Pega Platform '24.2

3 September 2025

CONTENTS

Rules management

A Rule is a basic building block of an application that defines its behavior. Pega Platform applications contain many types of Rules that specify different types of behavior. Because Rules are reusable, by implementing them, you can save time and resources when you build your application.

Rules define the display of forms, the UI fields, the flows for work processes, and multiple other software elements. The system can reuse Rules throughout your application. For example, in an application for ordering replacements parts, you can define a UI element to capture an address, and then reuse the Rule to capture the mailing address and billing address for the order.

You configure Rules to create a business solution for your organization and customers. Rules provide flexibility during the development process and help you design applications more efficiently so that you can implement the Rules again in future projects.

For learning materials about Rules, see Creating a Rule module on Pega Academy.

- **Creating Rules**
- **Organizing Rules into Rulesets**
- **Organizing Rules into classes**
- **Rule resolution**
- **Defining and changing the Scope of Rules**
- **Preparing for modifying Rules**
- **Exploring Rules in your application**

Related concepts

- Automating work by creating Case Types
- Designing the UI in Case Types
- Traditional UI

Creating Rules

To save time and ensure that your projects meet the needs of your clients, create Rules and data instances using reusable elements of your application. By combining a Rule Type, name, class, and Ruleset, you provide a unique identity for each Rule that you create.

1. In the header of Dev Studio, click Create, and then select the record category and type that you want to create.

   For example:

   To create a Data Transform, click Create > Data Model > Data Transform.

2. In the record configuration area of the **Create** form, enter a name for your record and define its key parts:
   a. In the Label field, enter a short description in the form of a sentence that describes the purpose of the record.
      As a best practice, use up to 30 characters. Pega Platform™ appends Rule information to the Rule name that you enter to create the fully-qualified name.
   b. **Optional:** To manually set the name key part of your record to a value that differs from the default, in the Identifier field, click Edit, and then update the name.
      The default for this field is To be determined. The system populates this field with a read-only value based on the sentence that you enter in the Label field. The system ignores spaces and special characters.

If you manually change the Identifier value, the system no longer populates this field after you enter a new value in the Label field.

c. In the remaining fields, enter additional key parts for your record.
   The number of key parts, types, and restrictions varies by record type. As a best practice, start each key part with a letter and use only letters, numbers, and hyphens.
d. **Optional:** To include other configuration options that this record type supports, click View additional configuration options.
   These options vary by record type and appear only for records that support Quick Create options. By using the Quick Create option for certain Rule Types, you can create Rules in the **Create** dialog box without having to open the Rule form. Rule Types with this option include properties, field values, When Conditions, flows, and activities.

3. In the **Context** section, if the Development Branch list appears, select a Branch in which to store your record:
   - To create the record in a branched version of the Ruleset, select a Branch name.

     If the branched Ruleset that you provide does not yet exist, the system automatically creates the Ruleset when you create the Rule.

   - To create the record in an unlocked Ruleset version, select [No Branch].

   The form displays the Development Branch list when you define Branches in the current application, or in one of the built-on Application Layers.
4. Select an Application Layer in which you want to store the record.
   You can store records only in Application Layers that have access to the selected development Branch.

NOTE: The Production Rulesets option appears as the first option in the stack if your current Access Group has production Rulesets. Choosing this option restricts only the Add to Ruleset field.

5. In the Apply to field, select the class to which this record applies.

   By default, the system populates this list with the Cases and data types that your chosen Application Layer can access. Choose the lowest class in the class hierarchy that serves the needs of your application.

   > **For example:**
   >
   > Choose *MyCo-LoanDiv-MortgageApplication* rather than *MyCo-LoanDiv-* as the Apply to class for a new flow or property, unless you are certain that the record can apply to all objects in every class derived from *MyCo-LoanDiv-*.

   > **TIP:** To select a class name that is not a Case or a data type, click the View all link.

6. In the Add to Ruleset field, select the name of a Ruleset that you want to contain the record.
   If the development Branch is `[No Branch]` or you have no available Branches to choose from, specify a version for the specified Ruleset name.
7. **Optional:** To override the default work item that your application associates with this development change, in the **Current work item** section, in the Work item to associate field, select a work item.
   For more information about your default work item, see Setting your current work item.
8. Click Create and open.

> **What to do next:**
>
> Complete and save the record form on the next screen that opens after you click Create and open.

