed are for the annual company meeting. TGB's application requirements include creating a default event for the annual company meeting. Default values populate the event form at rendering. If the user removes the default event, the event values do not populate again. The case type must also be suitable for all company travel requests.

You can use a data transform to populate the annual company meeting on the event form as a pre-processing action. The first time a user opens the form, the data transform populates the event on the form. Creation of the default event occurs when the user selects the flow action, or automatically if the flow action is the default action for the assignment.

Note: When you configure a flow action with a pre-processing action, Pega performs the action whenever a user selects the flow action and each time the user is presented with the assignment. In the preceding use case for a trip case type, if the user completes the assignment and later returns to the assignment — for example, to update the details of their trip — the event re-populates on the form. For this reason, add logic to a pre-processing data transform or activity to test whether to perform the action.
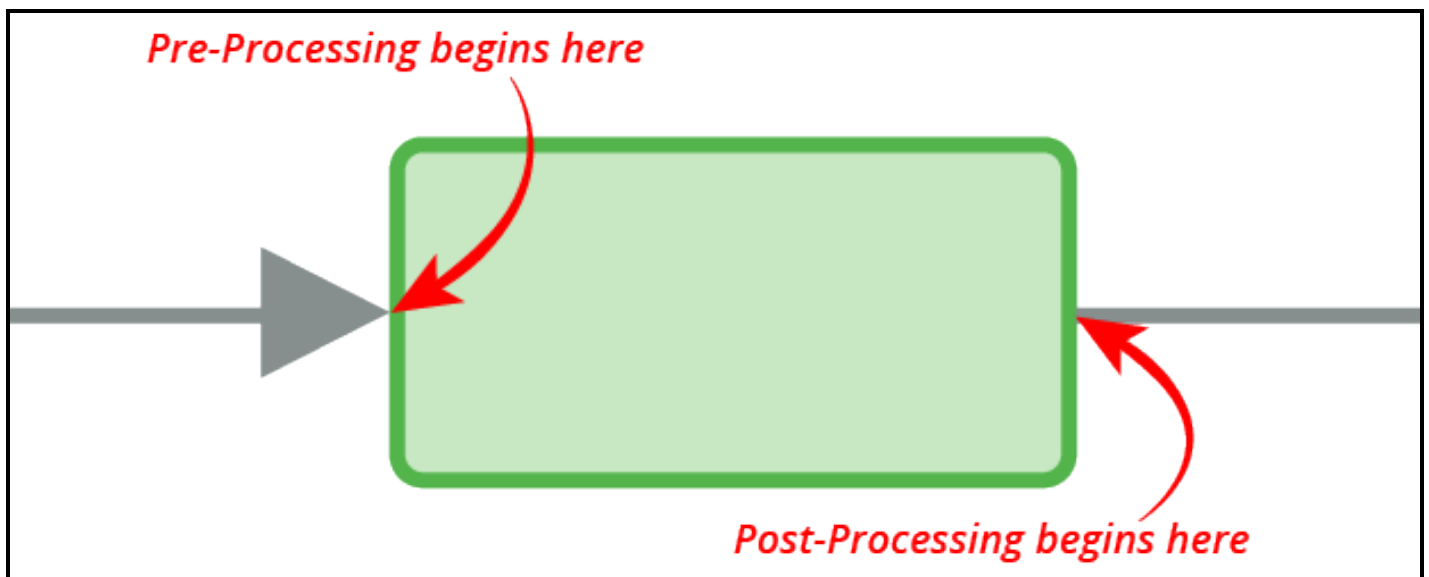
Another common use case for post-processing is when a customer's billing address is also the shipping address. A data transform copies the property values from the billing address page to the shipping address page when a box is selected. You add the data transform to the flow action as a post-processing action. When the user submits the form, the application copies the contents of the billing address page to the shipping address page.

When you configure a flow action with a post-processing action, Pega performs the action each time you perform the action. In the previous example of a billing address, each time the user submits the billing address form, Pega performs the post-processing action to copy the billing address to the shipping address.

Tip: Verify that adding an action to the flow action is the best way to perform the action. For example, when configuring concatenation of a user's first and last names, consider using a declare expression. The concatenation is performed only when needed with a declare expression. With a pre- or post-processing on the flow action, the concatenation is performed every time.

KNOWLEDGE CHECK

ANSWER   If a flow action includes a pre-processing data transform or activity, when does Pega perform the action?
Pega performs the pre-processing action each time the flow action is presented to the user.

How to configure pre- and post-processing for flow actions

When considering whether to add a data transform or activity to a flow action as either a pre- or post-processing action, analyze the requirement and the case type.

Consider the impact of a pre- or post-processing action when configuring the data transform or activity. For example, if you add a pre-processing action to initialize a value or a list, you may not want to repeat it if you reload the flow action. You should add logic to test if the value or list is already initialized.

For a pre-processing action, another consideration is flow action likelihood. Pega automatically loads the flow action with the highest likelihood, so a pre-processing action on the flow action automatically executes when the user reaches an assignment.

The primary concern is the use case. Reusing application components and the Situational Layer Cake are key Pega benefits. You can add actions and data transforms to a flow action both before and after processing. Whenever an application uses the flow action, Pega performs the pre-or post-processing action. If a pre- or post-processing action is only applicable to one case type, then specialize the flow action for the case type before adding the pre-or post-processing action.

You analyze the requirement and the case type. Identify the affected flow action. Determine the appropriate location for the new data transform or activity, before or after the flow action executes.

When the data transform or activity is ready, locate the locate the **Action** tab of the **Flow Action** form for the appropriate process. Select the data transform or activity to apply and enter it the **Pre-processing** or **Post-processing** section as appropriate. Selecting and entering the data transform or activity completes the form.

©2017 Pegasystems Inc.

For more detailed information on completing the Flow Action form, read the Help topic, Flow Action Form, Completing the Action Tab.

To see an example of creating data transforms and then using the data transforms as pre- and post-flow action processing, read Using Data Transforms in Flow Actions.

KNOWLEDGE CHECK

ANSWER	Why is analyzing the requirement and the case type important?

You must identify the impacted flow action and determine whether the new data transform or activity should be placed before or after the flow action. You must also create or update the desired data transform or activity.

Cirucmstancing rules on multiple variables

Introduction to circumstancing rules on multiple variables

In this lesson, you learn how to create a circumstanced rule that is based on multiple variables. You also learn how circumstancing impacts rule resolution.

After this lesson, you should be able to:

- Describe how circumstancing affects rule resolution
- Explain how Pega supports circumstancing rules on multiple variables
- Override circumstances with a base rule
- Circumstance a rule on multiple variables

How to circumstance a record with multiple variables

You use multivariate circumstancing when you want to use multiple properties to circumstance a record. The first thing you do to configure multivariate circumstancing is create a Circumstance Template. After creating a template, you create one or more Circumstance Definitions. Then, using the Circumstance Template, you circumstance the rule.

Creating a Circumstance Template

A **Circumstance Template** defines the properties used to determine if the circumstance is valid. For example, an internet provider wants to circumstance a correspondence based on the state the customer lives in and the reward level. You would create a Circumstance Template that defines the state and reward level properties.

Circumstance template

State = X
RewardLevel = Y
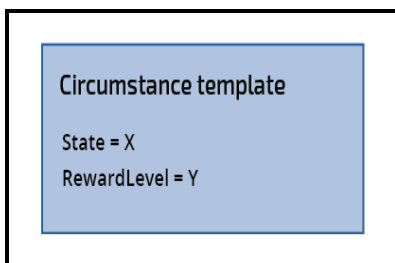
To create a new Circumstance Template, reference this Help topic: <u>New Circumstance Template</u>.

Creating a Circumstance Definition

After you define a Circumstance Template, you create one or more Circumstance Definitions that use the template. A **Circumstance Definition** defines the values for the Circumstance Template. The values determine if the Circumstance executes. In the internet provider example, you create Circumstance Definitions for each combination of state and reward level as needed. You do not have to account for all possible combinations. If a combination does not have a Circumstance Definition defined, the rule resolution algorithm uses the base rule.
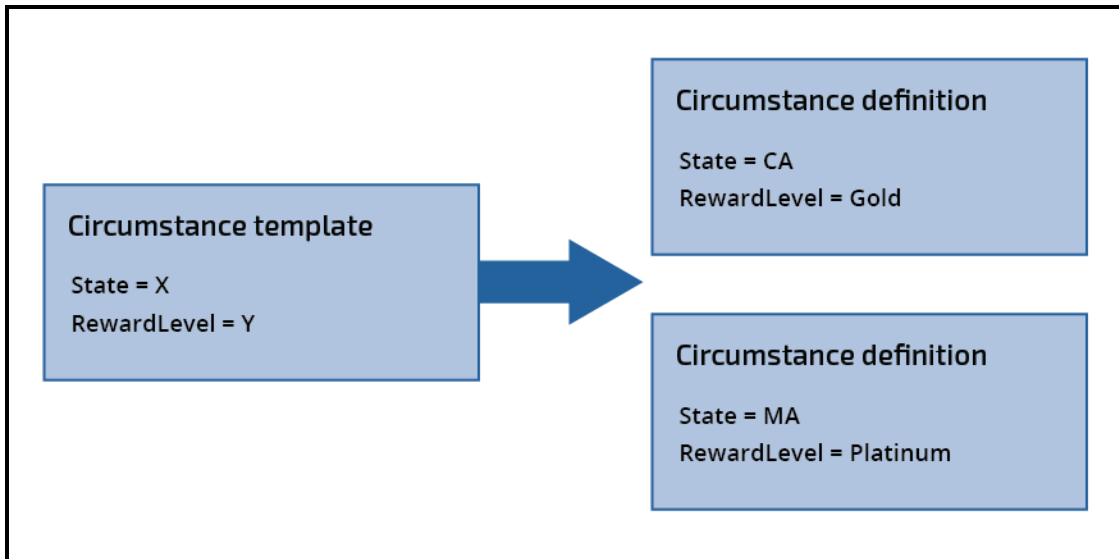
To create a new Circumstance Definition, reference this Help topic: New Circumstance Definition.

Circumstance a record using the Circumstance Template

After you create a Circumstance Template and one or more Circumstance Definitions, you circumstance the appropriate record. In the Internet provider example, you would circumstance the correspondence record and reference the Circumstance Template that uses state and RewardLevel.

To add a circumstance to a record, reference this Help topic: Specializing a record.

KNOWLEDGE CHECK

⌄ ANSWER	What are the three activities you must complete to configure multivariate circumstancing?

1. Create a Circumstance Template
2. Create one or more Circumstance Definitions
3. Circumstance the record and use the Circumstance Template

Circumstancing on multiple properties

You may have a business case that requires you to circumstance on multiple properties. For example, mortgage rates are typically impacted by the type of property and the type of customer. You can do the following things to implement this:

- Circumstance a base rule that is already circumstanced with a different property. Multiple property circumstancing in this way is possible only when the circumstanced rules are in different rulesets or Applies To classes. For example, a MortgageRate rule that is circumstanced by the PropertyType property can be circumstanced again by the CustomerSegment property.

Rule Type	Applies To	Rule Name	Circumstanced By	Value
Activity	PegaSample	MortgageRate	.PropertyType	"VacationHome"
Activity	PegaSample	MortgageRate	.CustomerSegment	"HNI"

- Circumstance a rule by a different property in the same ruleset by withdrawing the existing circumstanced versions. The circumstanced version must be in the higher ruleset version. For example, a MortagageRate rule that was circumstanced by the PropertyType property can be circumstanced again by a different property, CustomerSegment, in a higher ruleset version after the rule MortgageRate(PropertyType) is withdrawn.

	Rule Type	Applies To	Rule Name	Circumstanced By	Value
Withdrawn	Activity	PegaSample	MortgageRate	.PropertyType	"VacationHome"
	Activity	PegaSample	MortgageRate	.CustomerSegment	"LNI"

Note: If you withdraw a circumstanced rule in a ruleset version, you cannot specialize the base rule again in that ruleset version.

How circumstancing affects rule resolution

Rule resolution of circumstanced rules

When several circumstanced versions of the base rule are available, the rule resolution algorithm selects a rule based on the availability of rules in the higher ruleset versions as shown in the following table.

Circumstanced rule in higher ruleset	Circumstanced rule in lower ruleset	Base rule in higher ruleset	Result
True	True	True	Circumstanced rule in higher ruleset version is resolved
True	False	True	Circumstanced rule in higher ruleset version is resolved
False	True	True	Base rule in higher ruleset version is resolved
False	False	True	Base rule in higher ruleset version is resolved

Watch the following video to understand how rule resolution and circumstancing work.

KNOWLEDGE CHECK

⌄ ANSWER   You have a circumstanced rule in a lower ruleset and a base rule in a higher result. Which rule would run?
The base rule in the higher ruleset

©2017 Pegasystems Inc.

How to override circumstanced rule

When the rule resolution algorithm determines the ranking of a rule, the version of the rule is less important than the circumstance. Understanding how the rule resolution algorithm processes circumstancing is important when updating one of the variations of the rule, or the base rule itself.

Consider the following circumstance:

During tax season, complete the request in two days instead of three.

What if there is a requirement that requires the base rule to change from three days to four days? Updating the base rule is easy — you save the rule into a new version of the rule and update it. You do not need to save the circumstanced versions of the rules because the rule resolution algorithm ranks version as less important than the circumstance. So it is possible to have a circumstanced rule in version 01-01-01 and the base rule in 01-01-15. At run time, if the rule matches the circumstance, then the rule from version 01-01-01 executes. Otherwise, the rule from version 01-01-15 executes.

You have two choices to remove a circumstance: override the base rule, or override the rule by withdrawing it.

Overriding a circumstanced rule with the base rule

One way to override a circumstanced rule is by defining a new base rule. You use this method when you want to override all circumstances for a given rule. When a rule is marked as a base rule, all previous circumstances are ignored during rule resolution.

Any version of any rule that can be circumstanced can also be designated as a base rule. When setting the rule's availability, check the **Base rule** option to designate a rule as a base rule.
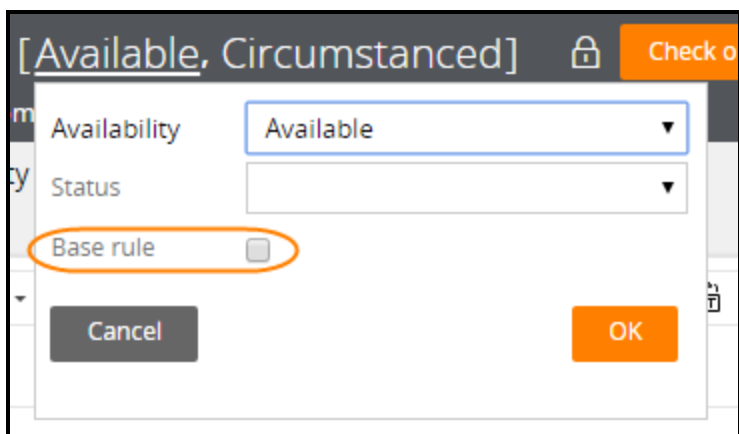
Selecting the **Base rule** option indicates that this version of the rule is now considered this rule's base and any previous circumstances no longer apply. Consider an example where you vary the welcome email based on the department a new employee is in.

The following table lists of all variations of a correspondence rule.

	Version	Circumstance
1	01-01-01	None

2	01-01-01	.Dept = Accounting
3	01-01-01	.Dept = Engineering
4	01-01-15	None
5	01-01-20	.Dept = Engineering
6	01-01-25	None, Base rule Checked
7	01-01-30	.Dept = Accounting
8	01-01-35	None

Based on the information in the list, if you execute this rule when *.Dept= Accounting*, Pega uses the version of the rule on line 7 (ver. 01-01-30). If you execute this rule when *.Dept = Engineering*, Pega Uses the version of the rule on line 8 (ver. 01-01-35).

This is because the rule on line 6 (ver. 01-01-25) has the base rule checked. All the rules previous to this version (rules 1 through 5) are no longer applicable to the ranking. When the system looks at only those rules available for ranking, the circumstance for Engineering is not applicable. As a result, the highest version of the rule with no circumstances defined is chosen.

Overriding a circumstanced rule by withdrawing a rule

You can also override a circumstanced rule by adding a withdrawn rule with the same circumstance. You use this method of overriding when the circumstance is no longer needed.

You must create this withdrawn rule in a higher ruleset version than the original rule to be deleted or blocked so it is chosen first in rule resolution. You must also define the new rule with the same characteristics as the rule to be withdrawn:

- Same defined-on (Applies To) class
- Same qualifiers (Circumstance Property, Property Value, Date Property, Date Property Value, Start Date, or End Date)
- Same ruleset name
- Same major version

Using the previous example, if the requirement is to remove the circumstance because the engineering department is no longer an exception, you would withdraw that circumstance and not impact the base rule or the accounting circumstance.

	Version	**Circumstance**
1	01-01-01	None
2	01-01-01	.Dept = Accounting
3	01-01-01	.Dept = Engineering
4	01-01-15	None
5	01-01-20	.Dept = Engineering
6	01-01-30	.Dept = Engineering, Withdrawn

| 7 | 01-01-30 | .Dept = Accounting |
| 8 | 01-01-35 | None |

Create a new rule as shown in line 6 to remove the circumstance where *.Dept=Engineering*.

KNOWLEDGE CHECK

ANSWER	What are the two ways to override a circumstance rule?
Override the base rule, or withdraw a rule in a higher ruleset.	

UI DESIGN

Customizing a user portal

Introduction to customizing a user portal

User portals provide application users with the tools and options needed to work with a Pega application. Each portal is tailored to a specific user role. In this lesson, you learn how to customize a portal for application users.

After this lesson, you should be able to:

- Explain the role of user portals in applications
- Explain the role of a harness in a Pega UI
- Explain how user portals are organized
- Customize a user portal

User portals

Applications often support multiple types of users, such as case workers and managers. Each type of user interacts with the application in a unique manner. For example, case workers create and process cases, while managers track the progress of cases. Each type of user needs access to tools and features that support their role in case processing.

A **user portal** is the application user's view into the application. Pega provides several user portals. Each portal is customized to the needs of a specific type of user. For example, case workers can use their portal to create new work, complete existing work, and run reports. Managers can use their portal to monitor cases in progress and run reports that show case worker and case metrics.

Pega provides default portals for case workers and managers. While these portals can be used in an application as is, some situations require that you customize the layout of the portal or the tools presented to the user.

KNOWLEDGE CHECK

What is the purpose of a user portal?

The purpose of a user portal is to provide users with a view into their application. Each portal is customized to the needs of a particular group of users.

Harnesses

Pega encourages architects to follow the principles of modular application design, including user interface (UI) design. To promote modular design, Pega provides different types of UI records for content, structure, and formatting. Harness records describe the structure of the UI.

Harness records

A **harness** organizes the structure of a portion of the user display. In Pega, you use a harness to organize either a work form or a portal.

Pega applications commonly use four standard harnesses to organize the content of user forms.

Harness name	Usage
New	Supports the creation of new cases.
Perform	Allows users to select a flow action to perform to complete an assignment.
Review	Presents an assignment in read-only mode, preventing data entry.
Confiirm	Presents a read-only confirmation of completion of an assignment if the next assignment is not performed by the user.

Pega also provides harnesses specialized for organizing user forms in screen flows. For a full list of the standard harnesses available in Pega, see the Help topic Standard harnesses.

Harnesses that allow users to select a flow action and complete an assignment contain an **action area**. When users select a flow action to perform, such as an approval form, Pega displays the content for the selected flow action in the action area.

KNOWLEDGE CHECK

What is the purpose of an action area in a harness?

An action area displays the content of a user form when users select a flow action. The action area describes the work users perform to complete an assignment.

Harnesses that organize a user portal contain a screen layout. A **screen layout** organizes the elements of the browser window into a main content pane and smaller surrounding panes. For example, the Header Left screen layout divides the portal into three areas: a header, a smaller left pane for navigation, and a larger content pane for displaying cases and reports.

Each pane of the screen layout references a section that contains the content displayed in the pane. To modify the content in these sections, you use Live UI to identify and open the section to configure.

KNOWLEDGE CHECK

⌄ ANSWER	What is the purpose of a screen layout in a harness?

An screen layout defines the structure or a portal. The screen layout defines the panes of the portal. The content of each pane is determined by referencing a section.

How to customize a user portal

You can customize the default end-user portals provided in Pega to meet many project requirements. If the requirements are complex enough, you can also create a custom application portal.

In Pega, a portal is represented with a **portal rule**. A portal rule identifies the type of user expected to use the portal, the harness used to organize the portal contents, and the skin that defines the branding applied to the portal.

To configure a Pega portal, you:

- Identify the intended user role and portal type

- Organize the layout of the portal

- Customize the branding of the portal

- Customize the content and tools available to users

- Configure an access group to reference the portal if necessary

Portal records are listed in the **User Interface** category in both the Records Explorer and the **+Create** menu.

 Note: Portal records are classless and do not appear in the App Explorer.

Identify the intended user role and portal type

In Pega, you configure a portal for use by either users or developers. User portals are intended for users who do not routinely need to update rules. Developer portals are intended for system architects and business architects, who routinely update rules. User portals require less memory on the user's workstation than developer portals. Unless the intended user configures rules on a daily basis, configure a new portal as a user portal.

 Note: Users can configure delegated rules in a user portal.

Pega supports two portal types: composite and custom. Composite portals are defined by harnesses and sections. Composite portals are cross-browser compatible and support Microsoft Internet Explorer, Mozilla Firefox, Apple Safari, and Google Chrome browsers. Custom portals are defined by an activity. As a best practice, configure a new portal as a composite portal.

You select the user role and portal type on the **Details** tab of the portal record.

Organize the layout of the portal

The organization of the portal affects how the contents of the portal are presented to the user. The default user portals are organized with a header, a left navigation pane, and a content pane. To change the layout of the portal, you change the screen layout used in the harness. To modify the content in these sections, you use Live UI to identify and open the section to configure.

To create a new layout for a portal, create a new harness in the *Data-Portal* class, and add a section layout to the portal. Then reference the harness on the **Details** tab of the portal record.

Change the branding of the portal

You can customize the appearance of a portal by applying a skin. Skins contain instructions for formatting elements of the user interface, such as text size, font style, and background color. To customize the appearance of a portal, you choose between applying the application skin to the portal and configuring a skin for the portal.

When you select the application skin, Pega applies the skin for the active application to the portal. If the user switches applications, Pega applies the skin for the new application to the portal. If you create a new portal, Pega configures the portal to default to the application skin.

To apply a skin to the portal, rather than reusing the application skin, select the **Other skin** option on the **Details** tab of the portal record, then enter or select the skin to apply. For example, a portal is used across an entire organization. Within the organization, each division customizes its branding, including fonts and color schemes. In this situation, consider applying a skin to the portal to prevent changes to the portal when users switch between applications.

Customize the content and tools available to users

You can add or remove content displayed in a portal by updating the sections referenced in the screen layout. To modify portal content, use Live UI to identify and open the section to update. To override records provided in the *UI-Kit-7* ruleset for standard portals, copy the record into an application ruleset.

Customize portal menus

You can add a menu to a portal by using a navigation record. A **navigation record** defines the entries in a menu and the action performed when a user selects the menu item. Navigation records are used to organize the menus displayed in standard portals such as the Case Manager and Case Worker portals.

A navigation record contains a list of menu items. For each menu item, you associate a click event and resulting action, such as logging off or displaying a harness.

For information on configuring entries in a menu using a navigation record, see the Help topic Navigation rule — Completing the Editor tab.

For information on configuring an action for an entry in a navigation record, see the Help topic Navigation rule — Completing the Node Details Action tab.

For an example of configuring a navigation record, see the PDN topic How to create context menus for grid layouts using navigation rules.

Navigation records are organized in the **User Interface** category in the Records Explorer and the **+Create** menu.

KNOWLEDGE CHECK

> A navigation record is used to do what?
>
> A navigation record is used to configure the entries on a menu. It lists a menu entry and identifies the action performed when a user selects the entry.

Replace the Pega logo

You can update the icon displayed in the upper left corner of the portal. Standard Pega 7 portals display the Pega 7 icon. You can update the portal configuration to display a different icon, such as a company logo. To add an image or other non-text file to a Pega application, you create a binary file record. A **binary file** record acts as a wrapper for the file, providing the security, inheritance, versioning, and deployment benefits of rule resolution.

To update the icon displayed in a portal, create a binary file record for the icon, then use Live UI to identify and update the section containing the icon. For instructions on configuring a binary file record, see the Help topic Binary File rules — Completing the Main tab.

Binary files are organized in the **Technical** category in the Records Explorer and the **+Create** menu.

KNOWLEDGE CHECK

What benefits does a binary file record provide for storing non-text files?

A binary file record is a container for a non-text file. It provides the security, inheritance, versioning, and deployment benefits granted through the process of rule resolution.

Customize the dashboard content

You can customize the content displayed on the dashboard of the Case Manager portal. For example, a manager often wants to view the status of cases processed by their unit or work group. You can configure the Case Manager portal to display one or more **dashboard widgets** that provide insight into the status and progress of open cases. Dashboard widgets are organized into two or more slots using a **dashboard template**.

To customize the dashboard, determine the template to use to organize the dashboard, then add widgets to each slot. For example, to display a report on the dashboard, add the Report widget to a slot, then select the report to display. For a list of widgets available in Pega, see the Help topic Dashboard widgets. For instructions on adding a widget to the dashboard, see the Help topic Adding a widget.

Tip: You can also create dashboard widgets. For instructions on creating widgets, see the PDN topic Quick start: Creating a custom widget.

After you finish adding or removing widgets from the dashboard, you can publish the dashboard as a default dashboard for use by managers.

Note: Managers can personalize their dashboard with any of the standard widgets provided in Pega, or any custom widgets you create. Once a manager customizes and publishes their dashboard, they no longer view any changes you make to the default dashboard.

Configure an access group to reference the portal

To provide users with access to a user portal, add the portal to one or more access groups. To add a portal to an access group, list the portal in the Available portals section of the **Advanced** tab of the access group record. For each access group, you select one portal for Pega to use as the default portal. Pega presents the default portal to a user when the user first logs in to Pega. The remaining portals are available to users from either the **Operator** menu or the **Launch** menu, depending on the portal.

Changing the logo image in a user portal

To replace the logo image in a composite portal, such as the Case Worker or Case Manager portals, you create a binary file record for the logo image, then update the portal header to use the correct logo image.

Create a binary file rule for the logo image

1. In the Designer Studio header, select **+Create > Technical > Binary File** to create a binary file rule for the image.

2. In the **Label** field, enter a short description of the record to use as the file name.

3. In the **App Name (Directory)** field, enter the server directory used for images. In most cases, the server directory is **webdb**.

4. In the **File Type (extension)** field, enter the file extension for the image, such as PNG.

   Tip: Use a PNG or GIF image if part of the logo is to be transparent.

5. Click **Create and open** to open the binary file record.

6. On the **File** tab, click **Upload file** to open the Upload file modal dialog.

7. In the Upload file modal dialog, browse to and select the image file.

8. Click **Upload file** to close the dialog and upload the image file.

9. Click **Save** to complete the configuration of the binary file record.

Update the portal header to use the logo image

1. From Designer Studio, use the Launch menu to open the relevant portal.

2. Use Live UI to identify the section used in the portal header. Standard Pega portals use the section *pyPortalHeader* to display the Pega 7 logo.

3. Save a copy of the section to the appropriate ruleset in your application.

   Note: Select a ruleset that matches the scope of the image. For example, for a company logo, save the header to an organization ruleset to reuse the customized header across all applications in the organization.

4. Locate the cell containing the image, and open the Cell properties panel. For the standard *pyPortalHeader* section, the cell containing the logo image is the blank cell to the left of the Case Manager cell.

5. In the Cell properties panel, in the **Image** field, enter the name of the binary file record you created for your logo image, in the format [*directory name*]/[*binary record name*].[*file type*]. For the Pega 7 logo displayed in the standard *pyPortalHeader* section, the image name is webwb/pega-7-logo.svg.

6. In the **Image size** field, select **Auto**.

7. Click **Submit** to update the logo used in the header section.

8. Click **Save** to complete your configuration.

Designing a mobile-ready application

Introduction to designing a mobile-ready application

Applications are accessed by desktops, laptops, tablets, phones, or other devices. When creating an application, you need to consider that users will access the application from both traditional and mobile devices.

After this lesson, you should be able to:

- Identify the design approaches that support a mobile ready application

- Incorporate mobile-specific features into an application

- Configure an application to support mobile users

How to build a mobile-ready application

Many applications are required to run on mobile devices such as tablets or phones, in addition to desktop and laptop computers. When the application you are developing will run on multiple devices, you must consider elements such as screen size, font size, and control position for your user interface.

When it comes to mobile-ready controls, layouts, and other user interface components there is no such thing as mobile-only in Pega. The advantage of Pega is that all the controls are responsive. You build a Pega application once and it works on all devices, and seamlessly adapts the user interface.
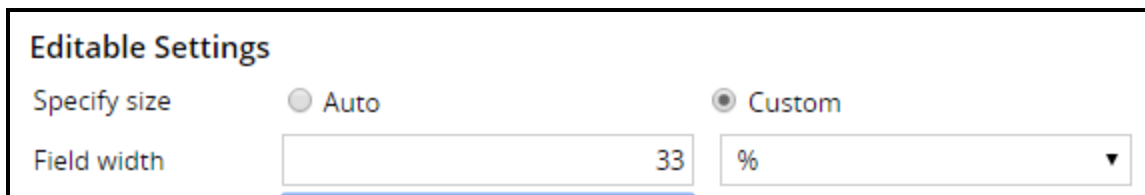
Mobile applications follow the same core user interface principles as other Pega applications. However, the designer and developer of the application must be aware that the user interface may be accessed on a mobile device. Follow these guidelines when building applications that are mobile ready.

Keep the user interface simple

Highlight the most important part of the screen by placing the important information in a section as visible. If there is supporting information, or less critical information that needs to be displayed, consider hiding it unless the user asks for the information. This can be done by putting less critical information under the most important content. Another technique you can use is progressive disclosure, such as using collapsible sections, menus, or some other paradigm.

Setting widths in percent (%)

When you are designing for mobile applications, avoid using pixels on layout formats. Using percentages enables the layout to automatically adjust for different screen resolutions. Using a fixed width in pixels has the risk of injecting a horizontal scroll bars on a smaller device.

User controls

Pega 7 comes with a wide repository of user controls. Always use out-of-the-box auto-generated controls. These are designed to work on all browsers and devices. They also have mobile specific configurations. For example, you can configure a date control to have a native rendering when on a mobile device. Set the data type of the property carefully so that the correct keypad choice appears (number, alphabetic).
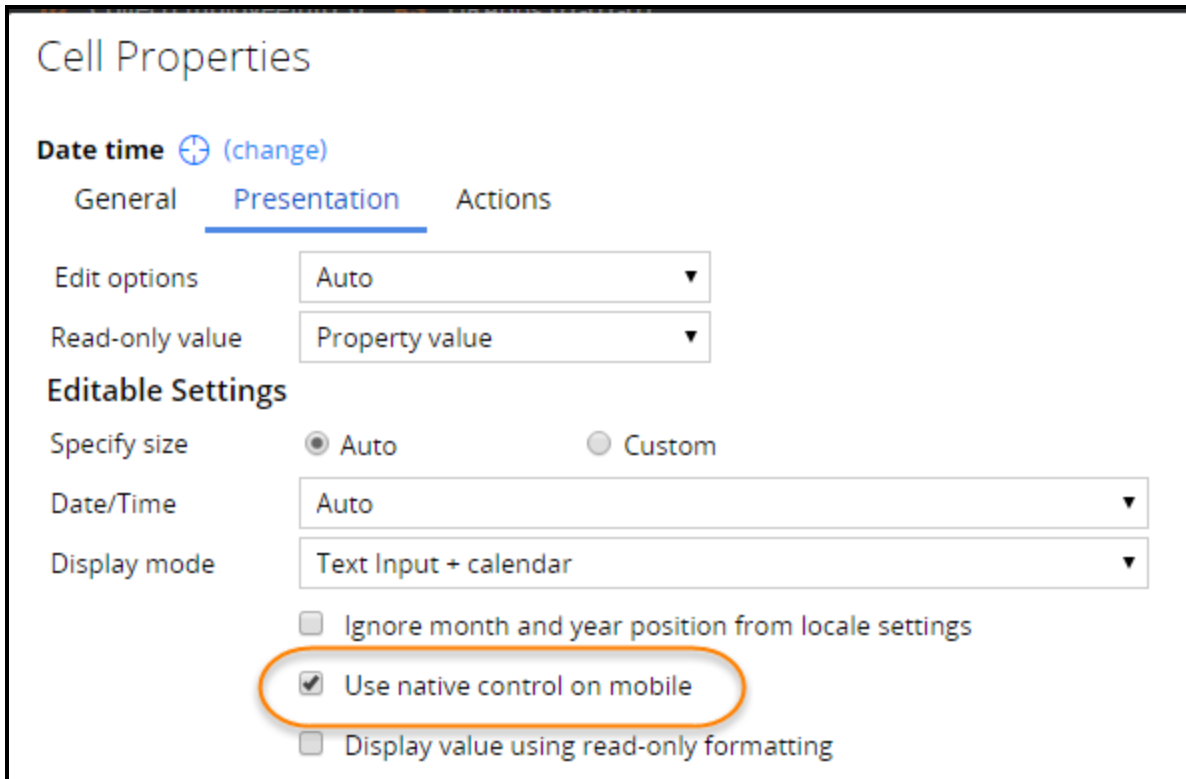
Design the application with Tap in mind

When using a mobile device, users may find buttons easy to tap, but not links. Many people find clicking a mouse easier than tapping a high pixel count touch screen. Ensure that actionable components are easy to tap. Generally, a tap target should be a minimum of 44 x 44 pixels. You can use layout level events and actions to help with that.

Events are configured to occur on a specific user action, such as navigating a menu using the mouse movements. This is harder to do on mobile devices but can be done by redesigning the menus to work when a user taps the screen. Use a Collapsible navigation menu to make this easier.

Designing layouts

A layout group is another key feature that helps when displaying layouts such as tabs, accordions, or stacked layouts depending on the screen size. Layout groups are formatted in the skin rule using responsive breakpoints to switch to a specific layout based on the resolution size. For example, you can see a layout change where you have a two column layout (like an inline grid double) responsively change to a one column layout (like stacked). The one column layout has better readability and usability on small screens.

Grids

Grids are great for displaying columnar data. However, grids do not work well on a mobile device when there are more than a couple of columns. Too many columns can be confusing to users trying to process information off-screen. If you have a grid with a lot of columns, identify the columns that are less important. Then identify the most important columns. Mark the columns in the grid as primary,

secondary, and other. The grid layout uses this information to adjust when displayed on a mobile device using responsive specifications. As the user resizes the screen or views the application on a mobile device, the user sees the primary column and only the other columns if there is room.

When you have a requirement to show a list, use a repeating dynamic layout instead of a grid. The repeating dynamic layout is more flexible than a grid, especially on a mobile device. Repeating dynamic layouts automatically adjusts the layout elements with respect to the screen size. This layout helps developers when creating an interface for displaying data that can be viewed on a tablet, monitor, or smartphone.

Sourcing controls and grids

Always use data pages as a data source regardless of whether the user interface is going to be used on a mobile device. This is required if you are designing applications to work on a mobile device when the device is not connected to a network, also known as offline mode. When a mobile application is in offline mode, all work completed is stored in data pages. The data pages are then synched to your application when the mobile device is online.
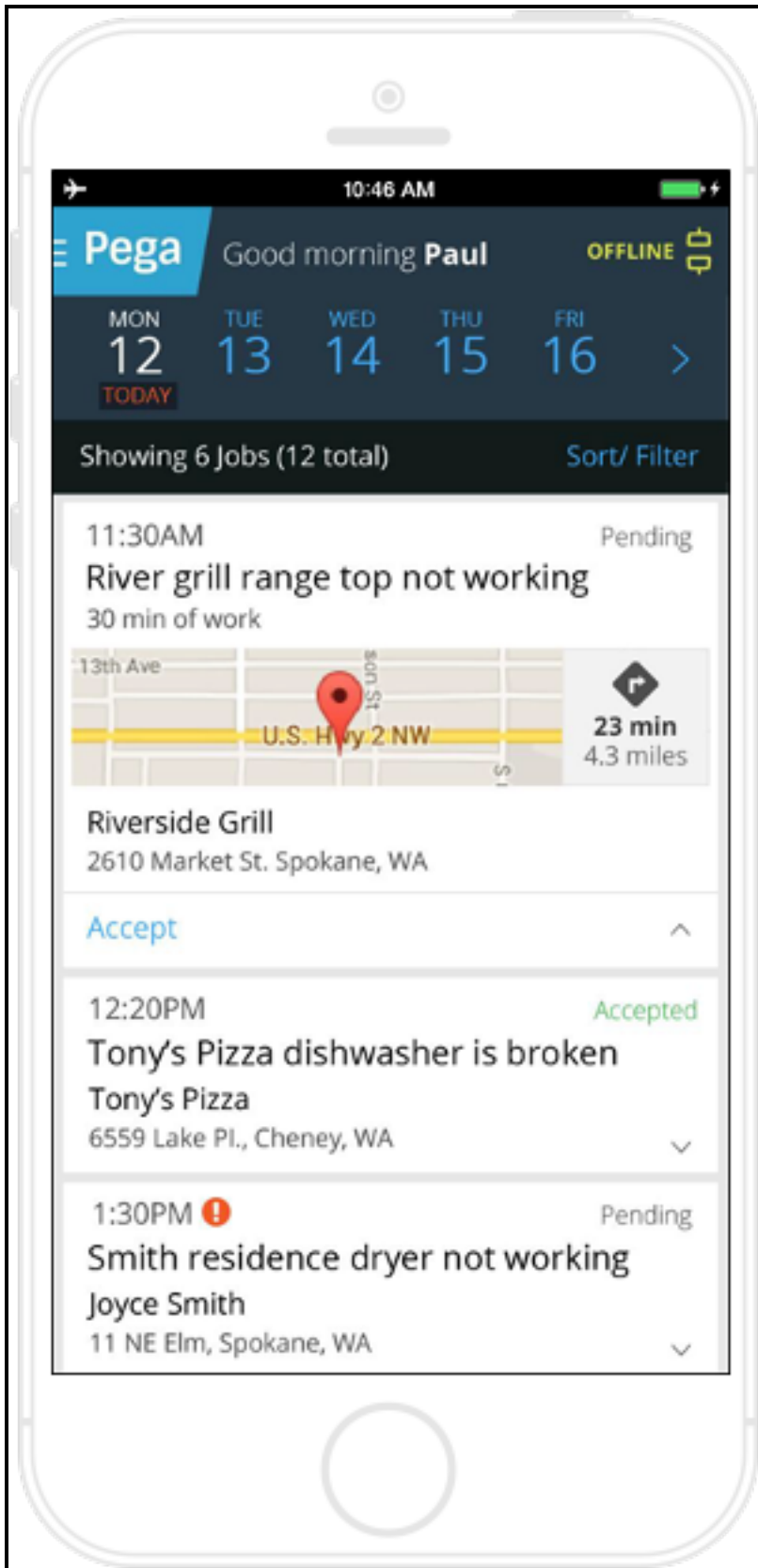
For more information about Pega's offline capabilities, see the PDN article [Offline Mobility](#).

Testing

Test the application on actual devices. Never rely exclusively on device emulators/simulators or on the Pega Mobile Preview for testing. Testing on actual devices ensures that your application performs well, provides the right experience, and works and behaves as you expect from a mobile application.

KNOWLEDGE CHECK

⌄ ANSWER Why is it important to use the out-of-the-box user interface controls and dynamic layouts?
Using the out-of-the-box user interface controls and dynamic layouts means your application is mobile ready without you having to do anything.

Mobile-friendly controls

Pega provides a set of controls that you can use to make your applications mobile-friendly. You add these controls to your application like other controls: determine the section where you want the control to display and add the control from Designer Studio.

Signature Capture

The Signature Capture control captures a user signature, either through mouse input or through a touch interaction on a mobile device.
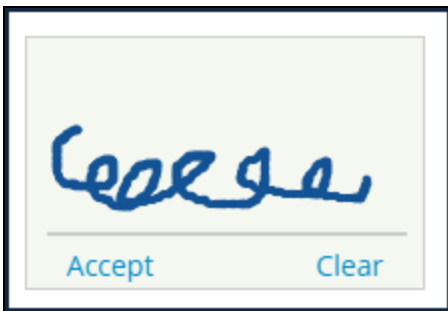
When a user clicks **Accept**, the entered signature is saved as an attachment in the case. Entering and accepting a signature can be repeated as many times as necessary, but only the last signature is saved.

When a user clicks **Clear**, the current signature is deleted so that a new one can be entered. If a signature has been saved as an attachment and then the user clears the signature, the signature attachment is still part of the case.

A signature is automatically cleared when the control is resized (for example, when changing the orientation of a mobile device).

To configure a Signature Capture control, go to the PDN article How to use the Signature Capture control.

Address Map

The Address Map control allows users to view and interact with location points in Google Maps from within the desktop and mobile applications. To display locations correctly on a map, geolocation must be active for your web browser or mobile device.

For the Address Map control to work correctly, you need to obtain a Google API key. In a production environment, the recommendation is to use an enterprise key since this allows for a greater number of geocoding hits per IP each day.

To learn more about configuring the Address Map control and obtaining a Google API key, go to the PDN article Using the Address Map control.
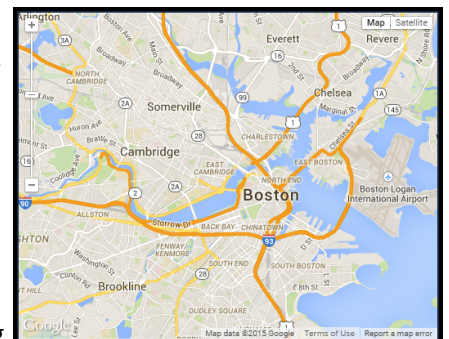
Attach Content

Add the Attach Content control to a desktop or mobile application to allow users to attach files to an application. This control can be formatted to display as a button, link, or icon.

When the Attach Content control is used at run time in a desktop application, the default file browser window opens.
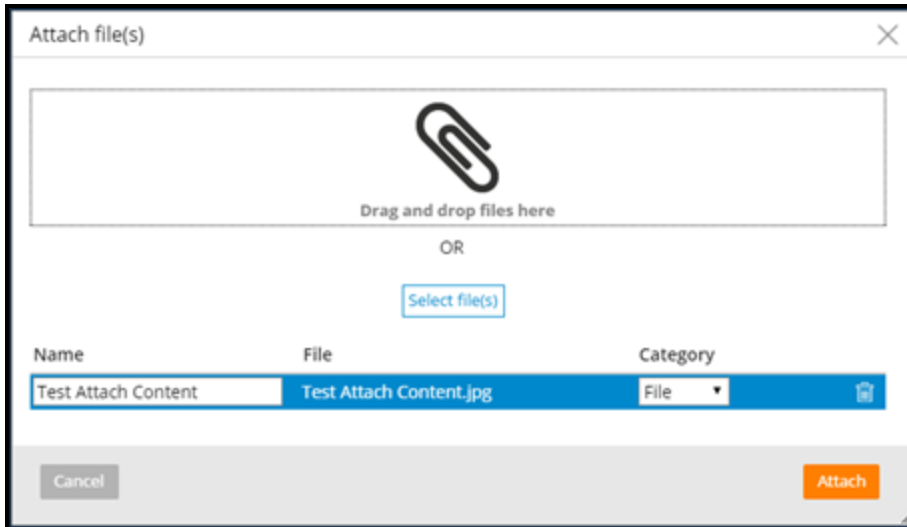
The control also works on certain mobile browser, such as Safari on iOS and Chrome on Android devices. In a mobile application, the actions are specific to the operating system.

On iOS devices, users can select an image file from the mobile device's camera roll or take a photo.
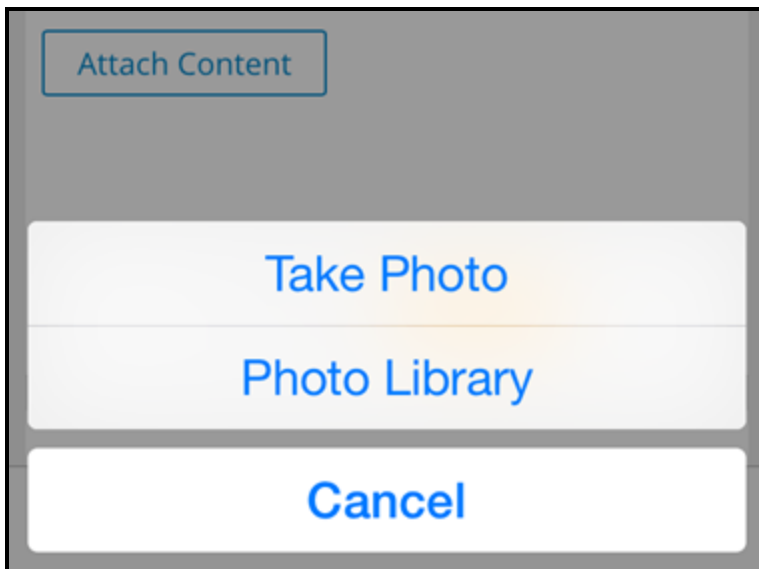
On Android devices, users can select where to retrieve an image file, using a tiled list of applications — including the device's image gallery, Google Drive, Dropbox, and any other related file storage application installed.

After initiating the attach process, users are prompted to attach a file. You cannot proceed until a file is attached unless the attachment process is canceled.

To learn more about using the Attach Content control, go to the PDN article [Using the Attach Content control](#).

KNOWLEDGE CHECK

How would you use an Attach control in layout to appear differently based on the type of device with which the user accesses your application?

You don't have to do anything. Add the control to the section and it will render differently based on the device.

Customizing the look and feel of an application

Introduction to customizing the look and feel of an application

If an application is easy to use, user adoption of it increases. Maintaining a consistent look and feel throughout the application helps users navigate it more easily. In Pega, you maintain a consistent look and feel using skins.

After this lesson, you should be able to:

- Describe the purpose of the skin
- Describe the components of a skin
- Update a skin to modify the user interface
- Create reusable style patterns using mixins
- Reference a style format in the application UI

Styling an application with skins

In any business, branding plays a vital role in presenting a uniform appearance across an application. A consistent look and feel provides familiarity for end users as they use an application. Well-designed formatting and styles help guide users through navigation and calls to action by applying consistent formatting to user interface elements.

In Pega, you style your user interface by configuring a skin. This generates the CSS for the application.

Skins

A **skin** defines the responsive behavior and formatting, such as colors, fonts, images, and layouts used in a Pega application. A skin generates the styling (Cascading Style Sheet) for the application. A skin also defines the responsive breakpoints applied to dynamic layouts. Responsive breakpoints enable your application to work on various devices, such as tablets and mobile phones.

For example, most companies have a corporate color scheme that their applications must follow. You implement the color scheme in a skin, removing the need for architects to manually specify the color scheme every time they use a UI element.

A skin applies formatting through the use of mixins and formats.

Mixins

A **mixin** defines a set of style attributes that can be reused in user interface elements. Mixins allow for defining efficient and clean style repetitions, and an easy way to update styling. For example, you can create a mixin that defines your corporate color scheme that is then reused for buttons, menus, and headers. If your corporate colors change from blue to orange, you only need to update the mixin with the new color, and any UI element that uses the mixin gets changed.
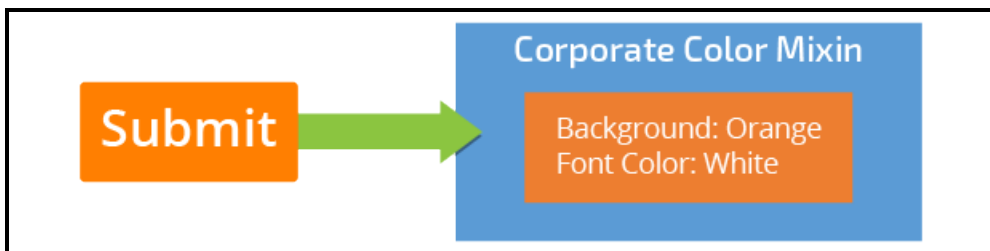
Mixins can either define a set of styles or inherit styles from another mixin. You can define four categories of mixins, listed in the following table.

Category	Description
Typography	Allows you to configure anything related to text, like font, font size, or color
Background	Allows you to configure background colors of elements
Border	Allows you to configure borders and gradient effects
Combination	Allows users to create complex mixins that incorporate multiple mixin types

Formats

A **format** defines the styling of a specific UI component. A component is an element that you can style within the skin — for example, a layout (dynamic layout), or a control (button).

You can define various style formats for each component. For example, you can define a style for all buttons. You define style formats in the skin and reference the formats on property panels in sections, harnesses, and controls. For example, you can define how inline grids or double grids are displayed to the user.

Every component can have one or more formats defined. These formats are then used throughout the application to control the appearance of the user interface. All components have a default format called either Standard or Default that is supplied out-of-the-box.

The following table lists the four categories of components where you add formats.

Category	Examples
General	Modal Dialogs, Errors
Layouts	Dynamic Layouts, Trees and Grids
Controls	Buttons, Dropdowns, Labels
Reports	List View, Paging Bar

You configure a format by setting the properties in the format or by inheriting styles from a mixin. When you update the mixin, any format based on the mixin is automatically updated. That instant update and reusability is incredibly powerful when designing user interfaces. Within a format, you also have the option to override part of the mixin. For example, define a mixin that specifies the color scheme for a menu. One of the attributes of the menu format allows you to set the color when a menu item is selected. You could override the mixin color when an item is selected to make it stand out from the other menu items.
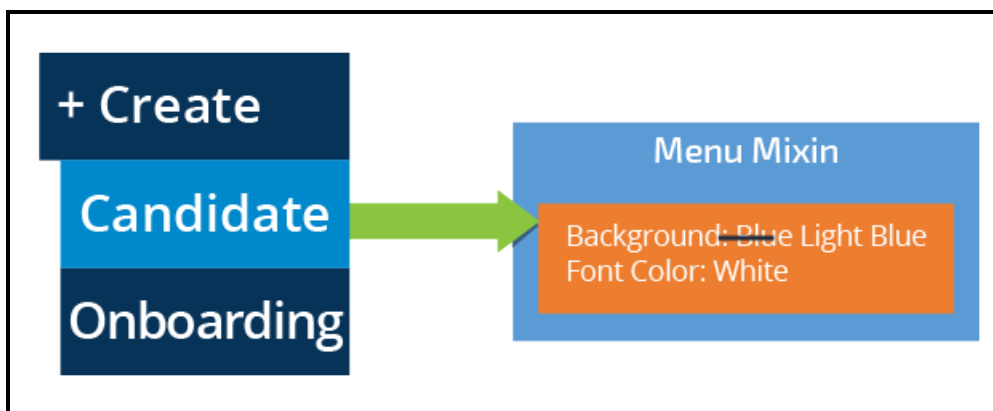
KNOWLEDGE CHECK

What element should you create to maintain a consistent group of styles?

Mixin

How to customize application appearance with skins

Skins are applied at an application level, but could also be applied to a portal. The best practice is to reuse the application skin in any portal. However, at times you may create a custom skin for a portal. For example, you could create a separate skin for the mobile application version. You only need to do this if you want to present the application differently where using the responsive layouts does not give you what you need.

The following example describes the relationship between mixins, formats, skins, and what you see when you view the application. The example uses the Attach New link — this adds an attachment to a case. You configure a link using a Link Control. One of the configuration options of a Link Control is to specify a format for the link. In this example, the Link Control specifies the Simple format.
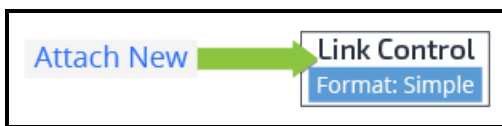
Every application defines a skin used for the UI. The formats are defined in the skin. In our example, the Link Control uses the Simple format — that is, the styles defined in that format determine how the link is rendered. For this example, the text style for Normal Text is set to use a mixin named Link.
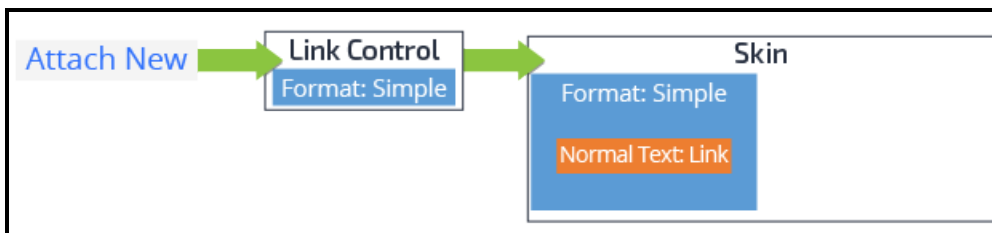
Finally, the styles of the mixin are loaded. In our example, the Link mixin defines the font color and font size for the text in the link.

Notes: There are other properties for both the Simple format and the Link mixin. The example discusses a subset of the overall properties.
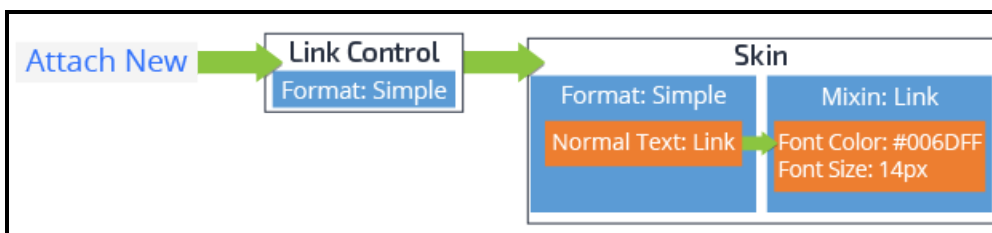
When creating a skin to customize the user interface, you need to answer the following questions:

- Will you inherit styles from another skin?
- Can you inherit styles from existing mixins?

Skin Inheritance

The first decision when creating a skin is to determine if a skin that you can inherit a set of base styles from already exists. This is known as skin inheritance.