- **Copying a Rule or data instance**
- **Naming conventions for records**

- Rule creation
- Setting Rule status and availability
- Searching for a Rule
- Configuring system settings

Copying a Rule or data instance

Save time and speed up the development process of your application by promoting reuse of resources. Instead of creating a new record that might be similar to a Rule that already exists, copy an existing Rule and then make the necessary changes.Consider a scenario in which you copy a Service-Level Agreement Rule that defines goals and deadlines for customer service representatives (CSRs) to follow when CSRs review loan requests, and then change some elements of the Rule to adjust it to reviewing mortgage requests. For example, you can make the goal longer.

Before you begin:

You must have the *@baseclass.ToolbarFull* or *@baseclass.ToolbarSaveAs* privilege to copy Rules.

NOTE: You cannot create a new Agent Rule by copying a rule or a data instance. Pega strongly encourages clients to complete activities run as background process processes using a Queue Processor or Job Scheduler instead of Agents. For more information, see Using job schedulers and queue processors instead of legacy Rule-type-agents.

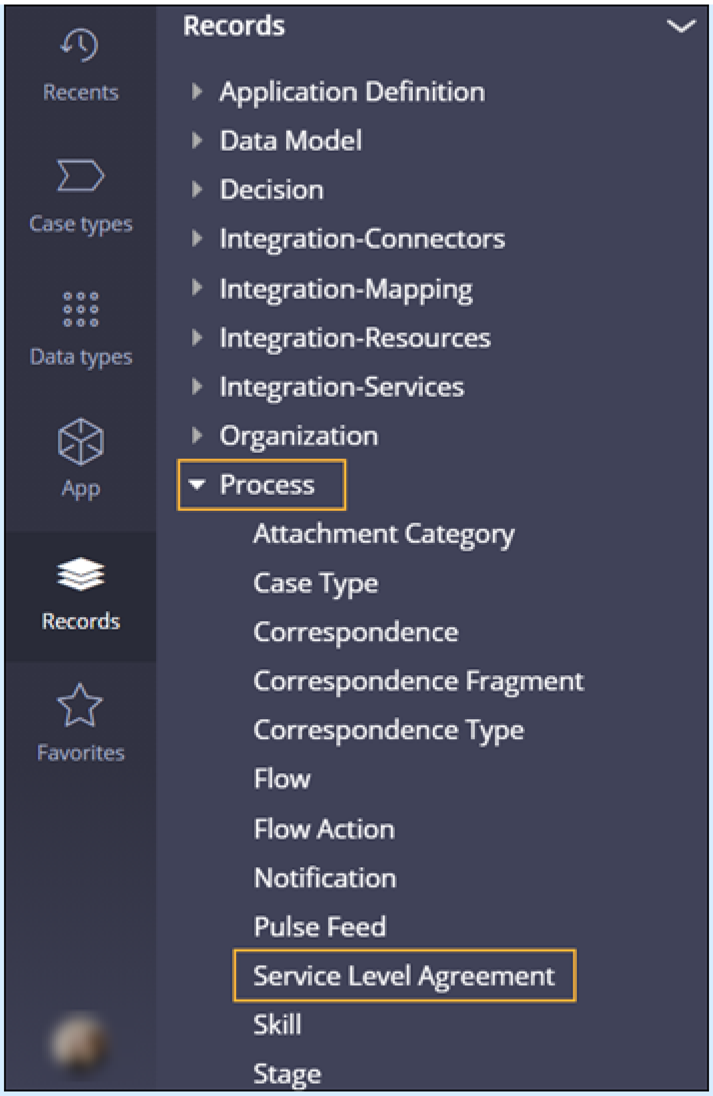
The result of copying an existing Rule is a new record with key parts that the system prepopulates with the values of the original Rule. To save the new record, you need to change at least one key part. The key parts are an identifier, a Branch that stores the record, a class that applies to the record, a Ruleset, and a Ruleset version.

NOTE: Because the Save As form displays a subset of the fields and functionality that are specific to each data instance, some fields might vary between different Rule Types.

1. In the navigation pane of Dev Studio, navigate to the record that you want to copy.

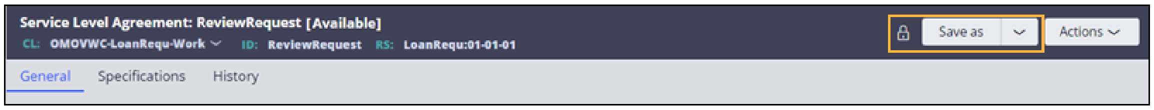
   For example:

   To copy a Service-Level Agreement Rule, click **Records** > **Process** > **Service-Level Agreement**, as in the following example:



Navigating to a record

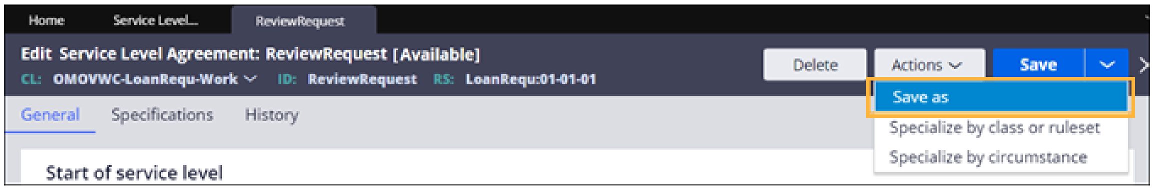
2. In the list of instances, open the instance that you want to copy.
3. In the Rule form header, select an action to copy the Rule:
    - If the record is in a locked Ruleset, click **Save as**, as in the following example:



Service Level Agreement: ReviewRequest [Available]
CL: OMOVWC-LoanRequ-Work ID: ReviewRequest RS: LoanRequ:01-01-01

Copying a Rule in a locked Ruleset

    - If the record is in an unlocked Ruleset, click **Save** > **Save as**, as in the following example:


Home
Service Level...
ReviewRequest
Edit Service Level Agreement: ReviewRequest [Available]
CL: OMOVWC-LoanRequ-Work ID: ReviewRequest RS: LoanRequ:01-01-01
Delete
Actions
Save
Save as
Specialize by class or ruleset
Specialize by circumstance
General
Specifications
History
Start of service level

Copying a Rule in an unlocked Ruleset

4. Review the record label and identifier to determine which key parts you can change.
   Consider the following factors:
   - If you plan to leave the context for this record unchanged, change the identifier.
   - If the record type that you want to copy has name restrictions, the Identifier field appears as not editable.
5. In the **Context** section, review the record context to determine what information about the storage of the record in your application you can change.
   Consider the following factors:
   - If you plan to leave the label and identifier unchanged, change at least one element of the context of the record. You can change a Branch that stores the record, a class that applies to the record, a Ruleset, and a Ruleset version.
   - If you copy a Rule with Final availability and leave the key parts unchanged, select the same Ruleset name and a higher version number. By default, you cannot override a Final Rule with another Rule in a different Ruleset.
   - When you copy a Rule, you cannot define the Rule availability. The system clears the Rule availability when the record identifier is unchanged, and the Rule meets any of following conditions:
     - the Ruleset is unchanged
     - the class that applies to the record is unchanged
     - the original record is specialized by circumstance

     For more information about Rule availability, see Setting Rule status and availability.

6. **Optional:** To override the default work item that your application associates with this development change, in the **Current work item** section, in the Work item to associate field, select a work item.
   For more information about your default work item, see Setting your current work item.
7. Click Create and open.

   > **Result:**
   >
   > The Rule form opens.

> **What to do next:**
>
> Make the necessary changes and then save the Rule form.

- Organizing Rules into Rulesets
- Organizing Rules into classes
- Rule resolution
- Defining and changing the Scope of Rules
- Creating a Rule specialized by circumstance
- Troubleshooting newly created Rules
- Creating Rules

Naming conventions for records

Give informative and descriptive names to the records that you create in Pega Platform™, such as Rules and data instances, to help you manage your resources and provide quick context for other developers. As a result, you speed up application development and promote reuse across your application.

For example, when you create a Ruleset to store Service-Level Agreement Rules for an HR department in your organization, name the Ruleset `SLAforHR` so that other developers can understand the purpose of the Ruleset and avoid duplicating the record.

As you create new items in your application, it is important to implement consistent and logical labeling. This approach makes it easier for members of an organization or a development team to perform the following tasks:

- Find and identify records as you build out applications and configure changes and enhancements.
- Convey the meaning of the development process and understand the structure of the application.
- Avoid record duplication and enhance processing efficiency and developer productivity.