Notes: Applications should use inheritance wherever possible. The application that you work on should either inherit from one of the base Pega skins or a corporate skin your company has already created.

Skin inheritance allows a skin to inherit formats and mixins from a parent skin. The parent skin defines formats and mixins that the dependent skin can use as is or update them in a new mixin or format. When a format on the parent skin is modified, the dependent skin automatically inherits those changes unless the format is overridden in the dependent skin.

The advantage of skin inheritance is that you can create an enterprise-wide layering of styles. In the following example, U+ bank (a fictional bank) uses a corporate base skin that defines a color scheme of orange and square-shaped buttons. The financial services application for U+ is branded using green. The financial services application would inherit from the corporate base skin and then update the buttons to use green. Then, in a loan application may be a requirement to use rounded corners on a button. This time, you inherit from the financial services skin and update the borders for buttons to use rounded corners.
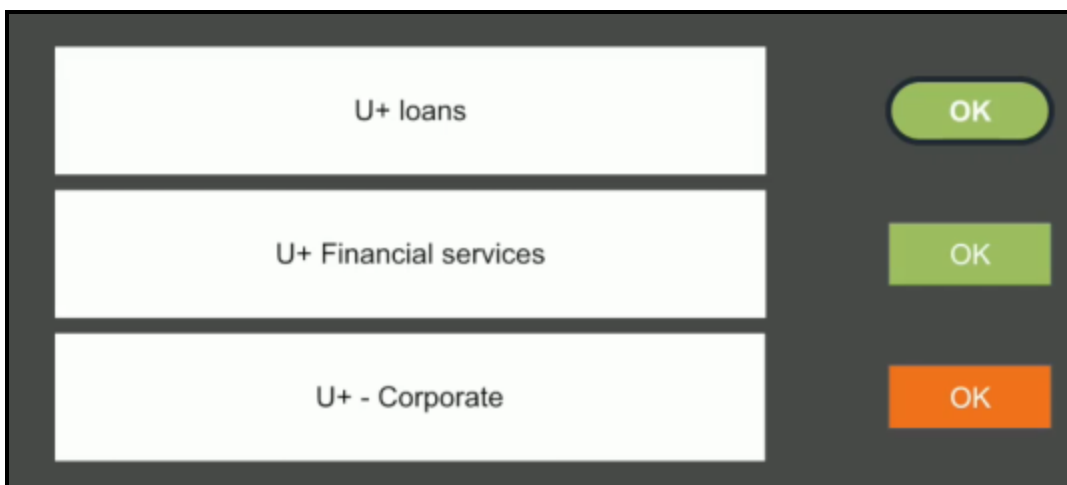
Another benefit of layering your skins through inheritance is that if a change is made, all skins that inherit from that skin get the changes. In the previous example, if the financial services application color scheme changes to purple, you can update the button color in the financial services skin. As a result, both the financial services and loan applications have purple buttons.

Mixin Inheritance

Mixins should be the first point of customization when customizing the look and feel of an application because they ensure that you maximize the reuse of styling. Formats then apply the style templates created in a mixin to render a UI component.

Like skins, mixins can also inherit from other mixins. A new mixin should be created when a new meaning for the style is needed. For example, you may create a generic notification style in a mixin, and then create a new mixin that inherits from the notification mixin for errors.

As with skin inheritance, using inheritance with mixins provides a layering effect where, if you change part of a style in a base skin, the change modifies any mixins that inherit from it.

Controlling application appearance with a skin

Skins are used to create a consistent look and feel in your application. Within a skin, you configure the formats and mixins used by the UI components in the application.

> **Notes:** This procedure demonstrates how to add a new Typography mixin. However, the procedure is the same for whatever type of mixin yow want to create. Each mixin type has its own configuration properties.

Accessing the application skin

1. In Designer Studio, on the Application menu, click **Open Application Skin**.
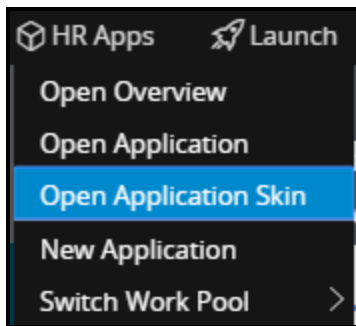
2. Click **Inheritance** to update the inheritance of your application's skin (if required).

Creating a mixin

1. In your application skin, click **Mixins**.

2. On the **My Mixins** tab, click **Create new mixin** for the type of mixin you need to create. For example, under **Typography**, click **Create new mixin** to create a typography mixin.
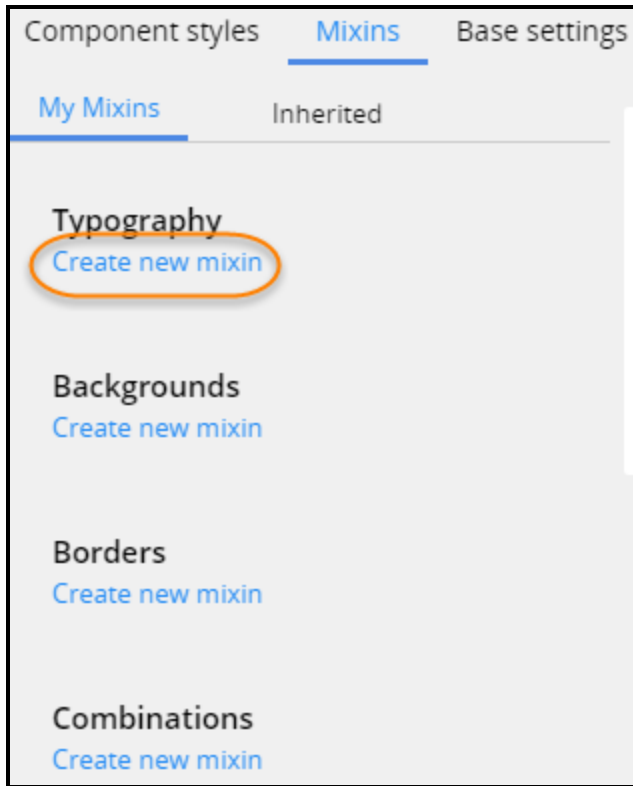
Notes: This procedure walks through the process of creating a Topography mixin.

3. In the **Mixin name** field, enter a name for your mixin, such as MyTypographyMixin.

4. In the **Mixin usage annotation** field, enter a description of the purpose of the mixin.

5. Click **Submit**.

6. Customize the mixin by specifying the style attributes. For example, define the mixin to render text in red using the Courier New font at 10px.

7. Click **Save**.

Using a mixin in a format

1. In the application's skin, click **Component styles**.

2. Click the slide out menu and select the format to modify. For example, from the slide out menu, select **Links** to update the format for links.
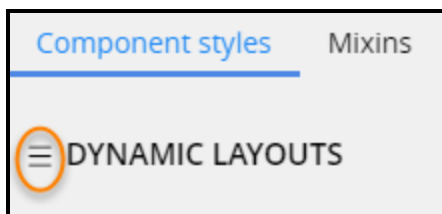
> **Notes:** This procedure updates the Link format.

3. Select the format to configure, such as the **Standard** format.

4. Click **override it** to update the Standard link format to your skin from the inherited skin.

> **Notes:** Alternatively, you could create a new Standard format in your format. However, as a best practice, inherit from the parent skin wherever you can.
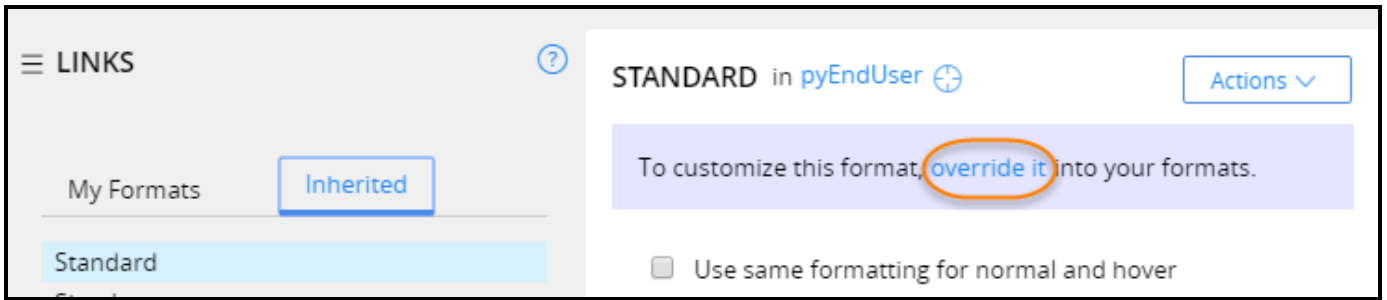
5. Expand the style property to update it. Select the mixin to specify the style. For example, choose the **myTopographyMixin** to update the **Normal Text**.
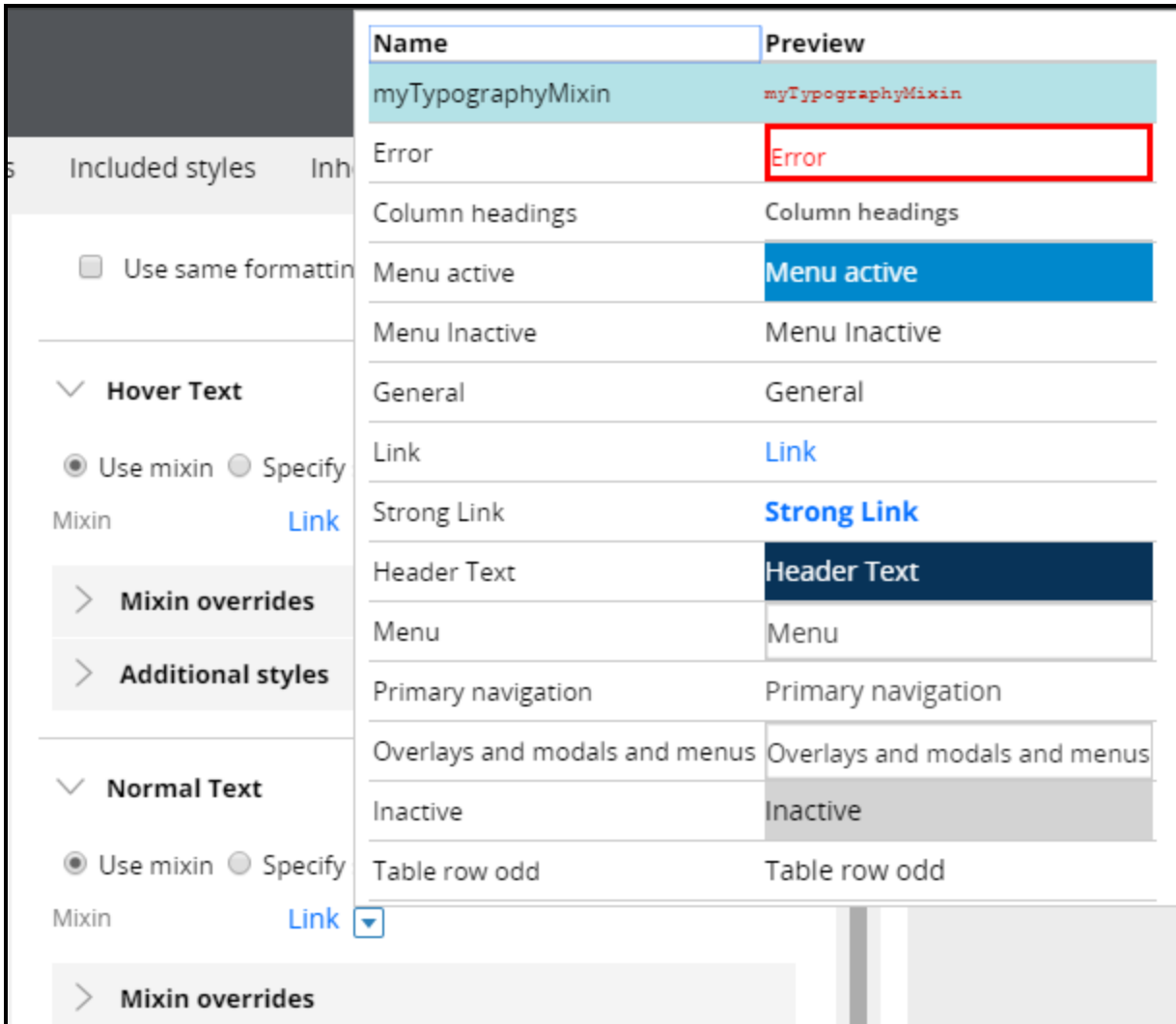
6. Click **Save**.

7. Click **Check in**.

Applying a format to a control

1. In any section, open a Link control from your application.

2. Click **Presentation**.

3. Update the **Control format** with the format used for this link.
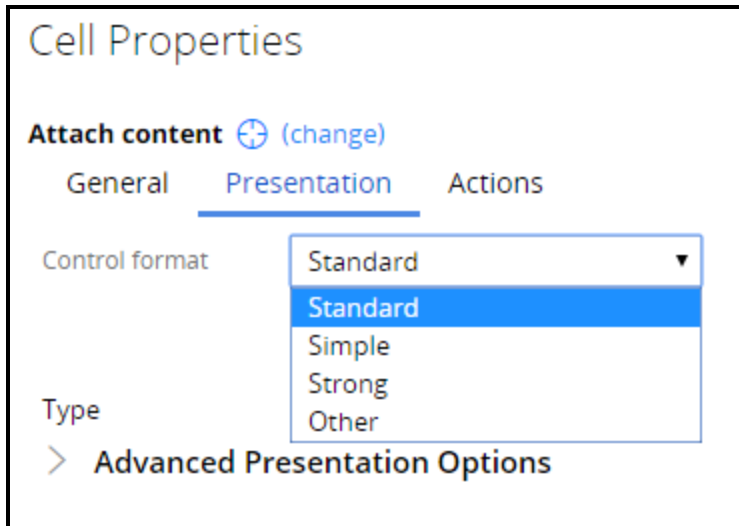
4. Click **Save**.

REPORT DESIGN

Creating reports that combine data from multiple tables

Introduction to creating reports that combine data from multiple classes

In this lesson, you learn three techniques to create a report that references data from multiple classes: combining data in a report by creating subreports, configuring class joins, and referencing reusable association rules.

After this lesson, you should be able to:

- Describe how Pega data can be stored in multiple database tables
- Describe the relationship of class mappings to database tables
- Explain how associations and joins combine data in different tables for reporting
- Explain how subreports combine data in different tables for reporting
- Use associations and joins to combine data from different database tables in a report
- Configure a report to incorporate data from a subreport
- Create a report that uses a class join

Data storage in Pega

Pega saves data across multiple database tables when a case is processed. The system uses Pega classes to organize and store the data in the appropriate table. When you create reports, the Pega reporting tool uses the Pega class organization to find and retrieve information from these tables. Efficient data organization and access enables the organization to generate reports that support key strategic decisions.

For example, managers may want to know which customer service representatives (CSRs) resolved the most cases during the past three quarters. The report requires you combine case information and historical processing information. This information may be stored in separate tables.

The following video highlights how Pega's flexible data model helps generate information immediately to suit reporting requirements.

KNOWLEDGE CHECK

⌄ ANSWER	How does the Pega reporting tool find and retrieve data from the appropriate table?
The tool uses Pega classes to find the table.	

Class mappings and database tables

Any Pega class that has instances, such as case types, can be mapped to a database table. For example, when users create cases, the system assigns the case an ID and saves the value as an individual row in a database table. When you generate reports, you are retrieving data from rows in database tables. Reports use **class mappings** to locate the data from one or more database tables.

When designing reports, you need to know which table has the data and how the data is mapped. For example, you may need to create a report that contains information about Candidate cases. These records are instances in the case work class. In the same report, you may also want to include workbasket information about each candidate case. Workbasket records are instances in a workbasket class. The information for each type of information is stored in separate tables. When you combine the information in a report, you use class names to identify in which tables the information is stored.
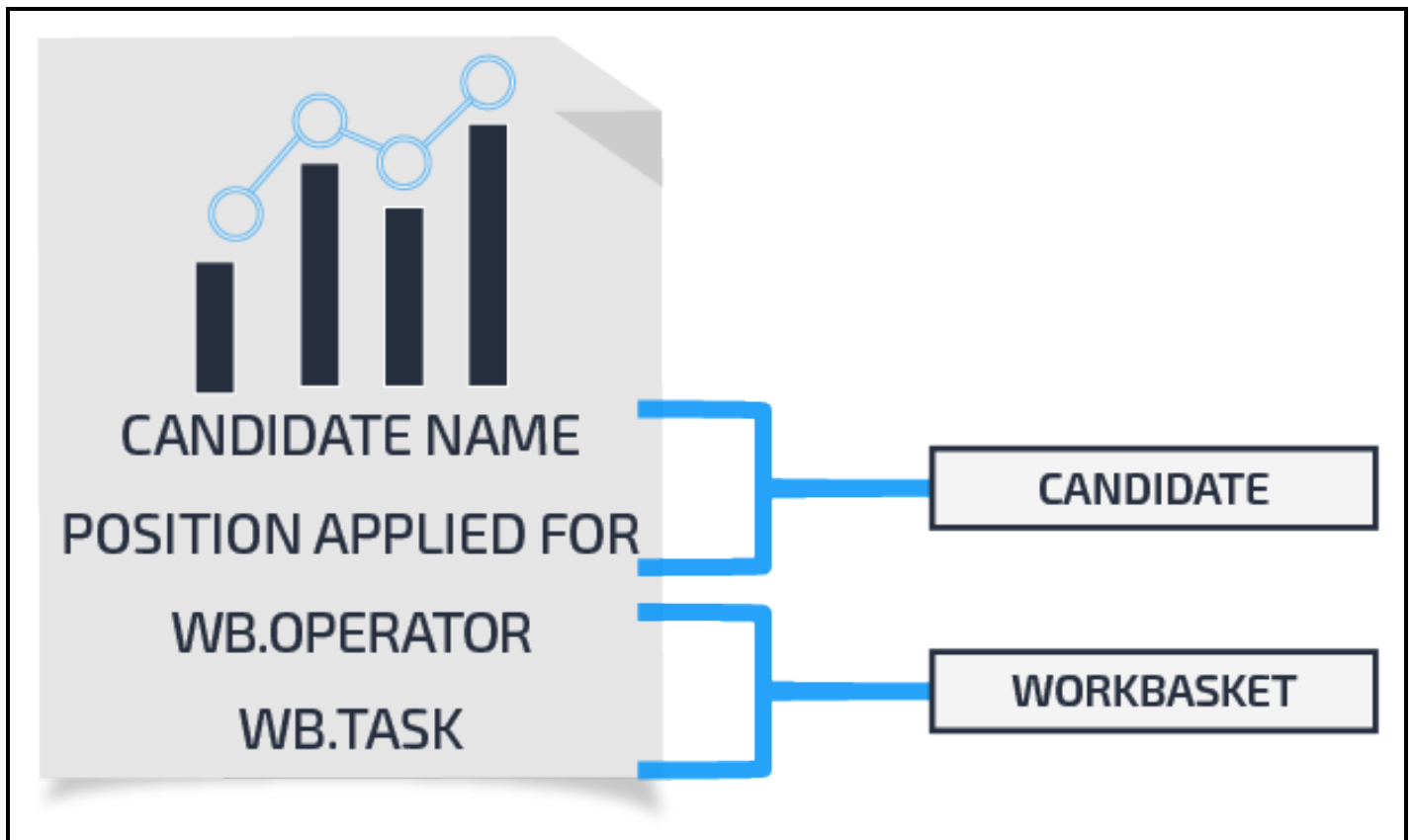
Records used for mapping classes to tables

Pega uses two records to identify the database table the class is mapped to: **Database** and **Database Table**.

- A Database record identifies how Pega connects to a specific database for the named database. The record contains connection information so Pega can access the database. The record establishes an alias that can be referenced elsewhere, such as a database table record. Database records can be configured to use either JNDI or JDBC url for the database connection. By default, Pega uses the following standard databases in a database record:

- **PegaRULES** maps to a database where all the rules, data instances, work objects, history, and other concrete objects from internal classes of your Pega system.

- **PegaDATA** maps to a database where all custom data instances are saved.

- A Database Table record identifies a specific table in a specific database, and specifies the corresponding Pega class. Pega uses this record to identify which table to write case data when a user creates or updates a case.

KNOWLEDGE CHECK

ANSWER How does Pega map classes to database tables?

Pega uses two records — a Database record and a Database Table record — to map classes to database tables.

Mapping multiple classes to a single table

In some situations, you want to have multiple classes store data in the same table. For example, you might have an application with three case types such as Candidate, Onboarding, and Benefits Enrollment, and you want to report on the work statuses of all cases in the application. Rather than create a database table for each case type, you can designate a class as a **class group** (also referred to as a **work pool**). Class groups cause the system to store instances of similar or related case types together in a single database table. If you create a report in a specific case type such as Candidate, your report returns only records in the case type. If you create a report for the class group, the report returns all instances in the classes that belong to the class group.

In Designer Studio, the mappings are displayed in the Database Class Mappings landing page. The following example shows the work classes that map to the class group database table pc_TGB_HRApps_Work.

Class	Table	Database
TGB-HRApps-Work	pc_TGB_HRApps_Work	PegaDATA
TGB-HRApps-Work-BenefitsEnrollment	pc_TGB_HRApps_Work	PegaDATA
TGB-HRApps-Work-Candidate	pc_TGB_HRApps_Work	PegaDATA

Three commonly used report types

You commonly generate three types of reports: work, assignment, and history. Each type of report uses properties in classes mapped to a variety of data tables.

Work reports

When a case is created, Pega uses standard properties in the Work-base class to define each case. This Work- base class includes properties that describe the following:

- A case identifier (*pyID*), the work parties participating in a case (*pyWorkParty*)

- The customer identifier such as an account number (*pyCustomer*)

- The work status of the case (*pyStatusWork*)

For descriptions of the important standard Work- properties used in case-based reports, see the help topic Standard properties in the Work- base class .

Standard properties that support subcase processing can be found in the Work-Cover- class. For descriptions of those properties, see the help topic Standard properties in the Work-Cover- class.

Standard properties in the @baseclass class are available for every case when it is created. The most important property is *pzInsKey*. Pega uses this property to internally identify each case. A subcase also has a property named *pxCoverInsKey* that identifies the parent case.

For descriptions of the standard properties see the help topic Standard properties in the @baseclass class.

Assignment reports

Cases requiring user interaction are assigned to a user during processing. Each time a case is assigned, Pega creates an assignment object. When the case is completed and advances to the next assignment, Pega creates another object. If the assignment is routed to an operator, Pega saves the object to the database table named *pc_assign_worklist*. If the assignment is routed to a workbasket, Pega saves the object in a database table named *pc_assign_workbasket*.

Some commonly used properties that are specific to assignments include the operator who has the assignment (*pxAssignedOperatorID*) or the name of the flow rule for the assignment *(pxFlowName)*.

When creating assignment reports, you often use *pxRefObjectKey* — this is mapped to *pzInsKey*. The *pxRefObjectKey* property allows you to relate the assignment to the case.

For descriptions of many standard properties used in assignment reports, see the help topic Standard properties in the Assign- base class.

History reports

When a case is being processed, the system automatically captures audit trail data in the history classes. The classes are mapped to the History database tables where the data is saved. For example, the history class *History-TGB-HRApps-Work* is mapped to *pc_History-TGB-HRApps-Work*.

History reports use properties in the History- and History-Work- classes. These properties include *pyHistory type* (identifies the event that caused the history event), or *pyPerformer* (identifies the operator who completed the event recorded in the history instance).

Properties in history classes can be used to design performance-based reports. For example, you can use *pxTaskElapsedTime* to report the total time spent on an assignment. If an assignment is routed to multiple users, you can use *pyPerformTaskTime* to report on the total time spent by all users. If *pyPerformTaskTime* is significantly lower than *pxTaskElapsedTime*, then the assignment has been idle for a long time.

For descriptions of the properties in the History- and History-Work classes, see the help topics Standard properties in the History- class and Standard properties in the History-Work- class.

For more information about using the statistics for performance reports, see the help topic Assignment statistics.

KNOWLEDGE CHECK

You create a report that shows how quickly cases were processed in a specific assignment. Which classes contain the standard properties you use in your report?

History-Work- and History- classes

How to combine classes using joins and associations

You can relate properties in multiple database tables or classes to combine data in a single report. The following examples show you how to use properties to make relationships between classes or tables.

- Use case and subcase relationships to show subcase data along with the parent case data. For example, assume you want to create a report that lists the purchase orders in a purchase request. You match the case identifier in the parent purchase order class to the case identifier in the child purchase request class.

- Use case and assignment relationships to show how the system processes assignments for a specific case or a subcase. For example, assume you want to show the operators working on specific cases. You match the operator identifier in the case database table with the operator ID column in the assignment table.

- Use case and history relationships to monitor performance. For example, assume you want to show the total amount of time required to resolve specific cases. You match the case identifier in the case data table with the case identifier in the history table.

You create class or database table relationships in a report definition. You do not specify database tables to define joins. You can either configure class joins or you can reference association rules.

Class joins

When you build a class relationship in a report definition, you configure a **class join**. For example, assume you want a report that identifies the current assignment and operator working on each candidate case. The candidate information is in the Candidate case type. The assignment information is in the assignment workbasket class. You would create a report definition in the Candidate class and configure a class join to the assignment-workbasket class. In the report definition, you would first specify the Assignment-Workbasket class as the class you want to join to your report. You would then specify a filter that matches key values in records for both classes. For example, you would match the candidate case key value *pzInsKey* to the workbasket assignment key value *pzRefObjectKey*. In this way, the report can correctly match, for each case, the records in both classes.

You follow these basic steps to create a class join:

- Determine the class to which you are joining.

- Create a prefix that in combination with the class name serves as an alias for the joined class.

- Decide whether you want to include or exclude instances that do not match.

- Create a filter that describes how you relate the classes.

> **Note:** In the report definition form, you specify the class as the primary join. If this work type is derived from Work-, determine whether the join is to an implementation class or to a framework class. This ensures that you are joining to the correct data set.

Class join settings

On the Data Access tab on a report definition form, in the **Class name** field you specify the class you are joining. For each class, in the **Prefix** field you enter a text prefix. The prefix combined with the class name serves as an alias for the joined class and its properties. For example, to join to a work class that describes benefits enrollment cases, you might use the prefix BE as shown in the following image.

When you add properties to columns in your report, the prefix helps you identify the properties in the joined class.

In the **Type** field, you specify how you want the system to join the data by selecting one of the following options:

Type	Description
Only include matching rows	To only include instances in each class that have a matching instance in the other class (referred to in database terms as an inner join)

©2017 Pegasystems Inc.

Type	Description
Select Include all rows in <class>	To include all qualifying instances of the Applies To class of the rule, even if there is no match found in the joined (prefix) class (referred to in database terms as an outer join)
Select Include all rows in <prefix>	To include all qualifying instances of the joined (prefix) class, even if there is no match found in the Applies To class (referred to in database terms as an outer join)

Filter conditions

You create a filter condition that defines the relationship between the classes. The filter uses one or more properties to establish the relationship. Consider the Benefits Enrollment class join to the Candidate class. You would create a filter that matches the .pxCoverInsKey property in the Benefits Enrollment class to the .pzInsKey property in the Candidate class.

Note: You cannot join to a class in a different database than the Applies To class of the report. The Column property must be an exposed column. You can use the Optimization tool to expose columns. For more information about the tool, see the Help topic Property optimization using the Property Optimization tool.

For descriptions of the fields you use when defining a class join, see the help topic Report definition access tab.

Association rules

You use association rules to join multiple classes. Unlike a class join (unique to each report), associations can be reused in any report. Managers can also use associations when they create reports in the Case Manager portal.

For example, assume you want to combine records in the Assign-WorkBasket class with records in work classes. You use the standard WorkBasket Assignment (*pxWorkbasketAssignments*) association rule to join the classes.

Note: Pega provides a set of standard association rules. You can use these rules for many class joins. For example, standard association rules allow you to join work to assignment classes or to history classes. For a list of standard association rules, see the Help topic Standard Association rules.

When you add a column, you specify the association rule class name as a prefix, then select the properties in the class.
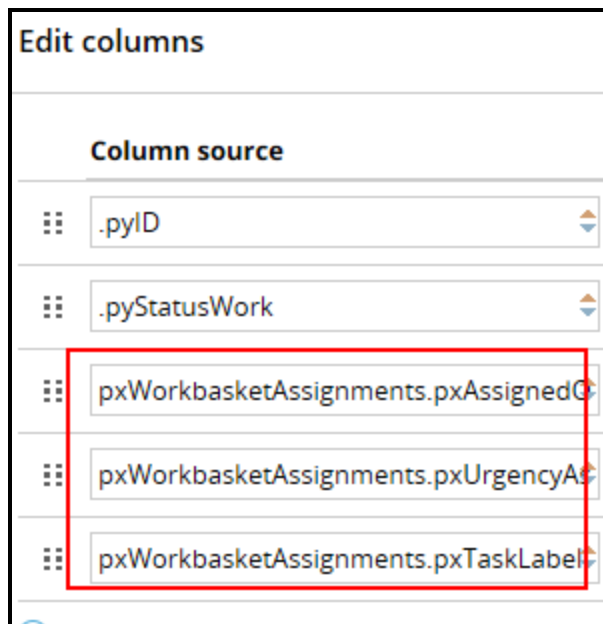
When you add the association rule prefix, it appears on the **Data Access** tab in the Associations section.

For information about creating your own association rules, see the PDN article [When and how to create an association rule to support reporting.](#)

KNOWLEDGE CHECK

When configuring a class join, how do you specify a property that defines the relationship between the primary and joined classes?

Create a filter condition

Creating class joins and associations in reports

You can report on records in multiple classes mapped to one or more data tables. To create these reports, in a report definition you can reference an association rule or configure a class join.

Referencing an association rule

After you create a report definition in the class you want to report on, reference an association rule to join another class.

1. Open your report definition.

2. In the **Edit columns** section, in the **Column source** and **Column name** fields, enter the name of the association rule you want to reference. The system uses the prefix for the properties you want to include.

The following example shows the pxWorkbasketAssignments association rule prefix and properties.

3. Click **Save** to save your reference to the association rule. When you add a column that uses an association, the system automatically references the association. The **Associations** field on the Data Access tab displays the association.

Configuring a class join

After you have created a report definition in the class you want to report on, create a class join to establish a relationship to instances of another class.

1. Open your report definition.

2. On the report definition form, open the **Data Access** tab.

3. In the **Class joins** section, click **Add class join** to add a row.

4. In the **Prefix** field, enter a prefix for the class to join to the report definition. Use this prefix to reference properties in the class.

5. In the **Class name** field, enter the class to join to the report class. The form looks like the following image.

6. At the end of the row, click **Edit conditions**. The system displays the **Enter filter conditions** dialog.

7. In the **Column** field, select a property within the joined class.

8. In the **Value** field, enter a property in the report definition's Applies To class.

Enter filter conditions

Filter conditions

A

Condition	Column	Relationship	Value
A	BE.pxCoverInsKey	is equal	.pzInsKey

9. Click **Submit** to save your filter condition and close the **Enter filter conditions** dialog.

10. Open the **Query** tab.

11. In the **Edit columns** section, enter **Column source** and **Column name** values.

   Note: Use the class prefix to find properties in the joined class you want to include in the report.

   When you are done, the report definition looks like the following image.

Edit columns

Column source	Column name	Summa
.pyID	Case ID	<blank
.Employee.pyLastName	Last Name	<blank
BE.pyStatusWork	Benefits Enrollment Work St	<blank
BE.pyID	Case ID	<blank
BE.MedicalPlan.Name	Plan name	<blank

⊕ Add column

12. Click **Save** to save the join configuration in your report definition.

How to combine data from different classes using a subreport

Subreports enable you to reference results from any report definition in a main report. A report definition used as a subreport can be run like any other report.

 Note: Consider subreports as a way of combining data using IN, HAVING, and WITH clauses.

Subreports can be defined in classes different from the main report. You can access data in different classes similar to the way you would use a class join or an association.

You commonly use subreports to satisfy complex reporting requirements. For instance, you can use subreports to filter results. This approach allows you to include or exclude data. You can also use subreports to display aggregate calculations on specific rows in a main report.

You use two different methods to create a subreport: join filters or aggregation.

Using join filters to create a subreport

Assume you want to display the task information for each purchase request recently updated by each operator.

Do the following things to create the report:

- Create a subreport for purchase requests that retrieves the most recent update date by update operator.

- On the main report in the **Query** tab, add columns for the requested data for each case and specify the subreport you want to reference.

- On the main report's **Data Access** tab, add the subreport.

- Create a join filter condition for the subreport. The filter defines how you want to join the subreport data to the main report. As shown in the following example, the subreport Update Operator column is matched to the update operator value (.*pxUpdateOperator*) in the main report.

- On the **Design** tab, add a filter condition so that the update date value is equal to the update date value in the subreport.

When you run the report, it shows, for each operator, information about the purchase request the operator most recently updated. For each case, the report displays the update date and time retrieved from the subreport.

The procedure for creating the previous example is described PDN article When and How to Use subreports in a Report Definition. See Use Case 2.

Note: The examples in the PDN article were created in Pega 6.2. The styles on the rule forms have been upgraded in Pega 7. However, the fields and functions are still applicable.

Using aggregation to create a subreport

You can use subreports to display aggregate calculations on specific rows. For example, assume you want to list the managers in the Engineering division who have more than ten direct reports.

You would do the following things to create the report:

- Create a subreport that shows the managers who have direct reports, and how many they have. You use filter conditions to limit the data to managers who are part of the Engineering division and report directly to someone.
- On main report's **Data Access** tab, add the subreport.
- Create a join filter condition to join data from the subreport to the main report where the value in the subreport's operator column matches the value in the operator column in the main report.
- Create a filter condition to use only data from the subreport where the number of reports-to instances is greater than ten.

When you run the report, it displays managers in the Engineering divisions who have more than ten direct reports.

OVER TEN REPORTS

Filters: Count of > 10

MANAGER	COUNT OF DIRECT REPORTS
astjs	30
grovs	12
karaf	21

The procedure for creating the previous example is described in the PDN article When and How to Use sub-reports in a Report Definition. See Use Case 3.

For descriptions of the fields you use to configure a subreport, see the Subreports section in the Help topic Report Definition Data Access tab.

Subreports can be configured to support many types of report requirements. For example, you may wan to list the average number of direct reports for the managers in a specific division or list the operators who have not updated work items of a specific type within the past week. To see more use cases and design procedures, refer to the PDN article When and How to Use sub-reports in a Report Definition.

KNOWLEDGE CHECK

How can you use a subreport to list the operators who have not updated cases of a specific case type within the past week?

Use the subreport as a filter, comparing with a list of all operators and removing all entries where there is a match between the two reports.

DATA MANAGEMENT

Exposing an application with a service

Introduction to exposing an application with a service

In this lesson, you learn how to use web services to expose application functionality. You learn how to identify an application function and use the Service Wizard to wrap the function in a SOAP framework.

After this lesson, you should be able to:

- Describe the components of a service
- Explain how to create a service with the Service Wizard
- Explain how to add error handling to a service
- Create a SOAP service using the Service wizard
- Test a SOAP service

How to expose an application as a service

The two most common ways to expose your application as a service is either to create a web service or to leverage the Pega API.

Conceptually, these two options work the same way: a request is made to a URL, and a response is returned. The difference is how you communicate with the service.

Leverage the Pega API

The Pega API provides a standard set of services that includes new case creation, assignment processing, and access to data pages. These built-in REST/JSON services enable the rapid implementation of Pega-powered mobile and client applications.
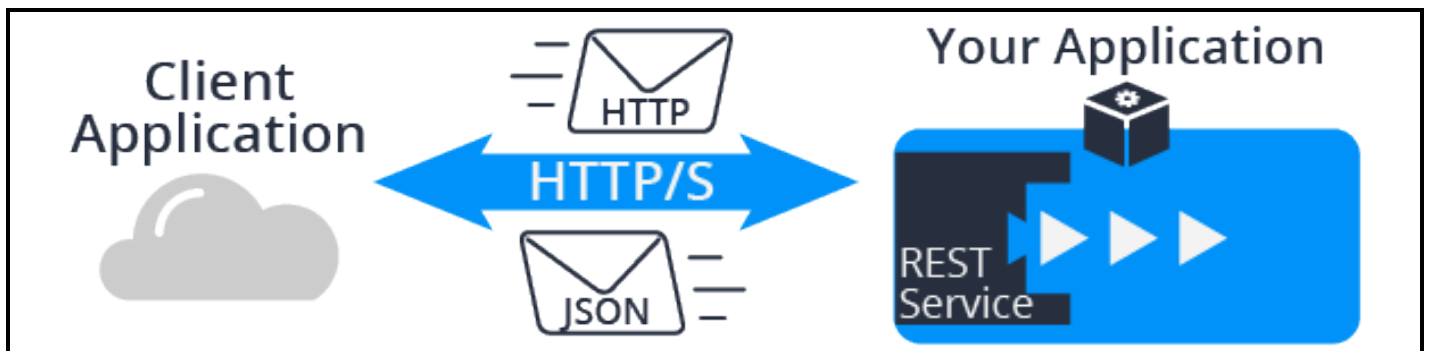
You can call any of the Pega API services by using standard HTTP methods (for example, GET, POST, or PUT). Refer to the Pega API resources page in Designer Studio (click **Resources > Pega API**) to see details of the request and response data requirements. This documentation is also available in JSON format in the Docs API (GET/docs).

Resource	Description
Assignment API	Provides the ability to obtain a list of assignments for a user, obtain the details of any specific assignment, and perform an assignment action
Authenticate API	Allows you to verify user credentials
Cases API	Provides the ability to obtain a list of cases for a user, create a new case, obtain case details, and update a specific case
Casetypes API	Provides the ability to obtain a list of case types for the authenticated user
Data API	Facilitates the process of obtaining the contents of a data page and obtaining the metadata for a specific data page
Docs API	Provides access to the complete documentation for the Pega API

You can learn more about the Pega API on the PDN at Overview of Pega API.

You can practice using the Pega API on the PDN at Getting Started with the Pega API.

Create a SOAP service

When you have a requirement to implement a SOAP web service to expose your application to other applications, you do this by creating a service. SOAP web services communicate using the SOAP protocol and pass XML messages from one application to another. Your application needs to convert that XML message to Pega objects to process them and then convert those Pega objects back to XML after the processing is complete.
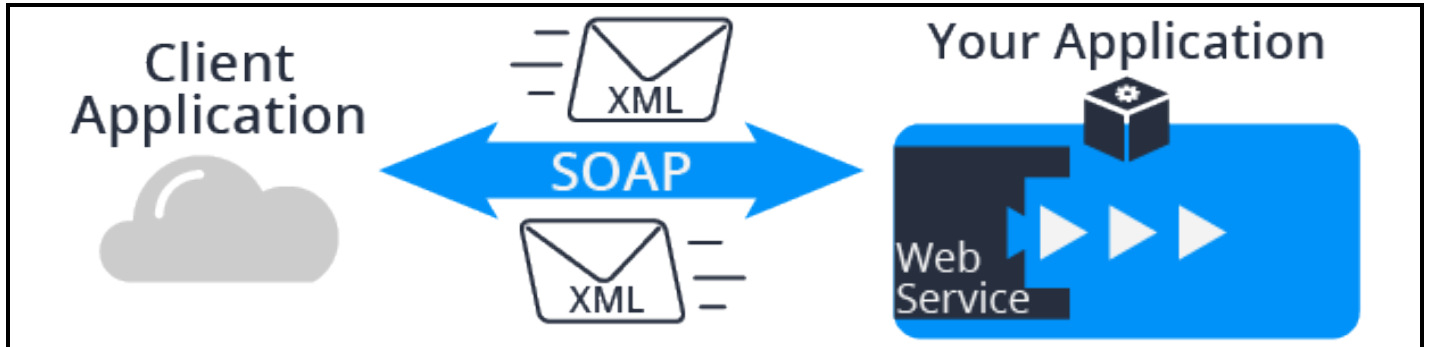
A SOAP service uses a combination of rules to process a request. The rules you use are:

Rule	Description
Service activity	An activity that performs the steps of what you want done in the service
XML Parser	Map data from an XML message into clipboard property values
XML Stream	Assembles and sends an XML document in an email message, a SOAP message, a file, or other types of messages
Service Package	Groups one or more service rules that are designed to be developed, tested, and deployed together

These rules work together to process a request and send a response back to another application. The following diagram shows how Pega processes a service request.
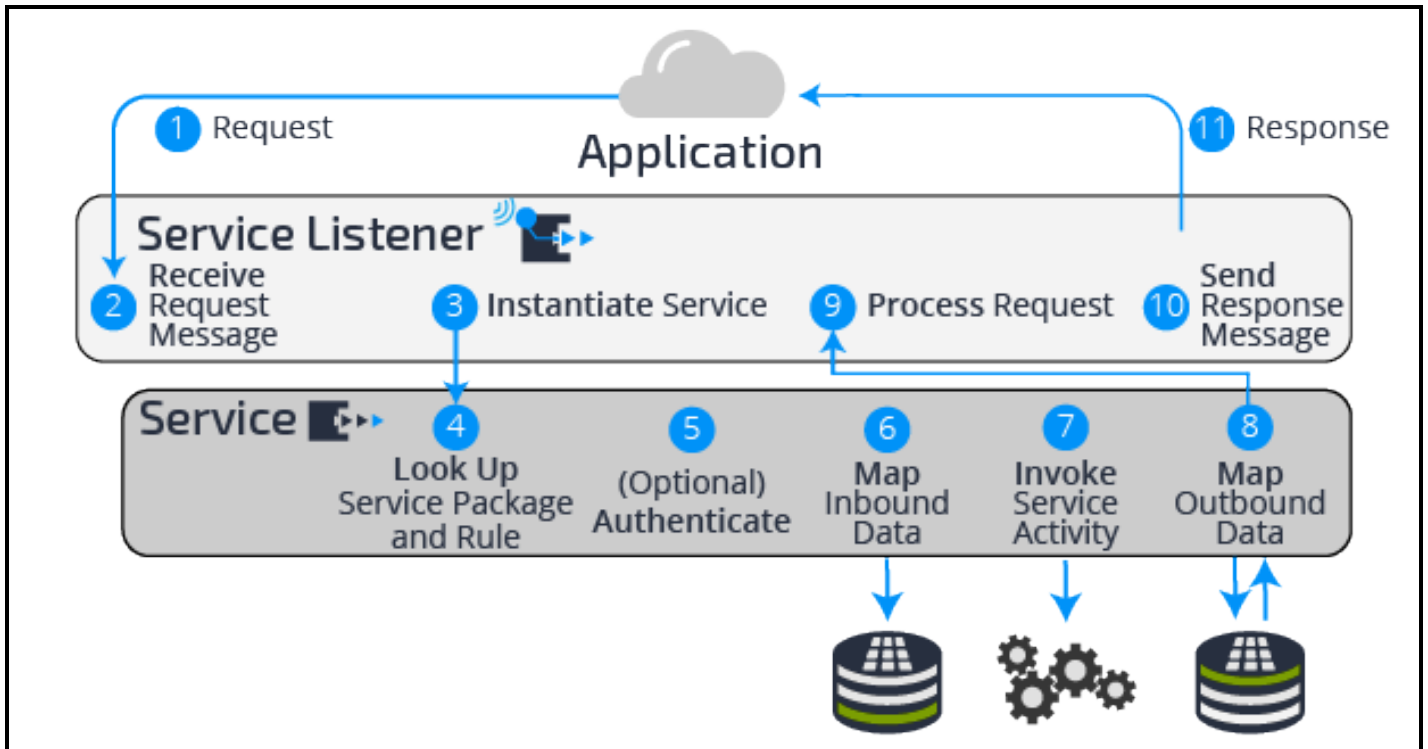
1. A client application sends a request to your application.

2. The service listener listens for incoming requests. This functionality is provided by either the Web Server, Application Server, or Pega Listener.

3. The service listener receives the request and instantiates the Service API to provide communication with Pega. Then, via the Service API, control is handed to Pega.

4. Pega looks up the service package and related service rule, using the access group that is specified in the service package.

5. Pega then establishes the service requestor, and optionally performs authentication based on security credentials that are passed in the request. Once authenticated, service processing continues using the authenticated user's access group, not the access group that is contained in the service package.

6. The request is mapped, using the instance of an XML Parser rule, onto the clipboard according to the specifications contained in the service rule.

7. Control is passed to the service activity, which provides the logic for the service.

8. Using the XML Parser rule, the service rule maps the clipboard data to form the response data.

9. The service listener receives the response from the Service API.

10. The service listener sends the response back to the application that made the request.

11. The client application receives the request.

KNOWLEDGE CHECK

⌄ ANSWER   What is the purpose of a service?
Services are used to allow other applications to access to your application.

Creating a SOAP service using the Service Wizard

You run the Service Wizard to create all the records needed to expose your application as a service. The Service Wizard walks you through entering the settings required to create the service. The Service Wizard does not create XML mappings for inherited properties, so you must add those properties as needed.

Run the Service Wizard

1. Select **Designer Studio > Integration > Services > Service Wizard** to start the Service Wizard.

2. Complete the Select Service Purpose section:

   a. Set the **Service Purpose** to Create and manage work.

   b. Set the **Service Type** to SOAP.

   c. Click **Next**.

Note: Create and manage work is the most common reason to create a service. You invoke an existing service activity if you have created the activity previously. The Process input or output data provides an empty activity that you configure to get or process the desired data. The options in the wizard differ based on the Service Purpose selected.

3. Complete the Provide Service Details — Select Work Properties section:

   a. Set **Work Type** to **TGB-HRApps-Work-Onboarding**.

   b. Set **Flow Type** to **pyStartCase**.

   c. Select **Create Work** and set the **Organization** to **TGB**.

   d. Click **Next**.

4. Click **Next**.

   Note: The *pyStartCase* flow does not have any flow actions that you want to consider a service to execute.

5. Select **Use XML for data mapping** to allow for list structures in the request.

   Note: Only properties defined directly in the TGB-HRApps-Work-Candidate case type are displayed. Inherited properties are not displayed and need to be added manually after the wizard has been completed.

6. Select the following input properties:

   - **.Employee.Manager**
   - **.Employee.StartDate**

   Click **Next** to confirm the proprieties to use.

7. Set the **Ruleset Name** to **HRApps**, and set the **Ruleset Version** to the version in which you are working.

8. From the **Service Package Options** drop-down list, select **Configure a new service package**, then click **Next**.

9. Keep the default settings in the Configuration Data Records screen. Click **Next**.

   Note: Requires authentication is selected by default. This means that you need to provide an operator and password when you call your service.

10. The Review and Save screen displays the records that will be created by the wizard. Click **Finish** to create the records.

The final page displays the records created by the wizard.

Add inherited properties

1. Open the *CreateNewWorkResponse* XML parse rule generated by the wizard.

2. Click **Add Element** to add the required input parameter that was not available for selection in the wizard.

3. Double-click the new property to configure it.

4. Enter a Node Name and specify the Property to map the value.

5. Click **OK** to update your changes.

Test the service

1. Open the service rule.

   Tip: You can access the service rule from the final screen of the Service Wizard. If you have closed this screen, you can open the Records Explorer and navigate to **Integration-Services > Service SOAP** to access it.

2. Click **Actions > Run** to test the service rule.

3. Select **Supply SOAP request envelope to enter values** for the test.

4. Update the request and enter values for each option. In the following screenshot, you see values have been entered for Employee, SSN, and FirstName.

5. Click **Execute** to test the service.

6. Your response will vary based on what you are attempting to do. If there is an error, you see a SOAPFault exception produced.

How to configure exception processing

Integrating with external systems can introduce additional problems such as network errors involving firewalls, incorrect authentication or credentials, or system failures. An application that handles exceptions properly is critical to the success of any application.

Many errors can be anticipated at design time and addressed with specific actions to help ensure a positive customer interaction. But unexpected errors may still occur and are just as important to address promptly.
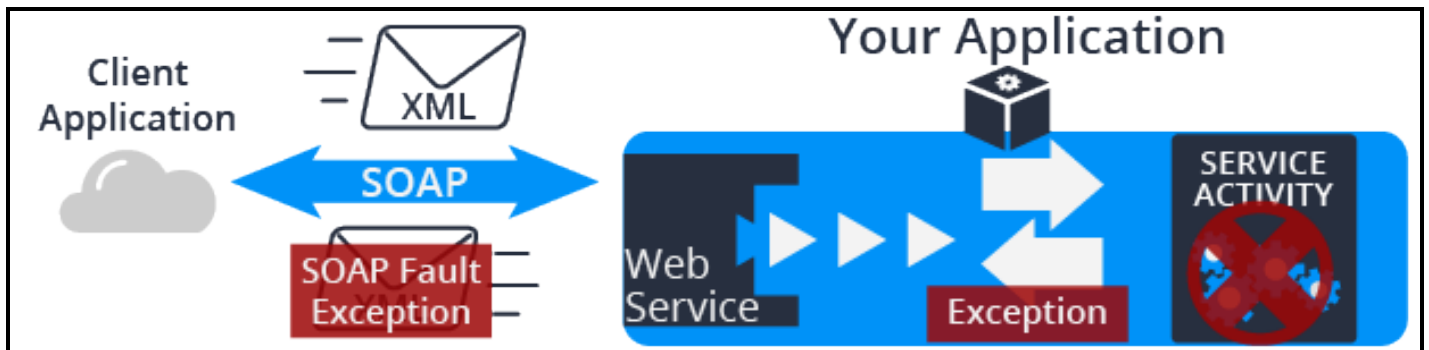
Exception processing is critical to catch errors and exceptions during execution of a service rule so that they can be communicated to the client application in a clean way. Best practice requires you to configure clipboard property values to communicate service errors to the calling application.

Many service types, including SOAP, have an exceptions or faults tab where you can define what the application does when a service error or exception is encountered. When the service encounters a processing error and a condition evaluates to true, the application returns a defined error message to the calling application. The following conditions are available for defining an error response message.

Condition	Description
When	The specified when rule returns true.
Queue When	If the specified when rule returns true, the request is queued and a Pega-specific SOAP fault that includes the queue ID of the request is returned.
Mapping Error	An error occurs while mapping incoming data from the request message to the clipboard.
Security Error	Authentication fails.
Service Error	A valid instance of the service activity cannot be found.

If the mapping, security, and service errors are not defined, the system returns standard exceptions, such as an AuthenticationException if the supplied credentials are not valid.

For more information about configuring the Fault tab of a service, consult the help topic Service SOAP form Completing the Faults tab.

KNOWLEDGE CHECK

ANSWER

What is the purpose of using a condition in a SOAP exception?

Conditions allow you to add a custom error message to the response.

Reading and writing data to the database

Introduction to reading and writing data to a database

Many Pega applications need to exchange data with each other as well as with external systems. In this lesson, you learn how to use Obj- methods and SQL connectors to read and write data to databases.

After this lesson, you should be able to:

- Describe the use of external databases in Pega applications
- Differentiate between options for connecting to an external database
- Determine when to use Obj- Methods for database integration
- Determine when to use SQL connectors for database integration
- Read and write data using activity Obj- methods
- Use symbolic keywords to navigate lists
- Connect to an external database

The PegaRULES database

Each system stores rules, work, and other data in a Pega relational database known as the **PegaRULES database**.

The PegaRULES database contains all of the permanent data for all the Pega applications. Detailed knowledge about the database is not needed for most application tasks. Designers can create new rules and new properties, and save data without updating the database structure. Built-in tools and wizards help you understand how well the database is operating and where to focus efforts to tune application performance.

As users interact with a Pega system, rows of the database are automatically created, updated, saved, and deleted. The database includes user inputs, decisions, and the results of calculations. For example, for a purchase order request system, as users create purchase orders, the PegaRULES database is updated with the each new purchase order requests. Input from a single user — for example, approving a purchase order request — causes updates to multiple tables in the PegaRULES database.

Database updates occur simultaneously (known as a transaction commit) so that PegaRULES is always up-to-date. Internally, database updates are transmitted as Structured Query Language (SQL) queries. Use Obj- methods to query tables present in the PegaRULES database. By default, Pega uses only the database table name to identify tables for database queries. This can cause issues in some situations. To handle such situations, use fully qualified table names.

For additional information

PegaRULES database overview

Database tables in the PegaRULES database

External databases

Pega applications sometimes require access to data stored outside of the application. For example, an application to provide quotes for automobile insurance policies may rely on a system of record for customer policy data. Also, an order management application may rely on an external inventory database.

The information in an external database is often manipulated by an application. For example, an order management application needs to update the inventory database after a user submits an order. This ensures that future orders reflect current inventory levels. Also, customers may update their home mailing address for their automobile insurance policy.
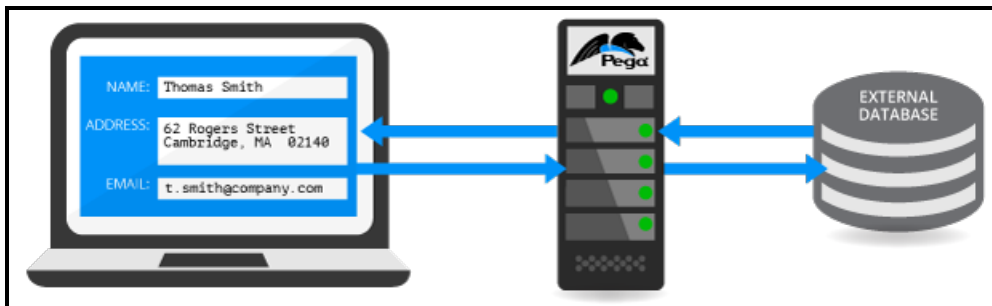
To manage the data read from and written to an external database, you create an external class in your application. An **external class** is a class that corresponds to a table in an external database rather than to a table or view in the PegaRULES database.

External classes differ from other Pega classes in three ways.

- An external class cannot belong to a class group. Instead, it corresponds to a database table not administered by Pega. Each external class should be mapped to a unique database table.

- An external class does not contain either the pzInsKey or pxObjClass properties. These properties are used by Pega as key columns to identify rows in tables in the PegaRULES database. Instead, the external class uses the key column specified by the database table to identify unique rows.