Before creating a record, consider the following actions:

- Look for an existing record, standard or custom, in your application that delivers the required capability or that you can copy and modify to reduce development time.
- Assess the likelihood of reuse of the record, as this assessment helps you determine where to place the record in the class and Ruleset hierarchies.

General guidelines for record naming conventions

When you create new records, consider the following naming conventions:

- Choose names that reflect the purpose of the record.
- Keep names short but long enough to make the purpose of the record evident.
- For readability, start all names with a capital letter. Start additional words within names with capital letters. Capitalize any acronyms in names.
- Do not include underscores or other special characters, except as noted in particular scenarios.
- When you create records, the system automatically generates the record identifier by using text that you enter in the **Label** field. The system removes spaces and

special characters that you include in the label and capitalizes the first letter of each word in the identifier.
- Names of records that are part of specific functional groups can start with the Function name, or an abbreviation of the name, followed by an underscore, and then a record name. Do not use this naming convention for Rules from which other Rules inherit. For example, names of Data Pages start with `D_` characters.
- Start the names of active Rules, such as activities and flows, with a verb indicating the main action of the Rule. Include a noun or noun phrase that indicates the element on which the Rule operates.
- For passive or object records, such as parameters, properties, work parties, libraries, and roles, choose nouns or noun phrases that describe the main purpose of the Rule. For example, `ProjectGoalDate`, `ClaimID`, or `DefenseAttorney`.
- Use hyphens in class names to control pattern inheritance. For example, when you name a class `SLAforHA-ReviewClaims`, you indicate that the parent of this class is the `SLAforHA` class.

Limitations on name lengths for Rule Types and objects

When you create objects and Rules of different types, consider the following limitations:

- A Ruleset name that uses *pyRuleSet* property can have the maximum length of 128 characters.
- An operator ID value that uses the *pyUserName* property can have the maximum length of 128 characters.
- The internal key of an object that uses the *pzInskey* property, can have the maximum length of 255 characters for most tables. Longer file names cause compilation issues in Windows environments.
- A class name that uses the *pxObjClass* property, for a class you create can have the maximum length of 56 characters. The system generates parallel `History-` classes for most classes that you create, and consequently reaches the 64 character limit for class names.

- The visible key that uses the *pxInsName* property can have the maximum length of 128 characters.
- The short description for a record that uses the *pyLabel* property can have the maximum length of 64 characters. As a best practice, enter a maximum of 30 characters.
- A property name can have the maximum length of 64 characters.
- The system name, that is a key of the *Data-Admin-System* data instance, can have the maximum length of 32 characters.
- A node name can have the maximum length of 32 characters.
- An Assignment name can have the maximum length of 32 characters.

NOTE: When you save your changes, the system does not always enforce size limits. Names that exceed the defined system limits might cause unusual or unexpected behavior.

Naming conventions for flow actions and Assignments

Use noun and verb combinations for flow actions and Assignments, for example `SubmitPurchaseOrder` and `ReviewJobApplication`. Ensure that names are meaningful to users as the text appears in numerous locations in the end user interface, for example Worklists and menus in user Portals.

Additionally, flow actions can reference other records. When you associate a flow action with an activity or validation record, prefix the flow action name with one of the following prefixes:

- For pre-activities, use `Pre`, for example, `PreAcceptCorrespondence`.
- For post-activities, use `Post`, for example, `PostAcceptCorrespondence`.
- For validation records, user `Val`, for example `ValApproveCorrespondence`.

When you name sections and privileges that are related to flow actions, apply the same guidelines as for flow actions.

Naming conventions for Access Groups

Choose a name that is unique system-wide, both in the current system and in other Pega Platform systems that may later host your application. Ensure that the Access Group name contains one colon character. Use an `ApplicationName:RoleName` format, for example, `BakingOperations:Customer` or `LoanRequests:Administrator`. As a result, you ensure that the name is unique across multiple applications. Begin the name with a letter and use only alphanumeric, ampersand, dash characters, and a single colon.

For more information about Access Groups, see Learning about Access Groups.

Naming conventions for data objects

Data objects are classes that contain the information such as properties and Data Pages that are necessary for a Case to accomplish actions or achieve business outcome. For example, in a banking application, a Customer data object might include properties such as Name, Phone number, and Address, so that you can conveniently identify and contact customers. Use nouns that clearly communicate what a data object includes when you name data objects.

For more information, see Creating a data object.

Naming conventions for database objects

Database objects include tables, views, indexes, and so on created in the Pega Platform database or other databases that a Pega Platform application uses. In general, name database tables in accordance with the location of the respective work or data class in the *Customer* Enterprise Class Structure (ECS). Names of standard Pega Platform database tables begin with `pr_`, `pc_`, or `pi_`. Numerous use cases require creating your own tables to supplement the standard tables. As a best practice, use the application/ Ruleset name as a suffix, for example `MyApp_`, and to try to resemble the existing name. For example, name the work table `MyApp_work`.

For database objects, use the same name as the class or class group and use one of the following ending suffixes:

- `TB` for a table
- `IN` for an index
- `SY` for synonyms
- `SQ` for a sequence

Work tables

Map work tables on the implementation layer so that multiple lines of business (LOB) can use the same framework application to map to different database tables. Name work tables by using the following format: *LOB name*`_work_`*workpool name*. Ensure that abbreviations of line of business name and workpool name match the abbreviations in the *Customer* ECS, except for DB2 table names.

The following example include sample work tables names:

- For ABCD line of business and Order Intake workpool, name the table `abcd_work_orderintake`.
- For Claims line of business and Contracts workpool, name the table `claims_work_contracts`.

Work history tables

Name work history tables by using the following format: *LOB name*`_work_history_`*workpool name*. Ensure that abbreviations of line of business name and work pool name match the abbreviations in the *Customer* ECS. For example, for ABCD line of business and Order Intake workpool, name the table `abcd_work_history_orderintake`.

Data tables

Name division-wide data tables by using the following format: *LOB name*`_data_`*domain name*. For example, for a Claims line of business and Provider domain, name the data table `claims_data_provider`.

Name enterprise-wide data tables in the following format: `ent_data_`*domain name*. For example, for a Networks domain, name the data table `ent_data_networks`.

Ensure that abbreviations of line of business name and domain name match the abbreviations in the *Customer* ECS.

Constraints

Name constraints on a database tables that contain work or data by using the table name and suffix _PK, which stands for `primary key`. For example, for a `claims_work_contracts` database table, name the constraint `claims_work_contracts_pk`.

Naming conventions for files

For records that derive from *Rule-File-* class, select a name that uses only lowercase characters to avoid confusion between systems which have case-sensitive file names systems which do not have case-sensitive file names. Begin the names of files for related Functions with a common prefix character so they appear grouped together in listings.

Naming conventions for flows

Name flows by using a combination of a verb and a noun that clearly describes a purpose of the flow, for example `Submit an order`.

For more information, see Case Lifecycle elements.

Naming conventions for Functions

Follow Java naming standards, which include initial lower case, then camel case, for the Function Name key part of a Function record, because Functions define Java Functions.

Naming conventions for HTML streams

For an HTML stream, choose a noun or a noun phrase that indicates the main data that the stream displays. If possible, relate the name to the name of the activity or action that displays the stream. For example, `WorkSummary`. Use camel case in HTML stream names.

Naming conventions for router activities

Start the name of a router activity with To and end in a noun phrase that indicates the route destination, such as `ToWorkbasket`. The following table contains sample router activities:

Routing activity	Purpose
`ToAgent`	Assign to agent entered in parameter.
`ToCorrPartyRole`	Assign to Correspondence party role.
`ToCostCenterManager`	Assign to operator designated as the manager of the cost center.
`ToDefaultWorkbasket`	Assign to the current operator's default Work Queue.
`ToOrgUnitManager`	Assign to manager of the current operator's organization unit.
`ToOverallSLA`	Assign to a list for someone to monitor the service level. This is a placeholder activity for copying to your Ruleset and modifying to reference your own service level Work Queue. Invoke the standard *Work-.OverallSLA* routing when the work object is created.

Routing activity	Purpose
`ToProblemFlowOperator`	Assign to designated operator responsible for investigating flow issues.
`ToWorkbasket`	Assign to the Work Queue specified as the parameter.
`ToWorklist`	Assign to the operator specified as the parameter.
`ToWorkParty`	Assign to the appropriate work party.

For more information, see Assigning tasks to users.