- An external class maps Pega properties to database columns. This mapping identifies the database columns that correspond to the properties used in the application. You can use this mapping to provide more meaningful names for cryptically named database columns. For example, an external class can map the database column CustAdrHome to property .CustomerStreetAddressHome.

Note: For a class mapped to the PegaRULES database, Pega uses the name of the property record as the database column name.

The external class allows you to operate on the data stored in the external database. For example, properties in an external class are present on the clipboard, and you can set property values with data transforms and declare expressions.

Assume that an office furniture purchase order form includes fields for three target property items: chairs, desks, and lamps. You have selected rulesets and versions for the table mapping, and created a class to map to the external table that holds inventory data for chairs, desks, and lamps. A declare expression uses the two source properties — item cost and quantity — to calculate the target property: item total. The declare expression multiplies the item cost from the external database by the quantity to calculate the item total.

KNOWLEDGE CHECK

ANSWER How does an external class relate to an external database?

An external class maps the contents of the external database to properties in Pega. This allows Pega to use the data from the external database in applications.

©2017 Pegasystems Inc.

Obj- methods

When you create a class mapping to an external database table, you can read from and write to the database using activity records. Activities consist of a series of steps that represent the operations performed when the activity runs. Each step includes a method that describes the action to perform.

Note: For information on determining when to create an activity, see the PDN article Introduction to Activities. For more information on the activity methods provided for activities, see the Help topic Methods and Instructions by Name.

When you configure an activity to read from or write to a database, you use a subset of activity methods called Obj- methods, provided by Pega. Obj- methods operate on one or more objects, or rows, in a database table, to open, save, and remove database records.

Tip: You can use Obj- methods to read or write class instances in the Pega database, in addition to records in an external database.

To configure an activity for reading from or writing to an external database table, use the methods described in this lesson.

Read from a database

The obj- Open methods are used when you want to update the object you read. Use **Obj-Open** or **Obj-Open-By-Handle** to load an instance of a class stored in either the PegaRules database or an external database. Obj- Open and Obj-Open-By-Handle methods create a clipboard page for the instance opened.

Note: If you want to fetch read-only data on a Data Page use a lookup and report definition.

Use **Obj-Browse** to search the database, retrieve multiple records, and copy them to the clipboard as an array of embedded pages. Use the Obj- Browse method optionally followed by the Obj- Filter method to create a list of search results.

Use **Obj-Refresh-and-Lock** if the instance is already present on the clipboard as a page. Use Obj-Refresh-and-Lock to test whether a page is current and if a lock is held. If the object is locked, Obj-Refresh-and-Lock has no effect. However, if the object is not locked, the activity acquires a lock and refreshes the page if it is stale.

Save to a database

Use **Obj-Save** saves the contents of a clipboard page to the database. Obj-Save immediately writes the object to the database only if the **WriteNow** parameter is selected. If the WriteNow parameter is not selected, the Obj-Save operation becomes a deferred save. A deferred save adds a deferred operation to an internal list of operations to be performed on the next commit. The internal list is also known as deferred operations list or deferred list.

Until the next commit occurs, either explicit or automatic, changes are not reflected in the database. The benefit of this deferral is the ability to perform multiple, back-to-back saves on the same object instance. Pega combines the updates into a single cumulative update at the time of commit. This decreases the interval of time in which locks are held within the database and maximizes concurrency within the application.

A valid reason to perform a WriteNow is the need to immediately reread this instance before issuing a commit. Using this parameter can cause incorrect information if you do not use proper locking and error checking in your configuration. For more information, see the PDN article When to set the WriteNow parameter in the Obj-Sav and Obj-Delete methods.

Delete a database record

Use **Obj-Delete** or **Obj-Delete-By-Handle** to remove an instance from the database. Obj-Delete deletes a database row based on the contents of a clipboard page, or marks it for later deletion with the Commit method. Use Obj- Delete-By-Handle to delete an instance identified by its handle.

Cancel or rollback operations

Use **Obj-Save-Cancel** to undo the last change. Obj-Save-Cancel cancels the most recent uncommitted Obj-Save method so that the instance is not written as part of a later Commit operation. You can also use this method to undo an Obj-Delete that has not yet been committed. The page identified in the most recent Obj-Save or Obj-Delete method is removed from the set of pages pending a Commit method. When a later Commit method executes, this page is not included in those committed. This method updates the *pxMethodStatus* property.

Use the **Rollback method** to cancel or withdraw any previous uncommitted changes to the PegaRULES database (and to external databases accessed from an external class) from the current Thread. When you use the Rollback method, all pending Obj-Save and Obj-Delete methods are canceled. As a result, all uncommitted database operations are withdrawn. If an instance is locked and the **ReleaseOnCommit** box was selected, the lock is released. Locks on instances where the ReleaseOnCommit box was not selected are retained.

Important: Using the Rollback method cancels all uncommitted Obj-Save and Obj-Delete operations, not only the most recent one.

KNOWLEDGE CHECK

⌄ ANSWER _____ and _____ methods create a clipboard page for the instance opened.
Obj- Open and Obj- Open-By-Handle

How to connect to an external database

Pega applications frequently need to exchange data with external systems. For example, an application requirement may be to respond to requests for order status updates. Order status data is stored in an external system of record database. The Pega application has to integrate with the SoR database to retrieve and modify data hosted outside of the Pega 7 system.

Pega provides the External Database Table Class Mapping wizard to guide you through the process of connecting to an external database.

Select a connection method

If your application performs simple SQL statements against a single external database table, use the **External Database Table Class Mapping wizard** to create an external class mapping. The External Database Table Class Mapping wizard maps the database contents to an external class in your application. Read and write data to the database using Obj- methods in activities.

If your application performs more complex SQL statements such as joins or stored procedures against external database tables, configure Connect SQL rules.

Tip: In certain situations, you may need to use a combination of Obj-methods and Connect SQL rules. For example, you may need to configure a complex search using an SQL Connector; the remaining operations can be performed with Obj-methods.

Connect to the external database

Database table mapping and the SQL connector both require you to perform three steps:

- Register the external database.
- Identify the database table containing the needed data.
- Map the database table columns to application data elements.
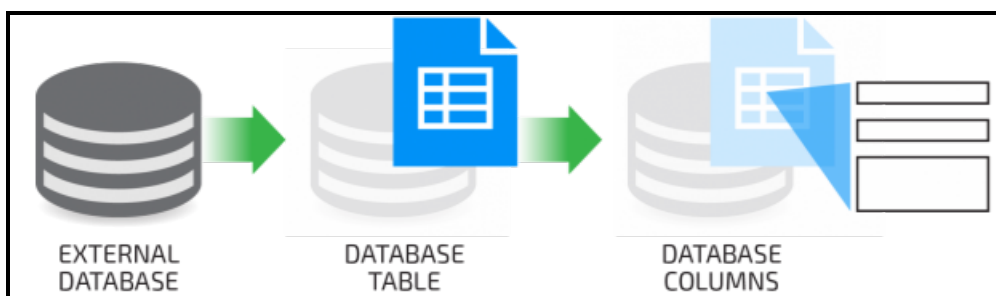
Register the external database

Before you can read from or write to a database, you must register it with the Pega server. To register a database in Pega, you create a *Database* record. For example, if the system of record for customer data is an external database, you create a database record named *Customer*. In a Pega application, you reference the database record to connect to the external database.

©2017 Pegasystems Inc.

A database record allows you to identify connection information for either the Pega database or an external database. The **use JDBC Connection pool** and **use configuration in preferences** options are used for connecting to the Pega database.

The database record defines an alias for the database, and contains the information needed to connect to the database. Configuring the database record allows you to then connect to the database using either the Database Table Class Mapping wizard or the SQL connector. The following example shows one possible configuration to an external database.

When configuring a database record, consider how you plan to connect to the database. For external databases, select the **use JDBC URL listed below** option. This option allows you to specify the database URL and authentication information needed for Pega to connect to the database. For more information on providing authentication information, see the Help topic [Completing the database tab](#).

To test your connection to the external database, click **Test connection**. This test allows you to verify that the information you entered leads to a connection. Pega tests the configuration and displays a status page indicating whether the test was successful or not.

KNOWLEDGE CHECK

What is the purpose of a database record?

A database record specifies the connection information needed to establish a link to a database, such as a system of record.

Identify the database table containing the needed data

Once you identify the database that contains the required data, you identify the table that contains the relevant data. To link a class to a database table, you create a *Database Table* record.

Edit Database Table: TGB-Data-ABARoutingNumbers

ID TGB-Data-ABARoutingNumbers   RS HRApps [Edit]

Database   **History**

Database

Database
ExternalDatabase

Reports database

Catalog name

Schema name
pegadata

Table name
ach_lookup

Test connectivity

Note: Database table records can be found in the Records Explorer, under **SysAdmin > Database Table**.

To configure a database table record, you specify the database to connect to and the table to read from or write to. In the previous example, Pega instances of the data class *TGB-Data-ABARouting Numbers* are read from and written to the table *ach_lookup*.

Both the External Database Table Class Mapping wizard and the Connector and Metadata wizard create a database table record for you. In either wizard, you enter or select the database name and Pega queries the database for a list of tables for you to choose from. If necessary, enter the name of the schema that contains the table.

To test your connection to the database table, click **Test connection**. Pega tests the configuration and displays a status page indicating whether the test was successful or not.

Note: Creating a *Database Table* record for every concrete class used in your application is not necessary. For classes written to the Pega database, such as the classes that represent case types, Pega uses pattern inheritance and class groups to locate the appropriate *Database Table* instance for an object. If none is found, Pega uses the table *pr_other* as default.

KNOWLEDGE CHECK

What is the purpose of a database table record?

A database table record maps instances of a class to a database table.

Map the database table columns to application data elements

After you identify the database table, you identify the columns in the table that contain the data used by your application. Both the External Database Table Class Mapping wizard and the Connector and Metadata wizard use your selections to create an external class with properties mapped to the columns you select.

One you complete either wizard, you can reference the properties in the external class in your application.

How to use symbolic indexes to reference lists

Pega provides a set of **symbolic indexes** to access items in a page list without using an explicit index number. For information about symbolic indexes, refer to Symbolic indexes - APPEND, CURRENT, INSERT, LAST and PREPEND in the PDN articles Expressions - How to reference parts of aggregate properties.

When data is read into a Pega application, the data may include a set of records recorded in a page list. Pega needs to navigate the entries in the list to locate a specific item. Often this is done by iterating through the list to find an item, or appending an item to the list using symbolic keywords.

For example, consider the vehicle inventory for an automobile dealer. When a salesperson sells a vehicle, it must be removed from inventory. In order to remove the vehicle, you need to know which entry in the list corresponds to it. Otherwise, you may remove the wrong vehicle from inventory.

You can configure an iteration that performs a repeated computation over a collection, such as all the elements in a Page List property. In the example, you need to iterate through the list to identify a specific item in the list. When you add a loop to a data transform or an activity, you can use the **<CURRENT>** index keyword to read the current item as the loop iterates. Or you can use **<INSERT>** index keyword to write a new item at the current index and push the remaining items down in the list.

For more information about entering loops, refer to Completing the Steps tab - Entering loops.

Read and write list items using an activity

In Pega, you may use a data transform or an activity to iterate over a list. For instructions on configuring an iteration in an activity step, see the Help topic Activity form — Completing the Steps tab — Entering loops.

Read and write list items using a data transform

You can use a data transform to iterate over a data page that uses a list structure. To review a sample procedure, see Iterating over a page list in a data transform using Pega 7.

 Note: You cannot use a data transform to write to a database. See Apply-Data Transform method.

Use symbolic indexes

If you know the index number in a list, you can add an item sequentially. If you do not know the index number, use symbolic indexes to iterate over a list.

To write a new entry to the end of a list, use **<APPEND>** as the index for the new entry. Use the <APPEND> keyword with the **Update Page** action in a data transform, and in an activity step.

 Important: When iterating over a list in a data transform by using the *For Each Page In* action, you cannot use the <APPEND> keyword. Instead, add a step to the iteration and use the **Append to** or **Append and Map to** actions. For more information on using these actions in a data transform, see the Help topics Data Transform form — Completing the Definition tab — Append to action and Data Transform form — Completing the Definition tab — Append and Map to action.

To write a new entry to the beginning of the a list, use **<PREPEND>**.

To identify the last item in a list, use **<LAST>**.

If you want to enter an item sequentially in a list, and you do not know the index number, use the <CURRENT> and <INSERT> keywords when iterating over the entries in a list. For example, for a purchase order application that allows users to order items, there is a list of unique vendors. For each line item ordered, business users select a vendor from a list of preferred vendors for the purchase.

Follow these steps to create a list of unique vendors.

1. Create a data transform to iterate over the list of order items.

2. Identify the vendor for the item.

3. Compare the vendor to the list of unique vendors.

If the vendor is not listed, copy the vendor from the order item to the list of unique vendors. Within the iteration, use the <CURRENT> keyword as the index for the order item.

> **Tip:** Pega also provides a standard activity parameter, *pyForEachCount*, for identifying the current entry in a list. Unlike <CURRENT>, *pyForEachCount* can be passed as an activity parameter and used in expressions.

To configure a data transform step to iterate over a list, starting with the first entry, select the *For Each Page In* action. The following example shows a data transform configured to iterate over a list of order items in a purchase request that creates a list of unique vendors.

	Action	Target	Relation	Source	
• 1	Remove ▾	.PRVendorList			🗑
▾ • 2	For Each Page In ▾	.LineItems		☐ Also use each page as source context	🗑
▾ • 2.1	When ▾	IsDistinctVendor			🗑
• 2.1.1	Append to ▾	Top.PRVendorList	an existing page ▾	.LineItems(<CURRENT>).Vendor	🗑

☐ Collapse All   ☐ Expand All

☑ Call superclass data transform
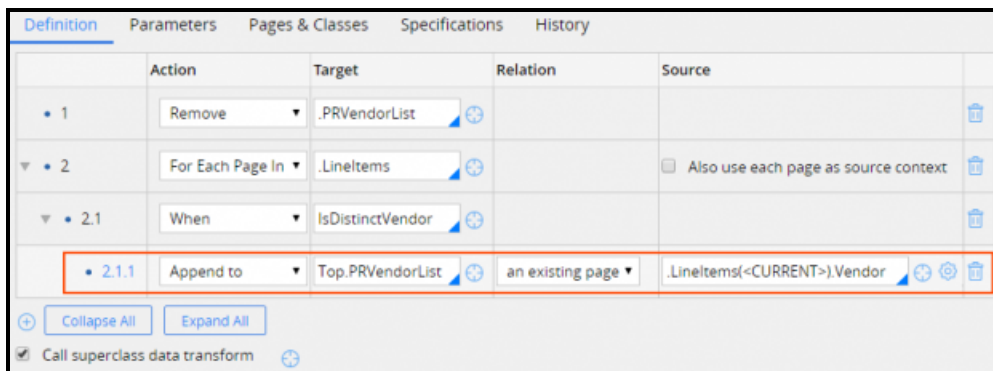
Navigate nested embedded pages

When navigating a hierarchy of embedded pages, you may need to refer to a parent page in the hierarchy. Pega provides two keywords for accessing the correct page in a hierarchy of embedded pages.

Top – Refers the top most page of the embedded page

Parent – Refers to the parent page of the embedded page

> **Note:** If the embedded page has only one parent and no grandparents in the hierarchy, Top and Parent refer to the same parent page.

How to execute SQL using Connect SQL rules

When you need to execute a SQL command or stored procedure as part of a database operation, you may use a **Connect SQL** rule to interact with an external database. Connect SQL rules can be invoked from a data page or from an activity using RDB methods. The Application Explorer lists the Connect SQL rules in your application. The Records Explorer lists all Connect SQL rules that are available to you. For information about creating records, see Connect SQL rules Completing the Create, Save As or Specialization form.

Keywords and notations

Keywords and notations describe mapping to the external database. These keywords and notations provide a two-way mapping between Pega features — the clipboard, Database Name instances, and Database Table instances — and the tables, rows, and columns of the external database. Employ the syntax described in Connect SQL form Data mapping to cause Pega to make run-time substitutions to the SQL before sending it to the external database.

The Connect SQL rule provides four tabs for supported operations Open, Delete, Save, Browse. In addition, the History tab provides information about the rule history. Use these tabs to enter SQL statements to execute when performing the operation.

A dynamically created WHERE clause can add flexibility to SQL connections. See the ASIS keyword to build a dynamic WHERE clause in Connect SQL rules.

Use a Connect SQL rule to map to a table

The following example includes a Connect SQL rule used to access an external *POSTALCODES* table. In the example, the query requests a city and region for a postal code. To map the columns in the select to properties in the clipboard page, use the keyword "as" for the select.

The Applies To class is *ADV-Purchasing-Int-PostalCode*. The mapping between columns and properties takes place within the Connect SQL rule. The mapping does not need to be defined in the class rule. The Package Name indicates the SQL language variant used. The Request Type field holds the name of the rule.

Note: A best practice is not to hardcode table names in the SQL statement. As an alternative, use the class keyword that uses the Database Table instance configured for the class to determine the table name.

The where clause references the ID property in the step page.

Use the characters /* and */ to surround comments within the SQL statement.

To include SQL debugging and return status information from the database software, enter a line at the top of the SQL code defining the name of the page on which to record the messages.

Edit Connect SQL: PostalCode (Available)
ADV-Purchasing-Int-PostalCode • Oracle • PostalCode | PurchasingInt:01-01-01

Open   Delete   Save   Browse   History

OPEN SQL

```
{SQLPage:OpenPostalCodeErrors}
/* this is a comment */
select CITY as ".City", REGION as ".Region"
from {Class:ADV-Purchasing-Int-PostalCode}
where POSTALCODE = {.PostalCode} and COUNTRY = {.Country}
```

Error Handler Flow

Test Connectivity

In this statement, the query searches for a city. The Asis keyword prevents the system from placing spaces or quotes around the value. Using wildcard characters is possible because the Asis keyword is included in the syntax.

KNOWLEDGE CHECK

Describe one way to add flexibility to SQL connections.

A dynamically created WHERE clause can add flexibility to SQL connections.

Connecting to an external database

Use the Database Table Class Mapping landing pages to map database tables to an external class in Pega. The wizard guides you through the steps to create the records needed to map a database table to an external class in your application.

Pega builds upon commercial relational database products from Oracle, IBM, and Microsoft. Detailed knowledge about the database is not needed for most application tasks. Senior system architects and system architects can create new rules and new properties, and save data without updating the database structure and without the need for a database administrator.

Register an external database

Each relational database you need to connect to must be defined as a database instance. When the Database instance has been created and a suitable JDBC library has been installed on the server, you can connect to and interact with that database.

Test connectivity

A database table instance maps classes and class groups to relational database tables or views in a database. Before creating a Database Table instance, you must confirm that the table or view is present in the database schema.

You do not need to create a Database Table instance for every concrete class. At run time, the system uses pattern inheritance and class groups to locate the appropriate Database Table instance for an object. If none is found, the system uses the table pr_other as default.

However, each class representing a table in an external table must have a Database Table instance.

Specify the database table

1. From the Designer Studio menu, select **Data Model** > **Classes and Properties** > **Database Class Mapping** to open the Database Class Mappings landing page.
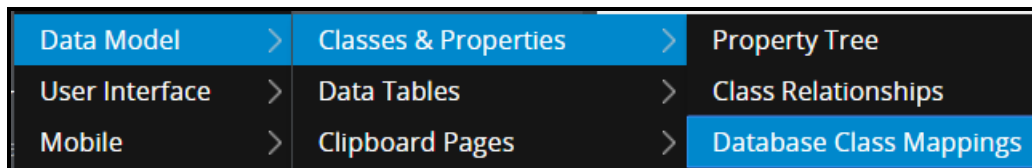
2. In the upper-right corner of the landing page, click **New External Database Table Class Mapping** to

save an existing one or create a new database table instance .

Map database table columns to properties in the ruleset class

1. In the **Database Table Class Mappings** dialog, select a database name and table name:
   a. For **Database Name**, select the database instance that holds the table to which you are connecting.

   b. For **Schema Name**, select the database schema to which you are mapping.

   c. For **Table Name**, select the table you will use to generate the class, properties, and database table instance in your application.

2. Specify the ruleset and version for generated rules
   a. For **Ruleset Name**, select an existing Ruleset that will contain the generated rules for the table mapping.

   b. Enter the **Ruleset Version** for the generated rules.

   c. Enter the **Class Name** of the class to create and map to the external table.

©2017 Pegasystems Inc.

Now that you have selected rulesets and versions for the table mapping, and created a class to map to the external table, the information is saved in the **External Mappings** tab of the class rule.

Map the database columns to properties in the ruleset class

1. Select the **Key** column check box to indicate that the associated properties will be designated as keys for the generated class. Columns that are keys to the table will also be keys to the generated class.

    Note: The Key cannot be edited once the external table has been mapped.

2. Select the **Column Name** for the name of a column in the external table.

3. The **Data Type** is the database-specific data type that was defined when the column was added to the table.

4. In the **Property Name** column, enter a name for the new property. The Property Name is the name of the property in the external class that represents the table column in the Column Name Field. Leave the field blank to use the column name from the database as the property name.

5. In **Property Type**, enter the type of property that will be created. This is dependent on the data type.

6. In the **Map All/None** column, select the:
    a. Check box to map the property to the specified class

    b. **Map All/None** check box to select or deselect all the columns specified class

To learn about the purpose of each of the tables in the PegaRules database, refer to Database tables in the PegaRules database.

Simulating integration data

Introduction to simulating integration data

Pega provides the ability to simulate integrations with services for testing purposes or when the data source is unavailable. You simulate an integration to unit test the integration connector.

After this lesson, you should be able to:

- Explain the need for simulated data in testing integrations

- Explain how to simulate a connection to an external system

- Create an integration simulation

Integration simulation

Integration simulation is useful in situations where the external service is not available or when you want to dictate the response returned. You can simulate any external service as long as you know what data the external service is expecting and returning. The exact interface definition does not need to be in place.

For example, an external service might not be implemented yet. Simulating the service allows you to implement case processing before the service is available.
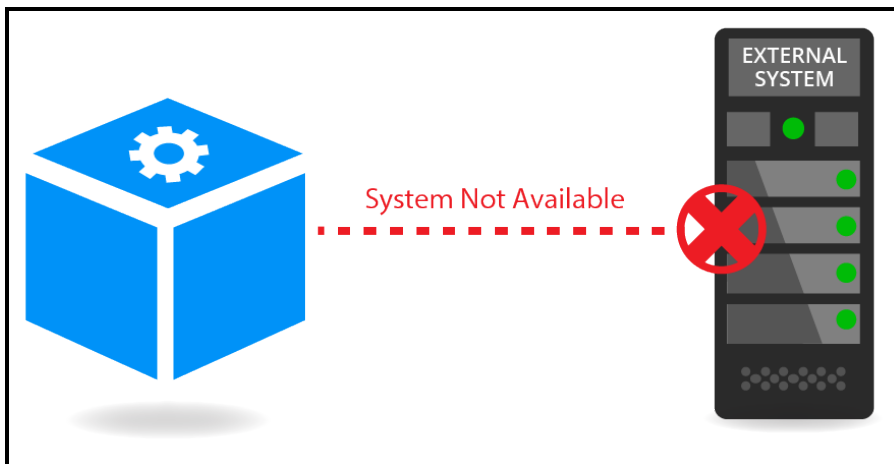
Simulation is also useful in testing since it allows you to dictate the response returned. For example, when testing all permutations of a flow.
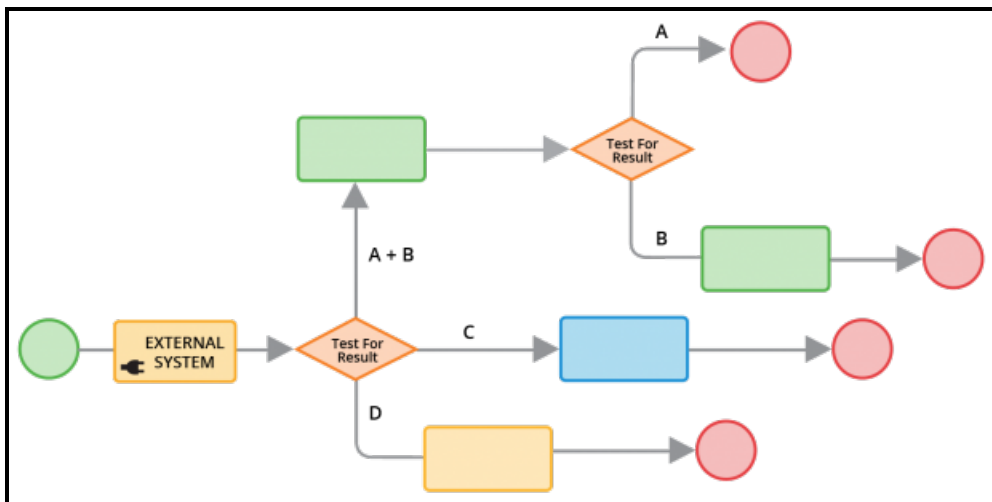
KNOWLEDGE CHECK

When would you simulate an integration?

You would simulate an integration when the service is not available or when the response needs to be dictated.

How to simulate an integration

Pega uses connectors to integrate with external systems. A connector can be invoked from either a data page or an integrator activity. Data pages are used to read, or pull, data from an external system. Activities are used to write, or push, data to an external system.

If the interface definition is available, a connector can be generated using one of the connector wizards. How an integration is simulated depends on whether or not a connector is available.

If you need to simulate an external service which has not yet been defined, but you know what data the service is going to return, then simulate the data page source or integrator activity. Simulate the connector if it already exists.

Simulating a data page

In the data page, select the **Simulate data source** option to simulate the data source in the data page. You can simulate a data page using a data transform, report definition, or activity. Selecting simulation disables any data source configured.
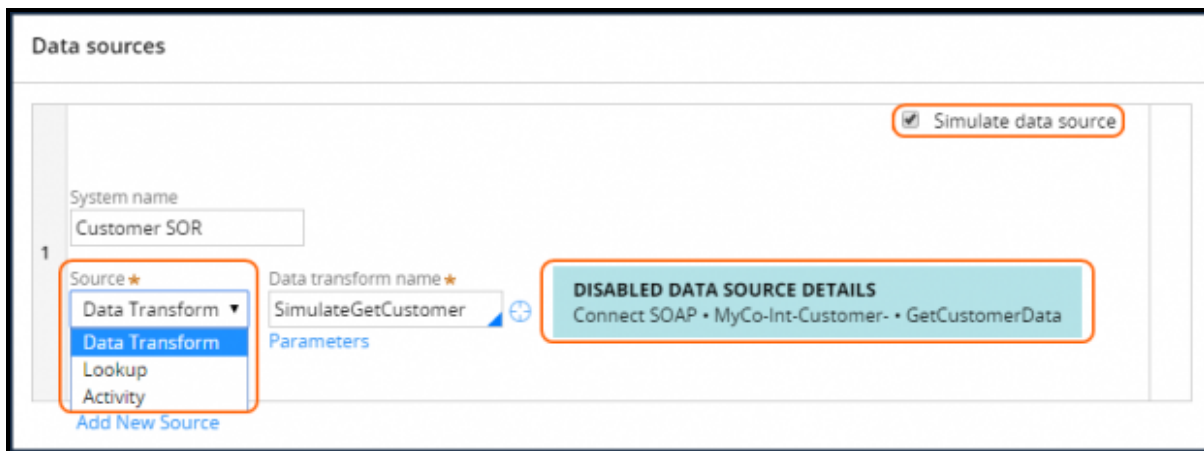
Select **Designer Studio > Data Model > View external data entities** to open the External Data Entities landing page and get an overview of simulated data pages in your application. Source systems marked with a green dot are production ready. Source systems marked with an orange triangle are simulated.

Simulating an integrator activity

If you are invoking the connector from an integrator activity, a simulation data transform can be applied to populate the case with response data.

Use a data transform to define simulation data if your connector is invoked from an integrator activity. Set the simulation data directly in the case in the same way you would map the response data returned from the connector.

Configure connector simulation

You can simulate a connector when the interface of the service to which you are integrating is defined and the connector exists, but the service may still be under development or not yet deployed.

The Connectors landing page (**Designer Studio > Integration > Connectors > Connector Definitions & Simulations**) provides you with an overview of available connectors and their simulations. No connector simulations are available out-of-the-box.

Note: Simulation is not available for SQL connectors.

Connectors are simulated using a simulation activity. The simulation activity sets the properties to be returned by the external service on the integration page used by the connector.

A connector can be simulated either for all users in the application using the *Global* option, or for the current user only using the *User session* option.

Follow these steps to enable simulation of a connector.

1. Navigate to the connector you want to simulate.

2. Specify the simulation activity.

3. Select the scope: Global or user session.

4. Apply the changes to activate the simulation.

KNOWLEDGE CHECK

ANSWER	When would you use the simulation option in a data page?
When the connector has not yet been generated	

Simulating connector data

Connector simulators can be set up for most connector types. First create the simulation activity for the connector. Then configure the activity for the connector.

Create a simulation activity

1. Create a data transform in the same class as the connector.

2. Use the values on the request page to create responses based on incoming values.

3. Set values on the response page to simulate the response returned by the external service.
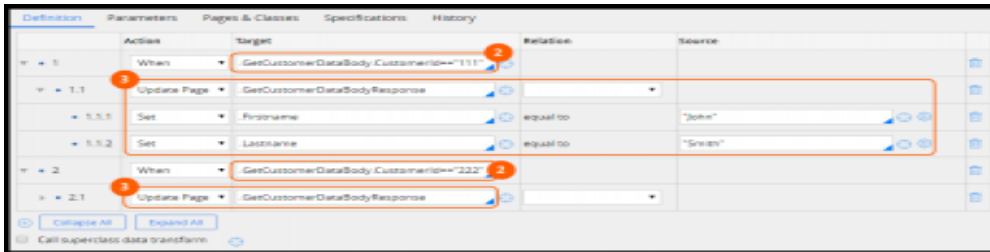
4. Create the simulation activity in the same class as the connector it simulates.

5. Apply the data transform in the simulation activity.
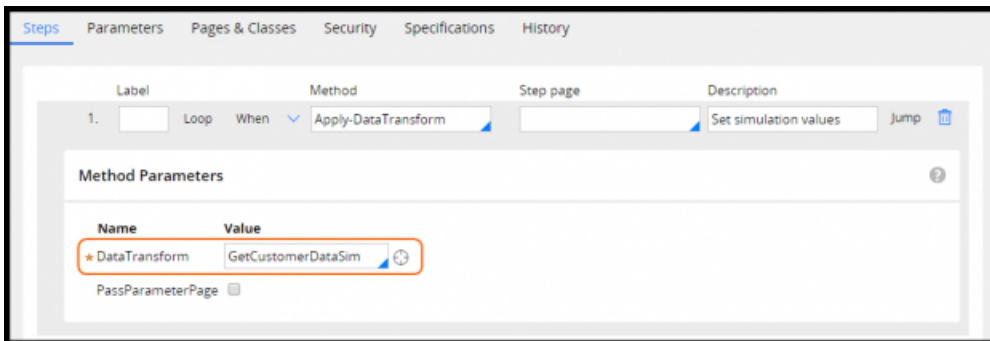
Configure simulation for a connector

1. Open the simulations settings for the connector. You have two options:

   a. At the bottom of the connector record, click **Simulations**.

   b. On the Connectors landing page (**Designer Studio > Integration > Connectors > Connector Definitions & Simulations**), click the **Simulations** link for the connector.

2. Specify the simulation activity.

3. Select the simulation scope.

4. Click **Apply Changes**.
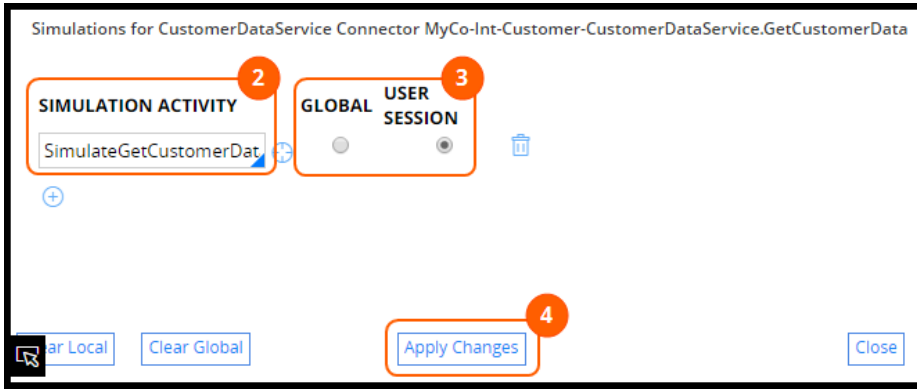
Addressing integration errors in connectors

Introduction to addressing integration errors in connectors

Connectors are used to integrate with external systems. Errors may occur when a connector invokes a service. Errors might be related to the infrastructure — for example, the connector may not be able to connect to the server. Errors may also be due to the configuration — for example, a service might return a response that the connector does not understand.

There are different mechanisms for handling errors, depending on how the connector is invoked. This lesson demonstrates a pattern to build reusable error detection mechanisms for connectors.

After this lesson, you should be able to:

- Describe the role of error detection when configuring connectors
- Configure error detection for an integration
- Address errors returned by a connector

Error handling in connectors

Connectors are used to read data from or write data to an external system. Two types of errors can occur when integrating with an external system:

- **Transient errors** typically do not last long; they correct themselves over time. For example, the connector is unable to connect because the application is being restarted and is temporarily unavailable.

- **Permanent errors** are typically due to a configuration error or an error in the remote application logic. For example, an invalid SOAP request is sent to a SOAP service. In this scenario, the error persists until the SOAP request message is fixed.
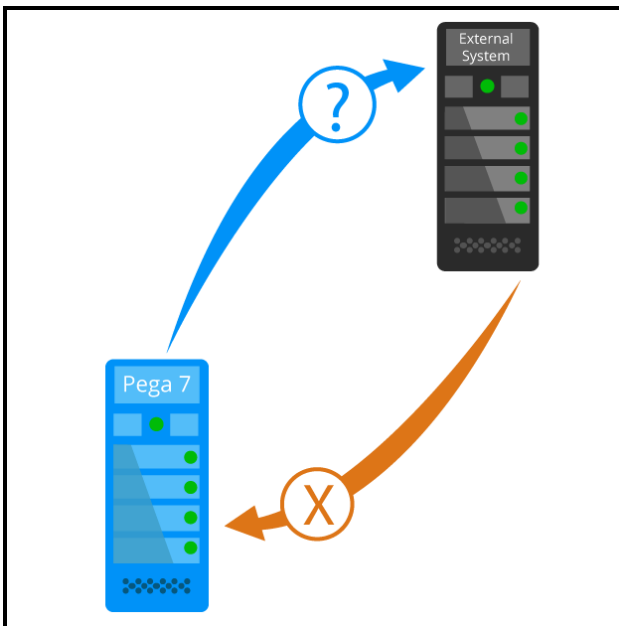
For transient errors, post a note to alert the end user that the integration failed. Ask the user to retry at a later time. Alternatively, if the response is not immediately needed, the connection can be automatically retried.

For permanent errors, write the details to a log file so that errors can be investigated and fixed. In addition, you might want to implement a process for the investigation and fixing.

Since connectors communicate with external systems, the possibility of something going wrong always exists. Therefore, a best practice is to include error handling for all connectors.

KNOWLEDGE CHECK

What type of errors correct themselves over time?
Transient

©2017 Pegasystems Inc.

How to configure error detection

The way errors are detected depends on how the connector is invoked. Connectors can be invoked by data pages or activities. When data pages and activities invoke a connector, the best practices are to:

- Add error detection to all data pages and activities

- Invoke a reusable data transform to handle errors

Pega provides a template data transform called *pxErrorHandlingTemplate*. This can be used to create a reusable error handling data transform. The error handler data transform can be used with both data pages and activities. The *pxErrorHandlingTemplate* data transform is in the base class and is shipped as part of the product.

In addition, each connector has an error handling flow. Pega automatically invokes the error handler flow if the error is not detected by another mechanism.

Data pages

Connectors are used with data pages to read data — such as customer and insurance policy data — from an external system. Data pages are loaded on demand. All data source errors must be handled as part of the data page load mechanism. The type of data source being leveraged to load the data page affects how errors are detected and handled.

- Connectors, report definitions, and lookups use the response data transform to detect errors.

- Activities use a transition condition in the activity step to detect errors.

The response data transform is invoked after the connector call is complete. If there is an issue, messages are added.
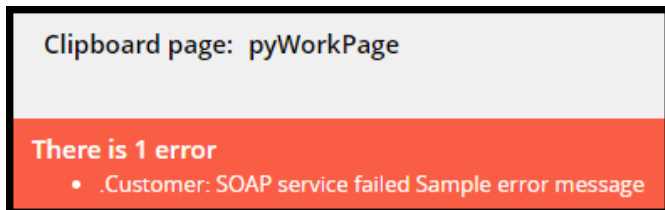
In your response data transform, use a when condition to check for any error messages on the page. If an error has occurred, apply the reusable error handling data transform.

Activities

Connectors are used with activities to write data to external systems. For example, a connector may be used to update data in a system of record. When the connector is invoked from an activity, use a transition condition in the activity step invoking the connector. If an error is detected, apply the reusable error handling data transform.

Error handling flow

The error handling flow feature detects errors that are not detected in a data page or an activity. This feature is always enabled.

The error handler flow allows you to implement a process for handling the error. It is typically used when the connector response is not required immediately — for example, when updating a legacy system.

<<image: error handler flow>>

KNOWLEDGE CHECK

⌄ ANSWER	In data pages, a best practice is to detect errors in the _____.
response data transform	

Configuring error detection for integration

The following procedure explains how to configure error detection and handling for data pages and activities. Data pages and activities invoke connectors to read data from and write data to external systems.

Configure error detection for a data page

To configure error detection for a data page, create a new data transform for error handling. This data transform is applied when an error is detected.

1. In Designer Studio, search for the standard data transform called *pxErrorHandlingTemplate*.
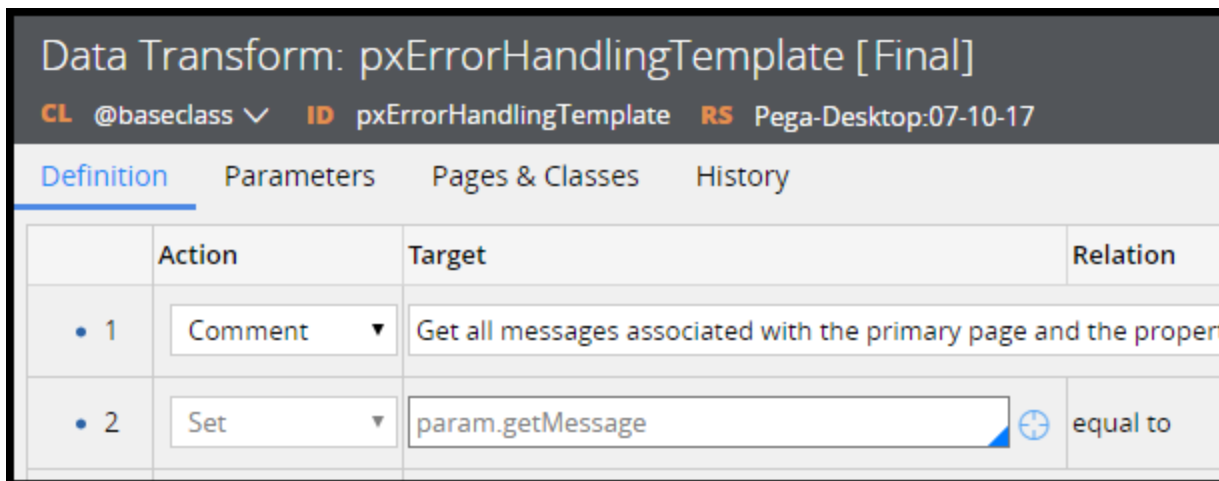
2. Save *pxErrorHandlingTemplate* with a new name to create a new data transform in your application.
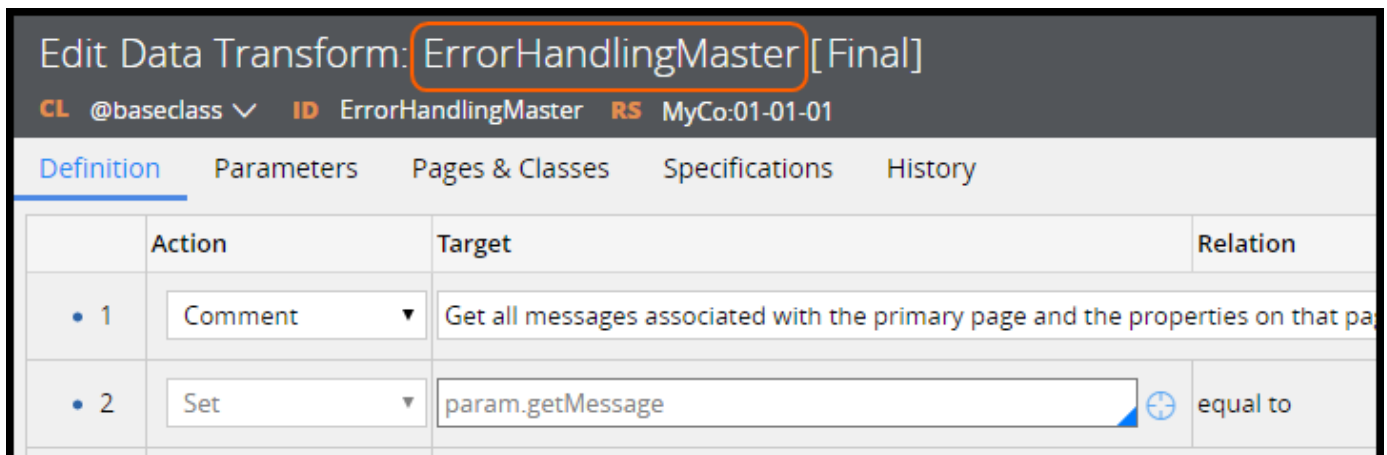
Next, in your response data transform, apply the error handling data transform you just created.

1. Open the response data transform. Create a new response data transform if one does not exist.
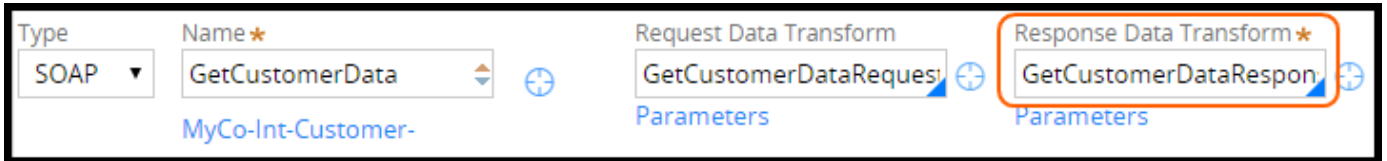
2. Use the standard when condition *pxDataPageHasErrors* to detect errors in the data page.

> **Important:** The steps highlighted in the following image are automatically added to response data transforms by default. When you change the name of the error handling data transform, you must update step 2.2 to apply your error handling data transform instead of the standard error handling data transform.
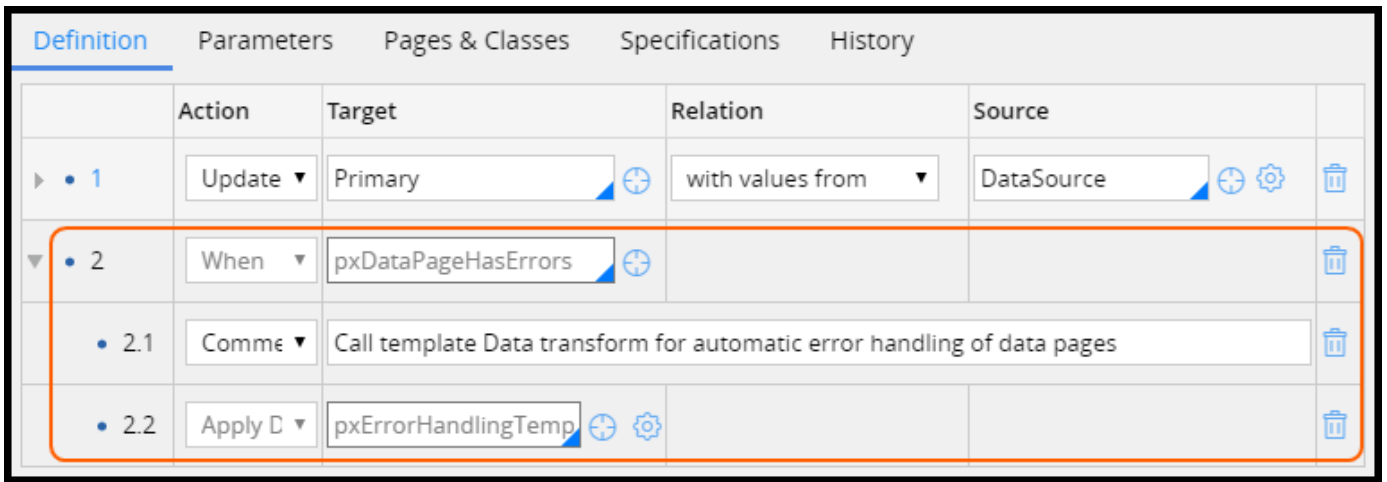
Configure error detection in an activity

The error handling data transform may be used when connectors are called from an activity as well.

1. Define an error label for the error handling step.

2. Identify the steps in your activity that include error detection. Click **Jump** to define a transition condition.
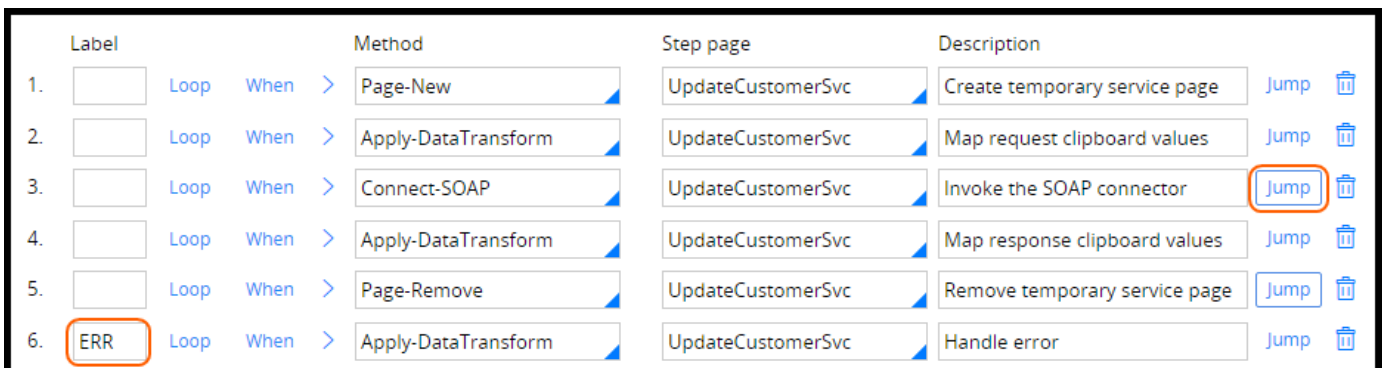
3. Select the *StepStatusFail* when condition to detect errors. Jump to the error step when an error occurs.

4. You can use the error handling data transform you created previously that specifies how to handle the error. Select the Apply-DataTransform method. In the error handling step, select your data transform.

How to address errors returned by a connector

(missing or bad snippet)

The following process describes how to address errors returned by a connector. Options to address integration errors depend on how the connector is used. Use an error handling data transform if the errors are detected using a:

- Response data transform in a data page
- Transition condition in an activity

If the error is not detected in the data page or the activity, then the error handler flow for the connector is invoked to detect the error.

Error handling data transform

If there is an immediate need for the response to be returned by the invoked service, you should:

- Display an error message
- Write the error to the log file

Writing a message to the log file helps troubleshoot errors. For example, a log file can be analyzed to identify patterns related to a specific error. In the log message, include details about the connector request to help identify the cause of the error.

Another option is to generate an email that includes error details. The template error handling data transform includes examples of standard utility functions to:

- Get available messages
- Clear messages
- Add a message to the data page
- Write a message in a log file
- Send an email

	Action	Target	Relation	Source	
• 1	Comment ▼	Get all messages associated with the primary page and the properties on that page as a String.			🗑
• 2	Set ▼	param.getMessage	equal to	@(Pega-RULES:Default).pxGetPag	🗑
• 3	Comment ▼	Removes all messages from the primary page. Also removes messages from all properties embedded on			🗑
• 4	Set ▼	param.clearMessage	equal to	@(Pega-RULES:Default).pxClearM	🗑
• 5	Comment ▼	Associates a message with the primary page and invalidates the page.			🗑
• 6	Set ▼	param.addMessage	equal to	@(Pega-RULES:Default).pxAddMe	🗑
• 7	Comment ▼	Logs the message for error level in pega logs.			🗑
• 8	Set ▼	param.logMessage	equal to	@(Pega-RULES:Default).pxLogMes	🗑
• 9	Comment ▼	Send an email message using data from input clipboard page. Copy the data page to a new page and pass			🗑
• 10	Set ▼	SampleStepPage	equal to	D_SampleDataPage	🗑
• 11	Set ▼	param.sendMessage	equal to	@(Pega-IntegrationEngine:Default	🗑

Retry a connector invoked from a data page

If the returned error is temporary, you may give the user the option to retry the connector. To retry the connector, configure the data page refresh strategy:

- Create a when condition that returns true when there are no error messages.
- Set the **Do not reload when** setting so the data page does not reload if there are no error messages.

Error handler flow

When the response to be returned by the connector is not immediate needed for further processing, consider using an error handler flow. Configure the error handler flow in the service tab for the

connector. The error handler flow feature is always enabled. The error handler flow is not executed if the error is detected in the response data transform or in a transition in an activity.

By default, connectors use the standard *ConnectionProblem* flow. The flow can be copied and customized. You may also choose to create an alternative error handler flow.
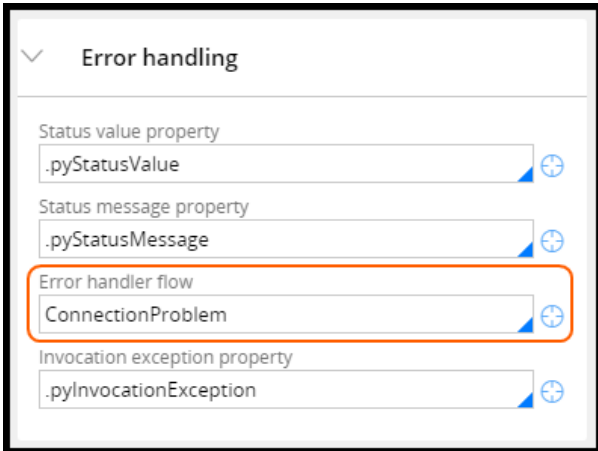
When an error occurs, the original flow execution is paused. Control is handed over to the error handler flow. If the resource is unavailable, a transient error may be preventing processing. If there is no transient error, the connector is retried, and processing continues in a flow called *FlowProblems*.
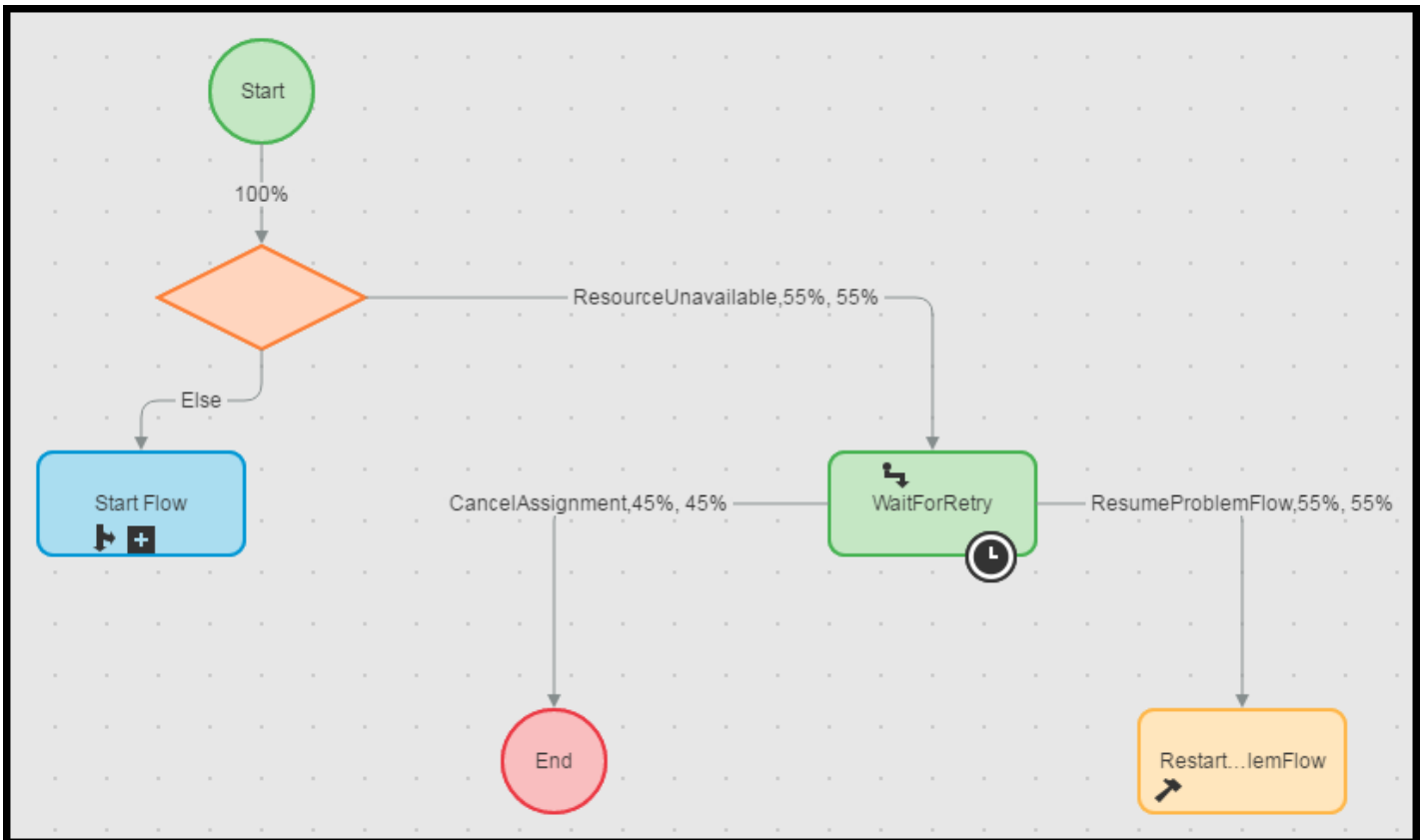
The *FlowProblems* flow either routes the work item to a problem flow workbasket or notifies an operator about the issue. The operator may:

©2017 Pegasystems Inc.