Naming conventions for service packages

A service package supports the development of services that your application offers. A service package name becomes the first key part of a set of service records that are packaged and deployed together. Create a service package data instance for each service. As a naming convention, use the following format:

```
<application name><service type><optional deployment type><optional unique part
>
```

The following list includes sample service package names:

- `LOANSSOAP`
- `LOANSEJBV2`
- `LOANSEMAIL`
- `LOANHTTPA`, `LOANHTTPB`, `LOANHTTPC`

For more information, see Service Packages.

Naming conventions for Work Queues and work groups

Use the name that clearly identifies the purpose of the Work Queue followed by the suffix `WQ`, for example, `AnalystsWQ` or `UnderwritingWB`.

Work groups usually identify a group of workers that have similar process responsibilities. Use a name that clearly identifies the purpose of the group followed by the suffix `Group`, for example, `HelpDeskGroup` or `BenefitsGroup`.

For more information, see Creating a Work Queue and Creating work groups.

Naming conventions for specific records

For more information about naming conventions for records, refer to the specific documentation for each record:

- Creating constraints Rules
- Creating Declare Expression Rules
- Creating a privilege
- Creating a ticket
- Creating a When Rule
- Creating Rulesets
- Naming conventions for classes
- Naming conventions for properties
- Creating an activity
- Case Status values
- Configuring a Data Transform
- Updating the organizational structure by using the organizational chart
- Creating an operator ID
- Case Types
- Stages in a Case Lifecycle
- Processes in a Case Lifecycle
- Steps in a Case Lifecycle

Naming conventions for other objects

For indexes, views, triggers, and stored procedures, use the same naming criteria as for activities. Start with a verb that indicates what the database object does, followed by a noun or a noun phrase that indicates what the view does. End the name with the appropriate suffix. Maximum length for such names is 15 characters. The following list includes the suffixes that you can use when you name the objects:

- `vw` for views
- `tr` for triggers
- `in` for indexes
- `rp` for reports
- `sp` for stored procedures
- `te` for trigger events, with further distinctions:
  - `te_ai` for After Insert events
  - `te_bi` for Before Insert events
  - `te_au` for After Update events
  - `te_bu` for Before Update events
  - `te_bd` for Before Delete events

For example, database view to retrieve employee Data Records has the name `get_employee_data_vw`.

NOTE: Avoid using the prefixes `pwvb_` or `pcv_`, because the system uses these prefixes in standard tables.

- **Enabling Unicode character support for object names**

Related concepts

- Rules management
- Organizing Rules into Rulesets
- Organizing Rules into classes

Enabling Unicode character support for object names

Enable the use of Unicode characters for object names by enabling Unicode character support in Pega Platform™. App Studio supports Unicode characters in the names of properties, Case Types, and data types. An administrator can enable support for Unicode characters, including multibyte characters.

Before you begin:

Support for naming Rules by using Unicode characters is designed for use in Pega Cloud® services. If you want to enable this feature locally, your application server and database must support UTF-8.

The *EnableUnicodeRuleNames* dynamic system setting (DSS) in Dev Studio determines whether a user can name objects by using Unicode characters. An administrator must configure this setting as part of the system setup. The default value for the DSS is false. If the value is false and a user enters characters that require Unicode support, the system displays an error message.

1. In the navigation pane of Dev Studio, click Records.
2. Expand the SysAdmin category, and then click Dynamic System Settings.
3. In the list of DSSs, click Pega-RulesEngine RuleManagement/ EnableUnicodeRuleNames.
4. On the Settings tab, in the Value field, enter `true`, and then click Save.
5. Test the setting by entering names that require Unicode character support:
    a. In the navigation pane of App Studio, click Case Types.
    b. Click New.
    c. In the Create Case Type field, enter the following Kanji characters: □□□□□□□□
    d. Click Next, and then, add Stages to the Case Type and enter Stage names in Kanji characters.

e. Click **Save**.

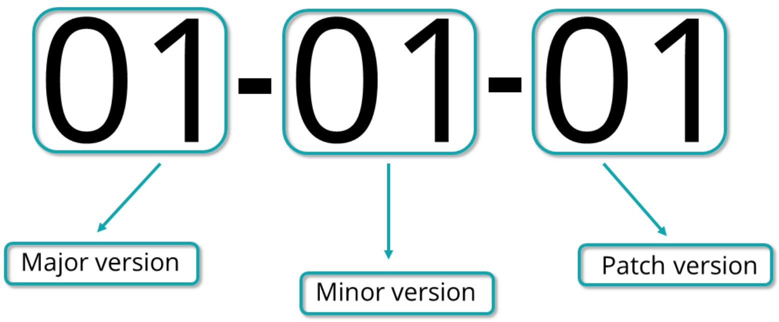
Organizing Rules into Rulesets

Collect a group of Rules that define your application into an instance of a Ruleset. Rulesets provide a simple and time-saving way to store, manage and identify Rules that can be shared between several applications. Instead of reusing individual Rules separately or creating new Rules, you can reuse an entire Ruleset while developing a new application with similar functionalities.