- Retry the connector

- Resume the flow without retrying the connector

- Restart the initial flow

- Cancel the error handling flow

KNOWLEDGE CHECK

	When would you use an error handling flow for your connector?
You would use it when the response is not immediately needed and a process for handling the error is required.	

Managing integration settings

Introduction to managing integration settings

The use of global resources settings for references to external systems, rather than fixed text values in rule forms, allows greater flexibility for changing values such as port numbers, addresses, and URLs.

In this lesson, you learn how to apply global resource settings to reference any property of the appropriate type on a data page. You also learn how to apply global resources settings in rule forms.

After this lesson, you should be able to:

- Explain the need for global resource settings
- Compare mechanisms for setting global resource settings values
- Describe the global resource settings pattern
- Add resource settings for an integration

Global resource settings

Before an application is live, it moves through many environments. Typically, applications go through development, staging, QA, and production. When migrating an application from one server or environment to another, references to the external systems connected to the application (such as endpoint URLs and JNDI servers) typically change. The information required to connect to these external systems must be modified, depending on your environment.
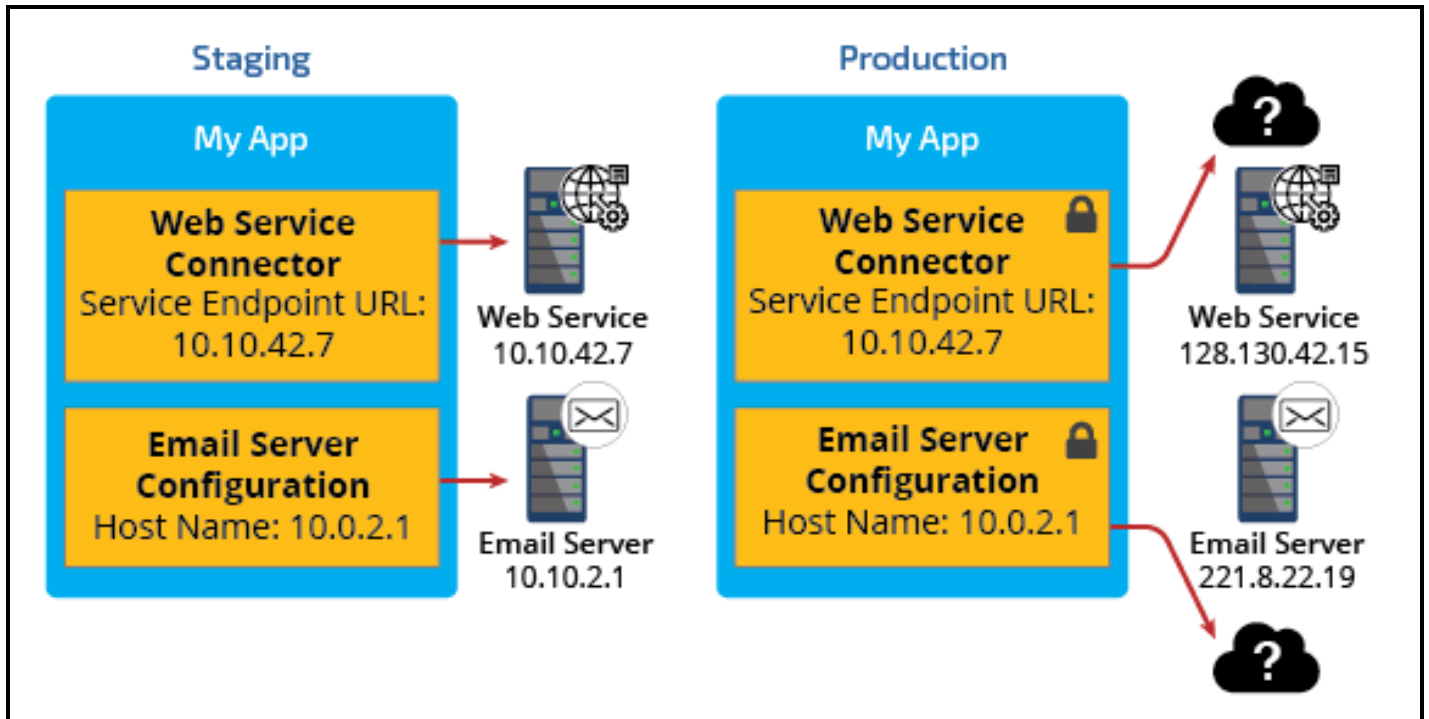
In this example, you have a web service connector that accesses a web service. There is also configuration for the email server used to send and receive email in the application. When the application moves from the staging environment to the production environment, the URLs and host names all have to change because the production web service and email server have different values.

The example provides only two sets of settings, but an application could have dozens of connectors and setting information. You do not want to have to remember all of the different resources and update each one individually. Additionally, some of the values are in rules — these belong to locked rulesets. You do not want to unlock your ruleset after promotion. The risk is that you would miss one or two settings and delay the application going live.

To avoid missing a setting, use the **Global Resource Settings** pattern to reference the external systems. Global Resource Settings allow you to define values for settings that can vary depending on environment, without requiring the update of integration rules and data instances.
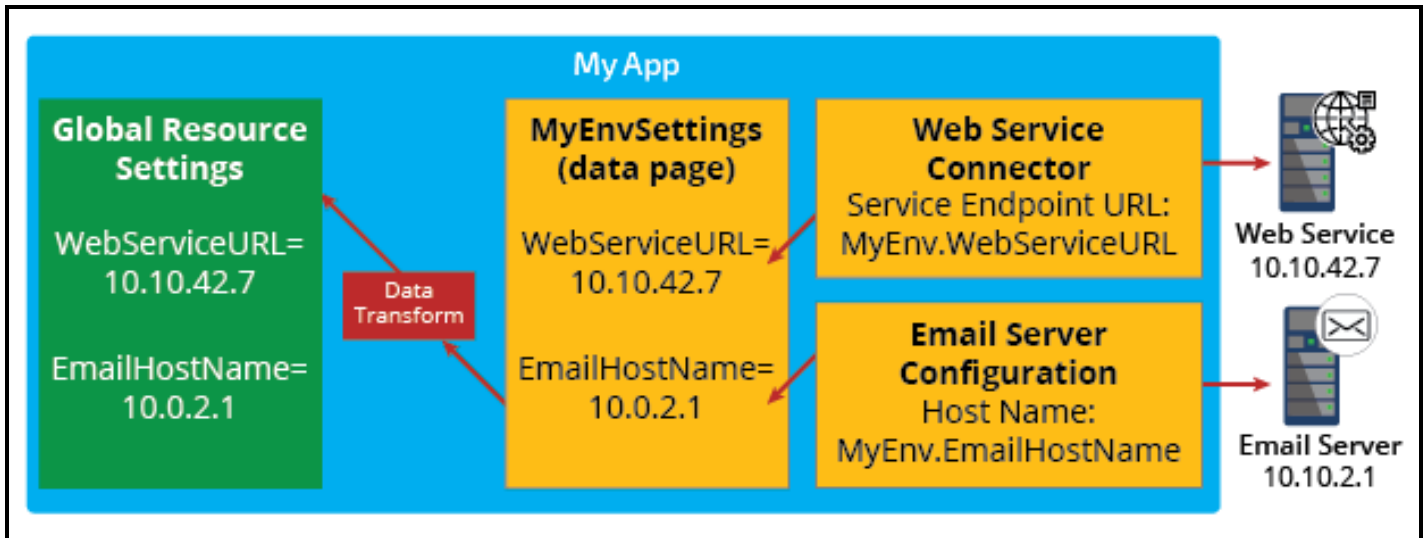
In this pattern, you create a class that contains the configuration settings for an integration that has values able to change from one environment to the next. You then have your resources access a data page to load those settings. This allows you to have a place to maintain and update these settings.

KNOWLEDGE CHECK

ANSWER	What is the largest benefit of using global resource settings?
The largest benefit is the ability to reference external systems in a way that can vary depending on environment, without requiring the update of integration rules and data instances.	

How to configure a global resource setting for a connector endpoint

When you configure a global resource setting (GRS) for an integration, you first create a class for the references. You place all GRS rules in the same ruleset as the integration rules. Next, you determine which environment references to external systems will use the feature. Then, you create a page property for each environment reference. Continue the process by creating a data transform to assign values to the environment properties using utility functions. Finally, you create a data page to tie these artifacts together.

Assume that you have generated a connector for an inventory check SOAP service using the Create SOAP Integration wizard.

Designate a class for the references

The first step in implementing GRS is to create or identify a class to contain environment properties that represent those references.

As a best practice, create the class in the integration's base class. This helps avoid later confusion when users access multiple integrations, since each integration has its own data page. Create a class called *Env* in the base integration class.

In this case, create the *Env* class in the Inventory class generated by the Create SOAP Integration wizard.
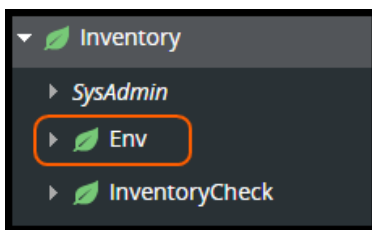
Create the environment properties

The next step in implementing the GRS is to determine which environment references to external systems will use the feature. This [table](#) contains classes that support the GRS syntax. Create a page property for each environment reference. For SOAP connectors, use the class *Embed-Env-SOAP*.
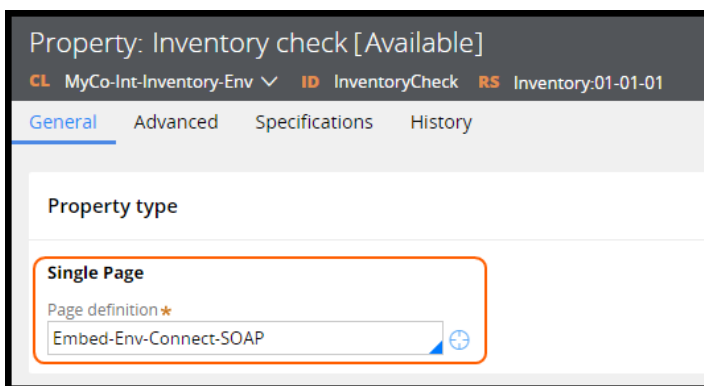
Create a data transform to assign values to the environment properties

After you create the environment properties for the integration, you must create a data transform to assign values to those properties. This data transform is used to source a data page.

Note: Hard coding the environmental variables in the data transform requires you to unlock the ruleset to update the values. As a best practice, never unlock a locked ruleset.

Pega 7 provides several ways of specifying environmental variables without requiring unlocking rulesets. The following table provides the options with relevant utility functions used to obtain the value.

Type	Function	Description
Dynamic System Settings	getDataSystemSetting	This function reads the specified Dynamic System Settings from the database and returns it as a string.
Java System Property	getJavaSystemProperty	This function returns the specified Java system property as a String.
Java Properties File	getJavaPropertyFromFile	This function loads the specified Java properties file and returns the value of the property specified in the function's Key parameter as a String. If the property cannot be found, the function returns an empty string.
JNDI Entry	getJNDIEntry	This function does a lookup for the specified JNDI entry and returns it as a string. If the lookup cannot be performed, the function returns an empty string and writes an error to the log.

Note: The environmental variables should not be packaged as part of the application since that would overwrite the settings on the target system when an application is migrated.

In this example, a Dynamic System Settings is used to store the values. Since Dynamic System Settings are data, they can be updated without the need of unlocking a ruleset.

Consider a naming convention to categorize the Dynamic System Settings entries and assure their uniqueness. For example, the first entry could be GRS, a namespace for these types of entries. The second is the name of the interface, and the third is the name of the property.
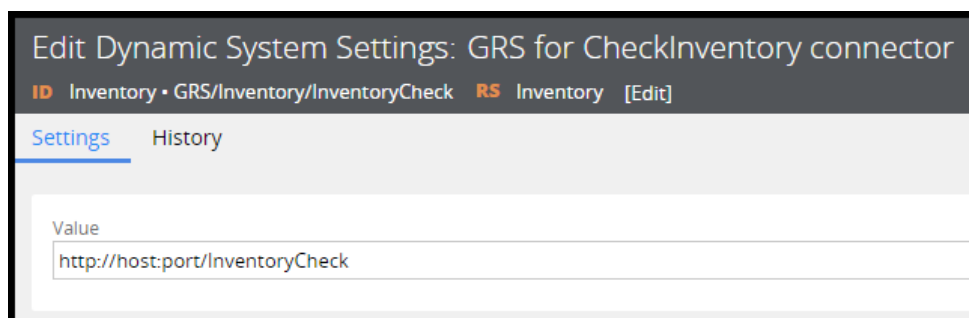
In this example, the *getDataSystemSetting* utility function rule is used to obtain the value in the data transform.

Note: You must Base64 encode any passwords before you use them in rules or data instances. Use the Encode() function to encode passwords.

Create a data page

The final rule you create is a data page that ties everything together. As a best practice, choose a name that includes the name of the integration. This helps avoid later confusion when users access multiple integrations (and each integration has its own data page).

For object type, enter the class created in the first step and select the node scope. Use the data transform created to populate the data page.

Use the Global Resource Settings syntax for references to external systems
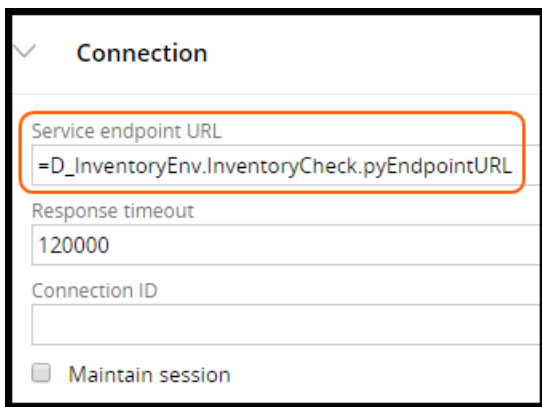
After you have set up the data page for GRS, use the following syntax to refer to the values:

=DataPageName.IntegrationPropertyName.FieldPropertyName

Where:

- *PageName* is the name of the data page, and
- *PropertyName* is one of the properties

This is how the reference looks if you want to refer to a SOAP Service Endpoint URL, your data page is named *D_InventoryEnv*, and the property is named *InventoryCheck*. The setting is on the SOAP connector's Service tab.

Execution-time sequence

The following list shows the execution-time sequence for determining the Endpoint URL for the SOAP connector:

1. The SOAP Connector is invoked.

2. A data page property is referenced.

3. The data transform for the data page is executed if the page is not already available on the clipboard.

4. The data transform invokes a utility function to obtain the value of, for example, a dynamic system setting.

5. The value is used by the SOAP connector to invoke the service.

APPLICATION DEBUGGING

Reviewing log files

Introduction to reviewing log files

Pega generates log files to track application and system activity. In this lesson, you learn how to use the PegaRULES Log Analyzer (PLA) to read and interpret log files.

After this lesson, you should be able to:

- Describe the importance of Pega log files
- Identify when to review log files to aid application development
- Describe the purpose of the stack trace file
- Analyze system logs using the PegaRULES Log Analyzer (PLA)
- Access system logs in Pega
- Set up remote logging

Log files

Pega writes errors, warnings, and other debug information to log files. Logs track exceptions and other events that impact your application, and provide valuable insight into their cause. Each log is managed by an **appender**, which determines the type of events written to the log file. Pega manages logs based on the appender configuration in the prlogging.xml file for the node.

In Designer Studio, logs are available from the System Operation landing page. Depending on your settings, you may see the following log files:

The **PEGA** log contains warnings, errors, and information messages about internal operations. This file, also referred to as the Console log or System log, is used for debugging the application. By default, the Pega log is filtered to entries that match the current operator ID.

The **ALERT** log contains performance-related alerts. Use the PLA tool to parse and summarize this file into a Microsoft Excel spreadsheet. See Understanding Alerts.

The **ALERTSECURITY** log contains alerts (identified by the prefix SECU) that suggest improper configuration of Internet Application Composer facilities, or overt attempts to bypass system security features through URL tampering. See Performance alerts, security alerts, and AES.

The **BIX** log is created by the Business Intelligence Exchange during extract operations. BIX is an optional add-on product that provides the extract functions of an ETL (extract, transform, and load) utility.

The **SERVICES-PAL** log contains performance data saved from services. See Performance tool — Statistics for services in the SERVICES-PAL log.

Filter criteria

When you view a log file, Pega displays only lines of the log for your own Operator ID (for all your sessions on this node), in pages of 25 lines each. You may view or set log filtering criteria. To view other entries, you have the option to change or remove the filter.

Logging level settings

Use the Logging Level Settings tool to control which logging events appear in the Pega log. The *prlogging.xml* configuration file defines the levels of logging events. In a multinode Pega system, you can create separate prlogging.xml files for each node.

Note: Rulesets and the Pega class hierarchy are not relevant to logging. If you set logging events for an activity named Data-Party.Research and your system includes several activities of that name (in various RuleSets and versions), executions on the current node of any of these activities may produce logged events.

You may temporarily override the severity settings in the *prlogging.xml* file for the current node. When you make this temporary change, the prlogging.xml file is not altered. Logging on nodes other than your current node is unaffected.

For example, you can change the logging level for activities in the *Work-* class from FATAL to DEBUG. Thereafter, olog() calls in Java steps that have a severity setting of DEBUG or lower appear in the Pega log.

The changes you make take effect immediately and remain in force until the server on the node is stopped, or until you or another developer uses the Logging Level Settings tool again to reset the logging level.

For more information about how to change the diagnostic logging level in a Pega application see Logging Level Settings tool.

KNOWLEDGE CHECK

⌄ ANSWER	What is the purpose of an appender?
An appender manages the writing of errors, warnings, and other debug information to a log file.	

How to access system log files in Pega

System administrators monitor logs throughout the development process. Using the logs to debug your Pega application as you work results in fewer errors during testing and speeds time to delivery.

In Designer Studio, logs are available from the System Operation landing page. Log files can also be obtained from the System Management Application (SMA).

The Pega log file is used for debugging. The other logs in the list illustrated in the following image apply to performance.

You can view logs in Designer Studio or download the current log from the server to your workstation. To view a log, click the text link or zip link to download the log. You can import the file into Microsoft Excel, using a single asterisk character (*) as the field separator character, for ease in reading, sorting, or searching.

Application server authentication may be required to read PEGA log files. Access is controlled by the *PegaDiagnosticUser* role in the web.xml file for the DiagnosticData servlet. Consult your Pega 7 Installation Guide for instructions.

Log messages

In an activity, use the **Log-Message method** to add a message to the Pega log. For more information, refer to the Log-Message method help topic.

How to use the PegaRULES Log Analyzer (PLA)

Regular monitoring of log files during development and production helps ensure your application is operating properly. The **PegaRULES Log Analyzer (PLA)** is a standalone web application that developers and system administrators can use to view summaries of console logs.

Use the PLA to test new or reconfigured Pega applications during user acceptance testing (UAT), performance and stress testing, and immediately after deployment into a production environment. Testing reconfigured applications during UAT, during performance testing, and right after deployment is important because performance, stability, and scaling issues are most likely to occur during these times.

The PLA consolidates and summarizes three types of logs — Alert, Pega, and Garbage Collection (GC) — from individual [Pega JVM](#) server nodes in your application system. The log data provides key information about operational and system health.

View log information to quickly identify, diagnose, and remediate issues that may be degrading or compromising your application:

- To monitor performance, review the **Alert log**. The Alert log contains diagnostic messages that identify individual system events that exceed performance thresholds or failures. Alert messages contain a set of field values that identify the alert, the conditions that caused it, and the state of system when it occurred. The Alert log supports performance-related monitoring.

- To monitor system stability, review the **Pega log**. The Pega log gathers system errors, exceptions (with their stack trace statements), debug statements, and any other messages not specified as alerts. The Pega log can contain messages created by your activities as well as messages created by standard rules. The Pega log is also referred to as the console log or system log.

- For insight into how your Pega application is using memory, monitor the **JVM garbage collection** log.

Each node on a Pega system produces two log files, the Alert Log and the Pega log. The Pega log contains messages created since the most recent start of the server. The log file is usually named *PegaRULES-YYYY-MMM-DD.log*. The date portion of the name indicates the date the application server was recently started (on the current node).

KNOWLEDGE CHECK

ANSWER   Name two specific times in the project life cycle when reviewing log files is important.
During UAT performance and stress testing
Immediately after application deployment

For more information about the PLA, refer to:

[Understanding the PegaRules Log Analyzer](#)

[How to User the PLA](#)

For more information about alerts, refer to:

[Understanding Alerts](#)

[Working with the My Alerts display](#)

[Performance Alerts, Security Alerts and AES](#)

For information about how to customize logs, refer to:

[How to customize logs with the prlogging.xml file](#)

Monitoring logs remotely

Use the System Management Application (SMA) to perform remote logging. Remote logging establishes a connection between the server and a standalone Pega application running on a desktop. The remote logger is updated near real time.

Note: The logs are updated in near real time. If you execute the steps too quickly, the remote logger will be unable to keep up.

1. Select **System > Operations > SMA** to open the SMA.

2. Select a Node. The SMA displays PegaRULES Node Information. Each server hosting the Pega rules engine software (but sharing one system name) is known as a node.

3. Expand **Logging and Tracing** to run the remote logger, and select **Remote Logging**.

4. Click the link at the bottom of the Remote Logging dialog to download the log4j socket server.
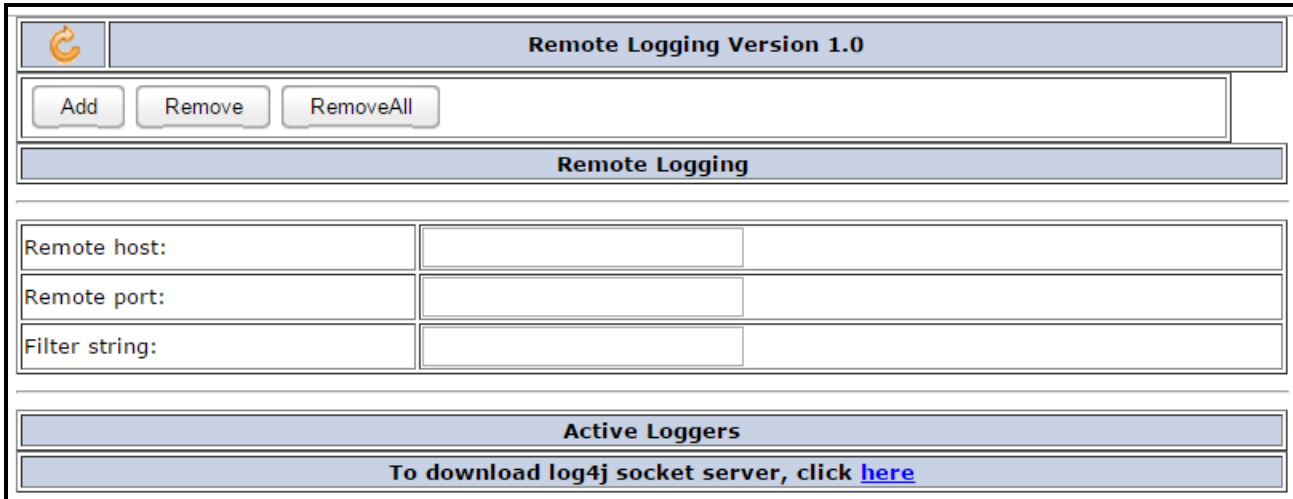
5. Extract and run the log4j socket server. The remote logger user interface opens.

6. Now that the log4j server is running, establish a connection by returning to the SMA and specifying:

- The host

- The port displayed in the first line of the logger (default is 8887)

- Any filters that are appropriate

You are now ready to monitor logs files in near real time as you step through a process to identify when exceptions or events occur.

For additional information, see How to install and use remote logging.

Unit testing rules

Introduction to unit testing rules

Unit testing allows you to test the smallest testable parts of your application. In this lesson, you learn how to unit test individual rules.

After this lesson, you should be able to:

- Describe the role of unit testing in application development
- Unit test a rule with the Run Rule window

Unit testing

An incorrect rule configuration in an application can cause delays in case processing. When an error occurs, end users may need to reassign work, or a case may become broken and require repair by an administrator. For example, consider a case that should be routed to the Fulfillment department. If the case is routed instead to the Accounting department, an accountant must reroute the case to the Fulfillment department. The accountant wastes time rerouting the assignment, while the Fulfillment department is idle. The result is a delay to the customer while the case is reassigned to the Fulfillment department.

To avoid configuration errors such as incorrectly routed assignments, developers test their applications. The most basic form of application testing is **unit testing** individual rules. The purpose of unit testing is to verify that each element of the application — for example, a decision table or a report definition — works as expected. Unit testing reduces the risk of a configuration error in one rule propagating to other rules in the application and causing significant delays to case processing.
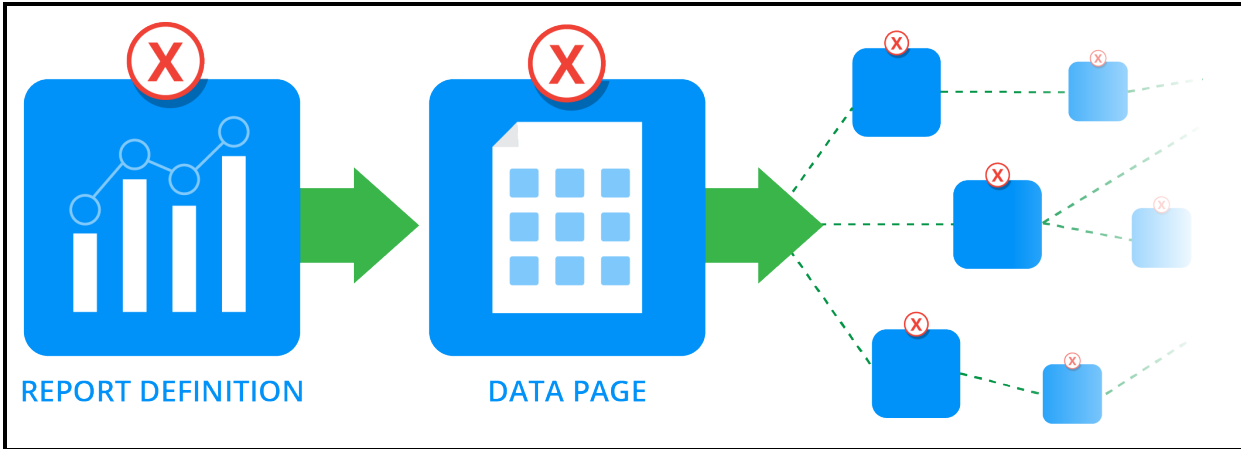
Use unit testing to reduce configuration errors. Consider a decision tree that evaluates a property. The application reads the property from a data page sourced from a report definition. By unit testing the individual rules as you configure them, you know that each rule works as expected. If the decision tree returns an incorrect result, but the data page contains the correct data, you can isolate the error to the decision tree.
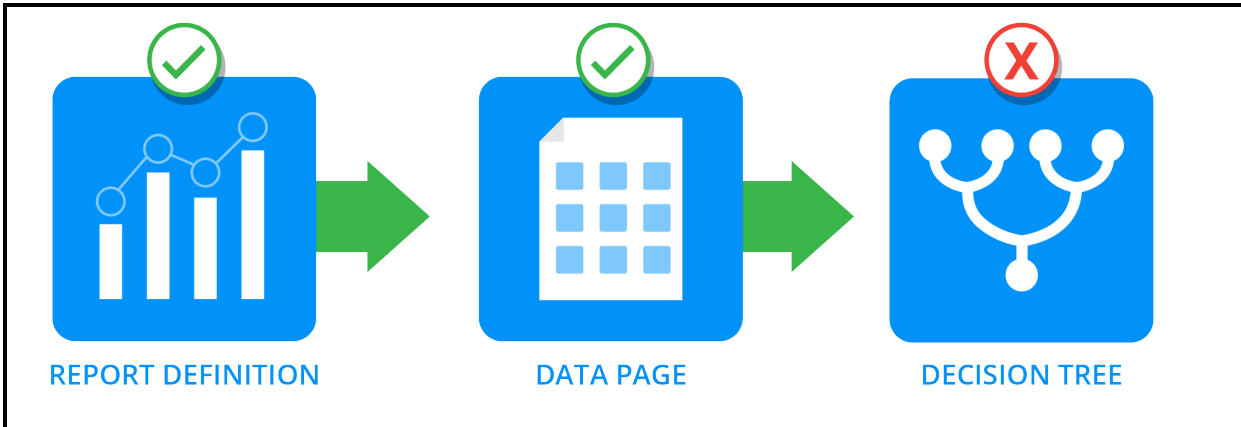
KNOWLEDGE CHECK

The purpose of unit testing is to _____.

identify configuration errors in a rule before the errors are compounded in an application when the result of the rule is depended upon by another rule

How to unit test a rule

Pega allows you to unit test many of the rules used to configure application behavior.

Perform the following tasks to successfully unit test a rule:

- Open the Run Rule window from the rule form.

- Initialize the rule with test data.

- View the result returned by the rule.

Open the Run Rule window from the rule form

To unit test a rule, open the rule form and select **Actions > Run**. For some rule types, such as binary file rules, Pega does not provide a option for unit testing. If the rule cannot be unit tested, the Run entry does not appear in the **Actions** menu.

If the rule operates on data in memory, Pega displays the Run Rule window. For example, declare expressions, decision rules, and UI rules operate on data in memory, so Pega displays the Run Rule window when you select **Actions > Run**.

Note: Report definitions run against the contents of a database table, rather than memory. As a result, Pega skips the Run Rule window and returns the report data when you select **Actions > Run**.

Use the top part of the window to specify a page of test data for the rule. The bottom part displays the result returned by the rule.
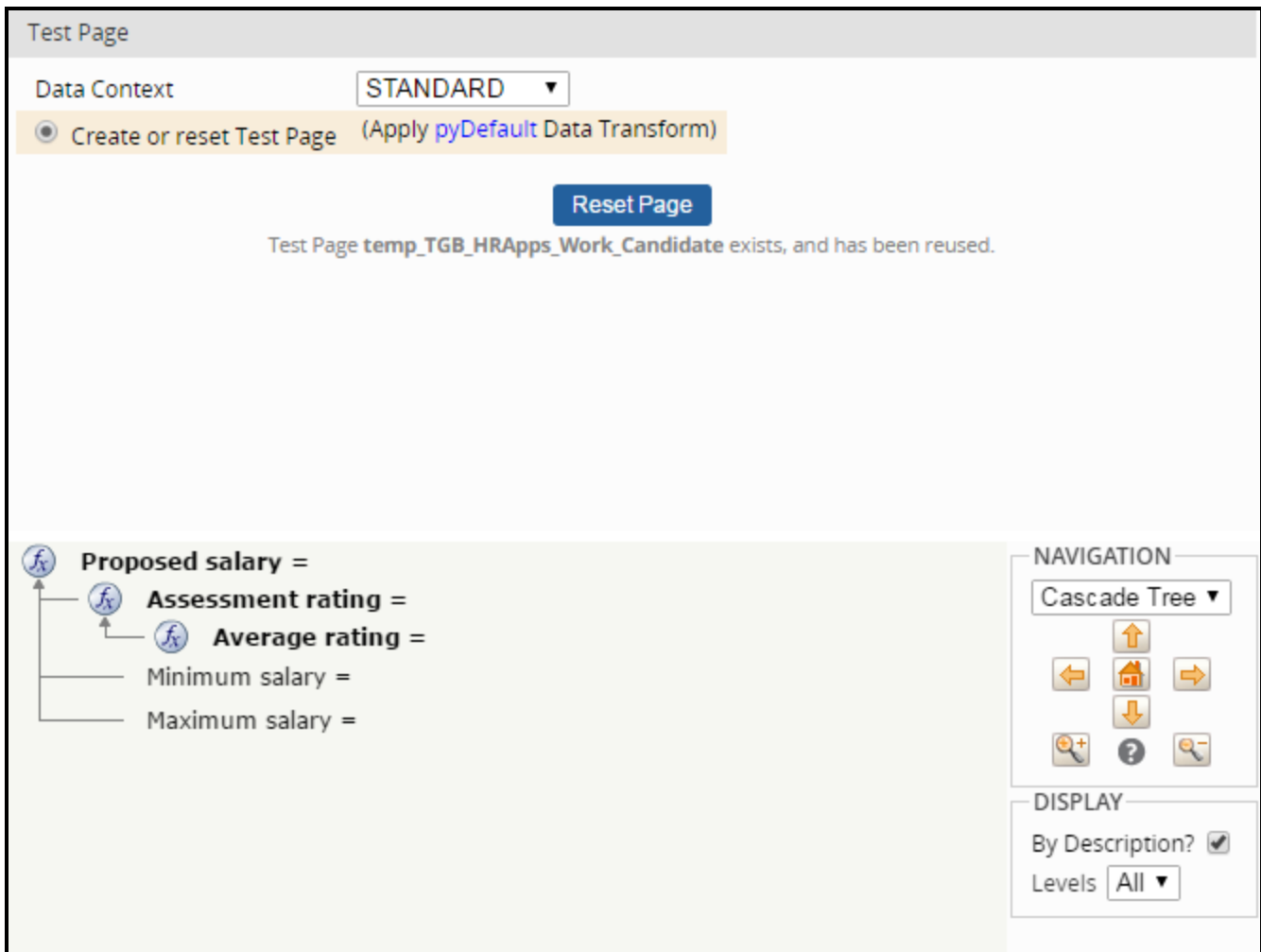
The Run Rule window applies rule resolution when testing a rule. If the application contains a higher version of the rule, Pega runs the higher version of the rule rather than the version you selected.

Initialize the rule with test data

Use the Test Page section of the Run Rule window to identify a page of data used to unit test the rule. This page is the **test page**. The Run Rule window creates the test page on the clipboard. The name of the test page is the same as the Applies to class of the rule, with the prefix "temp_" applied. This isolates the test data from other pages in memory.

When you unit test a rule, determine how you want to source data for the test page. The window provides several options to add data to the test page, depending on the type of rule tested.

Typically, you can create a page of test data using a data transform. By default, Pega applies the *pyDefault* data transform to populate the test page with data. Alternately, you can select another data transform to apply. For example, to unit test a decision table, you can create a data transform to provide values for the properties evaluated by the table, rather than manually enter values when you run the rule.

You can also copy a page of the appropriate class already on the clipboard. For example, to test a decision table used by a case type, you can create a case and copy data from pyWorkPage.

©2017 Pegasystems Inc.

To test a circumstanced rule, populate the test page to ensure that the circumstance conditions are correct for the rule. Otherwise, the base (unqualified) rule runs instead of the one you intend to test.

KNOWLEDGE CHECK

ANSWER

The purpose of a test page is to _____.

store data to use when unit testing a rule

View the result returned by the rule

Once you select a method to populate the test page, click **Reset Page**. This populates the page with data and runs the rule to return a result.

Test Page

Data Context	TABTHREAD3 ▼

◉ Copy existing Page   pyWorkPage ▼

○ Create or reset Test Page   (Apply pyDefault Data Transform)

Reset Page

Page **pyWorkPage** has been copied to Test Page **temp_TGB_HRApps_Work_Candidate**.

fx **Proposed salary =120000**
    fx **Assessment rating =**
        fx **Average rating =**
    Minimum salary =120,000
    Maximum salary =150,000

NAVIGATION

Cascade Tree ▼

DISPLAY

By Description? ☑

Levels All ▼

If the rule accepts inputs, you can enter values to test. The values you enter in this section of the window override the values on the test page. For example:

- For a declare expression, assign values to each input property and view the calculated result. As you add values, Pega updates the result automatically.
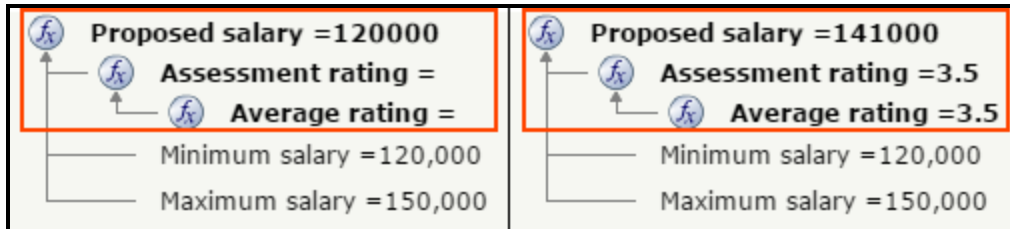
- For a decision table or decision tree, assign values to each input property and click **Run Again**. The window displays the result of the decision table or tree, and provides a link to the rule. Click the link to view the table row or tree branch that returned the result.
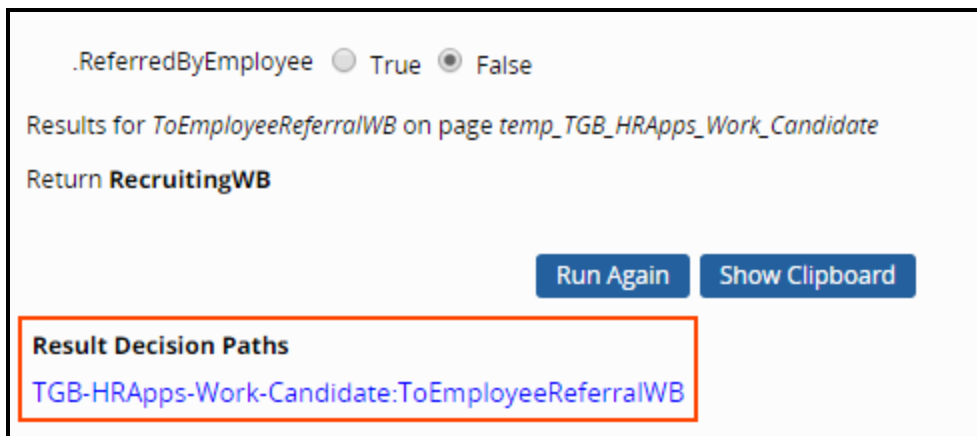
You can also review the result of the unit test by viewing the output of the test with the Clipboard tool. The Run Rule window generates several pages on the clipboard. These pages provide information about how the rule was tested:

- **RuleToRun** is the clipboard representation of the rule that you tested. If more than one version of the rule is present in the application, view this page to identify which version of the rule was tested.

- **runRulePage** holds the output from the rule. To view the result of rule execution, open the Clipboard and examine this page.

- A **temp_** page was created or copied when you tested the rule. The names of these pages begin with the string "temp_". View this page to examine the data used to initialize the rule for the unit test.

- **pySimulationDataPage** is used when testing service rules. View this page to examine information about the simulated request and response.

Analyzing application performance

Introduction to analyzing application performance

Pega provides tools to evaluate application performance. Performance statistics help you distinguish between performance issues that arise in the Pega 7 server, the PegaRULES database, or external systems called by the workflow. In this lesson, you learn how to use the Performance Analyzer, Performance Profiler and Database Trace to evaluate performance.

After this lesson, you should be able to:

- Describe the importance of performance testing
- Describe the purpose of the Performance Analyzer (PAL)
- Describe the purpose of the Database Trace
- Describe the purpose of the Performance Profiler

Performance testing

Performance is an important aspect of any system. User satisfaction and business value are often measured by the perceived quickness of the system. Pega provides a full suite of tools to monitor and address performance. You use these tools during development to ensure that your configuration performs optimally.

The Performance landing page provides access to the three performance tools: Performance Analyzer (PAL), Database Trace, and Performance Profiler. These tools are also available by clicking Performance tool in the toolbar, and they are described in detail in the Help files. Use the performance tools to collect performance statistics. Performance statistics can help you distinguish between performance issues that arise in the Pega 7 Platform server, the database, or external systems called. In all cases, the statistics can help you determine how to improve performance.

Performance Analyzer (PAL)

The Pega 7 Platform always accumulates cumulative resource statistics. Use the Performance Analyzer (PAL) to understand the system resources consumed by processing a single requestor session. PAL works on existing data, and this means that it does not degrade processing.

Database Trace

The Database Trace tool is useful in tuning the application in case of any database performance issues. Run Database Trace if PAL readings indicate performance issues in the database operations. Database Trace can trace all the SQL operations like queries or commits that are performed.

Performance Profiler

Use the Profiler to obtain a detailed trace of performance information about the execution of activities, when condition rules, and data transforms executed by your requestor session. The Profiler traces every execution (in all Threads) of rules of these three types in all rulesets. The Performance Profiler should be run in conjunction with Performance Analyzer to narrow down the specific step (Performance Profiler) of the cause (Performance Analyzer).

Additional information on these tools can be found in the Help file.

KNOWLEDGE CHECK

ANSWER	Which performance tool is used to display resource statistics recorded by Pega 7?
Performance Analyzer (PAL)	

How to analyze performance with the Performance Analyzer

The Performance Analyzer (PAL) provides a view to all the performance statistics that Pega Platform captures. You can use PAL to understand the system resources consumed by processing a single requestor session.

PAL is available on the Performance landing page (**Designer Studio > System > Performance**) or from the Performance tool in the toolbar.

Measuring performance

The first step to measuring your application performance is to take measurements. You start by clicking **Reset Data** to clear any data in the tool. Since the system is constantly monitoring performance, you are eliminating any previously recorded entries from your results by resetting data.

You have two options for adding a reading: **Add Reading** and **Add Reading with Clipboard Size**. The only difference between the two readings is the addition of the clipboard size, which takes extra time to calculate.

When adding a reading, the best practice is to define points that identify what occurred during that reading. For example, use one reading per flow action or screen render, depending on what process you are measuring.

Click **Save Data** to download the results to an Excel file.

Analyze performance data

The **INIT** row shows the totals when this Performance display first appeared. Each reading added is shown as a **DELTA** — this indicates the change from a previous reading. The **FULL** reading is the total sum of all the statistics from the last time the data was reset.

In the readings displayed, you can see the top delta has a reading that shows 1.61 for RA Elapsed. All values are in seconds. **RA Elapsed** represents the time spent in rule assembly. These results can skew performance readings as rule assembly, also known as first use assembly or FUA, is expensive but only occurs once. This is evidenced by the results you see here. The total elapsed time was 2.82s, and 1.61s of that time was spent in rule assembly. If you did not have the additional 1.61s, the total time would be less than half the measured number. FUA also affects the other readings such as the total rules executed, the reads from the database, and various I/O counts.

Add Reading		**Add Reading with Clipboard Size**								
FULL	9	9	2.82	1.61	0.29	0.16	0.00	0.55	1.78	0.00	0.00

	Int #	Int Count	Total Elapsed	RA Elapsed	Rule I/O Elapsed	RDB I/O Elapsed	Connect Elapsed	Other I/O Elapsed	Total CPU	RA CPU	Rule CPU	Other CPU
INIT	9		9	2.82	1.61	0.29	0.16	0.00	0.55	1.78	0.00	0.00	0.00

Reset Data　　　Save Data

0.00	6	59	2,701	62	64	0	40	1		653,619

RA Count	Rule Count	Total Rules Used	Activity Count	RDB I/O Count	Connect Count	Other I/O Count	Alert Count	Total Bytes
6	59	2,701	62	64	0	40	1	653,619

To obtain results not affected by FUA, you should run through the process once to ensure all rules have been assembled before taking any measurements. That was not done here in order to demonstrate the impact this has on performance readings.

Clicking **INIT**, **DELTA**, or **FULL** displays more details about the reading. Many different results are available to you for analyzing the performance.

Snapshot

	Int #	Int Count	Total Elapsed	RA Elapsed	Rule I/O Elapsed	RDB I/O Elapsed	Connect Elapsed	Other I/O Elapsed	Total CPU	RA CPU
FULL	9	9	2.82	1.61	0.29	0.16	0.00	0.55	1.78	0.00

Elapsed Time Detail For This Requestor (seconds)

2.82 Total Elapsed time for the reading
0.55 Elapsed time accessing non-rule-resolved instances from database
0.29 Elapsed time retrieving rule-resolved Rules from the database
0.00 Elapsed time executing Declarative Rules
0.00 Total Elapsed time building Declarative Networks
0.00 Elapsed time constructing Declarative Networks
0.00 Elapsed time retrieving Declarative Rules
0.00 Elapsed time retrieving non-Rule database lists
0.00 Elapsed time processing Storage Streams for non-Rule database lists
0.01 Elapsed time retrieving Rule database lists
0.00 Elapsed time processing Storage Streams for Rule database lists
0.01 Elapsed time inferring the Java
0.13 Elapsed time inferring the Java (High Level)
0.01 Elapsed time performing Rule assembly
1.61 Elapsed time performing Rule assembly (High Level)
0.00 Elapsed time spent determining if existing Rule assembly cache entry is applicable to user
0.47 Elapsed time generating Java
0.46 Elapsed time compiling Rules

Results have no magic number. A result of 10 minutes may be acceptable in one situation, anything over 100 milliseconds is considered too slow in another. You must work with the lead system architect and the business to determine an acceptable result for each step of the process.
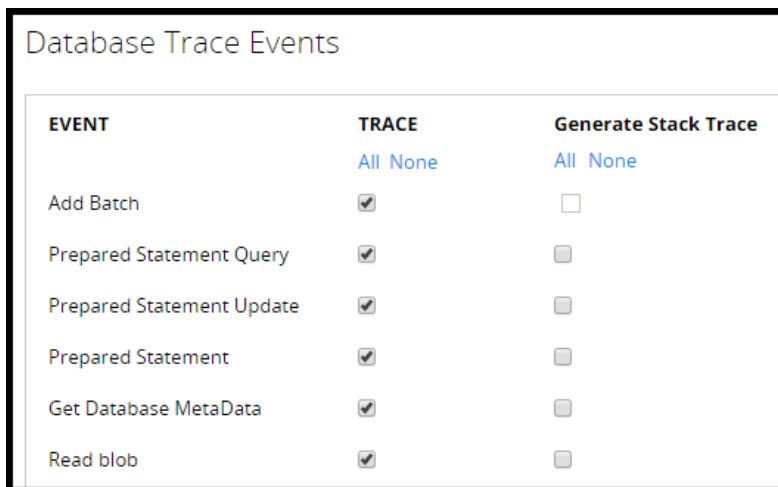
How to analyze application performance with Database Trace

The Database Trace produces a text file containing the SQL statements, rule cache hit statistics, timings, and other data that reflect your requestor session's interactions with the Pega 7 Platform database or other relational databases. Familiarity with SQL is not required to interpret the output.

The Database Trace is available on the Performance landing page (**Designer Studio > System > Performance > Database Trace**) or from the Performance tool in the toolbar.

Configuring the Trace settings

Click **Trace Options** to open the settings window. The settings window lists all possible events to trace. If an event does not apply to a situation, it should be removed from the list to streamline the results. You also have the option to generate a stack trace. Generating the stack trace is an expensive process and should only be used when required.

Taking readings

Click the green **Play** button to start Database Trace. Click the red **Stop** button after you have performed the steps you want to trace. After stopping the tool, the table is updated with the results for all the threads it traced.

You need to identify the thread corresponding to the process where you performed your work. When in doubt, look for the largest size — it is most likely the one in which you performed your work.

©2017 Pegasystems Inc.

Click the **Download** icon to save the results in a tab delimited file format so they can be opened using any spreadsheet program, such as Excel, to review the Database Trace readings. The following image shows an example of the file to review.

@BASECLASS GETWC activateCc null		Connection obtained from JDBC Connection Pool for database pegarules and data source java:comp/env/jdbc/PegaRULES	
@BASECLASS GETWC threadCor null		Connection obtained from the thread connection store for database pegarules	
@BASECLASS GETWC prepared! null			select pzPVStream from pegarules.pr4_rule_sysgen where pzInsKey = ?
@BASECLASS GETWC listResulti null	0	RDB operation returned result set with 0 rows	select pzPVStream from pegarules.pr4_rule_sysgen where pzInsKey = ?
@BASECLASS GETWC packageR null		list	select pzPVStream from pegarules.pr4_rule_sysgen where pzInsKey = ?
@BASECLASS GETWC deactivate null		Connection returned to JDBC Connection Pool for database pegarules and data source java:comp/env/jdbc/PegaRULES	
@BASECLASS GETWC assignToT null		Connection assigned to Requestor H9D49B9DD05BD5A1F9EFFE63116C35B33	
@BASECLASS GETWC assignToT null		Connection assigned to Requestor H9D49B9DD05BD5A1F9EFFE63116C35B33	

How to analyze application performance with Performance Profiler

The Performance Profiler is useful when determining which part of the process might be having performance issues, or identifying the particular step of a data transform or activity that might have a performance issue.
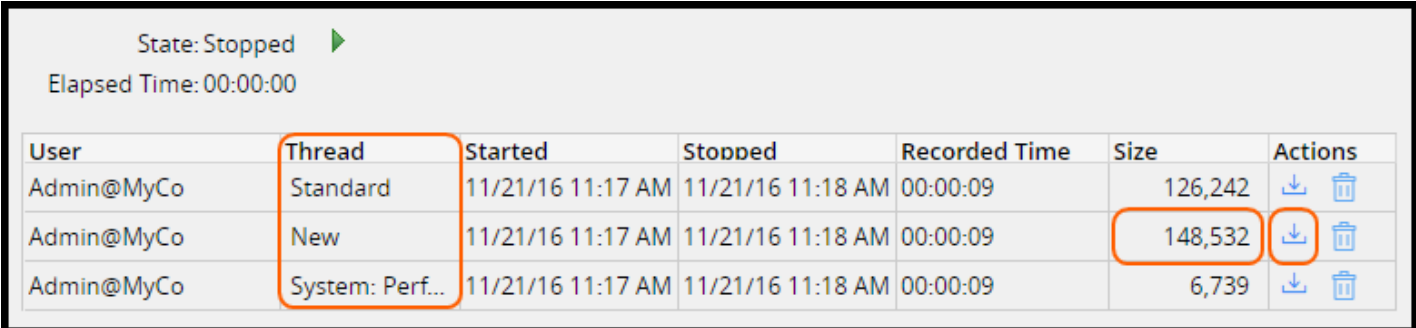
When you use the Performance Profiler, you first record readings of the application's performance. Then you analyze the readings to identify problems.

The Performance Profiler is available on the Performance landing page (**Designer Studio > System > Performance > Performance Profiler**) or from the Performance tool in the toolbar.

Once you locate the Performance Profiler, click the green **Play** button to start recording the steps you want to trace in your application. Click the red **Stop** button after you have performed the steps. After stopping the tool, the table is updated with the results for all the threads it traced.

Note: The Performance Profiler requires substantial processing overhead. Disable the Performance Profiler as soon as your data collection is complete.

When you review the trace log, you need to identify the thread corresponding to the process where you performed your work. When in doubt, look for the largest size. The largest thread is most likely the one were you performed your work.

Click the **Download** icon to save the results in a comma-separated-value file format. The results can then be opened using any spreadsheet program, such as Excel, to then review the Performance Profiler readings.

PegaRULES Application Trace										
System Name: pega										
System Node: 10.61.9.123										
System Port: 9080										
User: Admin@MyCo										
Requestor ID: H9D49B9DD05BD5A1F9EFFE63116C35B33										
Sequence	Interactio	Activity	Caller	Step	Method Name or When	Inlined?	Total CPU	CPU Time	Total Wall	Wall Time
2	10	RULE-OBJ-ACTIVITY CC	W		TRUE	TRUE	0	0	0	0
1	10	RULE-OBJ-ACTIVITY CC		1	Property-Set		0	0	0.001	0.001
3	10	RULE-OBJ-ACTIVITY CC		2	Java		0	0	0.001	0.001
5	10	RULE-OBJ-ACTIVITY CC	W		TRUE	TRUE	0	0	0	0
4	10	RULE-OBJ-ACTIVITY CC		3	Property-Set		0	0	0	0
7	10	RULE-OBJ-WHEN @BA	W		FALSE	FALSE	0	0	0	0
9	10	RULE-OBJ-A	RULE-OBJ-	1	Property-Set		0	0	0	0
11	10	RULE-OBJ-A	RULE-OBJ-	W	FALSE	TRUE	0	0	0	0
12	10	RULE-OBJ-A	RULE-OBJ-	W	FALSE	TRUE	0	0	0	0
10	10	RULE-OBJ-A	RULE-OBJ-	2	Property-Set		0	0	0	0
8	10	RULE-OBJ-ACTIVITY CC		5	Call setUserType	0	0	0	0

See [help](#) for a complete description of the data included in the results file.

APPLICATION ADMINISTRATION

Securing an application

Introduction to securing an application

Privileges assigned in an access group restrict access to application functionality.

In this lesson, you learn how to authorize users to define roles and assign privileges to members of an access group.

After this lesson, you should be able to:

- Explain how user roles relate to access groups
- Explain how privileges authorize users to perform specific tasks
- Create access groups and users
- Explain the role of the Access Manager in securing an application
- Configure access control with Access Manager
- Secure a workbasket
- Secure an attachment
- Secure a specific rule
- Require a privilege to perform a flow action

Access control

Each user of an application has a defined role in processing cases. Some users can only create cases, while other users may be responsible for reviewing cases and determining case outcomes. Ensuring that the correct user performs a task or action is **access control**.

Access control depends on two factors: authentication and authorization. **Authentication** confirms the identity of a user and verifies that the user is allowed access to an application. **Authorization** determines what data the user can view and what actions the user can perform.

KNOWLEDGE CHECK

⌄ ANSWER   _____ verifies a user's identity and access to an application.
Authentication

KNOWLEDGE CHECK

⌄ ANSWER   _____ determines the actions the user can perform in the application.
Authorization

In Pega, the records for the operator ID, access group, and application allow authentication of a user. For example, an access group named *PurchaseRequest:Managers* indicates that a user is a manager for a purchase request application. An access group named *TimeTracker:Users* indicates that a user is a case worker for a time-tracking application.

A user can belong to multiple access groups, but only one access group is active at a single time. When a user signs in, Pega identifies the default access group and opens the corresponding application in the specified portal.

Note: When users switch between applications, Pega changes the active access group and resets the user session. This automatically closes any open forms and discards unsaved work.

Authorization consists of identifying the types of users who access the application and determining the actions those users can perform.

Access roles

An **access role** categorizes users according to their job function. Each access group identifies one or more access roles. Each access role represents how a set of users interacts with an application to create and process cases.

Most applications allow one group of users to create and process cases, and a second group of users to approve or reject those cases. For example, in an application for managing purchase requests, any user can submit a purchase request, but only department managers can approve purchase requests. Each group of users performs a specific role in processing and resolving the case.

For example, in an expense reporting application you want to allow employees to submit expense reports, but not run reports. You define a role for employees, named *Submitter*, that allows users to submit expense reports. Remember, each access role describes the capabilities of a specific set of users.

KNOWLEDGE CHECK

What is the purpose of an access role?

The purpose of an access role is to identify the actions that a group of users can perform.

Privileges

A **privilege** is an authorization to perform an action. You grant privileges to an access role to deny, allow, or conditionally allow actions to users.
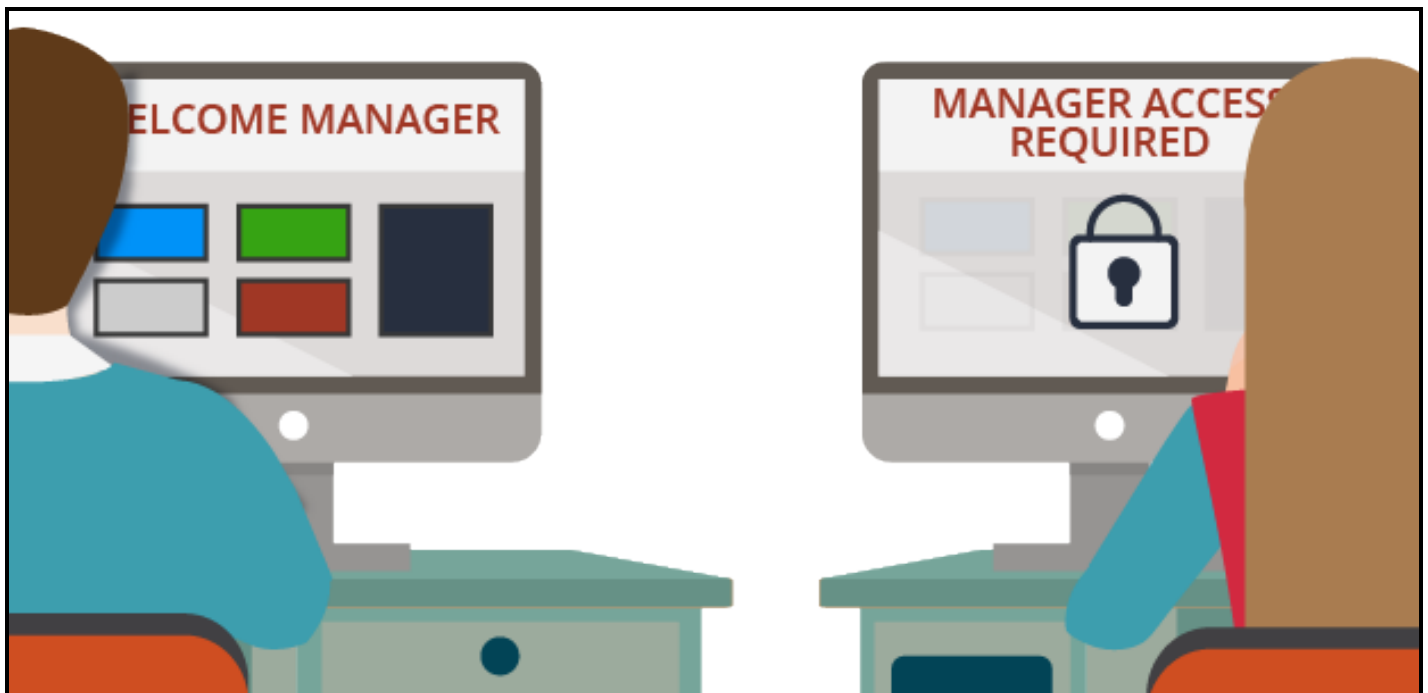
For example, consider a flow action to set a salary increase for an employee as part of their annual review. The process for employee reviews routes all assignments to calculate a raise for an employee to a common workbasket accessible by all employees in the Human Resources (HR) department. However, you want to only allow HR partners assigned to a specific department to perform assignments for employees in that department. So only an HR partner assigned to the Engineering department can determine raises for employees in the Engineering department.

In this situation, applying the privilege conditionally allows you to test whether the HR partner is assigned to the employee's department. If true, the HR partner can perform the action. Otherwise, Pega denies the action to the user.

KNOWLEDGE CHECK

ANSWER   What is the role of a privilege in access control?
A privilege determines whether a user can perform a specific action. If the user has the privilege, they can perform the action.

Access control record types

When creating an application, the New Application wizard creates test users and access groups. These test users represent the basic roles involved in case processing: users to create cases, and managers to approve or reject them. For your applications, you may need to create additional roles with different access control requirements.

In Pega, you configure access control through a combination of record types: *Access Role Name*, *Access of Role to Object* (ARO), *Privilege*, *Access When*, and *Access Deny*.

Access Role Name

An **Access Role Name** record describes a specific user role in the processing of cases. Pega uses an Access Role Name record to associate an access group with a set of actions that users can perform. Access Role Name records associated with an access group are listed on the Definition tab of the access group record, in the Available roles section.

An access group can reference more than one role. When an access group lists more than one role, Pega applies the settings for all the roles listed. For example, managers not only need to approve time-off requests for their direct reports, but they also need to submit their own time-off requests for approval. The *Manager* role identifies the actions that managers perform, such as approving cases. The *User* role identifies actions that users perform, such as creating and editing cases. By listing both roles, you allow members of the Managers access group to perform actions allowed for either role.

By applying more than one role to an access group, you can design modular, reusable roles and combine those roles to meet complex security needs. If two or more roles conflict over whether to allow or deny an action, Pega applies the most permissive setting across all the applied roles.

KNOWLEDGE CHECK

An access group lists two access roles. One role denies users the ability to run reports. The other role allows users to run reports. Can members of the access group run reports?

Yes. When an access group lists more than one role, Pega applies the most permissive setting across all of the listed roles.

Access of Role to Object

You use an **Access of Role to Object** record to define the access control settings for instances of a specific class. Each *Access of Role to Object* identifies the set of actions that a role can perform on instances of the specified class. For example, an Access of Role to Object named *PurchaseRequst:Administrators* associated with a class that describes purchase request cases may allow users to open cases, but not to run reports.
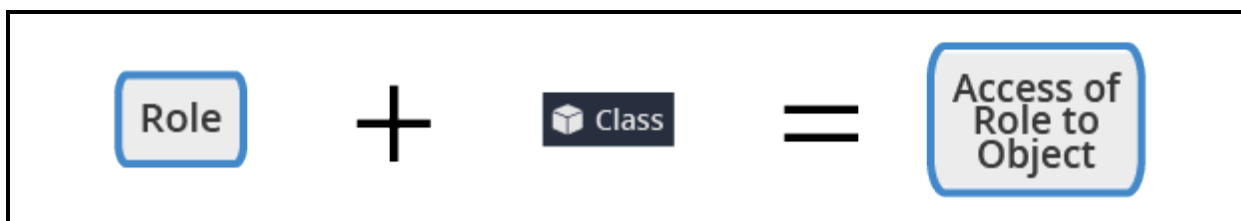
Each *Access of Role to Object* record corresponds to a unique combination of access role and class. This allows Pega to apply the permissions configured in the *Access of Role to Object* record to the users in a particular role.

> **Tip:** An *Access of Role to Object* record is sometimes informally referred to as an ARO.

KNOWLEDGE CHECK

The purpose of an *Access of Role to Object* record is _____.

to configure the access control settings for a specific set of class instances for a specific access role

Privilege

Security needs sometimes require more granular control over elements of an application. For example, you may need to prevent some users from running an approval process or performing a rejection action. To allow or deny access to individual records, such as flow actions and correspondence, you configure a privilege. For example, you add a privilege to a flow action to control which users can run the flow action.

A **privilege** is a token. If a record requires a privilege, then users can use the record only if you grant the privilege to a user role. A privilege record contains no information.
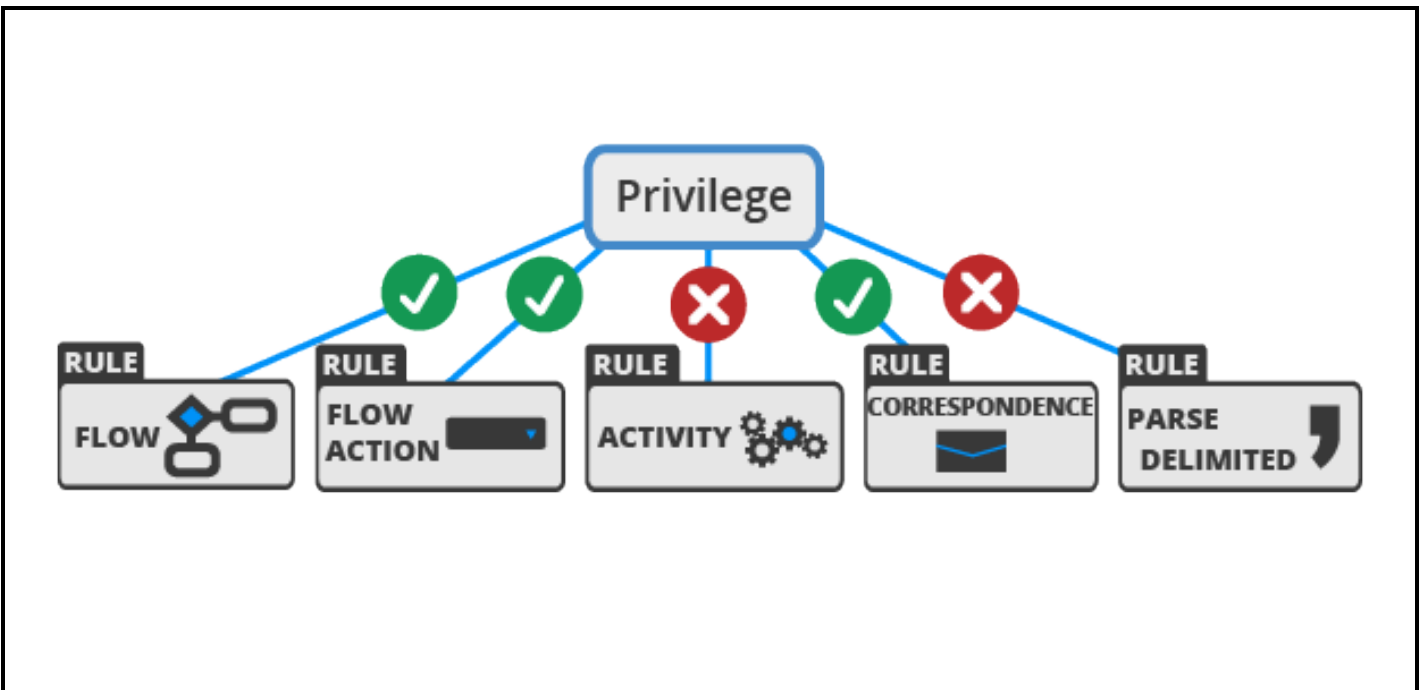
KNOWLEDGE CHECK

The purpose of a privilege record is _____.

to manage access control for an individual record, such as flow or correspondence record

Access When

Some actions are only performed under certain situations. To test whether to allow or deny an action, you configure a testable condition with an **Access When** record. At run time, Pega evaluates the condition to determine whether users can perform the action.

For example, an employee evaluation case requires employees to provide a set of goals for the upcoming year. At the end of the year, managers assess each employee's performance against the goals listed for the employee. You want to restrict an employee from editing their goals unless the evaluation case is in the Goal Setting stage. To do this, you create an *Access When* record to test the current stage. If the current stage is Goal Setting, the employee can perform the action. Otherwise, Pega denies the action to the user.

Note: Do not confuse *When* records with *Access When* records. Although both records consist of a testable condition that evaluates to a Boolean result, only *Access When* records are used for configuring access control.

KNOWLEDGE CHECK

The purpose of an Access When record is _____.

to define a testable condition, which is evaluated at run time to determine if an action is allowed or denied to a user

Access Deny

By default, Pega denies access to all instances of a class unless you explicitly grant access to users. In certain situations, government, or company regulations and policies sometimes require explicit denial of access to specific capabilities. For example, government regulations on personally identifiable information (PII) may require that a claims management application deny access to a patient's medical information for anyone not directly involved in patient care.

To explicitly deny access to an action for instances of a class, you configure an **Access Deny** record. An *Access Deny* record is similar to an *Access of Role to Object* record, but the logic is reversed. While the *Access of Role to Object* record denies an action unless explicitly allowed, the *Access Deny* record allows an action unless explicitly denied.

KNOWLEDGE CHECK

Why do you configure an *Access Deny* record, rather than denying an action on the *Access of Role to Object* record?

You configure an *Access Deny* record to explicitly deny access to an action, often to meet government or corporate regulations.

The following illustration shows the relationship between all these record types in managing access control.
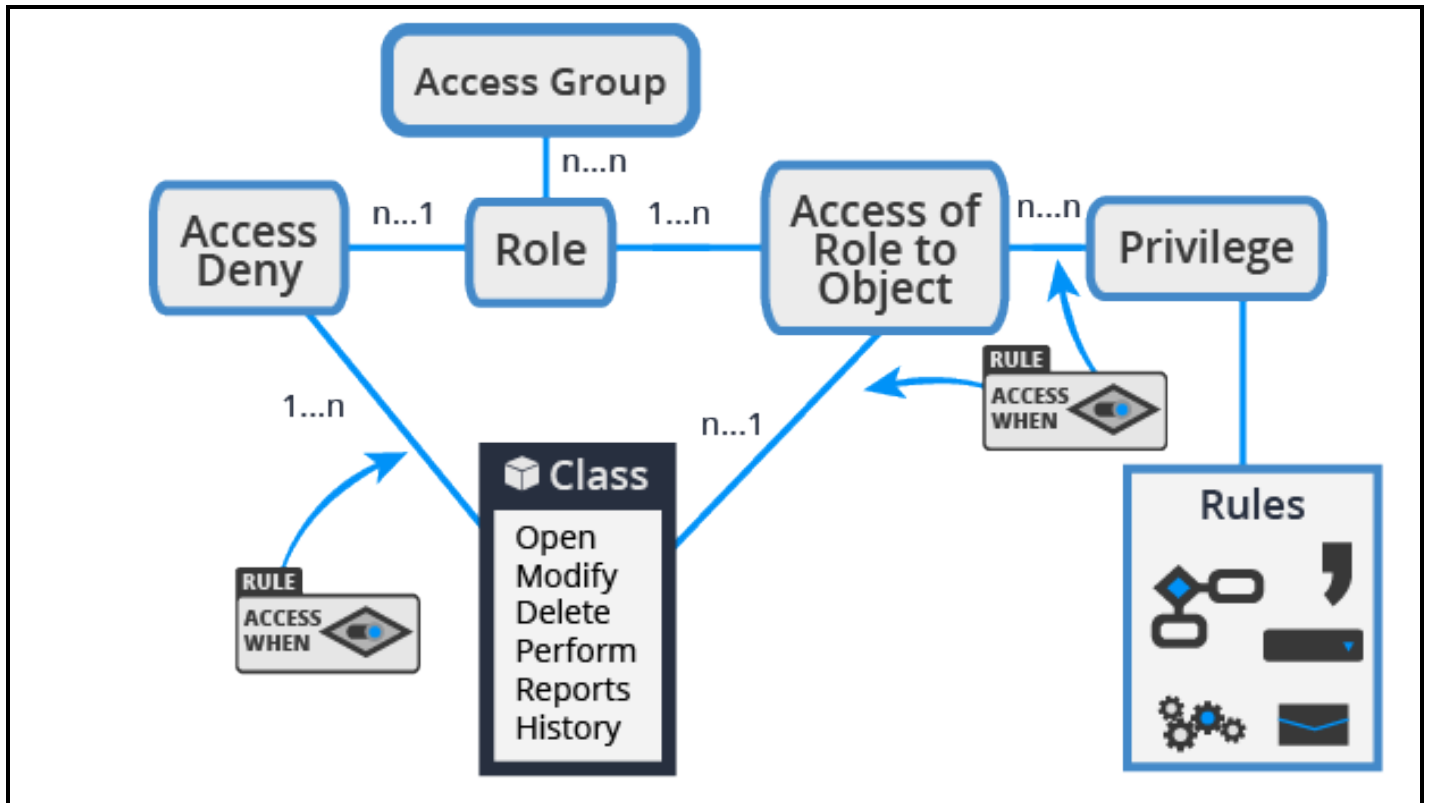
How to manage access control

To simplify the configuration of security records, Pega provides the Access Manager. The Access Manager presents you with an easy-to-use interface for managing application security. Navigate to the Designer Studio menu and select **Org & Security > Access Manager** to open the Access Manager.

Manage access control with the Access Manager

The Access Manager provides three tabs for configuring security settings in an application.

- Use the **Work & Process** tab to configure access control for instances of a specific case type.
- Use the **Tools** tab to configure access to Pega tools such as the Clipboard and Live UI for end users.
- Use the **Privileges** tab to manage access to specific records, such as flow actions and correspondence records.

The following screenshot shows the configuration for the *HRApps:Administrators* access group on the **Work & Process** tab of the Access Manager.

To configure the access control for a setting, click the icon in the column for the appropriate role. The Access Manager presents a pop-up window for you to select the level of access to grant — **Full Access**, **Conditional Access**, or **No Access**.

Note: The Access Manager displays *Conditional Access* for a case type when you select different access levels for the listed actions.

©2017 Pegasystems Inc.

Explicitly deny actions on class instances

The Access Manager manages the *Access of Role to Object* and *Access Deny* records for case types for a selected access role. When you update a setting, the Access Manager updates the appropriate record.

Note: *Access Deny* records only manage actions on class instances, such as opening cases or running reports. You cannot use an *Access Deny* record to explicitly deny access to tools or individual records.

An *Access Deny* record overrides an *Access of Role to Object* record applied to the same class and role. Below the indicator icon, the Access Manager displays a link to the *Access Deny* record to indicate the use of an *Access Deny* record. In the following example, the Access Manager indicates the use of an *Access Deny* record to explicitly deny privileges to delete cases and run reports for the *HRApps:User* role.

For a description of each action, see the Help topic Case type authorization.

Important: The View History action applies to the class group, so the setting is the same for all case types in a class group.

If necessary, the Access Manager creates *Access of Role to Object* records to reflect your configuration. For example, when you create an application, Pega creates an *Access of Role to Object* record for the class group. Any case types you create for the application inherit the settings for the class group. When you update an access control setting for a specific case type, the Access Manager creates an *Access of Role to Object* record for the corresponding class for the specified role.

Important: You must create an *Access Deny* record manually. The Access Manager creates only *Access of Role to Object* records.

Vary user access by system type

During development, you may want to configure more permissive access control to users to support debugging. However, you want to configure more restrictive access control on a production system. You can update individual *Access of Role to Object* and *Access Deny* records to automatically revoke access to actions and tools as the application advances towards production.

An *Access of Role to Object* record grants access for action on a scale of 0 to 5. A zero means the action is denied. The remaining ratings are compared to the production level value of your system. If the privilege level is equal to or greater than the production level value of the system, Pega allows the action. If not, Pega denies the action.

An *Access Deny* record denies access for an action on the same 0 to 5 scale. A zero means the action is allowed. If the privilege level is equal to or greater than that the production level value of the system, Pega denies the action. If not, Pega allows the action.

Production level values follow the software development life cycle. The greater the production level value, the closer the system is to a production environment.

Production level	Description
5	Production system
4	Preproduction system
3	Testing system
2	Development system
1	Experimental system

Note: For additional information on production level values, and how to set these values for the server, see the Help topic System form - Completing the Production tab.

When you update an access control setting in the Access Manager, Pega updates the *Access of Role to Object* or *Access Deny* record with a value of either 0 or 5. To apply a different value, click the access role name in the Access Manager to open the *Access Role Name* record, then click the access class to update the entry on the record.

Important: If you use access control levels other than 0 or 5, the Access Manager indicates the access level on the current system. For example, you set the access control level to 2 (Development system) for a role to delete instances of a case type. On a development system, the Access Manager indicates Full Access. On a production system, the Access Manager indicates No Access.

Configure conditional access

To conditionally allow access to an action, tool, or privilege, you configure an *Access When* record.

Unlike numerical access control values, *Access When* records are not tested against the production level of the system. If an *Access When* record returns a result of true, Pega grants access to the specified action regardless of the production level of the system.

To conditionally grant or deny access, click the **Indicator** icon and select **Conditional Access**, then enter the name of the *Access When* record. To create an *Access When* record, click the **Crosshair** icon to the right of the field.

You configure an *Access When* record as you do a *When* record. Create the when condition to evaluate on the **Conditions** tab of the *Access When* record form. For instructions on configuring the when condition of an *Access When* record, see the Help topic [Access When form — Completing the Conditions tab](#).

How to add roles to an access control model

When you create an application, Pega provides a default access control model to support three processing roles: users, managers, and administrators. In complex business processes, you may divide these roles into more distinct roles. For example, an accounting application may reflect the following roles for requests such as expense reports and purchase requests:

- Users — Employees who submit requests

- Managers — Employees who approve requests for direct and indirect reports

- Auditors — Employees who review requests for compliance with company policy

Each role requires that users perform different actions on a request. To ensure that users of an application only perform the actions they are allowed to perform, you extend the access control model by adding roles, then configuring the roles to allow or deny actions as appropriate.

Create a new role

You create a new role to customize access control for a specific set of users. For example, to differentiate between employees who submit accounting requests and employees who process requests, you define roles for each group of employees.

When extending the access control model with a new role, consider the following questions:

- What is the role of the user in processing a case?

- What actions do these users need to perform?

- How do these actions differ from the actions allowed for other users?

In Pega, you define an access role with an *Access Role Name* record, a label used to describe a specific set of application users with a unique job function. You apply the role to *Access of Role to Object* and *Access Deny* records to identify the actions allowed or denied to users assigned the role.

When you create a new access role, you must create a new set of *Access of Role to Object* records to associate with the role. You can clone an existing access role to create a set of *Access of Role to Object* records for a new role. Cloning an access role allows you to quickly create the needed *Access of Role to Object* records. You then update these records as needed.

Tip: When cloning an access role, identify the most similar role currently defined on the system to simplify configuration.

Note: When you create an application, Pega generates custom *Access Role Name* records by cloning default records. For example, Pega creates an *<ApplicationName>:Administrators* role for each application to manage security settings for developers. This role is a copy of the standard *PegaRULES:SysAdm4* role, which lists the default security settings for application developers.

Once you associate an *Access of Role to Object* record with an access role name, you can customize privileges from the *Access Role Name* record. When you do, Pega applies your changes to the corresponding *Access of Role to Object* record. To update an *Access of Role to Object* record, click the entry in the Access Class column to open a modal dialog that displays the contents of the *Access of Role to Object* record.

Important: The *Access Role Name* record lists both *Access of Role to Object* records and *Access Deny* records by class name. To identify the type of record listed, click the link.

After you create an *Access Role Name* record, you add the role to the appropriate access group. Pega then applies the access control settings in the role to users upon log in. If necessary, create a new access group to apply the role to the appropriate set of users.

How to manage access to individual rules

In certain cases, you need to control access to specific user actions, such as individual flow actions. For example, a bank organizes its account support agents into two roles: level one agents and level two agents. Level one agents can respond to customer complaints and open an account dispute case. Only level two agents can reverse a charge to an account to resolve an account dispute. In this situation, you need to allow level two agents to perform the flow action to reverse the charge, but deny the action to level one agents.

You create a *Privilege* record to control user access to a rule. When you add a privilege to a rule, users can access the rule only if they are assigned a role that has been granted the privilege. When a user attempts to use a rule with a privilege applied, Pega verifies whether the user is granted the privilege. If the user is granted the privilege, Pega allows the user to use the rule. If the user does not have the privilege, Pega denies the rule to the user.

To allow users to use a rule that references a privilege, you add the privilege to a user role. In the previous example, to ensure that only level two agents can perform a charge reversal, you first apply a privilege to the charge reversal flow action. Then you grant the privilege to the user role for level two agents. Pega then denies the action to level one agents.

Create a privilege record

You create *Privilege* records to configure access control for specific rules, such as flows, flow actions, and correspondence. *Privilege* records are listed in the **+Create** menu, under the **Security** category. When naming the record, identify the action that the privilege governs. This helps other system architects to select the correct privilege for other rules when configuring the access control model.

Caution: Whenever possible, save the privilege to the same class and ruleset as the rules that reference the privilege. This reduces the likelihood that Pega denies access to a rule because of a missing privilege.

Tip: When you create a privilege record, remember to complete the **History** tab. The contents of the **Full Description** and **Usage** fields are displayed in the Access Manager. Use these fields to identify the intent of the privilege and what rules require the privilege.

Require a privilege to use a rule

To add a required privilege to a rule, open the rule and list the privilege on the rule form. For most rules that support privileges, you add privileges to the **Security** tab. For flow rules, you add privileges to the **Process** tab. The following image shows a privilege applied to a flow action. At run time, Pega verifies if the user has been granted the privilege *RuleObjFlowAction:pyCascadingApprove* before allowing the user to perform the action.
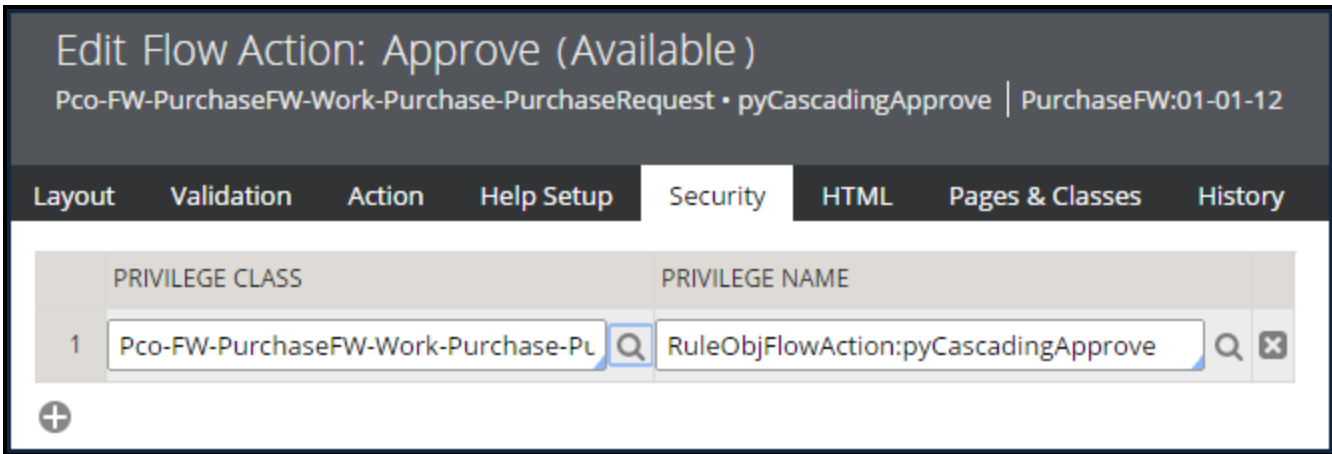
Grant a privilege to a role

You add a privilege to the user role on the **Privileges** tab of the Access Manager. In the Access Manager, you can deny, explicitly grant, or conditionally grant a privilege to users. To conditionally grant the privilege, you configure an *Access When* record to test when to grant and deny the privilege.

> **Tip:** You can also configure the access control model to grant or deny privileges according to the production level of the system. To do this, add the privilege to the role using the Access Manager, then adjust the privilege setting on the *Access of Role to Object* record for the role and class.

The following example shows a set of privileges configured for the *HRApps:Manager* role. Users with the *HRApps:Manager* role on their access group can use any rule that requires one of the listed privileges.

To view the set of privileges granted to a role, select the role and class. To add a privilege to a role, add the privilege to the table.

Tip: The Access Manager filters the class list to display either case types or data types. Under **Type of class**, select either **Case Type** or **Data Type** to switch between a list of case types and a list of data types.

By default, the Access Manager lists privileges applied to the selected class. To view inherited privileges, select **Show inherited privileges** to display the privileges inherited from parent classes. For example, you can define privileges in the class group to extend to all case types in your application. These privileges are not displayed in the Access Manager unless you select **Show inherited privileges**.

How to manage user access with access groups

In Pega, user access to an application is determined by the access group to which a user belongs. Each access group references an application version that the user can access, and the roles that the user belongs to when logged in. For each role in an application, you allow or deny actions and privileges to determine what actions users assigned to the role can and cannot perform.

When you extend the access control model for an application, you may need to create additional access groups to implement the entire model. When creating a new access group, consider how the access group differs from existing access groups.

- Identify the application and application version for the access group.

- Identify allowed portals for user interaction.

- Identify allowed access roles for group members.

- Identify the cases that users can create.

If the access group you plan to create does not provide a unique combination of these factors, re-evaluate the need for the access group.

Note: For best performance on a production environment, minimize the number of distinct access groups in use on a system.

When you create a new access group, enter the access group name in the format *ApplicationName:JobDescription*. For example, to create an access group for auditors for a Purchasing application, use the name *Purchasing:Auditors*.

Tip: Before you create a new access group, review the access groups defined for the application. Access group records are listed in the Records Explorer, under the **Security** category. If an existing access group closely matches the needed configuration, you can save time and reduce configuration errors by copying the access group and updating the configuration as needed. To create a copy of an access group, open the access group record and click **Save As**, then enter a unique name.

Identify the application and application version for the access group

Access groups specify the application that members can access. When configuring the access group, identify the application and application version that users will use. This identifies the rulesets that are added to the ruleset list for the user, which determines how users process cases.

To associate the access group with an application, enter the application name and version to the **Application** section of the **Definition** tab of the access group record. The following example shows an access group configured to allow users access to version 01.01.01 of the HRApps application.

| Definition | Advanced | Operators | History |

Application

Name* `HRApps`   Version* `01.01.01`

Note: If your application uses production rulesets, list the production rulesets used by the access group on the **Advanced** tab of the access group record.

Consider creating a new access group when migrating users to a new application version for user testing or as part of a phased roll-out of an application update. For example, when developing a new minor version of an application, you may want to provide users with early access to the new version without removing their access to the current version. This early access allows users to become familiar with changes to the application, and allows users to provide feedback during development.

Since each access group can only reference one application, you need to create an additional access group to allow users to access both application versions. You create a new access group to reference the new application version, and add the access group to the operator ID record. Users can then switch between the two access groups from the **Operator** menu.

Note: In Designer Studio, you switch between access groups from the **Application** menu.

If you need to migrate all members of the access group to a new application version, update the application version on the **Definition** tab of the access group record rather than create a new access group.

Identify allowed portals for user interaction

An access group specifies the portal or portals that members of the access group use to perform work. Access groups for end users reference one of the standard end-user portals. When identifying portals to add to an access group, select portals that align to the roles assigned to the access group. For example, if members of an access group are managers, add the *Manager* portal to the access group record. To add a portal to an access group record, list the portal in the **Available portals** section of the **Definition** tab of the access group record.

When adding portals to an access group, identify a default portal to present to users upon log in. The following example shows an access group configured to allow users access to the *Manager* and *User* portals. In this configuration, Pega presents the *Manager* portal to members of the access group after they log in.
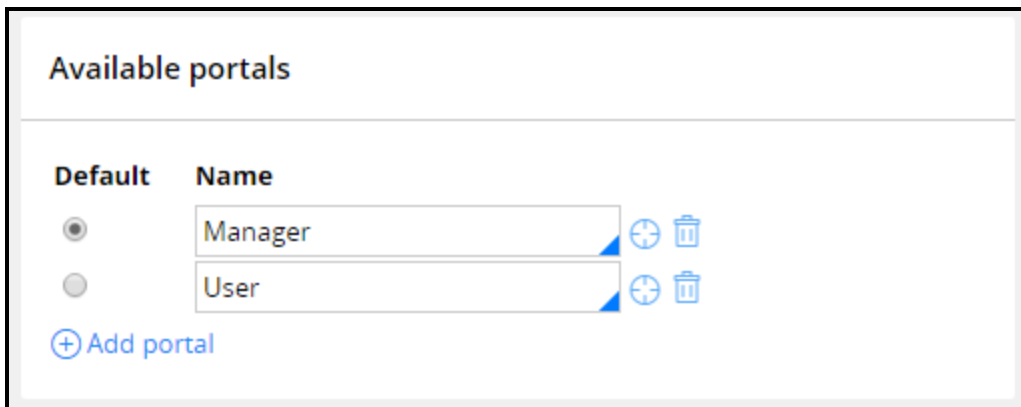
If an access group lists more than one portal, the remaining portals are available to users from the Operator menu.

Note: In Designer Studio, additional portals are listed in the **Launch** menu, rather than the operator menu. In Pega Express, additional portals are listed in the **Application view** menu.

Identify allowed access roles for group members

An access group identifies the access roles granted to members of the group. As you extend the access control model for your application, you add new roles to an access group. Adding a role to an access group grants the access control and privileges for the role to the user. To add an access role to an access group, list the Access Role Name record in the **Available roles** section of the **Definition** tab of the access group record.

When configuring an access group, identify the access roles that grant or deny privileges to match the needs of members of the access group. An access group can reference more than one access role.

For example, an application allows employees to submit expense reports for reimbursement. The application then assigns the case to the manager of the user for review. However, managers also submit expense reports. So managers perform tasks associated with both the user and manager roles. In this situation, you apply both roles to the access group for managers.

When an access group references more than one access role, Pega applies the most-permissive setting across all the access roles.

Identify the cases that users can create

An access group identifies the types of cases that members of the group can create and process. To identify the case types that members can create, you identify one or more work pools for the access group on the **Advanced** tab of the access group record.
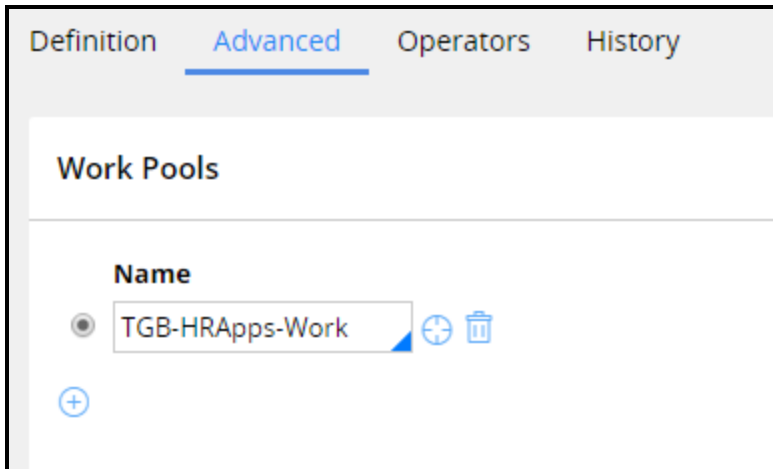
A **work pool** is a set of case types a user can work on in an application. Each work pool corresponds to a class group defined in the application or a built-on application. Users in an access group can only create cases from class groups identified as a work pool for the access group. For example, Tom is a member of the Users access group, while Mary is a member of the Managers access group. The Managers access group lists a class group that contains the *Transaction override* case type, but the Users access group does not. Mary can create a *Transaction override* case. Tom cannot create a *Transaction override* case.

Note: If your application uses case types defined only in the Framework layer, add the class group for the framework layer to the access group to allow users to create cases using case types defined in the framework application.

How to secure workbaskets and attachments

The Access Manager allows you to configure and manage access control settings for many of the elements of an application. Security for two application elements cannot be configured in the Access Manager: workbaskets and attachment categories.

Access control for workbaskets

In Pega, access control for workbaskets is role-based. By default, any user who can access a workbasket can perform an assignment in the workbasket. To control whether users can perform assignments in a specific workbasket, you apply one or more roles to it. When you apply a role to a workbasket, only users assigned to a listed role may retrieve an assignment from the workbasket.

For example, the standard workbasket *default@pega.com* lists three roles. A user must be assigned the *PegaRULES:ProArch4*, *PegaRULES:Batch*, or *PegaRULES:Basics* role to perform an assignment in the *default@pega.com* workbasket.
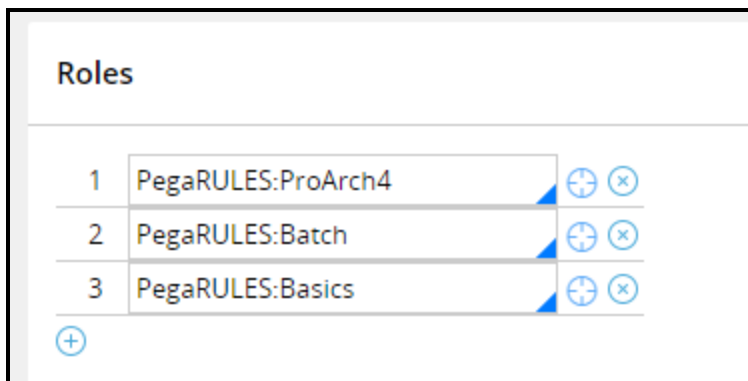
Note: Pega uses the *default@pega.com* workbasket as a last resort for routing. Pega routes assignments to the *default@pega.com* workbasket when no other more specific or local workbasket can be found.

To apply a role to a workbasket, enter or select the role in the **Roles** section on the **Workbasket** tab of the workbasket record.

Note: Do not confuse access control with skill-based routing. You prevent users from performing specific assignments through the use of skill-based routing. You assign a role to a workbasket to prevent unauthorized users from performing any assignment in the workbasket.

Access control for attachments

Pega provides two levels of access control for attachments. You apply a privilege or when condition to an attachment category to allow or deny attachment actions to users. You enable attachment level security to restrict access to the attachment itself.

Attachment category access control

Use a privilege or when condition to control access to an attachment category. When you add the privilege, select the actions to allow if the user has the privilege. For each when condition, select the actions to allow if the condition is true.

Note: Configure access control for an attachment category using a *When* rule, not an *Access When* rule.

Users can perform an action on attachments in the category if they have at least one of the required privileges, and all of the when conditions for the action are true. Consider the following configuration for an attachment category.

Access control list by privilege

Privilege	Create	Edit	View	Delete own	Delete any
DeleteOthers					x
DeleteOwn				x	
AdministerWorkbasket				x	x

Access control list by When Rule

When	Create	Edit	View	Delete own	Delete any
IsCurrentStageSubmit	x	x	x		

For this attachment category, the following conditions hold:

- Users can delete any attachment if they have either the *DeleteOthers* or *AdministerWorkbasket* privilege.
- Users can delete their own attachment if they have either the *DeleteOwn* or *AdministerWorkbasket* privilege.
- A user can create, edit, or view an attachment if the *IsCurrentStageSubmit* when rule returns a true result.

Caution: If you use a *When* rule to control access to a category, deselecting an action is not sufficient to deny access to the action. In the example above, the when rule *IsCurrentStageSubmit* is not sufficient to prohibit users from deleting an attachment if the condition returns a value of false. For more information, see the support article [Access control 'When' rule does not affect Attachment Categories](#) on the PDN.

Tip: You can use the standard when rule *Never* to create an always-false condition to deny an action to users.

Attachment-level access control

Configure attachment-level access control to allow users to determine who can access a specific attachment within the category. When users add an attachment to the category, they identify one or more work groups for which to allow access to the attachment.

To enable attachment-level access control, click the **Enable attachment-level security** check box on the **Security** tab of the Attachment category record.

Note: For more information on how attachment-level access control affects user access to attachments, see the PDN article [How to control access to categorized attachments](#).

Creating an agent for background processing

Introduction to creating agents for background processing

In this lesson, you learn how to create an agent for background processing. You also learn how to use the System Management Application (SMA) and use DB Trace to trace and troubleshoot agents.

After this lesson, you should be able to:

- Explain the purpose of agents
- Describe agent types and modes
- Describe the agent processing model
- Define agents rules and agent schedules
- Create a standard agent
- Be able to trace and troubleshoot a standard agent

Standard agents

An **agent** is a background process operating on the server that runs activities on a periodic basis. Service levels, inbound messages (Services), and data flows are examples of agents running in the background.

The assignment, queuing and management of background system operations is governed by agent configuration. Agents route work according to the rules in your application. Agents perform system tasks including:

- sending email notifications about assignments and outgoing correspondence

- generating updated indexes for the full-text search feature

- synchronizing caches across nodes in a multiple node system

The following video highlights how agents perform tasks in the background.

You can schedule agents to run at almost any time. For example, a case worker needs to check currency exchange rates for an application that processes international transactions. The database connection and currency retrieval can be done by an agent as a background task every five minutes. The case worker continues processing the work item without stopping to look up the exchange rate. The system retrieves the data in the background. If rapid updates were not needed , you could also schedule the agent to process the currency lookup task at a specific time, for example, overnight.

Agents are autonomous and asynchronous. The activities they call run individually on their own schedules. One activity does not have to finish before another one runs.

Typically, an agent can either be task-driven or schedule-driven.

Task-driven agents pick work from a queue. The queue entries are generally created during the work item processing. For example, a case generates correspondence and an agent sends it.

Schedule-driven agents directly run an activity to perform their tasks. For example, every morning an agent runs the system cleaner to clean up temporary database entries. A queue entry is not needed before the agent runs this operation.

KNOWLEDGE CHECK

ANSWER	The _____ agents pick their work from a queue.
task-driven	

How to perform background processing with an agent

When an agent starts, a master agent creates an agent thread and associates it with a requestor. A Pega master agent checks the agent's queue for entries. If there are no entries, the agent stops without running the agent. If the system finds one or more entries in the queue, the system retrieves and locks the entry, and then starts the agent activity. The activity processes the entries and stops when the maximum number of entries (that you define) have been processed, or until the queue is empty — whichever comes first. The agent starts again at the specified interval.

Note: For a description of master agents, see the Help topic [master agents](#).

You add agents to an **agents rule**. An agents rule provides a template that specifies the global settings for agents on all nodes. An agents rule is not itself an agent. Agents rules are instances of the Rule-Agent-Queue class in the SysAdmin category.

Note: A ruleset can have only one agents rule.

For each agent in the agents rule, you specify:

- Whether to enable or disable the agent

- Which activity the agent will run

- The maximum number of entries the activity will process before the agent stops

- The schedule that controls when the agent will start and run the activity. You can specify one of three schedule patterns:
  — A periodic interval measured in seconds, such as every 180 seconds.

  — A recurring time interval that defines a start time and schedule, such as 12 A.M. every Friday.

  — Once at startup, based on a specified parameter.

- The way the agent interacts with the queue when running an activity.

  You typically run the agent in **standard queue mode**. In this mode, Pega manages the agent queue functionality. The activity does not perform transaction tasks such as getting a lock on the entry or committing the changes to the database. The agent activity contains only business logic.

- Whether you enable **Auto Queue Manager (AQM)**

  AQM manages the state of each agent queue entry as it is moves from the scheduled agent queue through agent processing. AQM also controls how a queue entry is retrieved for processing. This includes locking and updating the state of the entry. For each entry, if agent processing succeeds, all entries are committed to the database. If agent processing throws an exception, then the system rolls back all entries associated with that operation. AQM then moves the entries to the Broken Process queue. You can troubleshoot entries by monitoring entries in that queue.

  As shown in the following example, an agent sends correspondence to customers approved for loans.

  1. Correspondence entries wait in the scheduled queue until the agent activity starts.

  2. AQM locks the entry and sends it to the agent activity for processing.

3. If processing is successful, the correspondence entries are sent. If processing fails, the entries are sent to the Broken Process queue for troubleshooting.
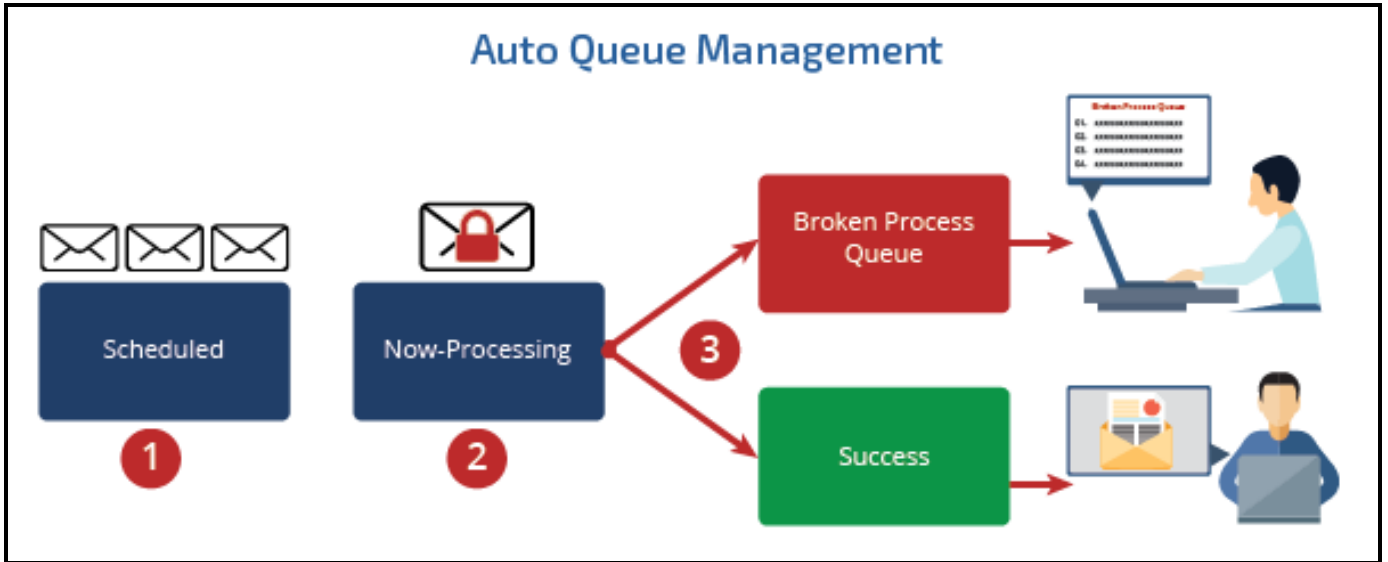
For more information about queue modes and AQM, see the PDN article How Agent Queues Work.

KNOWLEDGE CHECK

ANSWER	Under what conditions does an agent activity stop processing (assuming there are no errors)?

An agent activity stops processing when all the items in the scheduled queue have been processed or when the number of entries reaches the maximum defined in the agents rule.

Agent schedules

When you create an agents rule, a Pega master agent generates one **agent schedule** data instance for each server node running on your Pega system. Agent schedules are data instances of the Data-Agent-Queue class in the SysAdmin category. You do not create or copy agent schedules. The system must create agent schedules in order to associate them with an agents rule.

Agent schedules determine whether an agent is enabled, and on which server node that agent will run. A new agent schedule contains the schedule intervals and the enable setting you specified in the agents rule. The master agent uses the settings in the agent schedules when checking for agents. The master agent does not use the settings in the agents rule. Therefore, if necessary, make updates to the schedule or enable/disable the agent in the agent schedule. Be careful to make these changes to each agent schedule on every node. This ensures that the agent functionality is consistent across nodes in the system.

To modify the settings on an agent schedule, on the Agents rule form, select a node displayed on the **Nodes** tab to open its agent schedule.

Pega provides standard agents rules. These agents rules are in locked rulesets so you cannot modify them. To change the configuration settings for the agents listed in these rules, update the agent schedules generated from the agents rule.

> **Notes:** For information about standard agents rules, see the Help topic Atlas - Standard Agents.
>
> For more information on how agents process items, see the PDN article Overview of agent processing.

KNOWLEDGE CHECK

When the master agent checks for agents, which agent schedule settings does the master agent use?

The master agent uses the schedule settings in the agent schedule instance.

Creating agents or defining schedule intervals

For instructions on how to create an agent or how to configure agent schedules, see the PDN articles How to create an agent and Agent schedule intervals.

> **Note:** The procedures in the PDN articles were created in Pega 5.4. The styles on the rule forms have been upgraded in Pega 7. However, the fields and functions are still applicable.

How to troubleshoot issues with agents

Agents can fail to correctly process items for different reasons. For example, agents cannot locate assignments they are attempting to process, or the agents are unable to obtain a lock on an entry because they lack security access.

You have two options available when troubleshooting issues with agents. These options are available in the Pega System Management Application (SMA).

- Examine items in the Broken Process queue
- Trace running agents using the Tracer tool

Checking the Broken Process queue

If processing fails, all the changes are rolled back. If Auto Queue Management (AQM) is enabled and the system cannot commit a queue entry, AQM puts the entry into failure status and stores the entry into the Broken Process queue. From there, users who hold the *@baseclass.ReconcileBrokenQueueItems* privilege can view the issue, fix it, and resubmit or delete the broken queue entry.

Important: Enable Auto Queue Management (AQM) as a best practice. If it is disabled, when an entry is retrieved for processing, the system removes the task from the queue. If processing fails, that entry is lost and cannot be restarted in the Broken Process queue.

Tracing running agents

You can use the Tracer tool to troubleshoot agents while they are running. Tracing agents allows you to examine issues such as failed activity steps that may be preventing the agent from committing an entry. You can trace agent processing throughout the system. This enables you to troubleshoot agents even if AQM does not add failed entries in the Broken Process queue. When you trace an agent, SMA lets you control when to start the agent and when to run the tracer.

KNOWLEDGE CHECK

If AQM is enabled, what happens when agent processing fails?
All changes are rolled back and the entries are sent to the Broken Process queue.

Troubleshooting in SMA

The System Management Application (SMA) gives you access to a list of items in the Broken Process queue. SMA also gives you access to the Tracer tool so that you can monitor a running agent. Open SMA in **Designer Studio > Systems > Operations > System Management Application** .

The following demonstration first shows you how to locate and manage items in the Broken Process queue. Then the demonstration shows you how to identify issues by tracing a running agent.

Note: The style of the agent schedule rule form shown in the video has been upgraded in Pega 7. However, the fields and functions are still applicable.

You can administer broken queue items using the Broken Queue Items report. Select **Designer Studio > Process & Rules > Tools > Work Admin > Broken Queue Items** to access the report. For instructions on how to use the tool, see the PDN article [How to requeue failed agent items with the Broken Queue Items report.](#)

Note: For more information about identifying and correcting agent issues, see the PDN article [Troubleshooting agents](#).

Migrating an application

Introduction to migrating an application

An application is typically packaged and exported to move it to another system. For example, you package and export an application to migrate among development, testing, and production environments. In this lesson, you learn about the features and tools available to package, export, and import applications.

After this lesson, you should be able to:

- Describe the purpose of a product rule
- Associate data objects with a ruleset
- Describe the contents of the product rule
- Create a product rule with the Application Packaging wizard
- Generate an application archive
- Import an application archive

Product rule

Applications consist of rulesets, application data, system data, and other objects such as database schemas. As an application moves toward production, the application and its components must migrate among Pega systems. For example, after you have completed development, you migrate the application from the development environment to a QA environment. In some cases, you may want to migrate only specific application components such as updated rulesets or data objects that are included in a patch release.

Imagine that you are moving from one house to another house. You would likely create a manifest of household items you want to move. You would not include things like cabinets, plumbing, or wiring. Those items are already built into the house you are moving to. You would then load the items on the manifest into a moving van. When the van reaches its destination, the items are taken out of the van and unpacked in the house.

Like the manifest described in the previous example, you create a **product rule** that identifies the application components you want to move to a destination Pega system. A product rule lists the rulesets, data, and other objects that make up an application. The product rule usually does not include standard rulesets and data because those components are built into all Pega systems. A product rule is an instance of the *Rule-Admin-Product* class, and is sometimes referred to as a **RAP**. You can find product rules in the SysAdmin category in the Records explorer.

Like you would load the moving van, you put the contents of the product rule into a ZIP archive file (sometimes called a RAP file). The information consists entirely of XML documents in Unicode characters. You copy the archive file to the destination system and import the file's contents into the system.

You can create a product rule directly in the rule form. Alternatively, Pega provides a tool called the **Application Packaging wizard** that guides you through the creation of a product rule in a series of steps. Both approaches allow you to generate the archive file.

The following video highlights the basics of application migration.

KNOWLEDGE CHECK

In order to migrate an application between system environments, what actions must you perform?

First, identify the application components in a product rule. Then put the rule contents in an archive file. Finally, copy the archive file to the destination system and import the file.

How to create an application archive file

To create an application archive file, you must first create a product rule to identify the components you want to include in the archive file. Then, you create an archive file that contains the application components.
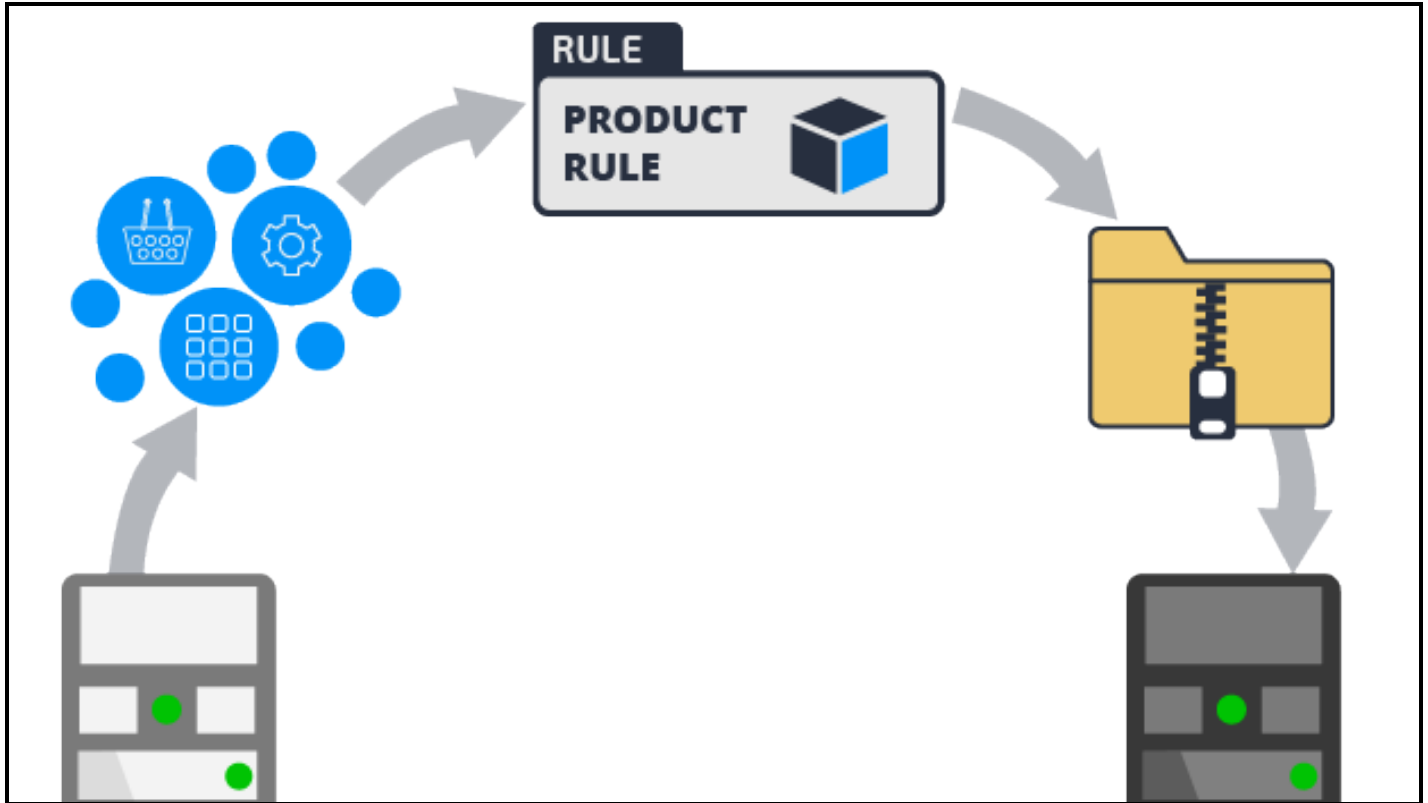
Create a product rule

You can create a product rule in two ways:

- Create the rule instance manually and add the information to the fields on the rule form. The rule form provides greater flexibility than the wizard. For example, you can set minimum or maximum ruleset versions to include in the archive rule. You can also include rulesets that are not in the application. However, manually entering information — such as selecting specific data instances — can be time-consuming and error-prone.

- Use the Application Packaging wizard — it guides you through a series of steps that populate and create a product rule. The wizard includes the rulesets in your application. Because the wizard presents an inventory of components that are in your application, selecting the components you want to include is easier and more accurate than manually entering them in the product rule. When the wizard creates the product rule, it enters your selections automatically on the form. For finer control, you can modify the product rule after you have created it in the wizard.

Create an archive file

Before you create an archive file, do the following things to ensure that the file is complete and correct:

- Associate your data records with rulesets. This ensures that all the data records required by your application are included in the archive file.

- Make sure that all rules are checked in so that the rulesets are complete and current.

- In most cases, lock the application rulesets included in the package. This helps ensure that the migrated application and its rulesets are synchronized among the source and destination systems.

- If you are exporting the application to a production system, you should merge branched rulesets and remove the branches from the application.

  Note: Do not lock delegated rulesets. Locking the rulesets prevents users from updating the rules in the destination system.

You can create an archive file using buttons on the product rule form or on the landing page that contains the wizard.

How to associate data instances with rulesets

Associating data instances to a ruleset simplifies application migration and maintenance. When you package an application for migration, you include an application's rulesets. However, a ruleset corresponds to a collection of rules, and does not include data instances such as operator IDs, access groups, database tables, and databases.

To help make packaging and migration of data instances easier, you associate data instances with rulesets. By doing this, you do not need to specify each data instance in the product rule. When you create the product rule, the system adds the data instances to the archive file automatically.

As a best practice, associate data instances with rulesets. As you create data instances of certain classes, either manually or with a wizard, the system automatically associates the instance with one of the application's rulesets.

The associated ruleset appears in the rule header. The following example provides an access group that is associated with the Purchasing ruleset. You can change the associated ruleset by clicking Edit. Update the ruleset in the **Associated RuleSet** pop-up dialog.
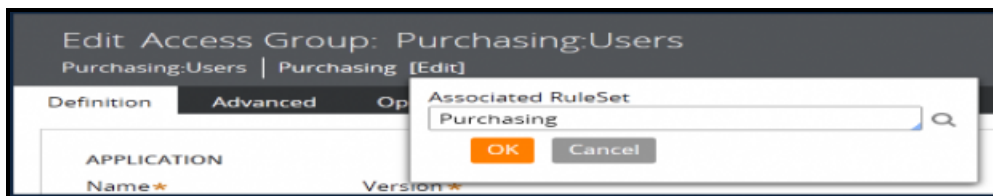
You can remove the ruleset so that the data instance is not associated with any ruleset. For example, you may not want to associate specific instances of operator IDs that you do not want to export. Removing the associated ruleset results in a guardrail warning.
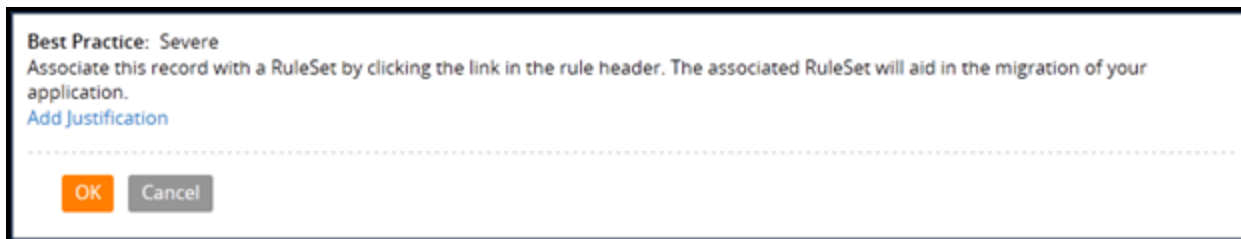
Associating a data instance with a ruleset does not affect or restrict any run-time behavior. The data instance remains available to all applications regardless of the associated ruleset.

How to configure a product rule

You define the application components you want to package in a product rule. Put the product rule in the same ruleset as the work class for the application.

Identifying the components

You specify the components you want to export on the Contents tab of a product rule. The tab includes these sections:

- Applications to include
- Rulesets to include
- Class instances to include
- Individual instances to include
- JAR files to include
- File details

For more information about the settings on the product rule Contents tab, see the help topic Product rules - Completing the Contents tab.

Applications to include

Specify the application rules that identify the rulesets and versions to include in the archive file. Using this option eliminates the need to specify the individual rulesets and versions in the **RuleSets to include** section. Select the application in the **Name** and **Version** fields. The order of applications in the array is not important.

Rulesets included in the archive must have all the prerequisite rulesets included either in the product rule or on the destination system. When the product rule is exported to an archive file, the archive contains information about all the rulesets associated with the application. This information is used by the Import wizard to warn you about missing rulesets on the destination system.

A ruleset using Application Validation (AV) mode has no prerequisites. Rules within a ruleset using AV mode can refer to rules in any rulesets in the application and its built-on application. An AV ruleset, therefore, should be exported within the context of an application.

Use the following settings to automatically include specific components or patch ruleset versions in the archive.

- Select **Include associated data** to automatically export any data instances associated with the application ruleset. Instances selected in the **Individual instances to include** section are not used. Typically, these data instances are derived from the Data- class and include operator ID, access group, database table, and database records.

- Select **Custom/Production rulesets** to automatically include the production rulesets listed on the application rule. You should select this check box if you are using delegated or localization rulesets (these are production rulesets).

- Select **Include rule history** to include the instances of History-Rule class. The instances are rows in the standard history rule table. These instances include the date, time, operator, and comments in a rule's History tab that are added when a developer checks in a rule. This information can be useful for auditing purposes when migrating applications to another development or testing environment.

- Select **Include data types** to include the instances of the custom data types (classes) that you have created for your application. For example, you may have created a Customer data type to manage customer contact information. This data type might include information such as customer's name, email, or phone number. The data types are exported even if they are not associated with a ruleset.

- Select **Delta mode** to include only the highest patch version of the application's rulesets in the archive file. For example, if your application references OrderEntry:02-03-05, the archive file produced from the product rule includes only rules in OrderEntry:02-03-05. This feature is useful when you are migrating a patch update and do not want to include the contents of all the prerequisite rulesets.

Rulesets to include

Specify rulesets that are not part of an application. The rulesets can be entered in any order. During import, the system determines a valid order to load the rules. Make sure that the prerequisites are included in the list or already exist on the target system. For example, when you include rulesets from other applications, you must verify that all the prerequisites are listed in the product rule or are already present on the target system. If the prerequisites are not present on the target system, some rules may be unavailable.

- Use the **Minimum version** and **Maximum version** fields to define a range within a major version that you want to include in the product. Leave both fields blank to include all versions.

- Select **Exclude non-versioned rules** to exclude rule types that are associated with a ruleset but not with a version, such as class and application rules.

- Select the **Include associated data** check box to include data instances associated with the selected rulesets.

Class instances to include

You can include all instances from any class by entering a class name. In addition to concrete classes, you can enter an abstract class that has concrete subclasses. This section is useful for specifying data classes (derived from the Data- base class) that are required for an application to run correctly on the destination system. Remember that if you have selected the **Include associated data** check box in the **Applications to include** section or **RuleSets to include** section, all data instances associated with the rulesets are exported by default.

Be careful to note any dependencies among data instances or between data instances and your rules. During import to the destination system, data instances already present are by default not overwritten. These data instances may require adjustment after they are imported.

You can use a when rule in the **When filter** field to filter the class instances you want include in the ZIP file. The **ListView filter** column is included for backward compatibility.

If you select **Include descendants?**, all the subclass instances are included and the **ListView filter** and **When filter** values are ignored.

Use the **Exclude classes** field to enter the names of descendant classes you do not want to include. You can enter more than one class using a comma (,) to delimit the names.

Individual instances to include

Complete this optional array by listing the specific class instances that you want to include in the archive file. For example, you may want to include only specific operators or access groups in the product rule, rather than all instances.

> **Note:** If you select **Include associated data** in the **Applications to include** section or **RuleSets to include** section, your selections are ignored if the instances are associated with application rulesets.

Instances are processed in the order listed on the product rule. This is important if your application includes views that reference each other. Listing the instances in the wrong order can create dependency tree errors. Drag and drop the instances to change the order.

As a best practice, when selecting large numbers of instances, use a filter in the **Class instance to include** section.

JAR files to include

The Pega 7 Platform allows you to extend the Java code built into the Pega 7 platform with your own code. The code can be used with activities (Rule-Obj-Activity rule type), user interfaces, or system interfaces. If your application uses JAR files, complete the array in the **Jar files to include** section to include JAR files in the product archive.

Jar files to include

Search jar files

[Query jars]

	Codeset	Version	Jar file
1			

To locate available JAR files, enter a class name in the **Search jar files** field and click **Query jars**.

File details

Use this section to prepare the product rule for packaging in a ZIP archive file.

The date in the **Creation Date** field displays on the destination system before the archive file is imported. The date value entered persists even when the file is copied or renamed. Enter a text description of the contents of this file in the **Short Description** field. This value appears on the destination system before the archive file is imported.

File details

Creation date

20160726

Short description

HR 01-01-04

☐ Exclude schema changes for 5-4 compatibility?

☐ Allow unlocked ruleset versions?

[Preview product file]

[Create product file]

As a best practice, lock the ruleset versions you are including in the archive file. This helps prevent updates to the rulesets after they have been packaged in the archive file. Select **Allow unlocked ruleset versions?** if you want to include unlocked ruleset versions. For example, select the check box if you are including a delegated ruleset.

If you want to include post-import instructions such as a Read-Me text, identify an HTML rule on the **Installation** tab. The file is displayed at the end of the import operation for this product rule.

The **Preview Product File** button lets you review the contents of the product definition before creating the archive file. It is a best practice to review contents before you create the archive file to ensure that the contents are correct.

The **Create Product File** button creates an archive file on your local computer.

Exporting an application using the Application Packaging wizard

The Application Packaging wizard simplifies the process of exporting an application by guiding you through the steps to create a product rule. In each step, you add the application components you want to include. After you complete the steps, the wizard creates the product rule. You can then review the product rule contents and generate the archive file.

Follow these steps to create a product rule and generate an archive file:

1. Start the wizard by selecting **Designer Studio > Application > Distribution > Package**.

2. On the Application step, use the **Application** drop-down to select the application you want to package. The drop-down contains the applications you have access to on the current system.

3. In the **Ruleset Name** and **RuleSet version** fields, select the ruleset and unlocked version that contain the product rule. If you specify a locked ruleset, you receive an error message.

4. Optionally, click **Check to display only Application Rulesets** to display the rulesets specified in the selected application's rule form. If unchecked, only rulesets in your ruleset stack, as shown on your operator profile, are displayed.

As you proceed through the steps in the wizard, the system automatically lists the components that are available in your application with a check box. Components that are used by your application are selected by default. If you want to exclude a component, deselect the check box

5. On the Application Stack step, identify the chain of built-on applications ending with the base PegaRULES application. Depending on the applications present on the destination system, you may not need to include the entire stack in your package.
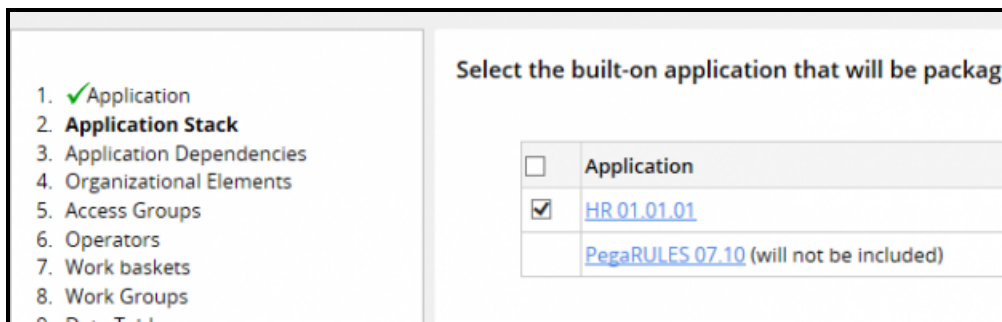
6. On the Application Dependencies step, identify the products that many Pega-supplied applications are dependent on. The Pega 7 Platform verifies that these dependencies are installed on the destination system before importing the Pega-supplied applications. If there are no required products for your Pega-supplied applications, skip this task.

7. On the Organizational Elements step, identify the organization, divisions, and units associated with your application.
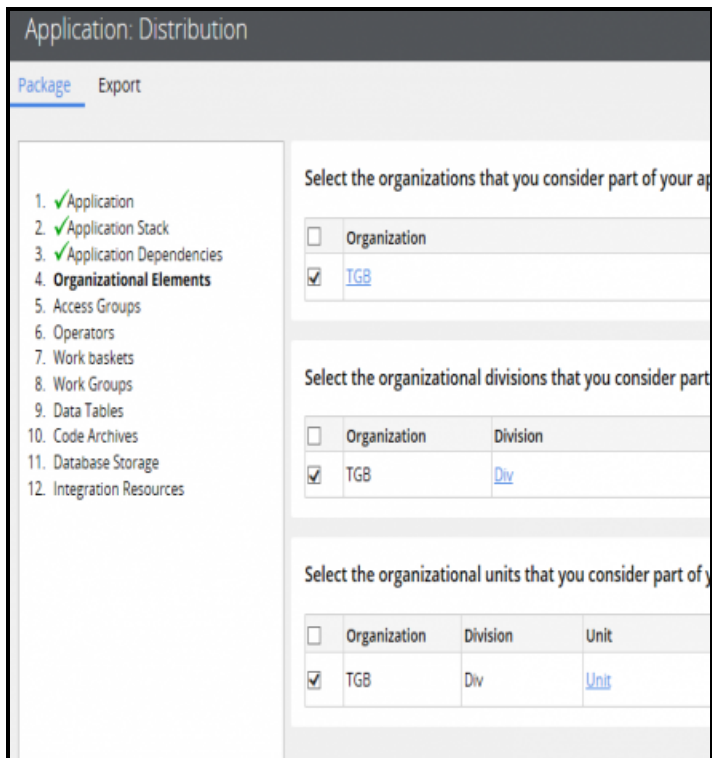
8. On the Access Groups step, identify the access groups associated with your application. If you have created access groups for the purpose of development or testing and are not part of the application, clear the check box next to the access group names to exclude them.

9. On the Operators step, identify the operators associated with the application. If you have created operators for the purpose of development or testing and are not part of the application, clear the check box next to the operator names to exclude them.

10. On the Work baskets step, identify the work baskets associated with your application.

11. On the Work Groups step, identify the work groups associated with your application.
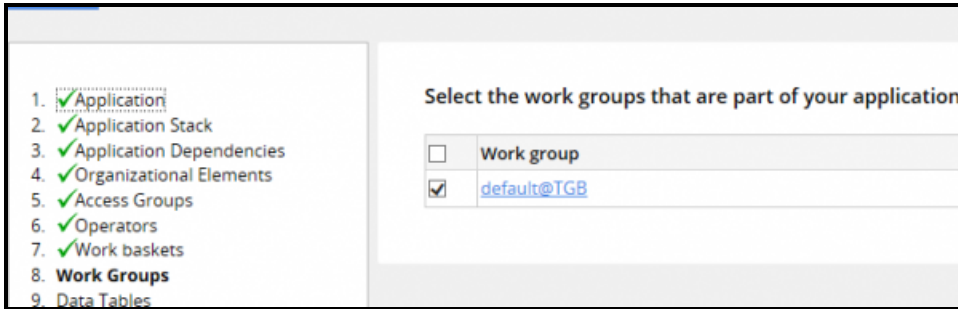
12. On the Data Tables step, identify the local data storage data types used in the application. A link named **No Connection** is displayed in the **Count** column if a connection issue arises. For example, an external data table may have more than one data class mapped to it. Click the **No Connection** link to open the rule, and click **Test Connection** to identify the issue.
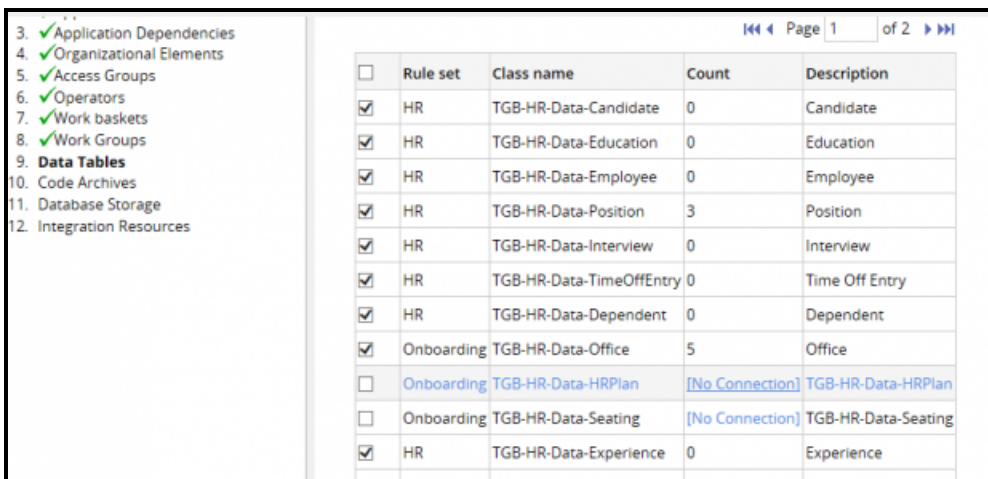
13. On the Database Storage step, identify the databases and database tables that are available to your application.
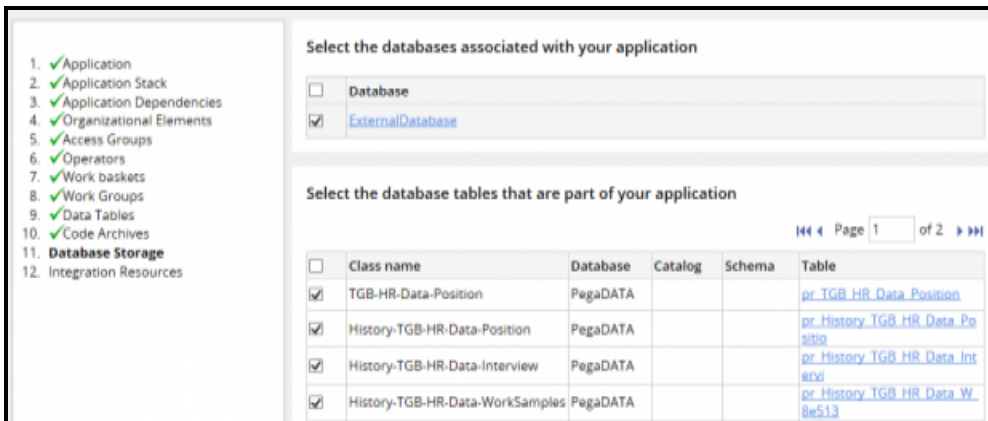
14. On the Code Archives step, identify the java code archives used by your application.

15. On the Integration Resources step, identify the integration resources available to your application. The integration instances associated with your application are automatically selected.
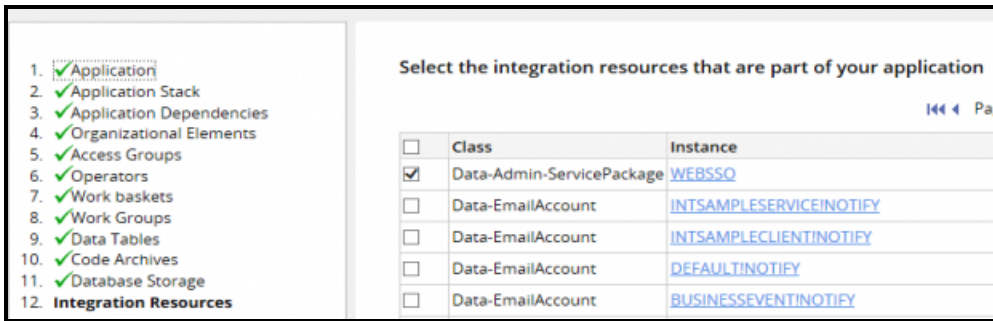
When you have completed all the steps, click **Finish**. The system creates the product rule in the ruleset you specified. The wizard names the product rule after the application you specified. The version is set to the date and time it was created.
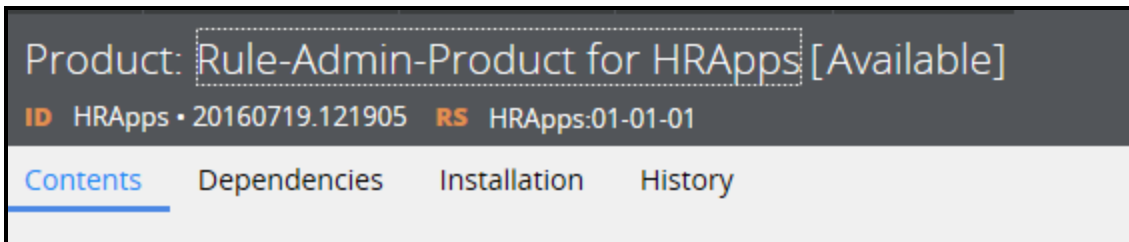
After you click **Finish**, the Package landing page displays a dialog that lets you perform various tasks.
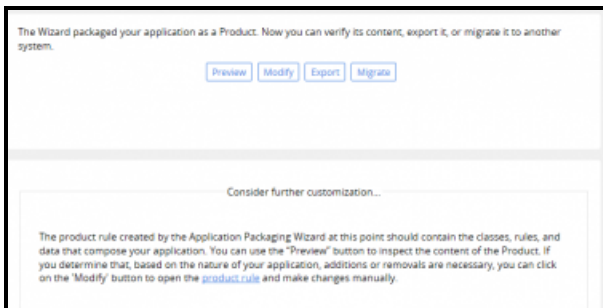
16. You can do the following:

- Click **Preview** to show the content of the ZIP file that is created by the product file.

- Click **Modify** to open the product rule so that you can manually add or remove instances or change default settings.

- Click **Migrate** to start the Product Migration wizard. The Product Migration wizard lets you automatically archive, migrate, and import a Product rule to one or more destination systems. See Pega 7 help for more information on the Product Migration wizard.

- Click **Export** to create an archive file on your local computer. The system displays a Create Product File dialog in which you enter a file name.
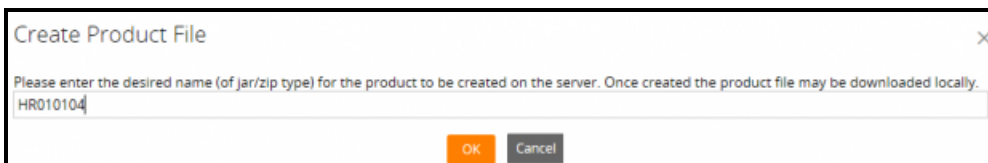
Click **OK**. A progress indicator is displayed. When the export process is complete, the indicator displays a link to the archive file.

Click the link to download the file.

Note: After you have created your product rule, you can also create the archive file using the Export gadget. The gadget is available from Designer Studio menu by selecting **Application > Distribution > Export**. In the gadget, provide the product information and click **Perform export**. You can also use the Export gadget to quickly package specific rulesets and data objects without having to create a product rule.

Updating the product rule after you use the wizard

The product rule created by the wizard uses default settings that you may have to override, depending on your requirements or selections you made in the wizard. Click **Modify** to open the rule and make your updates.

What you want to export	Where to make the update
You are exporting delegated rulesets.	In the **File details** section, select the **Allow unlocked ruleset versions?** check box. These rulesets must be unlocked so that users can access them on the destination system. In the **Applications to include** section, select the **Custom/Production rulesets** check box.
You have excluded specific associated data instances in the wizard.	In the **Applications to include** section, clear the **Include associated data items** check box. Otherwise, all associated data items are included in the archive file.
You have excluded specific data tables in the wizard.	In the **Applications to include** section, clear the **Include data types** check box. Otherwise, all data types are included in the archive file.
You do not want to include rule history instances.	In the **Applications to include** section, clear the **Include rule history** check box.

Importing an application archive

Use the Import gadget to import the contents of an archive on the destination system. You must have the *@baseclass.zipMoveImport* privilege to use the Import gadget. Rules, data, and other Pega system instances contained in the archive file are added to the rules already present in this system, optionally overwriting some existing rules.

1. On a destination system, open the Application Import wizard from **DesignerStudio > Application > Distribution > Import**.

2. Select the archive to upload. This field can be left empty if the file to import already exists on the server.

   Note: By default, uploading a file larger than 1 GB is not possible. You can increase the limit by changing a parameter value in the prconfig.xml file or the Dynamic System Setting. See the Notes section in the help topic [Import landing page](#) for more information.
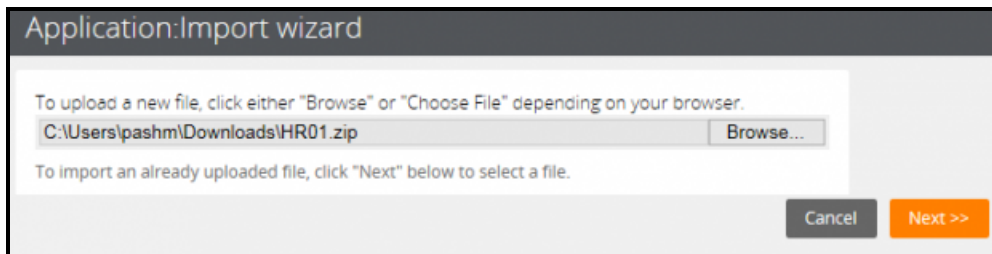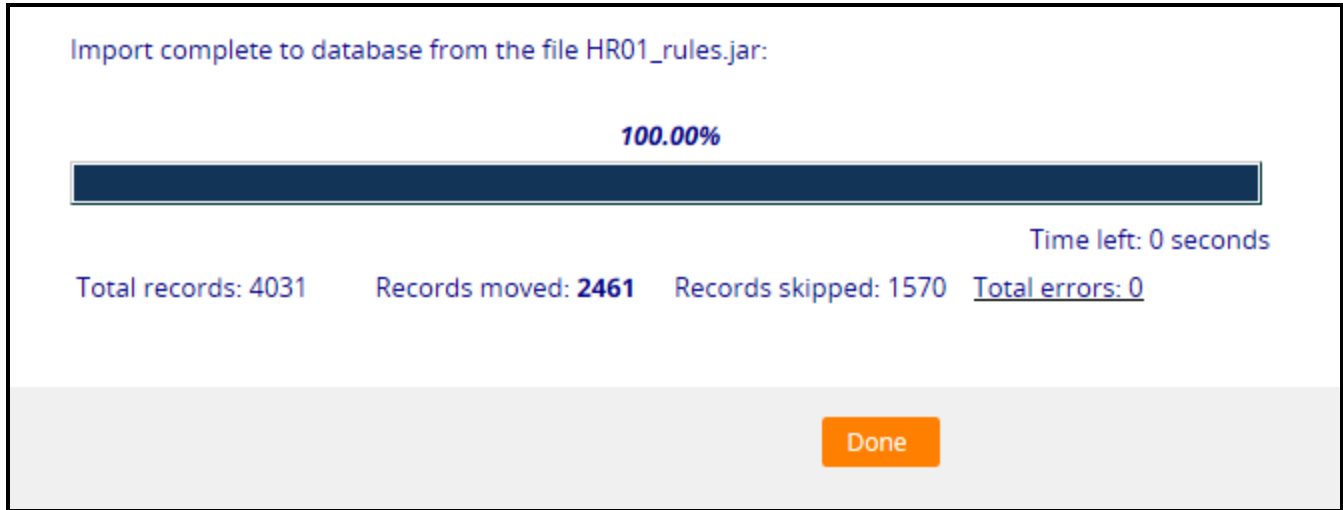
3. Click **Next**.
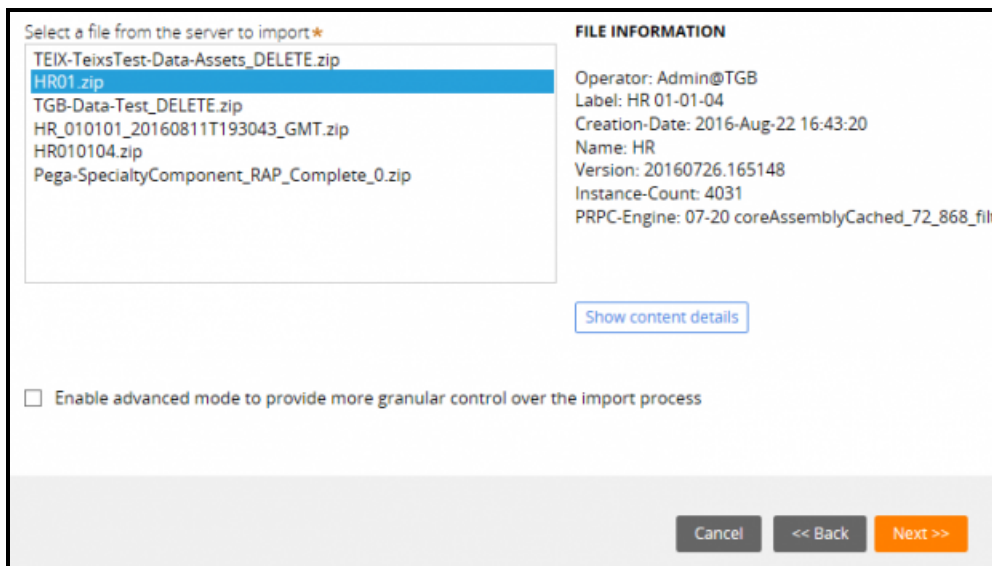
4. Optionally, click **Show Content Details** to show the details of the contents.

5. Optionally, select the **Enable advanced mode to select individual instances in the archive to provide more granular control over the import process** check box. If you do not select this option, continue to step 7.

6. Click **Next** and continue through the pages to review or edit instances that are skipped, inserted, replaced, or added. When you reach the last page, the import starts.

7. Click **Next** to start the import. The system attempts to upload the rules in an order that minimizes processing. When complete, the progress indicator shows 100.00%.

Import complete to database from the file HR01_rules.jar:

100.00%

Time left: 0 seconds

Total records: 4031   Records moved: **2461**   Records skipped: 1570   Total errors: 0

Done

8. After processing is complete, adjust access groups or application rules to provide access to the new rulesets, versions, and class groups as needed.

When imported rules are available

If you import rules in a ruleset that users already can access, the rules may begin executing immediately. These rules may execute before all the rules in the same archive have been imported. Similarly, declarative rules begin executing immediately. This means that the declarative processes might fail if the elements or properties they reference have not been uploaded yet. This needs to be planned for when an archive is imported on a system with active users.

COURSE SUMMARY

Next steps for Senior System Architects

Senior System Architect summary

Now that you have completed this course, you should be able to:

- Identify the tasks and responsibilities of the senior system architect on a Pega Implementation

- Create and extend a Pega application

- Manage rules and rulesets

- Leverage the Enterprise Class Structure (ECS) to promote rule reuse between case types and applications

- Configure roles, access groups, and privileges

- Manage data access within an application

- Create wizards to configure a sequence of assignments

- Design applications for multiple device types and screen sizes, including mobile devices

- Manage integration settings

- Incorporate next-best-action decision-making into applications

- Test your application design to analyze rule behavior and identify configuration errors

Next steps

Completion of Senior System Architect 7.2 helps to prepare students to take the Certified Senior System Architect exam. To practice taking the exam, enroll in the CSSA Practice Exam course in Pega Academy.

Certification

Become a Certified Senior System Architect

Certification Benefits
Industry Credibility

The Senior System Architect course and practice exam
prepare you to take the Certified Senior System Architect exam

Pega Discovery Network

Join the Pega Discovery Network (PDN) today. The PDN is an online community for Pega developers, customers, and partners to find everything they need to successfully implement Pega solutions.

Discover information on a variety of development topics.

Share ideas, ask questions, and participate in a variety of discussions with other members of the Pega community.

Learn at your own pace with our Pega Academy self-study courses or with a certified Pega trainer.

Have a product question? Want to engage with other Pega users, employees, and partners? Check out our Pega Developer Network to converse with the PDN community at pdn.pega.com

For self-study training on Pega applications, go to Pega Academy.

DECISION DESIGN

Improving the Customer Experience with Next-Best-Action

Next-Best-Action paradigm

Helping to determine the Next-Best-Action is the primary role of Pega Decision Management in customer-facing business operations.

Next-Best-Action is a proven method for improving the results of any type of customer interaction involving marketing, sales, service, retention, collections, risk, even data collection. Its goal is to determine the optimal action to take with a customer at a given moment under the circumstances. The optimal action is one that satisfies customer expectations while also meeting business objectives.

In a real-time inbound environment, Next-Best-Action is a process that begins with the customer taking some sort of action such as calling into the call center or responding to an email. The system then registers the customer's intent for this action.

Next, Pega Decision Management applies decision strategies to the customer's action/intent to create a mini business case that calculates the Next-Best-Action for the interaction. Once calculated, the decision engine communicates the Next-Best-Action to the channel in which the interaction is taking place. The Next-Best-Action could be to take no action at all, or to take a service action, or to make an offer, or any number of other things. The process begins again each time the customer responds or takes another action. This action-decision-action loop never ends for the entire lifetime of the customer relationship.
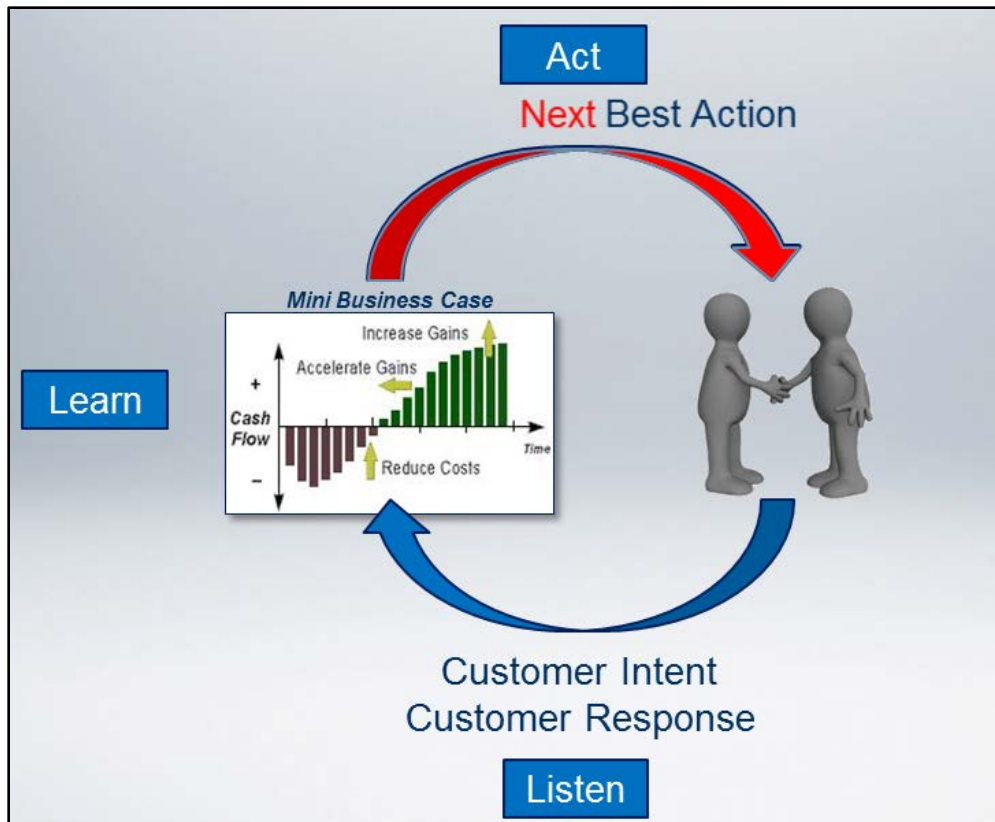
To put it in more human terms, Next-Best-Action follows a very old interaction paradigm: Listen, Learn, and Act accordingly.

To listen effectively you have to be able to remember what was said. Interaction History is what gives Pega Decision Management its long-term memory. Interaction History captures every customer response to every Next-Best-Action, even if the response is no response. Traditional business rules engines do not have an Interaction History component, so they are not capable of remembering. Pega Decision Management combines traditional business rules with predictive analytics and interaction history to determine the Next-Best-Action.

In real-time inbound environments in which an agent is involved, Pega implements Next-Best-Action via Next-Best-Action Advisor, which advises agents on how to interact with customers. Next-Best-Action Advisor is part of Pega Next-Best-Action Marketing, which applies the same principles to outbound campaigning to make all customer outreach as relevant and effective as possible.

Next-Best-Action in chess: Next-Best-Move

Another way of thinking about Next-Best-Action is to apply it to a game of chess. In chess the question would be, "What's the next best move?"

With every next move you make in chess, you would ideally consider all of the factors ABOVE. Who is your opponent and what do you know about them? Is a clock being used in this game? In other words, what are the constraints you have to work under? Also, what is your goal for this game, do you want to go for a win, or just a draw? Have you played similar opponents before and how did you do against them? What did you learn from the

last game that will help you improve your game this time? That's where analytics come in. Next-Best-Action considers all of these questions and more before deciding on an action, and the more data it has, the more accurate its decisions will be.

However, unlike a game of chess with two players, Next-Best-Action can be played with thousands of customers at the same time.

Optimizing Customer Value

The ultimate goal of Next-Best-Action is to optimize customer value. In other words, to maximize the profitability of each customer relationship and to help retain profitable customers.

To do this, Next-Best-Action bases each of its decisions on 2 essential things: first, what does that customer need right now, and then, what are the business objectives for this interaction?

And when it is deployed as a centralized decision hub, Pega Decision Management can deliver each Next-Best-Action decision to any available channel, from the call center to a mobile phone to a retail store, at any moment. Next-Best-Actions continuously build on each other across every interaction in any channel to maximize customer value over the lifetime of the relationship.

To satisfy customer needs, Next-Best-Actions must have 4 key characteristics.
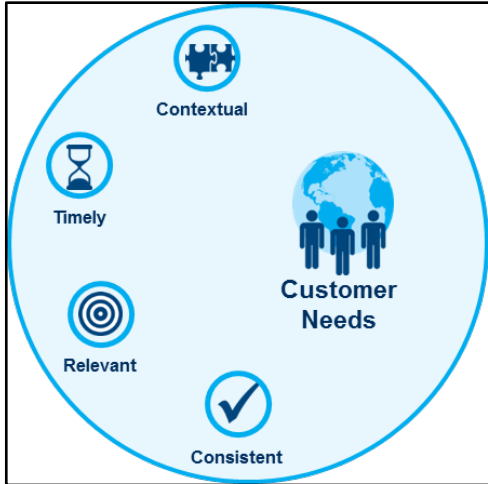
First, they must be Contextual. In other words, they must take into account the very last input the customer gave to the company. For example, if a customer said 'no' to a particular offer, that will influence what the next offer will be.

They must also be Timely. All Next-Best-Actions must be taken in an appropriate timeframe, via the right channel at the right moment. This means in real-time in the inbound channels.

Then they have to be Relevant. Every action taken must be right for that particular customer based on everything known about them at the moment of communication.

And lastly, Consistent. Every Next-Best-Action must be consistent across every channel. No matter which channel the customer uses to communicate with the company, the messages and propositions they receive must be the same.
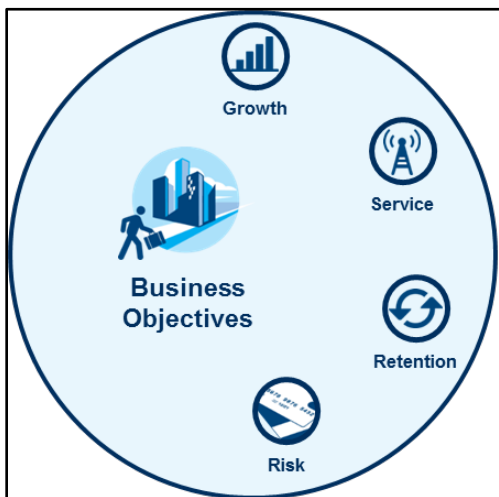
On the business side, Next-Best-Actions consider four key strategic business directions: Growth, Service, Retention and Risk mitigation. These lie at the heart of ensuring profitable customer interactions.

By considering these factors, Decision Management can answer key business questions during every interaction, such as: 'Do we want to try to make a sale with this customer or should we just focus on retaining them? If so, how much should we invest in trying to retain them? Or, maybe we need to send this customer straight to collections?

Other things such as budget and resource constraints must also be considered. For example, what if you're sold out of the product you want to offer to a customer?

All of these factors must to be weighed and balanced as part of every Next-Best-Action decision, and the weighing and balancing needs to happen fast.

Next Best Action

This is your customer. You want him to buy your products, use your services and have a great experience. And your competitors want the same thing.

To compete, you have to take the right action, at every customer touch, ensuring that each and every conversation delivers exactly the right message, the right offer and the right level of service.

You want to provide a great experience, while maximizing the customer's value to your organization.

Big data and analytics are at start. But the real value comes when you can turn data into insight, into action. Which means making coordinated decisions in real-time across all channels. Pega delivers this through a capability we call: Next-best-Action.
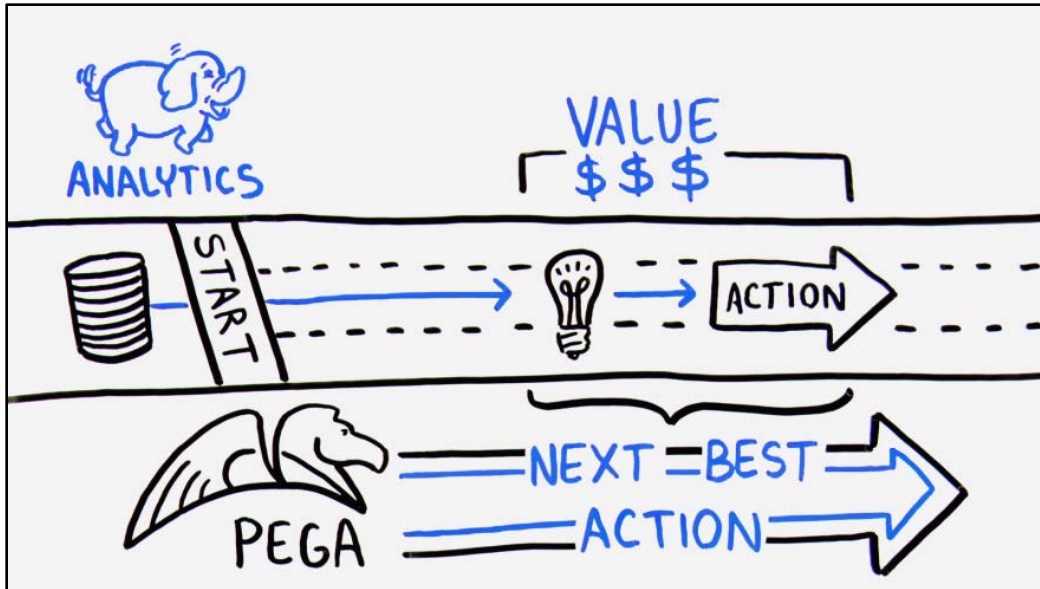
In Pega, business people uses browser based forms to build decision strategties visually. These strategies. These strategies combine predictive analytics with algorithms developed to mining large sets of data, adaptive analytics with self learning algorithms that improve with each interaction and traditional busienss rules that allow users to prioritize and arbitrate between decisions.
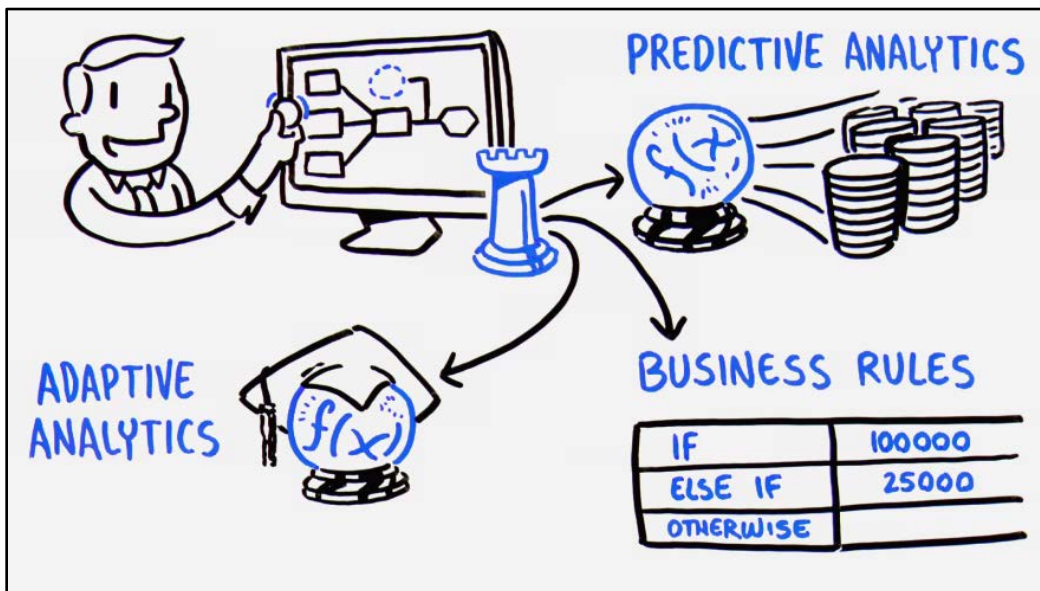
Pega uses a strategy to looks across all the potential actions that you may take with a customer. Make an offer, innitiate a retention plan, open a service case, start a collections process, and ensure that exactly the right action is taken at every interaction; and it works across al channels to give a consistent experience, weather the customer is on the web, a mobile app, in a store or calling a contact center.

Strategies can even be connected to streams like social feeds, or network events that detect paterns and drive the Next Best Action proactively. And stratgies are completely contextual. Any change in the customer context, a click a reply, a location change a tweet, will trigger the Next Bext Action so you can really listen to your customers and act accordingly.

Pega's 7 Next Best Action capability let's organizations optimize every customer interaction for experience and value. Our world is constantly changing. Only Pega let's you build for change.

Agent desktop

This video provides an overview of the Call Centre Agent desktop. Here's a typical desktop of the Next-Best-Action Advisor call center application.

Through this dashboard, agents are presented with all of the information they need to have an intelligent conversation with the customer.
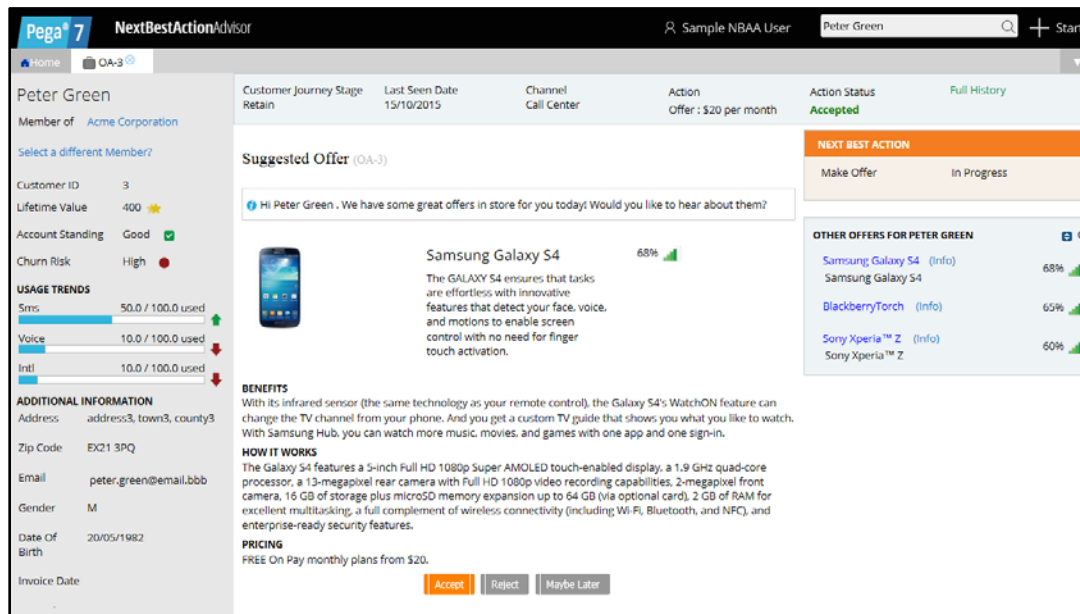
On the left is the customer profile and other contextual information such as their current Churn Risk.

All of this data is fetched in real-time from different data sources once an interaction begins.

In the center is the Next-Best-Action 'Suggested Offer', which in this case is the Samsung Galaxy S4.

Along the top we see more contextual information such as the last time we spoke with the customer and the main goal for this interaction, which in this case is to retain them.

By clicking on the Full History link, you can see a record of every interaction the company has had with the customer.

On the right, in the 'Other Offers' box are the top 3 offers the system as selected for them.

The Samsung is the most highly recommended, but if they reject it, there are 2 other highly ranked possibilities.

Call Center Interaction

In your role as a Call Center Agent, you're going to use Next-Best-Action Advisor to recommend two Next-Best-Action offers to customer Peter Green and then wrap-up the call with him. After logging in, you immediately see your Call Center Agent home screen. Here you can see recently completed assignments as well as the status of cases you're currently working on.
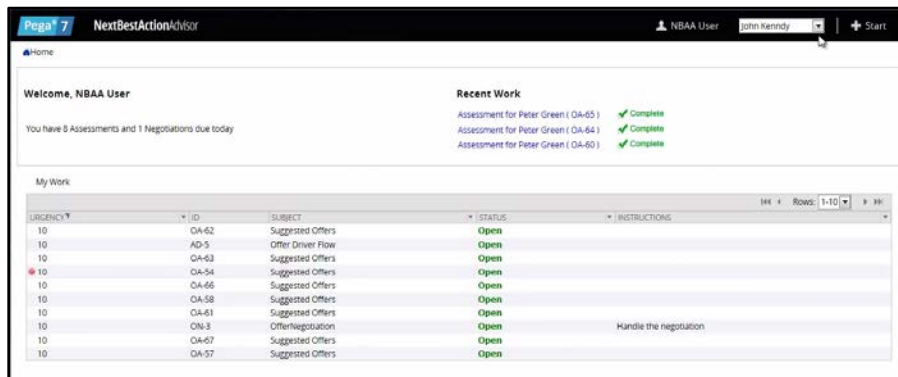
If you open the customer queue drop down menu in the upper-right corner of the screen, you can select a customer to start a new interaction with. In this case, select Peter Green and click Start.
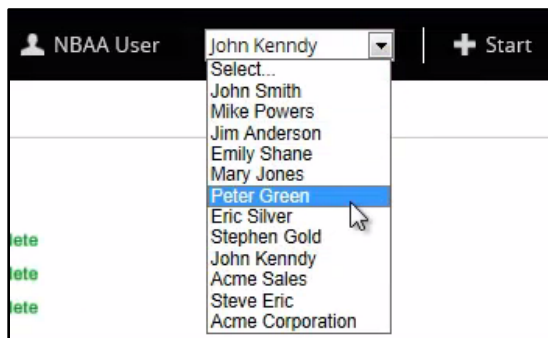
From the Start menu, you can select which process you want to run with the customer. In this case, you're going to kick-off a Next-Best-Action process.
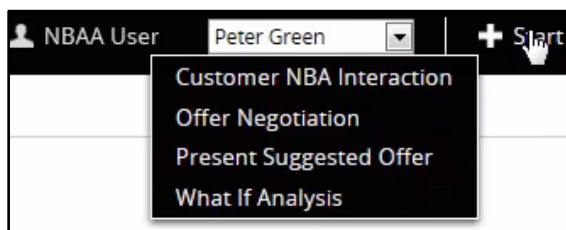
In the orange 'Next-Best-Action' box you can see that the recommended action is to make an offer.
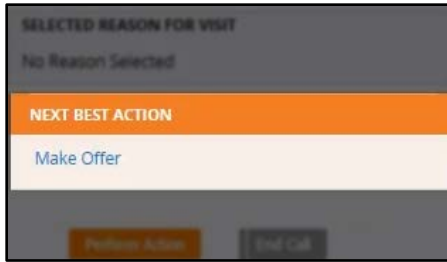
On the left is the profile information on Peter Green that has been fetched from the database.

In the middle of the screen is the 'Customer Intent' section. This is populated with potential reasons why the customer has called in.

Customer Intent (AD-11)

ℹ️ Hi Peter Green . We have some great offers in store for you today! Would you like to hear about them?

Why is customer here?

Please select a reason below

Account	Best Package	Billing
Change Mailing Address	What If Analysis	Dispute Bill
Other		Change Billing Address
Enquire About Leaving	Enquire	Change Billing Credit Card
Cancel Contract	Voice Plans	
	Other	Help
Complaint	Handsets	Delivery
Billing	Data Plans	Product
Bad Service	Best Offer	Service
Other		Other

In a real-world environment, the agent would select a call reason, and the recommended Next-Best-Action may or may not change based on that input.

In this demonstration, you're just going to take the current Next-Best-Action, which is to make an offer to Peter Green.

You can take the action by clicking on the 'Make Offer' link or the orange 'Perform Action' button.

SELECTED REASON FOR VISIT

No Reason Selected

NEXT BEST ACTION

Make Offer

Perform Action   End Call

The system now presents the Next-Best-Offer for Peter Green, which in this case is the Samsung Galaxy S4 phone.

If you look in the 'Other Offers' section, you can see that by default, the top 3 offers for Peter Green are shown.

But if you click on the blue 'Information' button you can see the other offers for Peter Green. Clicking on the 'Info' link for one of the offers will show you more information about it. This information can be customized with scripting and unique details for each customer.
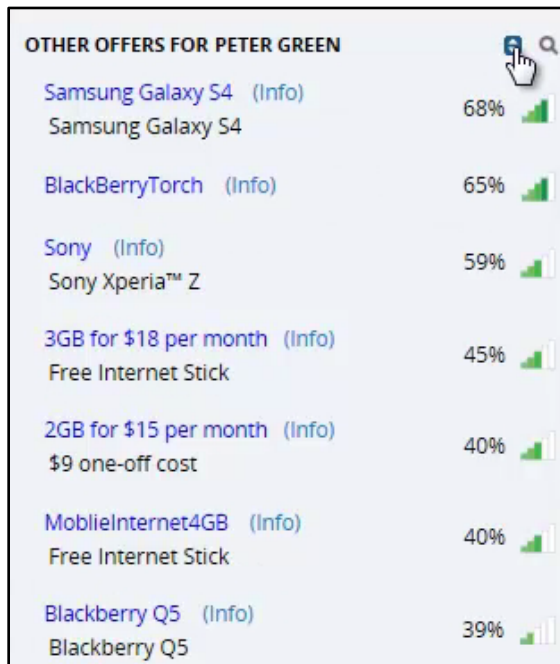
In the 'Other Offers' section, there is a percentage listed next to each offer. This is what we call the Propensity. In Pega Decision Management, the Propensity is the likelihood that a customer will accept a particular offer. The Propensity for each customer/product combination is determined by the predictive models in the decision engine.

Of course, the information displayed here doesn't have to be the Propensity, you can select something else based on your business priorities.

Let's select the Sony as the offer Peter Green is interested in. Now you can see that the 'Suggested Offer' section changes to highlight the Sony.

By accepting the offer for Peter Green, the offer fulfillment process is automatically kicked off and will be carried out by another application.
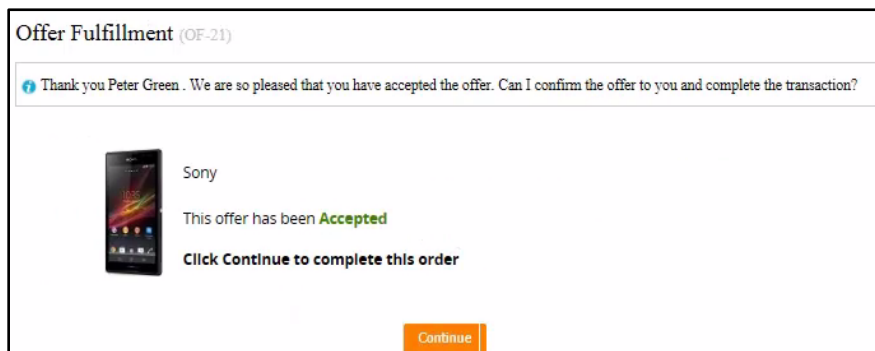
You will then be prompted to continue the Next-Best-Action process. At this point, you have the choice to either end the call, start from the beginning with the same offers, or carry on with Next-Best-Action.

If you click the 'Perform Action' button, the system immediately displays the Next-Best-Offer.

In this case, it's a Data Plan. This is because, using historical data, the decision engine has seen a strong correlation between a customer accepting a new handset and then accepting a new data plan. Notice now that in the 'Other Offers' section on the right, all of the top offers shown are data plans.

Click Accept again, and the order fulfillment process will kick off for the customer. Then click Continue, and this time, wrap-up the call.

Now you're back at the home screen and can go ahead and take the next customer in the queue.

Inside Pega Decision Management

Key features of Pega Decision Management

In this lesson, we're going to open the Pega Decision Management box and take a look at the features and components that come with the product. We'll also examine how they work together to make decisions and deliver Next-Best-Actions.

The key features of Pega Decision Management are: Proposition Management, Decision Strategies, Interaction History, Visual Business Director, Adaptive Decision Manager, and Predictive Analytics Director. You need to have at least a high-level understanding of each of the key features, so let's define them briefly.

Proposition Management is how propositions are defined, stored and managed in Pega Decision Management. Propositions are anything that can be offered to a customer. This can include things such as advertisements, products, offer bundles, even customer service actions.

Decision Strategies match propositions with customers. They combine data about the customer with rules and predictive analytics to determine the Next-Best-Action for a customer during each interaction.

Interaction History is what gives Pega Decision Management its long-term memory. It captures every customer response to every Next-Best-Action and provides it as input for determining what the Next-Best-Action should be.

Visual Business Director is a 3-D graphical environment for simulating and analyzing the results of decision strategies and customer interactions. This helps ensure that your strategies are on target to meet your goals.

Adaptive Decision Manager enables automatic development and deployment of adaptive predictive models. These models learn and gather data on-the-fly to predict future customer behavior.

Predictive Analytics Director provides a safe and heavily automated environment in which business users can easily develop fit-for-purpose, non-adaptive predictive models that use historical data to predict future customer behavior.

Decisioning artifacts

Each of the key features of Pega Decision Management leverages the different types of decisioning artifacts: rule types, landing pages, decision components, and flow shapes.

These artifacts are the building blocks of Pega Decision Management.

Let's look first at some of the main landing pages. Landing pages are where certain key capabilities are configured, monitored and managed.

The landing pages also provide a shortcut to viewing the rule types as well as the data behind the key capabilities we will discuss.
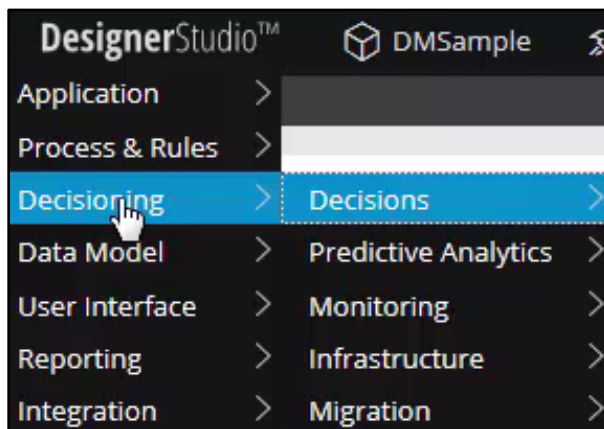
The Decisions landing page enables you to define and manage all aspects of the decisioning that determines Next-Best-Actions. This includes proposition hierarchies, simulations and data flows.

The Predictive Analytics landing page enables you to manage all aspects of the development and implementation of predictive and adaptive models.

The Monitoring landing page enables you to monitor decisions through three types of reports: the Adaptive Model reports, Interaction History reports and Visual Business Director.

The Infrastructure landing page enables you to configure, view and manage resource usage and various operational settings.

Next, let's examine the decisioning-specific flow shapes. Each element in a process flow is called a flow shape. These flow shapes allow seamless integration of Pega Decision Management with Pega Business Process Management.

There are 2 types of flow shapes: Decision and Run Interaction.
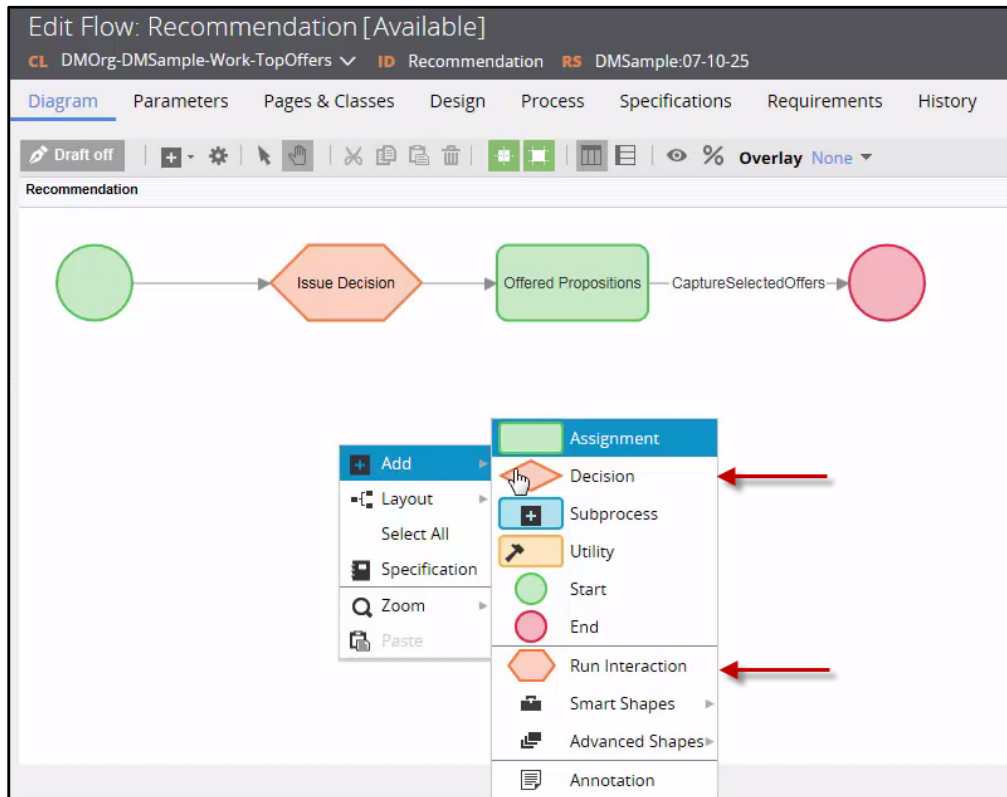
The Decision flow shape extends the capability of the Decision flow shape in the Pega platform by enabling it to invoke predictive models and Decision rules, such as Decision Tables and Decision Trees.
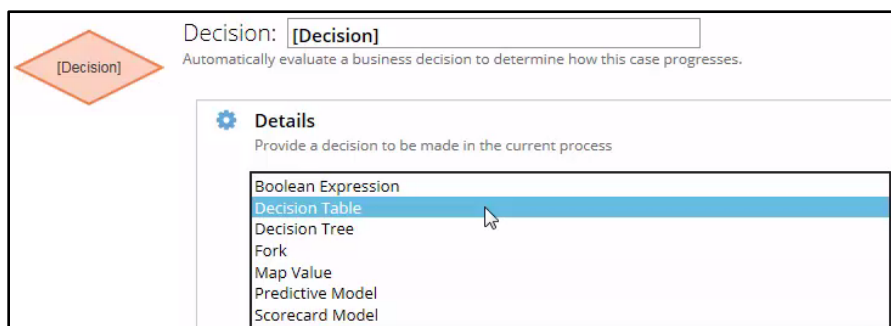
The Run Interaction flow shape invokes a decision strategy, which determines how and when to make a proposition to the customer, and then captures the customer's response to the proposition.
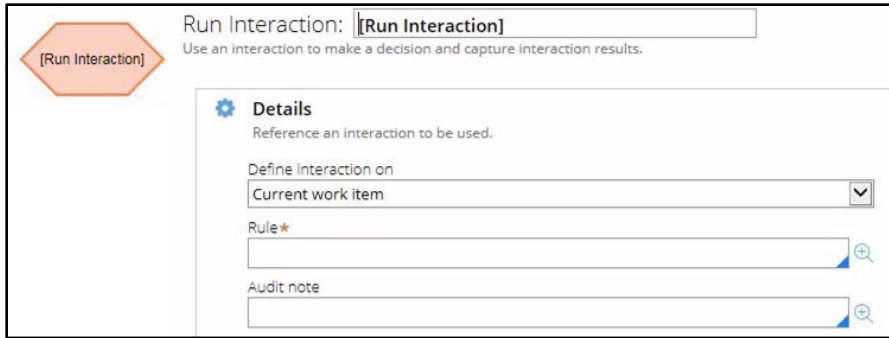
Decision Management also contains new rule types. These are Scorecard, Predictive and Adaptive Model, as well as the Interaction and Strategy rule types.

The Strategy rule type, which is the main rule type, utilizes all other rule types to determine what the Next-Best-Action should be.
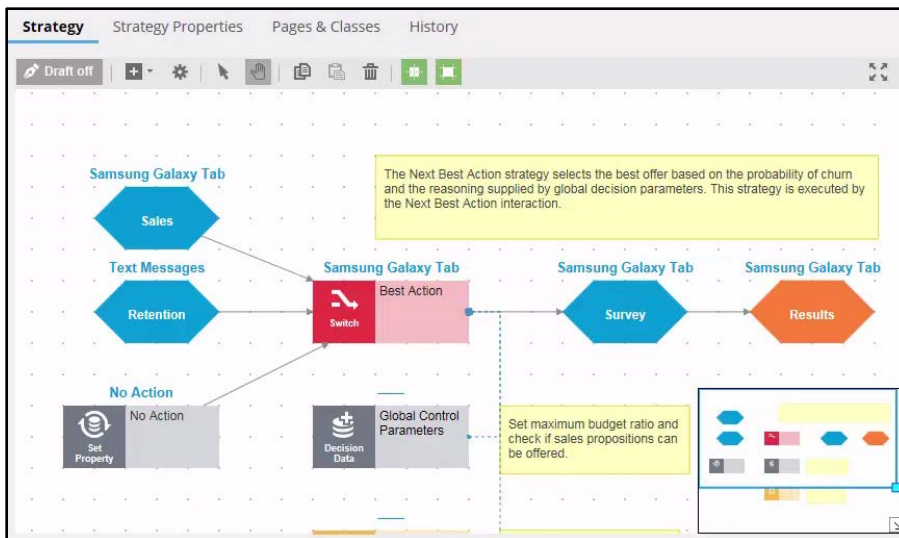
Decision Tables and Decision Trees are existing rule types that use different formats for doing largely the same thing, which is to classify customers by characteristics for the purpose of creating segments.

The Predictive Model rule type is used to execute predictive models created with Pega Predictive Analytics Director.
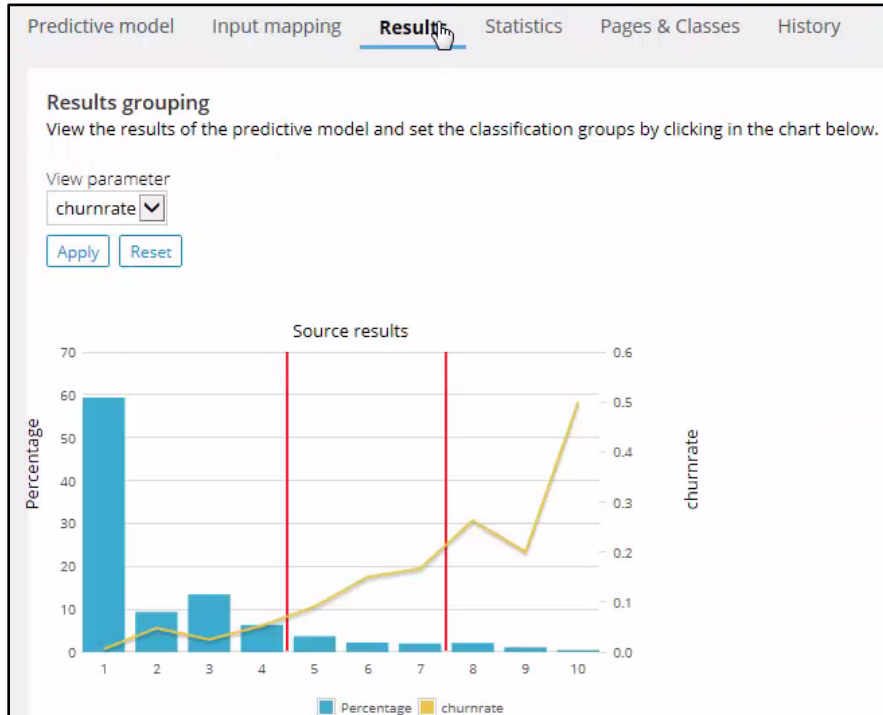
The Adaptive Model rule type is used to configure the adaptive analytics decision models that are managed by the Adaptive Decision Manager service.

The Interaction rule type is used to execute a decision strategy from a workflow and capture the details of the customer interaction.
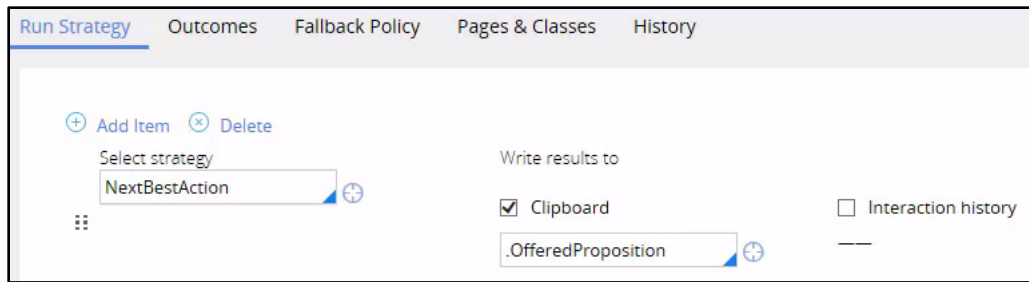
Scorecards are used to implement industry standard scorecards such as the FICO scorecard.
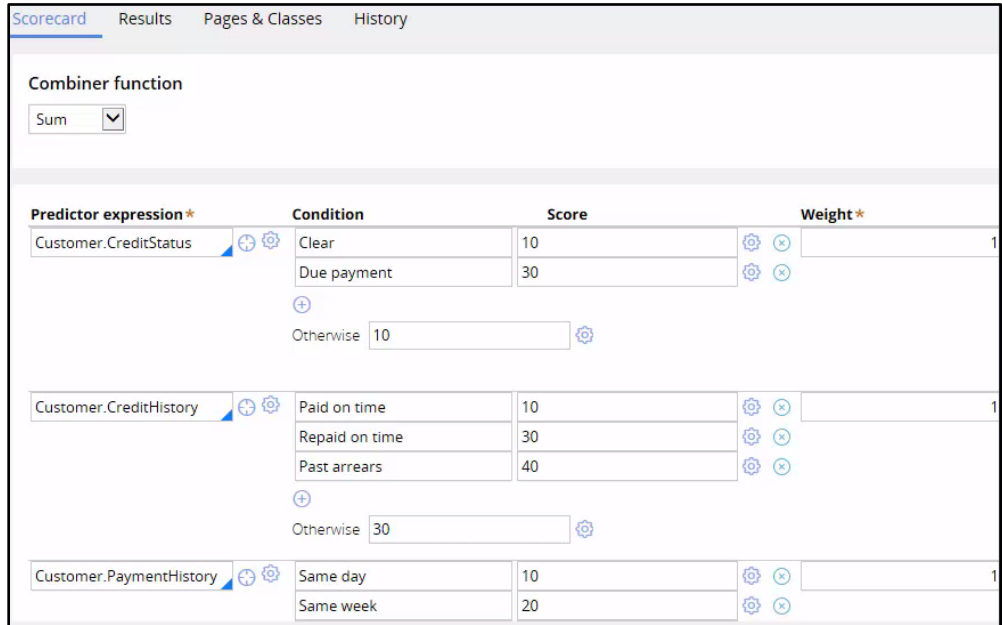
And lastly, the decision components are the building blocks of decision strategies. They are the language used by the business to express how they want to do business through their strategies.

There are 7 groups of decision components: Import, Business Rules, Decision Analytics, Enrichment, Arbitration, Selection and Aggregation.

Within each of these groups are a number of decision components, but for now we'll just give an overview of each of the groups.

The Import components bring additional data into the decision strategy, such as external data, Interaction History data, or the propositions that you want to offer to the customer.

Business Rules components apply business rules, such as eligibility requirements.

Decision Analytics components make predictions about the customer. They often make predictions about their propensity, which is the likelihood they will accept an Offer.

Enrichment components can do calculations, such as a margin calculation based on the price and cost of an Offer.

Arbitration components filter out propositions that are less relevant for the customer. They also filter out propositions that the customer is not eligible for.
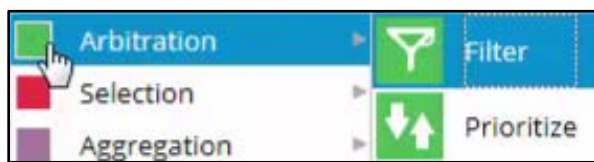
Selection components allow propositions to be ranked, so the top Offers for each customer can be selected.
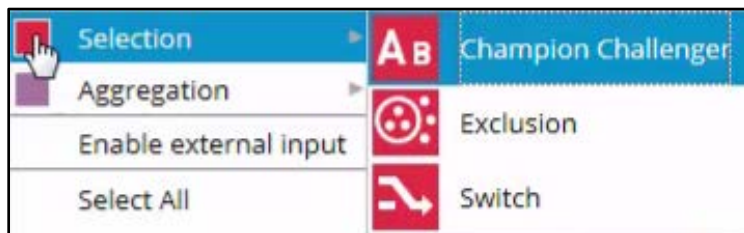
Aggregation components allow the calculation of aggregate numbers. For example, a communications company can calculate monthly usage, or a bank can calculate average monthly account balances before determining the customer's eligibility for, say, a credit card.
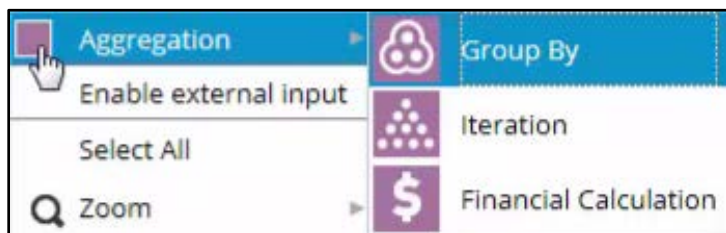
System architect view

In your role as the System Architect you will explore the artifacts that come with the 'Whats In The Box' application.

Start by logging in as the System Architect to the Pega 7 Designer Studio portal.

This is the Application Designer portal. It provides access to the back end of the sample application.
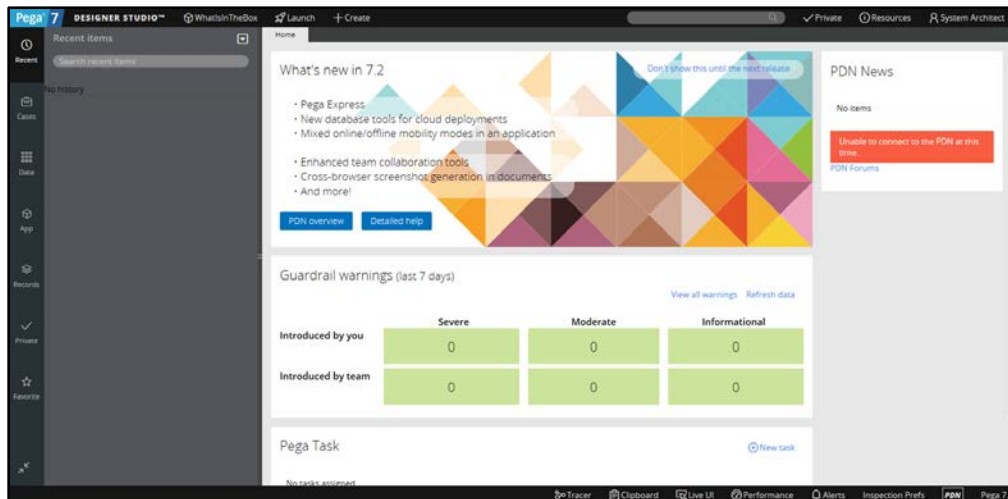
Most of the changes to the artifacts will be delegated to the Strategy Designer whose work will be carried out in the Decision Manager portal.

But before diving into the back end, let's take a look at how the application looks to an end-user and how it's used.

To experience the application from an end-user perspective, you have to switch to the Decision Manager portal.

From here you can run the Top Offers process against customers in the database.
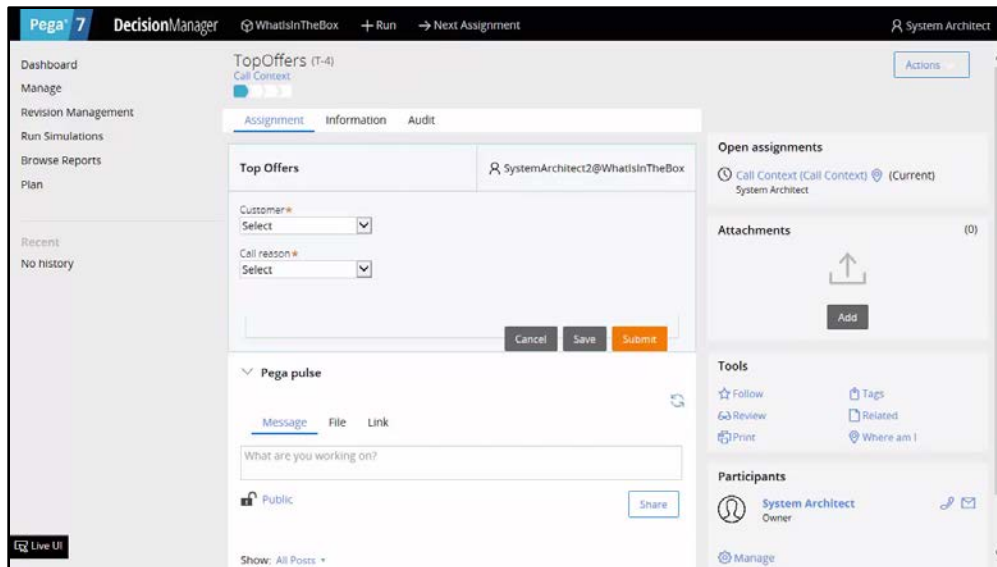
The Top Offers process has 3 stages. First, the customer that's being served is identified and the reason they're calling is determined.

Next, the top 3 Offers are presented to the customer, and their response to each proposition is recorded.

On the last screen, you can see the result of all of the interactions the company has had so far with the customer.

Now let's explore the decisioning artifacts in the application. This will help you understand how the application works and enable you to modify it. To do this, you need to switch back to the Designer Studio portal.

In the application explorer, open the application and expand the Process Flow menu.

Now click on the Top Offers business process.

Again you see the 'process flow' diagram. To view the underlying decision strategy, you need to open the Issue Decision flow shape.
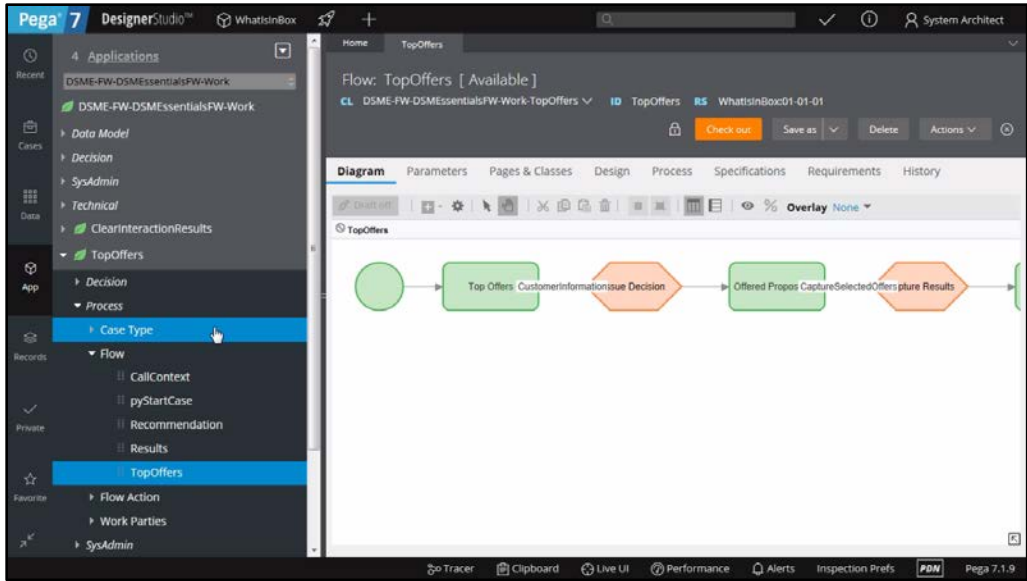
When you right click on it, you can see that it calls an 'Interaction rule'.

When you open the Interaction rule, you can see on the left that it invokes a strategy, in this case, it's called the Next-Best-Action strategy.
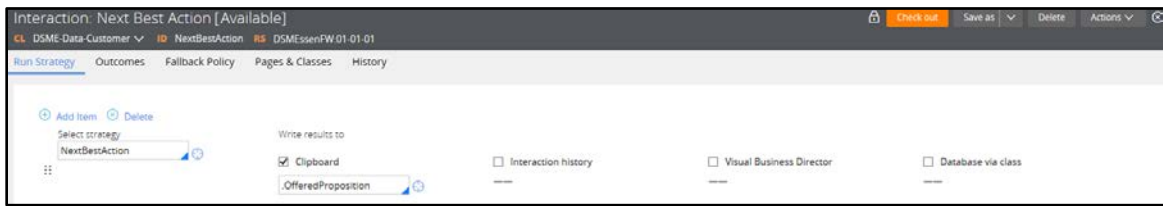
If you now open the Next-Best-Action strategy, you can see the underlying structure, which shows that the Best-Action is selected from either a Sales or Retention component, or it returns 'No Action'.
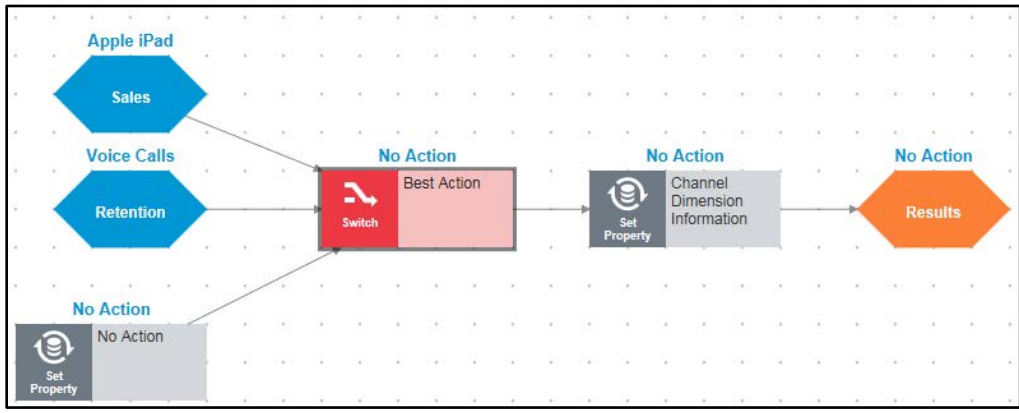
Let's examine the Switch component to determine when the different strategies are selected.

As you can see, if the Call Reason is "Customer care" then the Retention strategy will be selected, if it is "Product offer" then the Sales strategy will be selected.

If none of the reason codes match, the best action is 'No Action'.

If you drill down into the Sales component, you can see the strategy it uses to select Offers.

The Offers are selected from either all available Products or all available Tariffs.

If you open the Products component and go a third level down, you can see the strategy it uses to select Products. These are selected from either all available Phones or all available Tablets.
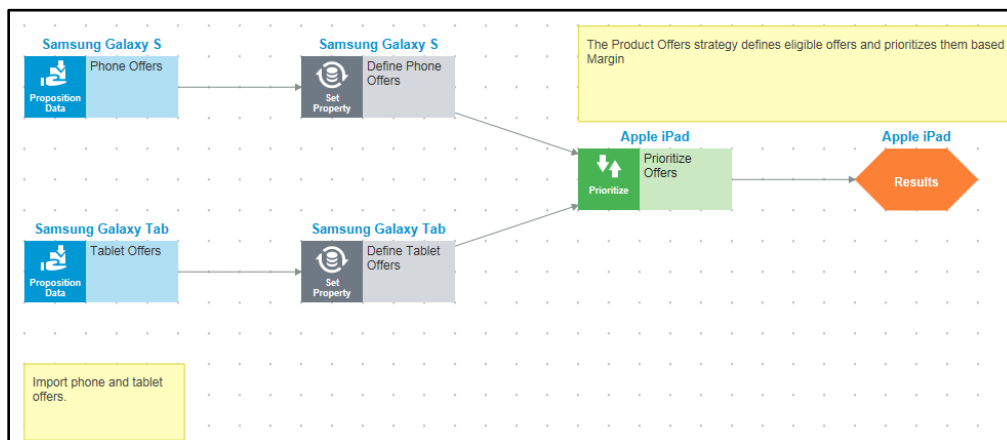
In this lesson you have explored the artifacts that come with the 'Whats In The Box' application. These are the Top Offers process flow, the 'Next-Best-Action' Interaction rule and the Next-Best-Action strategy.

Strategy designer view

In your role as the Strategy Designer, you want to modify the strategy to display all relevant Offers for the customer instead of only the top 3.

Normally, the TopOffers process selects the top 3 Offers for a customer.
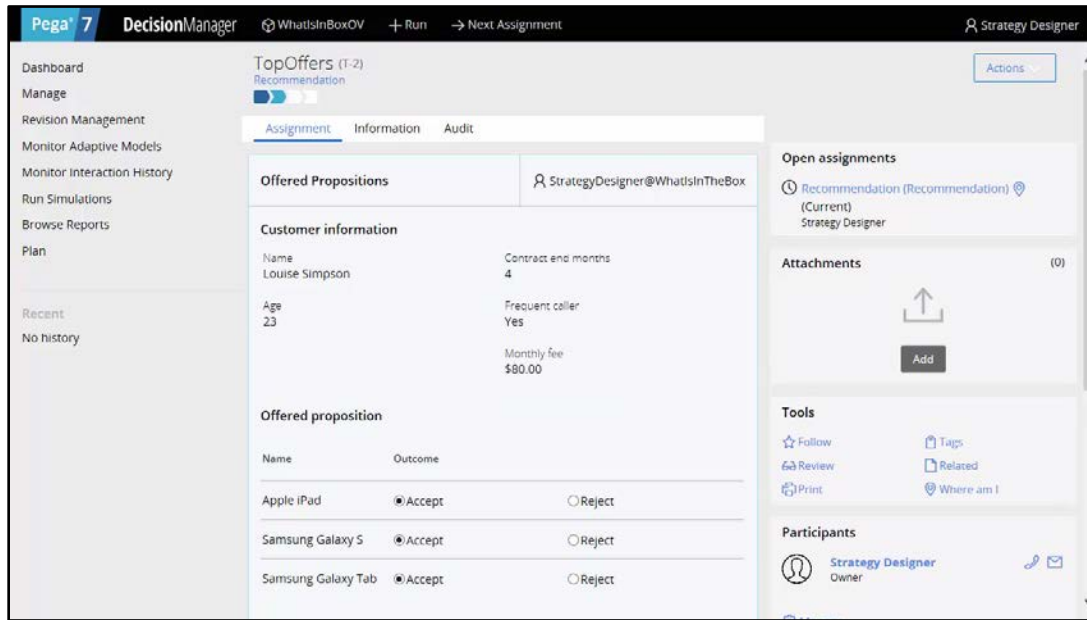
To change this, start by creating a change request detailing the work to be done.

Essentially, the Prioritization component in the Product Offers strategy needs to be modified.
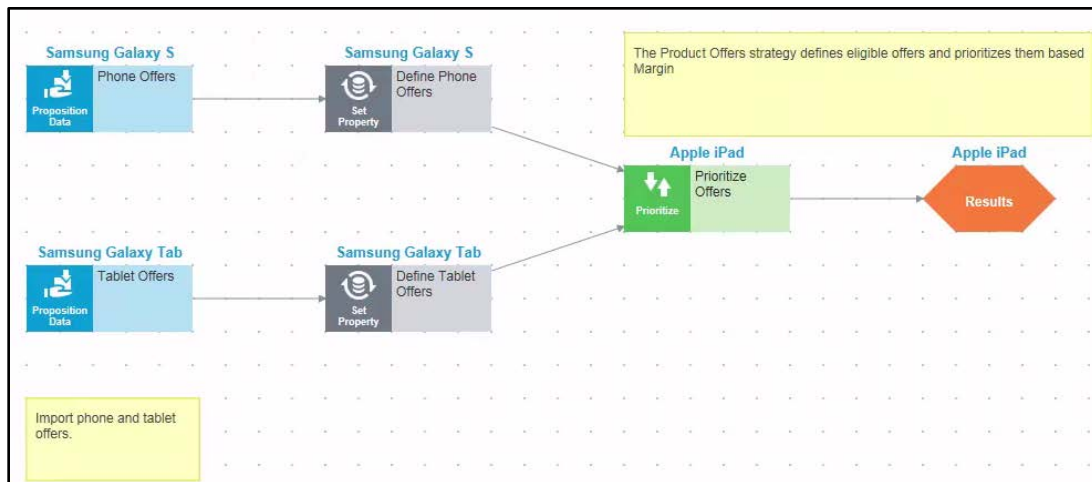
The value that needs to be changed is in the Prioritize Offers component, so you need to open this component's Properties panel.

Here you can see that the Output of the component is currently the Top 3 Offers. And in the Expression field, you see that they are sorted by Margin.

You want to change the Output from 'Top 3' to 'All'.

Now instead of returning the Top 3 Offers, the strategy will return all relevant Offers for a customer. By saving the change you can verify your work.

To see the result of the change you can step through the TopOffers process again.

For a direct comparison of the two runs, select the same customer and the same 'Call reason'.

Now you can see that the customer is presented with four propositions instead of three.

Enriching Business Applications with Next-Best-Action

Decision Management in a business process

This lesson will illustrate how Decision Management can be used from within an existing business process.

Typically, a Decision rule or Decision strategy is kicked off as part of a business process. In order to request a decision, you use a flow shape.

You can think of the flow shape as the link between Business Process Management and Decision Management.
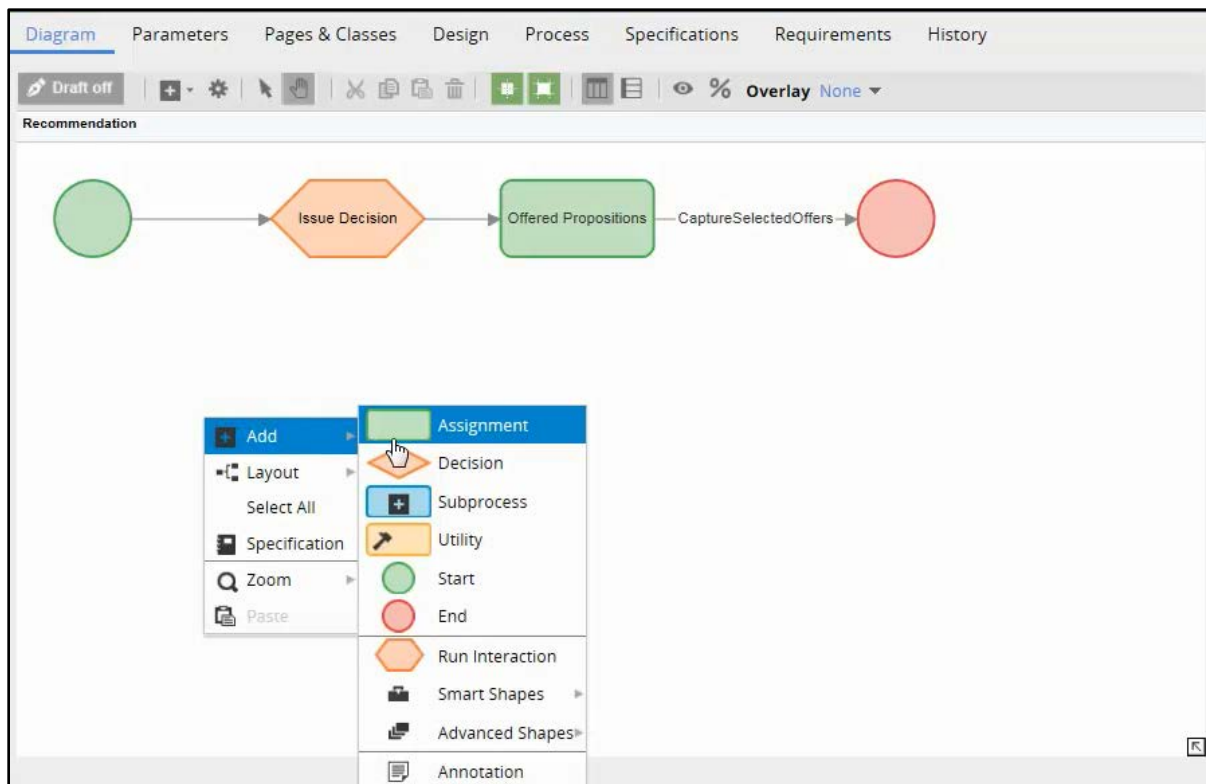
These flow shapes allow seamless integration of Pega Decision Management with Pega Business Process Management.

There are 2 types of decisioning-related flow shapes: Decision and Run Interaction.

The Decision flow shape extends the capability of the Decision flow shape in the Pega Platform by enabling it to invoke predictive models and Decision rules.

The Decision rule is either a Predictive Model, Map Value, Scorecard, Decision Table or Decision Tree.
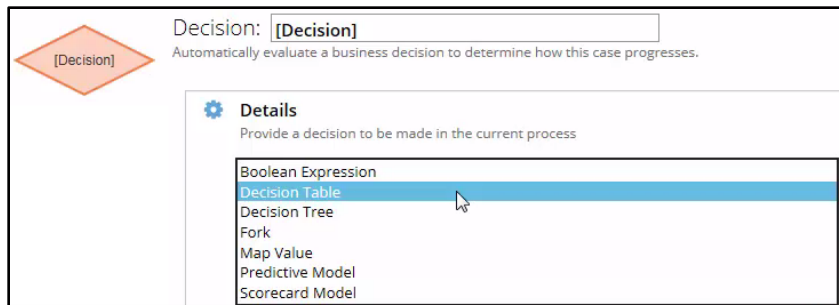
A 'Run Interaction' flow shape invokes a Decision strategy via the Interaction rule type. Having the Interaction rule in-between separates the Decision strategies from the business processes.

This allows you to swap out different strategies without having to touch the process flow. To view the Decision strategy, you need to open the underlying interaction rule.
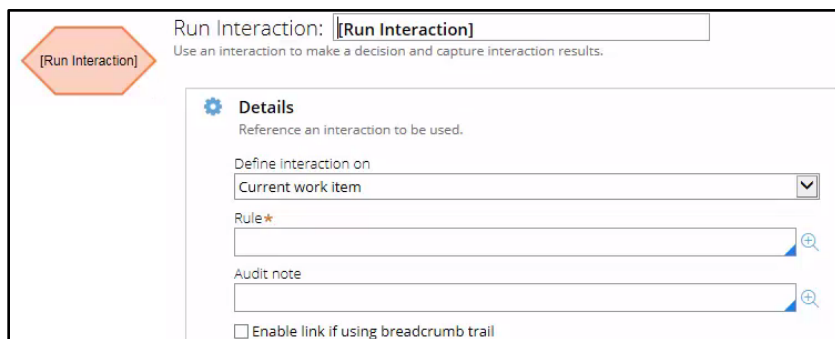
By right-clicking on the Issue Decision shape, you can see that it calls an Interaction rule.

By opening the Interaction rule, you can see on the left that it invokes a strategy, in this case, it's called the Next-Best-Action strategy.

As you can see, the results of the strategy are written to the Clipboard.

The Clipboard is the place where data can be stored and accessed by both the Business Process Management layer as well as the Decision Management layer of the application.
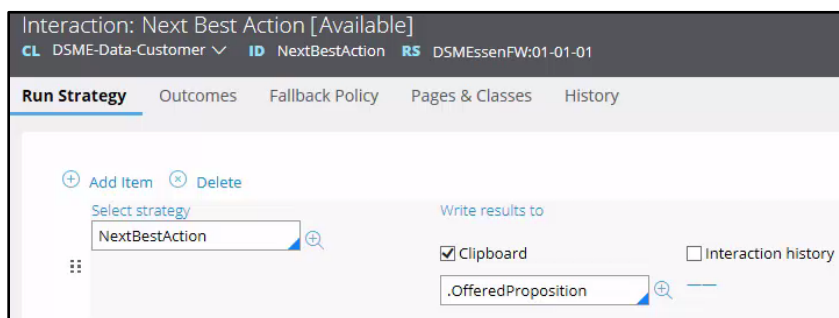
When you open the Next-Best-Action strategy, you can see the top-level decisioning structure. The Best Action is either a Sales action, Retention action or No Action.
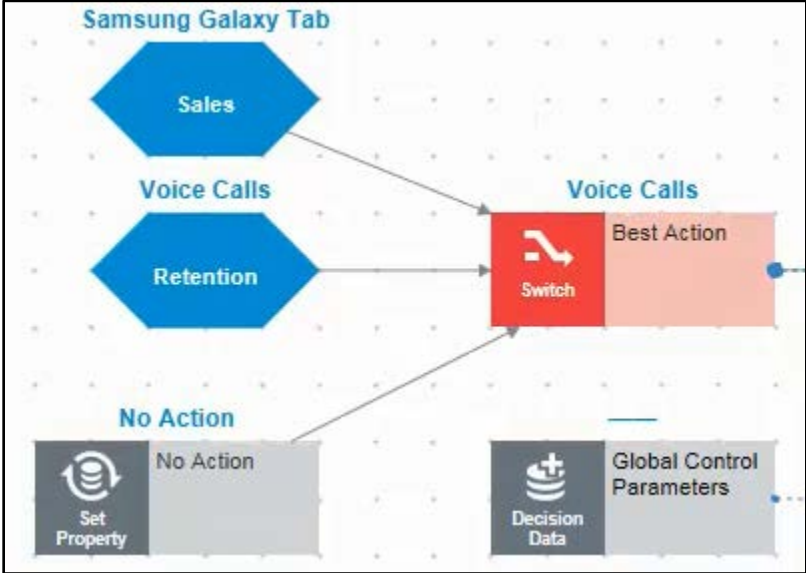
In this way you can enable a business application to optimize the customer experience by augmenting it with Decision Management.

The Importance of Propositions

What is a Proposition?

In Pega Decision Management, propositions are anything that can be offered to a customer. This can include things like advertisements, products, offer bundles, even service actions. Whatever is presented to the customer as the Next-Best-Action is called a proposition. The actual definition is: "The proposed course of action."

Under the covers, a proposition is just a data model that is organized into a three-level hierarchy. The highest level is the business issue level. This represents the ultimate purpose of the proposition, such as Sales, Retention, Service, Risk Management, etc. Next is the group the proposition belongs to, which can include Phones, Discounts, Tariffs, Tablets, etc. And then there are the names of the actual propositions themselves. Using this three-part hierarchy, every proposition in Pega Decision Management has a unique Identifier, which is a combination of its Business issue, Group, and Name. This 3-level hierarchy is customizable, and can be easily mapped to an existing product catalogue.
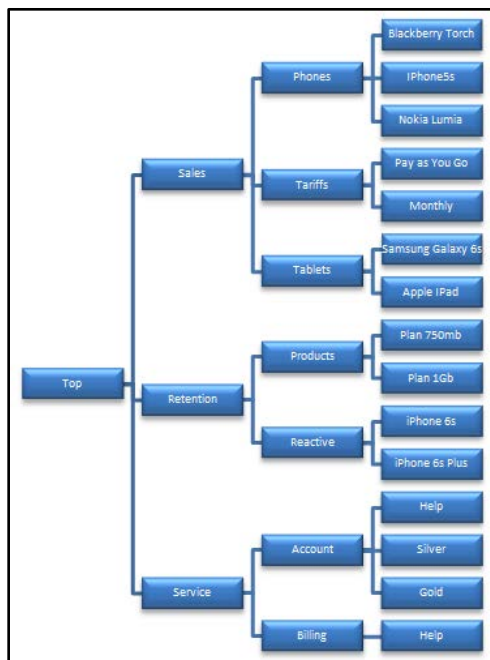
In addition to its Identifier (/Business issue/Group/Proposition Name for example; /Sales/Phones/IPhone5s) each proposition has properties. These properties can be a number of different things such as an image or description, a price, the actual cost, or the customer lifetime value threshold required for eligibility. However, not all properties are relevant for decision-making, so Pega Decision Management handles this information as smartly as possible, importing only the necessary properties into the decision strategy.

Proposition Management

Propositions can only be created under Business issue/Group hierarchy and the name of the proposition must be unique within the individual Business issue/ Group structure. That is to say, two propositions called 'Help' may exist providing there were defined under different Business issue/ Group structure.

To add a new property to a proposition you should first define the property on the Proposition Management landing page. During that process you declare the property name, property type and the property scope.

The property scope is a very important concept in decisioning and it follows the rules associated with proposition hierarchy. The properties scope can be summarized as follows:

- Scope global (top) - available to all propositions in the system

- Scope issue - available to all propositions within the same issue

- Scope group - available to all propositions within the same group

Once a property is created you can make it available to a proposition. To do so, open the proposition grouping and select the property from a property list. Only properties within allowable scope are displayed.

Monitoring Propositions

Reports

There are a number of reports based on the interaction history which are automatically created by the system. You will find them in the **Designer Studio** on the **Interaction History** landing page

(Decisioning→Monitoring→Interaction History)

and in Decisioning Management Portal under Interaction History Reports under Browse

| Dashboard |
| Manage |
| Revision Management |
| Monitor Adaptive Models |
| Monitor Interaction History |
| Run Simulations |
| Browse Reports |
| Plan |
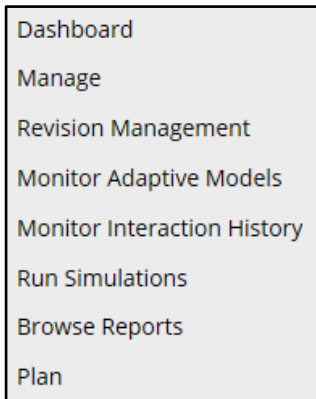
Reports.

There are four standard reports:

Accept Rate - Accept rate by a proposition group.

Volume by Channel - Information regarding which channel the propositions are accepted or rejected.

Volume by Proposition - How frequently a proposition is being offered to the customer.

Recent Interactions – A list of recent interaction in a chronological order.

In a real implementation, these reports would typically be based on millions of records for thousands of customers. Pega Decision Management typically interacts with thousands of customers at a time through multiple channels. All of the propositions and customer responses are captured in the Interaction History, which builds up Big Data, and we use that to do analytics. In fact, the more data there is the more analytics the system can do, the more it learns, and the better it becomes at making offers that will be accepted.

Visual Business Director

Visual Business Director (VBD) is Pega's 3-D graphical environment for simulating and analyzing the results of Decision Strategies in real time. This means that if you add a new proposition it will immediately picked up and analyzed by VBD. VBD enables you to monitor **Key Performance Indicators (KPIs)** which you can create and maintain based on previously defined interaction outcomes.

Create proposition hierarchy

This lesson will illustrate how to create a proposition hierarchy as a Decisioning Architect.

As you recall, a proposition is just a data model that is organized into a three-level hierarchy. The highest level is the business issue level, which you will now create.

First, navigate to the Hierarchy landing page, which can be found under Decisioning: Proposition Management. Here you can see previously created Retention and Sales business issues.

To create a new business issue, click "New business issue" and give it a unique name. In this case, Service.

Now, return to the landing page and click Refresh. As you can see, your newly added Service business issue is displayed.
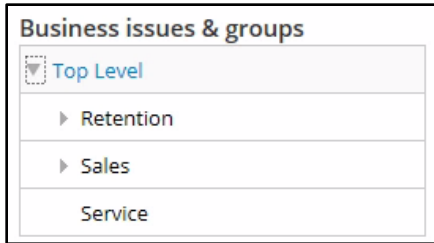
To create the second level of the hierarchy, click 'New group' and enter a group name. The name must be unique within the issue. Ensure that the box entitled "Create versioned proposition data" is selected. This will make proposition data available for version control. At this stage you can also select a business issue for which the group is being created.
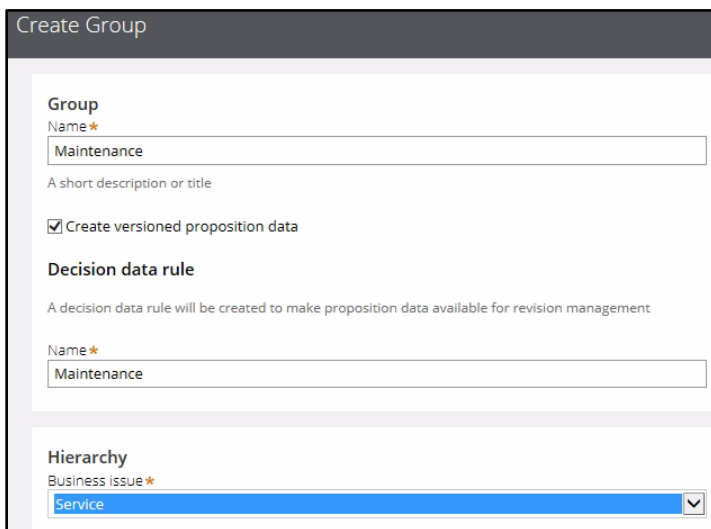
To see the new Group, you need to refresh the screen. Now you can select the new Group by expanding the Service issue.

To add a new proposition you need to open the Maintenance group.
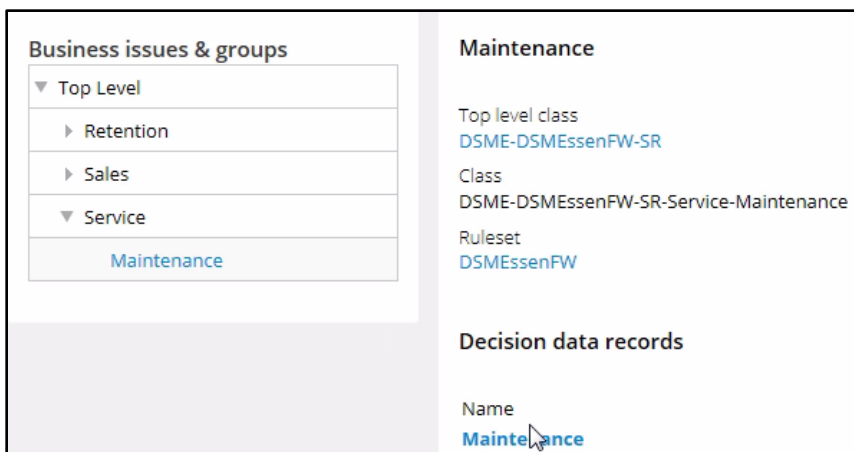
Here you can add propositions. To create a new proposition, check-out the Decision Data record and click New.
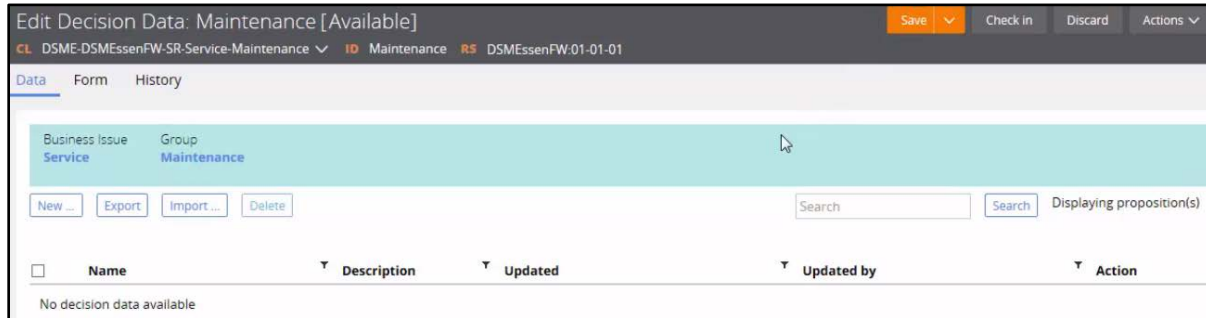
Enter the proposition Name, Description and Cost values.

In order to preserve the changes, save the record.

You can now add a new property to the Maintenance group. Let's define the MaxDiscount property for all propositions in this group. This is done on the Form tab. First you need to create the property, and then add the property to the group.
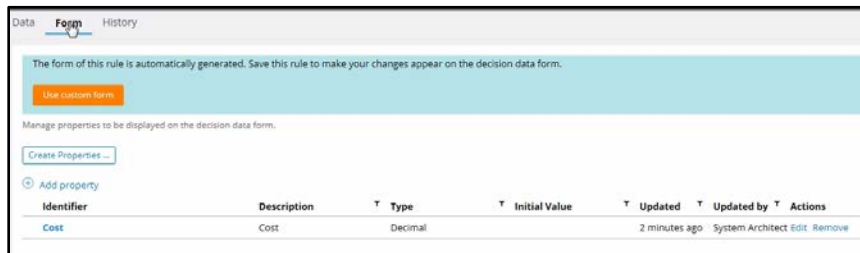
Enter the property Name and Type, then click anywhere on the form to save it.
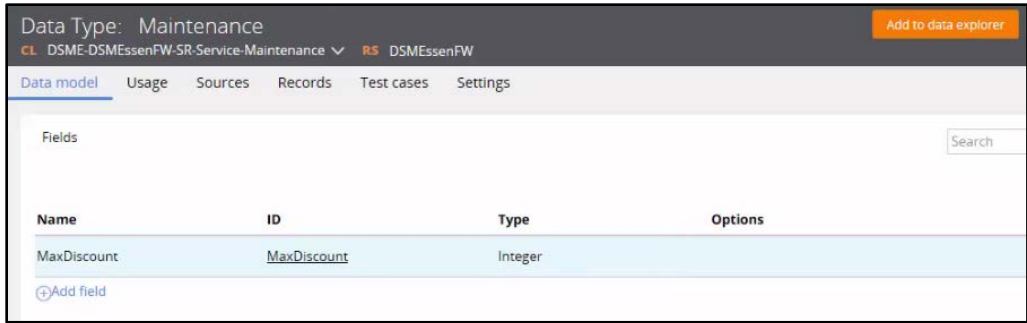
Now the MaxDiscount property is created, and you can add it to the proposition group.

From the Data tab you can open a proposition and edit the value of the new property.

This demonstration illustrated how to define a business issue, a proposition group, a proposition and a proposition property

How to create new propositions

This demo explores how propositions are created and managed, as well as how they are tracked and reported on once they have been offered to the customer.

The TopOffers process used in this demonstration uses the Product Offer strategy to determine which Sales Offer should be made.

Execute the TopOffers flow, to show how Interaction History records are collected.

The Product Offer strategy selects all currently eligible Offers and prioritizes them based on Margin. Accept one offer, and reject the others.

Offered proposition		
Name	**Outcome**	
Apple iPad	⦿ Accept	○ Reject
Samsung Galaxy S	○ Accept	⦿ Reject
Samsung Galaxy Tab	○ Accept	⦿ Reject
Apple iPhone	○ Accept	⦿ Reject

Four reports were created based on this Interaction History.

Let's examine each of the reports individually.

In the Proposition Distribution report you can see how frequently a proposition is being offered.
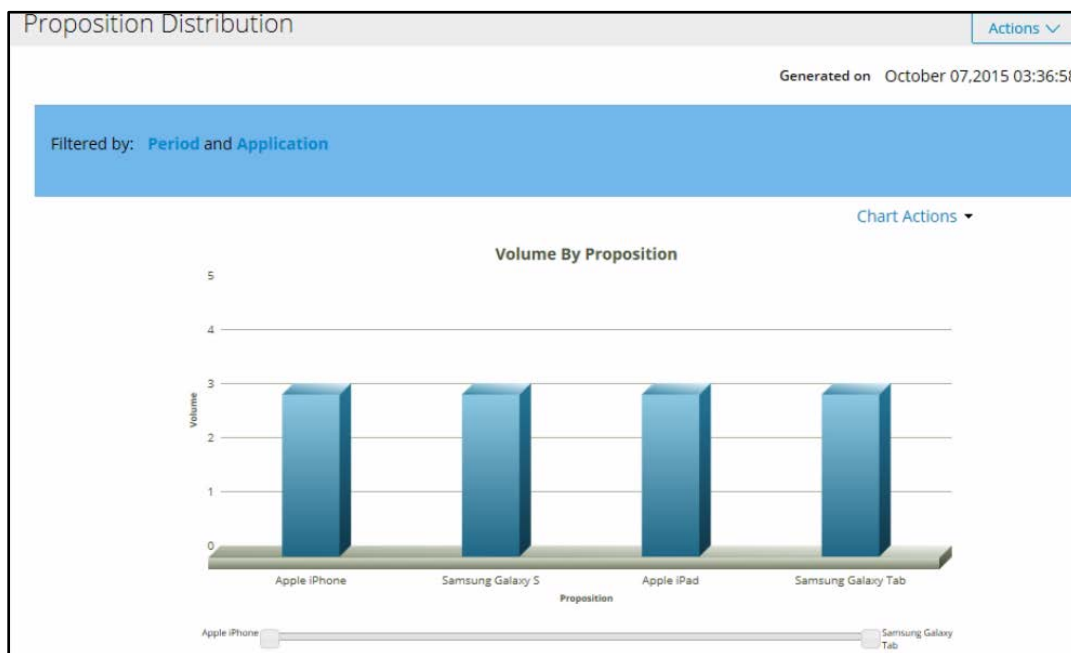
The Channel Distribution report displays proposition Accepts and Rejects per channel.
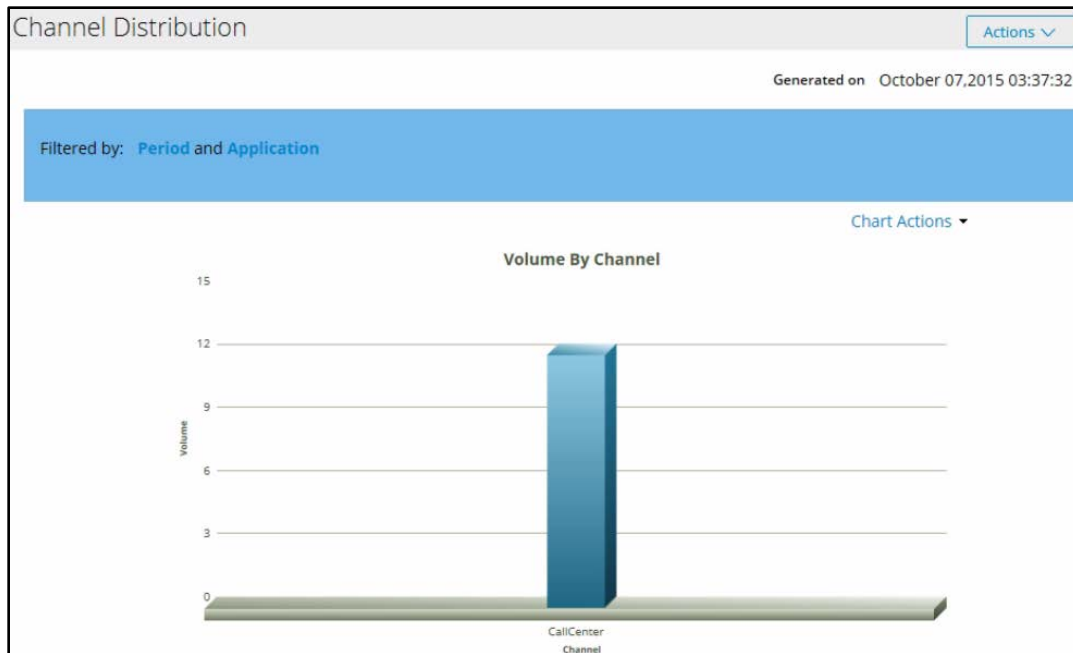
The Acceptance Rate report shows the accept rate of proposition groups, in this case Phones and Tablets.
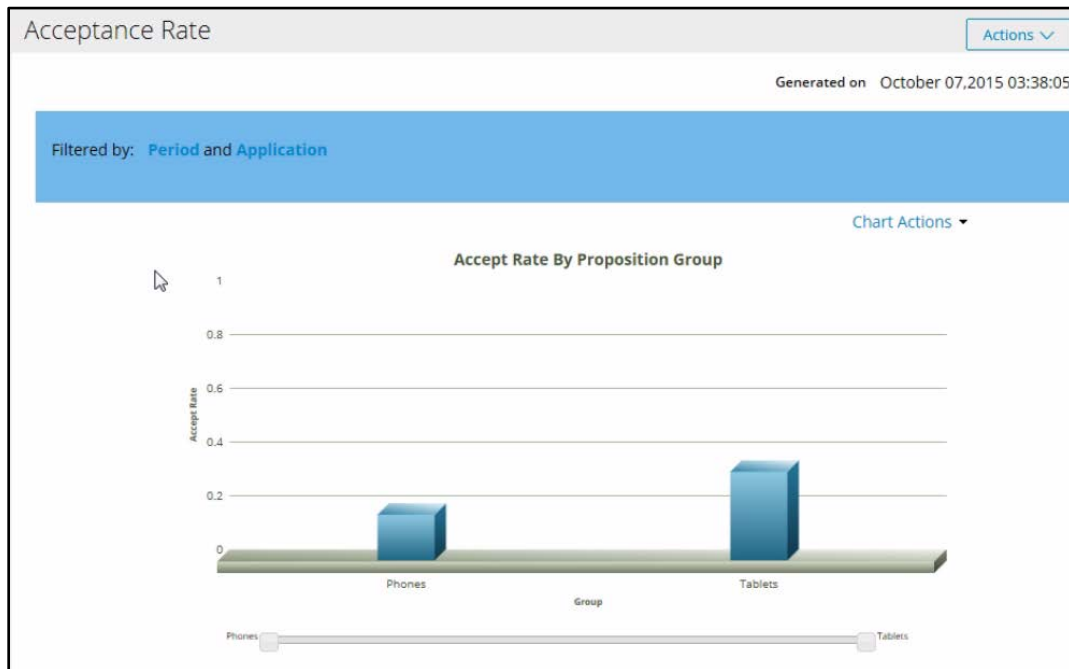
The Interaction History report lists recent interactions.

What you see here are 12 interactions with our test customers, Louise Simpson and Maria Shields, who are identified as CE-1 and CE-2 respectively.

In a real implementation, this report would typically be based on millions of records for thousands of customers.

Pega Decision Management typically interacts with thousands of customers at a time through multiple channels.

In the **Key performance indicators** tab, you can create and maintain KPIs based on previously defined interaction outcomes.

These metrics allow you to calculate things like 'accept rate' or 'conversion rate'.

For example, you can count the number of responses.

Or calculate the accept rate, which is the ratio of accepts to total volume.

Now let's take a look at how this data is rendered in Visual Business Director, which is Pega's 3-D graphical environment for simulating and analyzing the results of Decision strategies.

In the Saved Views tab you can define different perspectives on the data. Here you see only one view, the Sales Outcomes.

When you open this view, you will see a 3-dimensional depiction of the Accept and Reject volumes for each of the propositions.

This is a live, real-time visualization of the actual data. Listed along the Y Axis are the different propositions, and along the X Axis are the Outcomes. By right-clicking on the graph, you can rotate the view to get a look at the data from different perspectives. You can also reset to the original view, and zoom in and out.

You can drill down on the Y Axis to view Accepts and Rejects at the Group or Issue level. On the X Axis you can drill down to view the total volume of Accepts and Rejects. And you can change the view by clicking on a different KPI.

You can even bring the back panel forward for a 2-dimensional view and take screen shots to use in reports.

You can change the dimension of the X Axis by clicking on the All Outcomes label and selecting a different dimension.
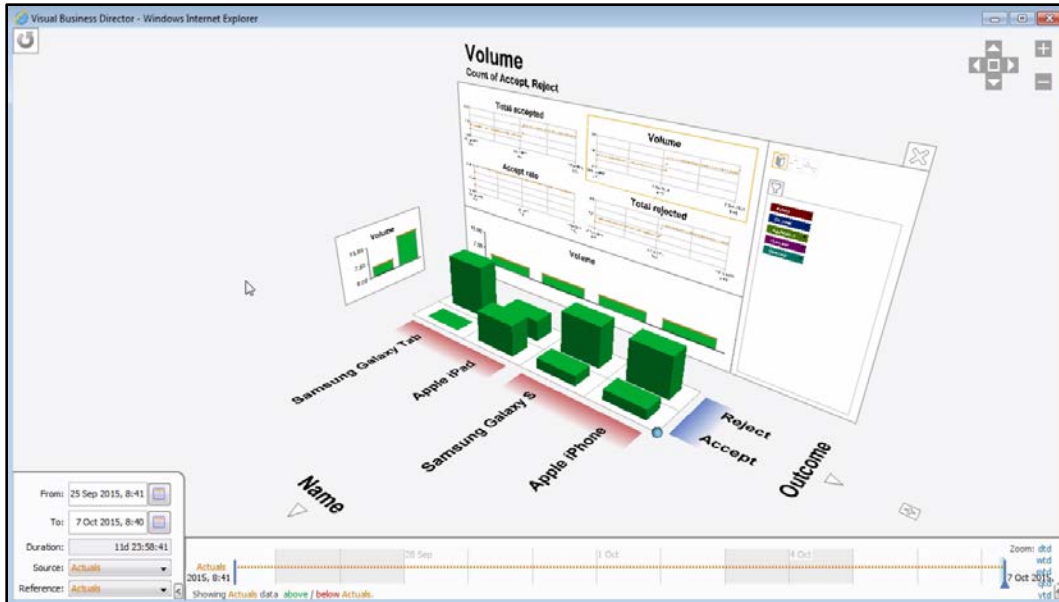
There are 6 dimensions available for the X Axis. These are: Actions, Channels, Applications, Operators, Outcomes and Times.

If, for example, you select Operators from the All Operators dimension, you can instantly see who was most successful in offering the Apple iPhone to customers.

Viewing this alongside Accept/Reject rates, you can quickly pinpoint which sales person or agent is the most successful at offering which propositions.

Let's now add a new phone proposition - the Nokia Lumia.

In this demo, in order to add a new proposition, you need to create a change request.

Open the newly created change request, and then open the Decision Data record, where you can add this new proposition.

The propositions in this Group have three properties: Cost, Points and Price.

Don't forget to save the record.

You can test to see that the new proposition is immediately picked up by the monitoring system.

To do this, re-execute the Top Offers process and accept the Nokia Lumia proposition. Then, examine the results in Visual Business Director.

Note that the Nokia Lumia proposition is immediately available without having to alter the strategy.

You can see that the list of historical interactions with this customer now includes the Nokia Lumia.

The new proposition is also part of the 3-D visualization. This is because Pega Decision Management monitoring and reporting happens in real-time.

The KPIs are also automatically updated with the Accept data for the new proposition.
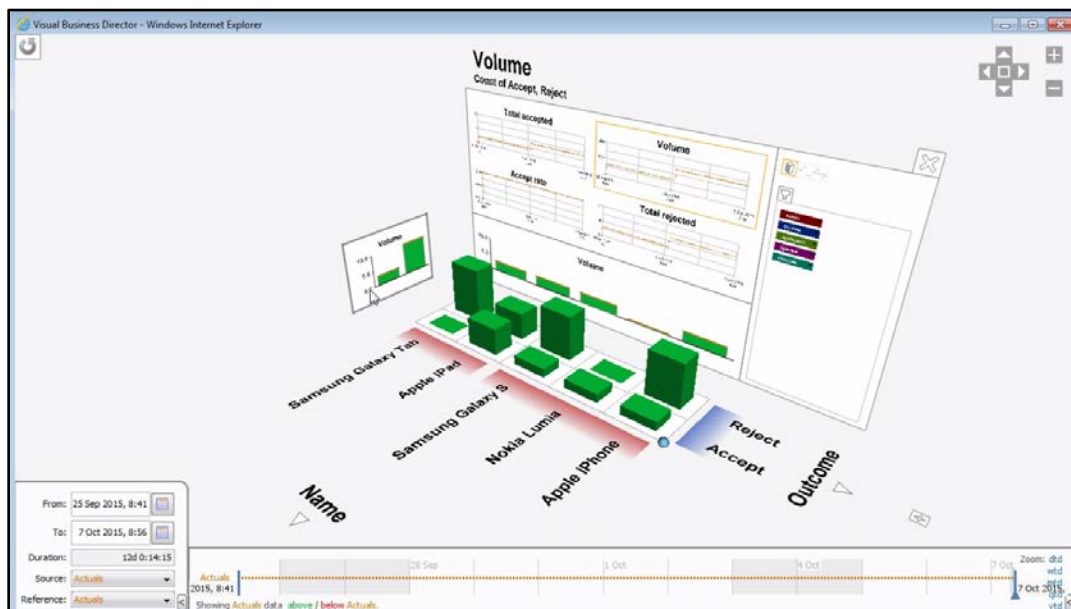
How to Design a Decision Strategy

Introduction to Decision Strategies

The Decision Strategy

Decision Strategies are used to select the best proposition for each customer. If you're familiar with Business Process Management, you may be thinking that Decision Strategies look very similar to Process Flows. However, where a business process comprises a sequence of steps represented by flow shapes, a decision strategy comprises a unit of reasoning represented by decision components. And each decision component adds distinct functionality to the holistic Next-Best-Action decision.
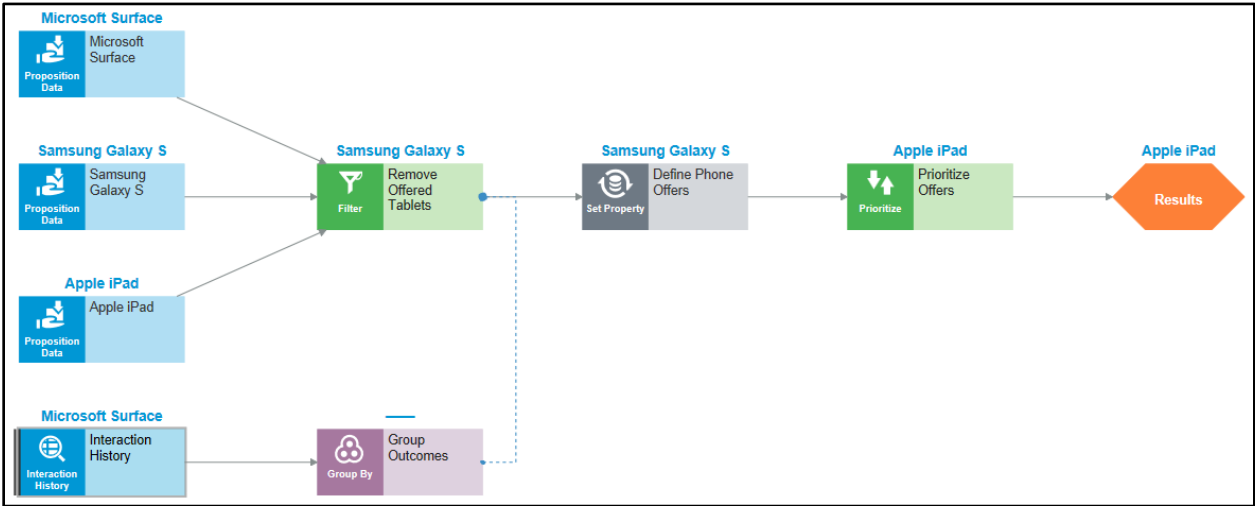
Decision Strategy Components

Decision components are grouped into seven categories.

Import components are used to bring data into the strategy. The data can include propositions or historical information about customer behavior.
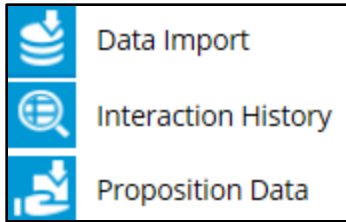
Business Rule components reference logic such as Decision tables and Decision Trees that are used to implement business policies and requirements as well as operational constraints.
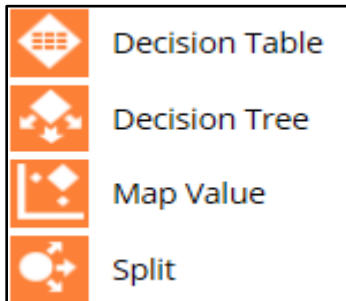
Decision Analytics components reference logic that makes predictions about customer behavior, such as adaptive models, predictive models and scorecards.
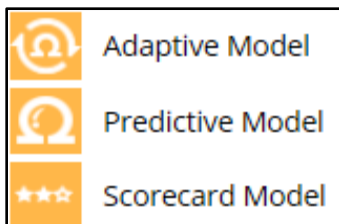

Enrichment components set values for properties that require a calculation, such as the margin on a product. These calculations are defined using the Expression Builder.
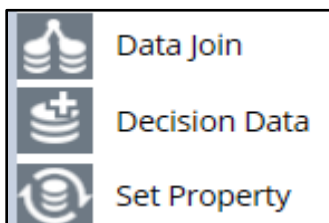
Arbitration components filter out inappropriate propositions and prioritize those that remain by characteristics such as margin or propensity.
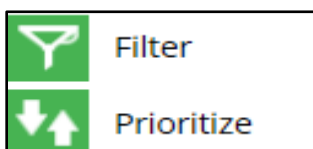
Selection components allow the strategy to switch between different business issues such as Sales, Retention, Service, etc. The Champion Challenger, for example, is used to try out alternative strategies or to compare the performance of different predictive models.
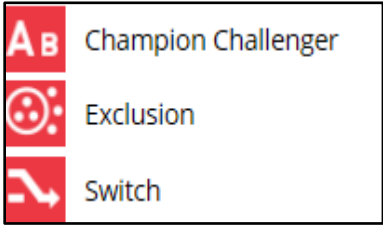
Aggregation components are used to calculate aggregate amounts, such as the sum of all transactions in the past month.
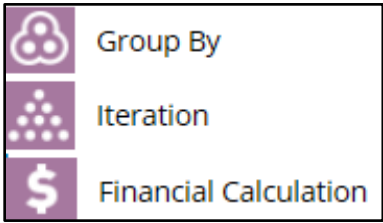
Sub-strategy components enable the strategy to consider the result of other strategies in its calculation, such as the outcome of a Sales or Retention sub-strategy.

Finally, each strategy contains a standard component, the Results component, which defines the output of the strategy. The contents of the Results component can be used as input by other strategies and rules.


Depending on the goal for our strategy, various component combinations can be used to express the reasoning. For instance, if we're trying to create a strategy that will help us retain customers, we could add a Predictive Analytics component to select propositions that will be mostly likely to entice a customer to remain with the company. The individual components and component combinations provide a lot of flexibility in designing decision strategies.

Applications can enrich the number of components available to the strategy designer. For example, Pega Marketing makes three additional components available: Contact Policy, Geofence, and Segment.
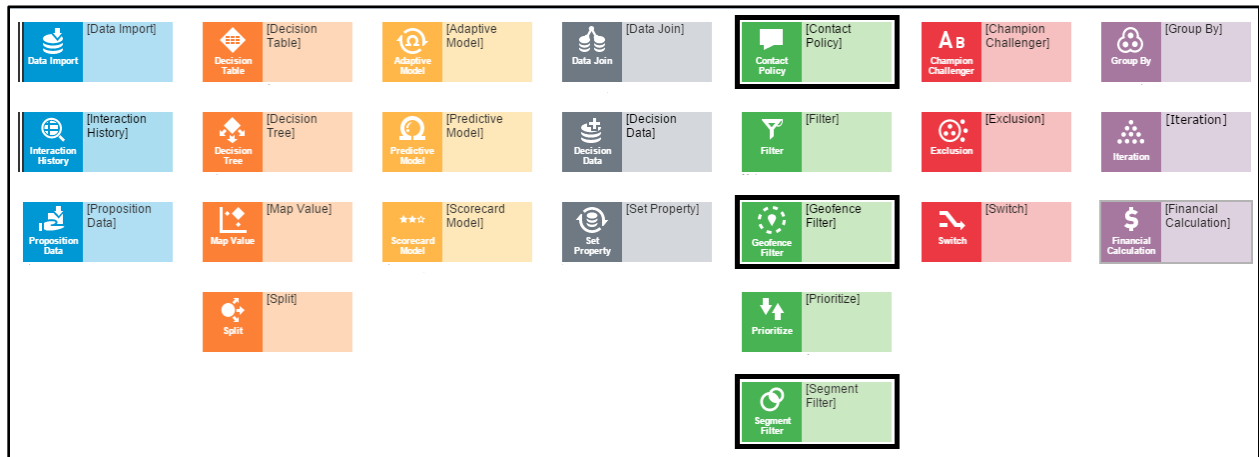

Strategy Flow
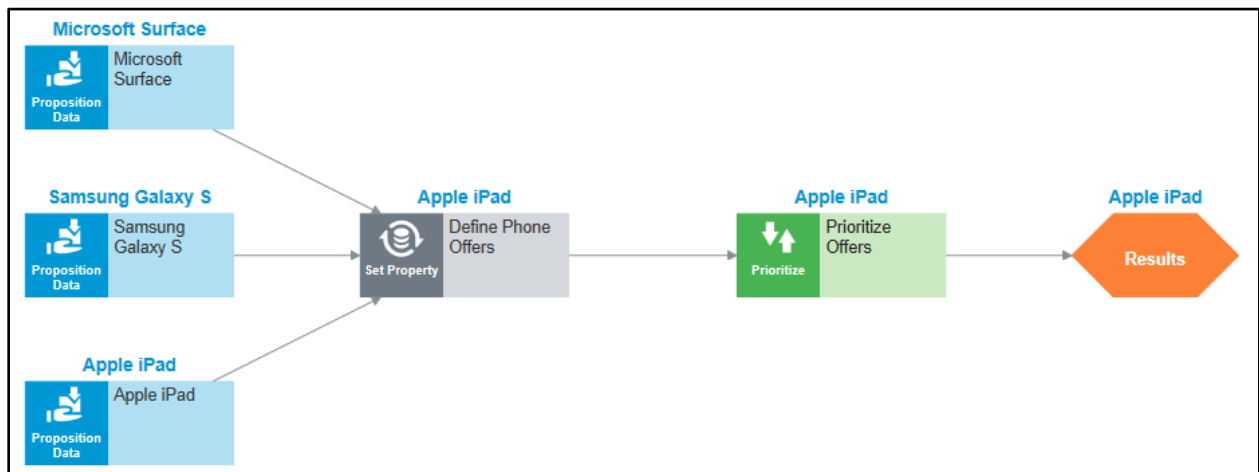
This is an example of a Next-Best-Offer strategy that uses 4 types of components: Proposition Data components, a Set Property component, a Prioritization component and a Results component. This strategy's flow can be broken down into the following phases. First, the propositions we would like the strategy to consider are imported via the Proposition Data components. Then, a Data Enrichment component calculates the margin for each of them. Once the margin is calculated, the arbitration phase begins, where, in this case, the Prioritize component prioritizes the propositions by margin, and the proposition with the highest margin is selected as the Next-Best-Offer. The selected proposition is collected by the strategy Results component where it is typically written to a database table or clipboard page to be referenced by other strategies or business processes.

More realistic strategies have 20, 50, or even hundreds of components. Typically, a Next-Best-Action strategy is broken up into smaller strategies, called sub-strategies, to make it more manageable. As such, most Next-Best-Actions are holistic decisions determined by a collection of sub-strategies. This is called the "Separation of Concerns" design principle. It enables each sub-strategy to address a separate aspect of how the Next-Best-Action is determined. This in turn allows different people to take responsibility for different aspects of the Next-Best-Action decision. The sub-strategies provide their results to the higher-level strategy via the Sub-strategy components.

How Strategies Determine the Next-Best-Action

Here is an example of a Next-Best-Action strategy that uses input from Sub-strategies to determine the Next-Best-Action.
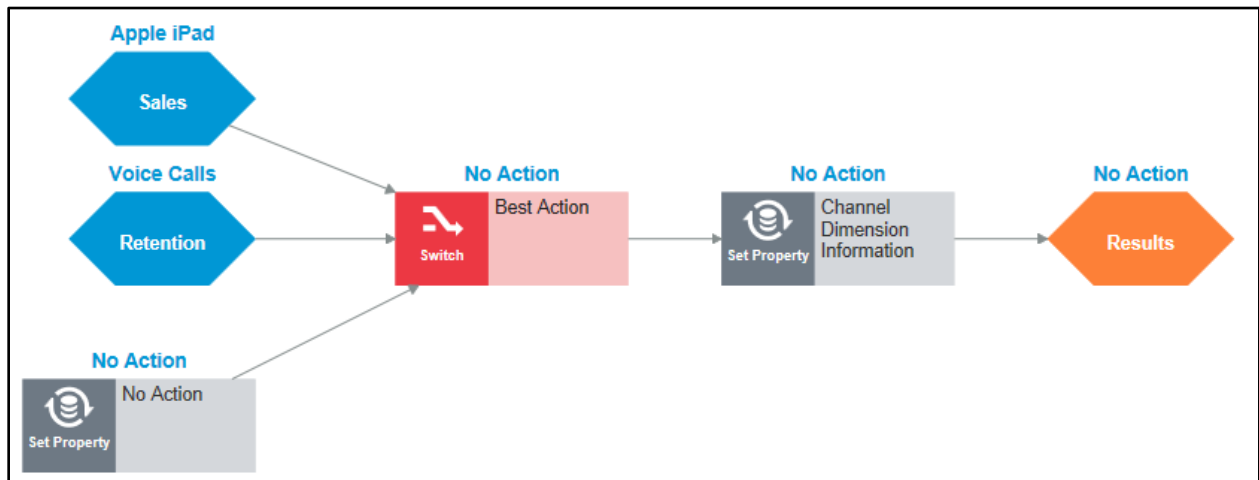
Input from the sub-strategies is delivered via the Sub-strategy components on the right. The sub-strategies make decisions regarding Sales and Retention.

The dotted lines in this image depict how each of the strategies is linked to the others.
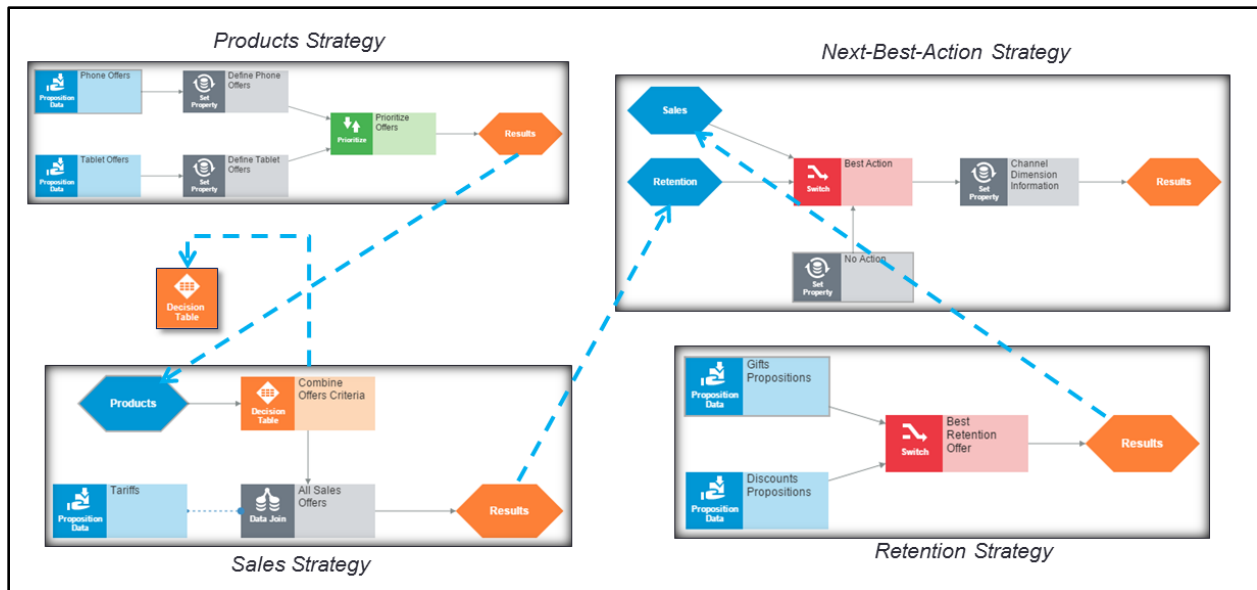
If we drill down further into the Sales strategy, we can see the Product strategy that it references. And in this case, the Sales strategy also references a Decision Table business rule.

The Result of each sub-strategy feeds into the corresponding component of the higher-level strategy. This chain of events takes place for each customer decision, with the final Result being presented by the Next-Best-Action strategy to the agent or customer in the appropriate channel.

Strategy Tabs

Strategy

Under the Strategy tab of our Next-Best-Action strategy you will find the Design Canvas. This is the place where you build and test your strategies. Expand the right-hand side panel and you will see a powerful Test Run panel that enables you to run simulations and test strategies for single and multiple customers. There you can identify the most popular propositions offered to customers, check overall strategy performance, and examine the performance of individual components.
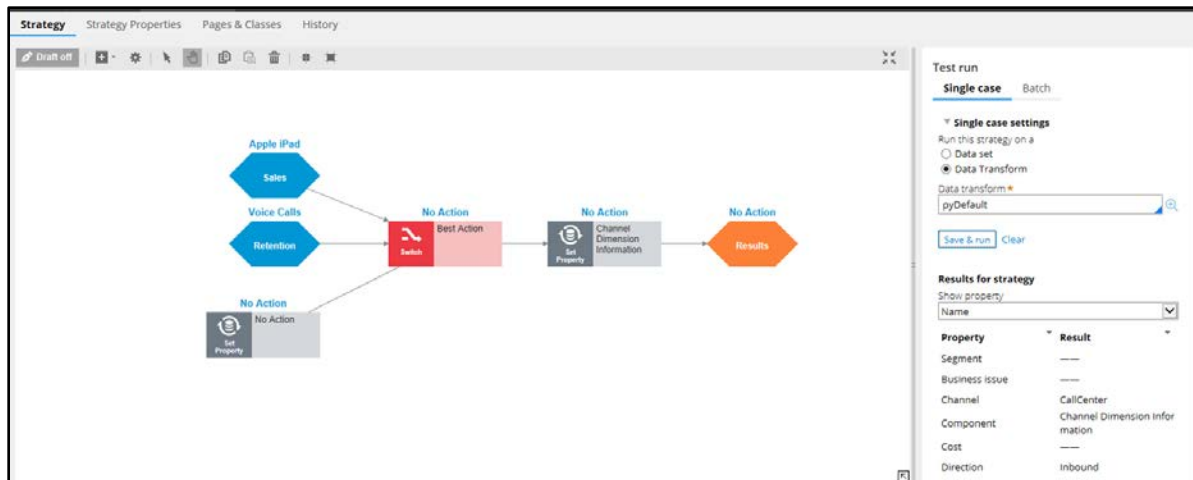
In Single Case mode, the strategy Test Run panel captures the result of a strategy each time it is run. The data values behind each strategy component are visible in the panel. In this example we can see that the Phones group consists of two propositions.

Propositions are listed as Pages that show all of the information that is stored for any particular proposition.

In Batch mode, the strategy Test Run panel enables you test the strategy against real data. You can run the strategy on all records or a limited number of records and view the results based on the highest ranked decision or strategy performance.

Strategy Properties

The Strategy Properties tab contains the underlying data model, or, set of properties that comprises the strategy. Creating and executing the strategy essentially sets values for this set of blank strategy properties.

Edit Strategy: Next Best Action [Available]

CL DSME-Data-Customer ⌄   ID NextBestAction   RS DSMEssenFW:01-01-01

Save ⌄   Check in   Discard   Actions ⌄   ⊗

Strategy   **Strategy Properties**   Pages & Classes   History

Strategy Results class	DSME-DSMEssenFW-SR
Business Issue	
Group	

Strategy Properties

New ...   Refresh

Name	Type	Scope	Action
Cost	Decimal	All DSME-DSMEssenFW-SR Strategies	⊗
pyOriginalComponent	Text	All Strategies	———
pxSegment	Text	All Strategies	———
pxPriority	Double	All Strategies	———
pyLongitude	Double	All Strategies	———
pxInteractionID	Decimal	All Strategies	———
pyComponent	Identifier	All Strategies	———
pyDirection	Identifier	All Strategies	———
pyGroup	Identifier	All Strategies	———
pyDivision	Identifier	All Strategies	———
pxOriginalRank	Text	All Strategies	———
pyName	Identifier	All Strategies	———
pyPreviousComponent	Text	All Strategies	———
pxOutcomeTime	DateTime	All Strategies	———
pyPropensity	Double	All Strategies	———
pyChannel	Identifier	All Strategies	———
pyInteraction	Identifier	All Strategies	———

434

Create a New Strategy

As a Decisioning Architect, you are going to create a stub for the Next-Best-Offer strategy and ensure that the strategy will be used by the TopOffers process.

In Decision Management, all of your strategies will be created in the context of making customer-related decisions.

So you need to switch to the Customer data class, which is where all the decisioning artifacts for your application live.

If you expand the Decision category, you can see the 3 Decision rule types that are currently in use in this application: Decision Table, Interaction and Strategy.
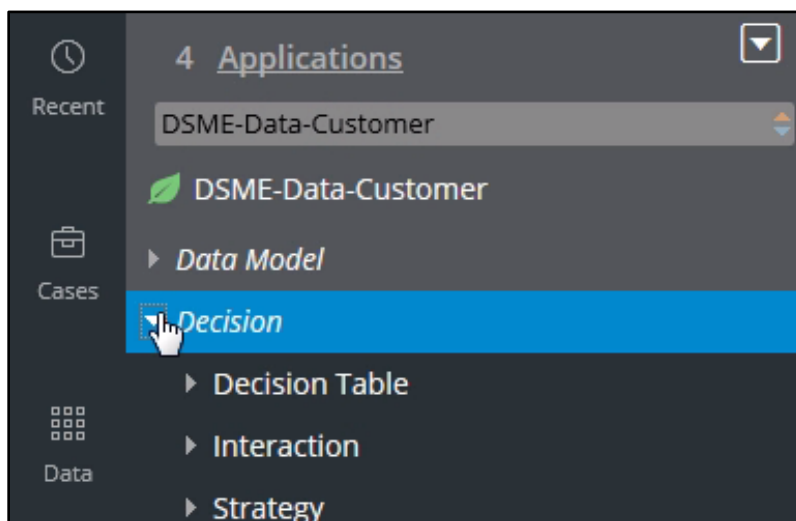
If you open the Strategy rule type, you can see the different strategies that exist in the system. Right-click on the Strategy menu and click Create to create a new strategy.

Next, give your strategy a name in the Label field. A unique identifier will be generated by the system for it. This will allow you to change the label name later if needed.

The Business Issue and Group of the Strategy Results class will limit the scope of the propositions the strategy will be able to consider.

By selecting the Sales Business Issue, the strategy will filter out all propositions that are not Sales propositions.

Next-Best-Offer strategies include a reference to DSME-Data-Customer. This is what we call the Applies-to class, which is always the Customer class.

The creation of the strategy stub is now complete. The actual development of the strategy will be completed by a Strategy Designer.

Next you need to update the Interaction rule. The Interaction rule is what the Top Offer process uses to request a decision from the Next-Best-Action strategy.

A business process calls an Interaction rule, and the Interaction rule invokes the strategy. This ensures that the entire business process does not need to change if you just want to change the strategy.

Within the Interaction rule, you can see that it currently refers to a Next-Best-Action strategy. But this needs to be changed to your new Next-Best-Offer strategy.

When you check the interaction rule back in, it becomes available to other users in the system. So it's important to write an appropriate comment to indicate to others what you've done.

Now when you run the business process, your new strategy will be executed.

Building a Strategy

In your role as the Strategy Designer, you are going to learn how to use decision components by building the Next-Best-Offer strategy.

Start by opening a change request detailing the work to be done. Now open the strategy.

Notice here that the complete definition of the Next-Best-Offer strategy includes a reference to DSME-Data-Customer.

This is what we call the Applies-to class. This classification indicates the context of the strategy.

This ensures that from within the strategy, you have access to customer-related properties, such as Age, Address, Name etc.
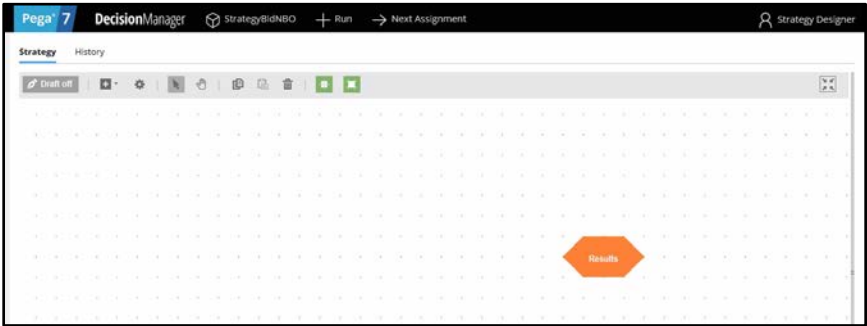
You can now start building the strategy. By right-clicking on the canvas, you get this context menu, which shows all of the component categories. The first component you need to add is an Import component.

By expanding the Import category, you can see the 3 available Import component types. In this case you need a Proposition Data component.
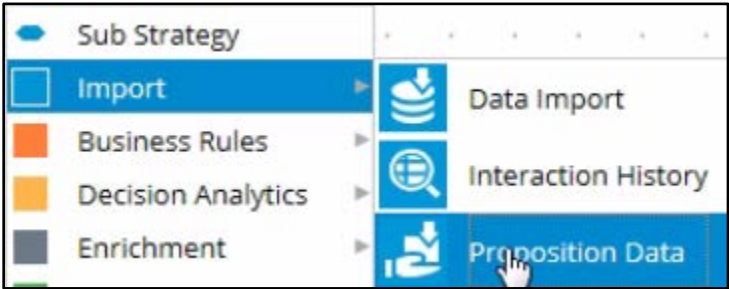
Now you need to configure the component. First, right-click to open the Proposition Data properties panel. Notice that the Business Issue is grayed out. This cannot be changed because Sales was previously selected as the Business Issue for this decision strategy.

As you can see, by default, the strategy will import all of the propositions within that Business Issue. But you can select a specific proposition Group. For this component, you're going to import only the Nokia Lumia, so let's select that.

Selecting the proposition from the drop-down menu automatically gives the component an appropriate name. The description, which will appear on the canvas, will also be generated automatically. If you want to create your own description, you can do so by clicking the 'Use custom' radio button.

You want to import 2 more propositions into the strategy. You can use the Copy and Paste buttons to quickly add two more Proposition components to the canvas. First, Select Nokia Lumia, then click Copy and Paste. Also, by clicking the highlighted button you can turn the grid display on or off.

This button is for "alignment snapping". By turning it off, you can place a component anywhere on the canvas, but it makes it more difficult to align shapes.

Now you need to add the next component in the strategy, which is an enrichment component called Set Property. You can add this component to the canvas by selecting it from the component menu.

Ultimately, we want the result of this strategy to be the proposition with the highest margin. The Set Property component is where we will calculate the margin for each of the propositions.

The information in the 'Source components' tab is populated automatically by the Proposition Data components connected to this component.

On the Target tab you can add properties for which the values need to be calculated. Click Add Item to create the equation that will calculate margin for each of the components.

Begin by setting the Target property to Margin. Then you need to make Margin equal to the calculation you create. To create the calculation, click on the 'Expression Builder' icon next to the Source field.

Using the Expression Builder, you can create all sorts of complex calculations, but we're going to keep it simple. Notice that we began each of the inputs with a dot. This is called the dot-operator, and it's unique to Pega.

Typing a dot before each input means that you want to use a strategy property to calculate the value of your expression. And when you type the dot, a list of available and relevant strategy properties appears.

This not only makes it easy to quickly find the property names you're looking for, it also avoids spelling mistakes.

But what if you want to use a customer property in your expression, such as Age or Name? In that case, you would have to type the prefix "Primary dot", instead of just dot. "Primary dot" accesses the properties in the "Applies-to" class to which your strategy belongs.

Even though you used the dot-operator to build your expression, it's best practice to validate it, so click Validate.

If the expression wasn't valid, you would receive an error message on screen.

On the canvas, you can see the automatically generated description for the component: Sets Margin using Points and Price.
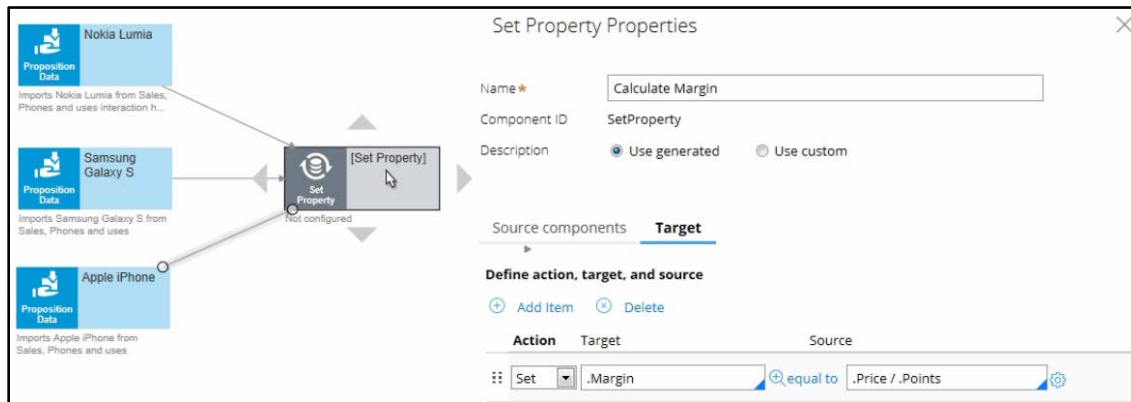
Now you want to ensure that the Offers will be prioritized based on their margin. So you need to add the Prioritize component from the Arbitration category.

The prioritization can either be based on an existing property or it can be based on an equation. Let's select an existing property using the dot construct.

Here you can select the order in which the top Offers are presented. You can also select the number of Offers that will be returned by the strategy.
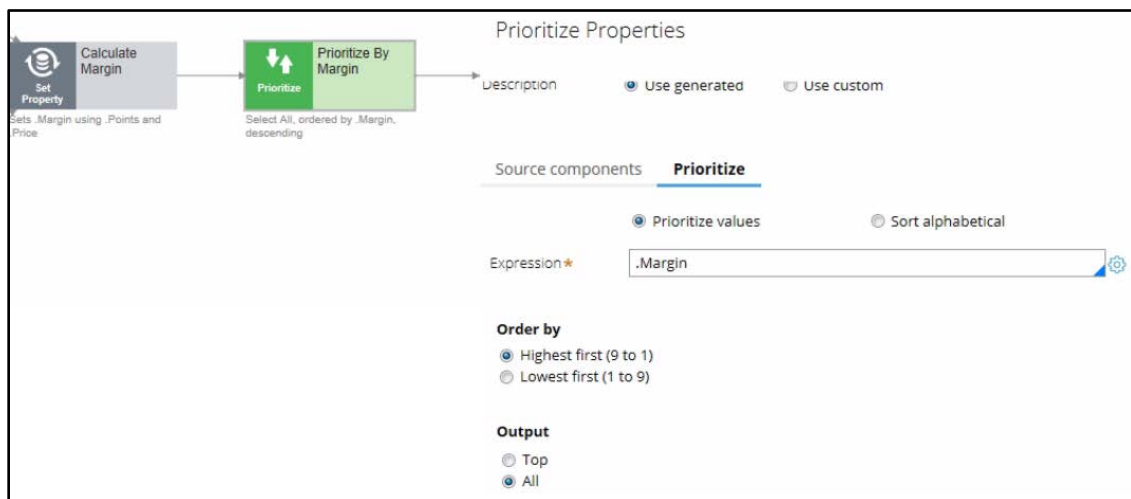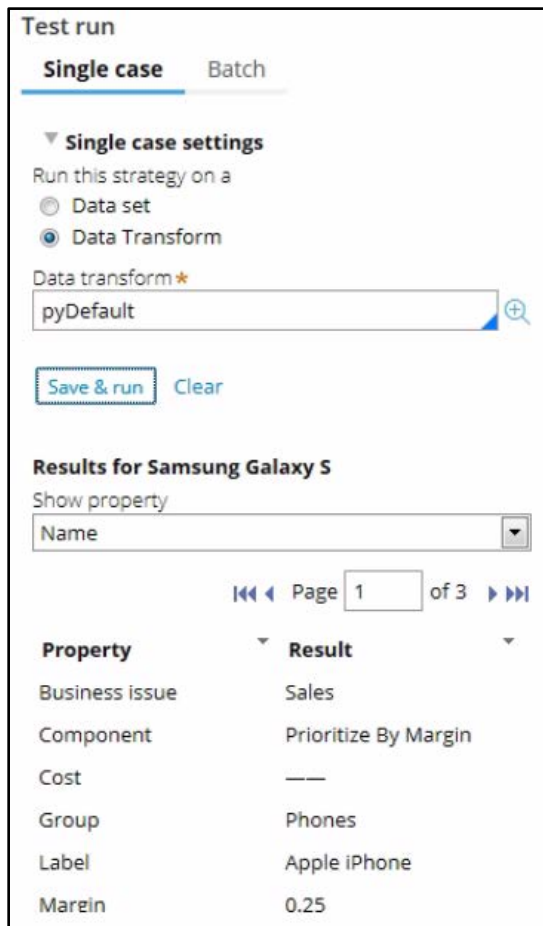
Now you can save and test the strategy. Start by expanding right-hand side panel. The test case can be defined and stored in something called a Data Transform.

You will currently execute this strategy against a Data Transform called pyDefault. But you can also select a Data Set that points to an actual live database table. Click 'Save & run' to examine the results.

In this case the result contains three propositions, which are presented as pages. The properties listed for each proposition's page are the Margin and the Points and Price used to calculate it.

The Points and Price come directly from the live product catalogue, and the Margin is calculated on-the-fly in this particular strategy. As you can see, the Margin values for the second and third propositions are lower than for the first proposition.

You can also view results for any of the components by selecting that component.

You can now execute the Top Offers process to verify your results. Note that the proposition with the highest margin is listed first.