For example, you can create a Ruleset that groups the Service-Level Agreement (SLA) Rules and reuse it in other applications that share the same SLA requirements for processing Cases.

Ruleset versioning format

You identify a Ruleset using its version to understand the significance of past changes to it. The default version when you create a Ruleset is `01-01-01`. A Ruleset version consists of the three separate placeholders:



The meaning of each placeholder in a Ruleset version

- The first two digits correspond with a **major version**, or the initial release of an application. A new release represents extensive changes to application functionality. For example, if an accounting department uses an application to process expense reports, and the department wants to extend the functionality to track employee time off for payroll purposes, you create a new release of the application.
- The middle two digits correspond with a **minor version** or an enhancement. For example, you can create a minor version of an application to include an automatic audit that processes an expense report.
- The last two digits correspond with a **patch version**. For example, you can create a patch version to correct a wrong field label in your application.

You can copy Rules from your Ruleset into a Ruleset of a higher version by using the skimming process. For more information, see Rule skimming for higher Ruleset versions

You can use the Ruleset Maintenance wizard to simplify Ruleset-related actions such as versioning, moving, or merging Rulesets. For more information, see About the Ruleset Maintenance wizard.

Beginning with the release of Pega Infinity™ '23, Pega is updating the minor version in application Rulesets to indicate the year that the Ruleset last updated. The Ruleset versioning format remains unchanged. For more information, see Pega Infinity product naming convention in Pega software maintenance program.

Ruleset security

When you create a Ruleset, you define whether the Ruleset is locked or unlocked. By creating a locked Ruleset, you protect it from accidental changes because developers must enter a password before editing the Ruleset. You can also define passwords for other actions, such as adding a new Ruleset version, or updating the Ruleset. For more information, see Defining the security of a Ruleset

To allow members of your development team to work simultaneously on different features, create Branch Rulesets that developers use as isolated sandboxes. As a result, you reduce the risk of overwriting work, and improve the quality of your application.

Categorizing a Ruleset

You can choose the category of a Ruleset based on the type of Rules that it groups.

- Standard - the default category for Rulesets that can contain all types of Rules.
- Component - a Ruleset that is designed for reuse in multiple systems and applications.
- Shared - a restricted Ruleset that is designed for reuse, which operates only on top-level pages.

For more information, see Categorizing a Ruleset.

For learning material about Rulesets, see Rulesets on Pega Academy.

- **Creating Rulesets**
- **Defining the security of a Ruleset**
- **Categorizing a Ruleset**
- **Deleting a Ruleset**
- **About the Ruleset Maintenance wizard**
- **Production Rulesets**

- Creating applications
- Application Structure landing page

Creating Rulesets

Reduce development time and costs by creating Rulesets. After you create a Ruleset, you can group Rules, which are reusable elements of your application, together in a logical way in the Ruleset. As a result, you conveniently manage, maintain, and reuse your Rules.For example, you can create a Ruleset that stores Service-Level Agreements (SLAs), and a separate Ruleset for Case notifications. If you want to make any changes to the SLA Ruleset, your modifications are granular and you avoid the risk of unintended changes occurring in the Case notifications Ruleset. You can also provide a way for members of your development team to work in isolated sandboxes by creating Branch Rulesets.

When you create new Rulesets, consider the following guidelines:

- Create a new Ruleset with a top-level class that inherits directly from @baseclass.
- Use names that are easy to remember and unique for each Ruleset.
- Use names that clearly convey the purpose of the Ruleset; avoid using acronyms that might be difficult to decode. For example, name your Ruleset `UPlusTelcoContracts` instead of UPTC.
- Always begin your Ruleset name with a phrase that clearly identifies your company and the business purpose. This convention also prevents potential Ruleset conflicts.
- Do not use `Pega` or `Pega-` as a prefix for your Ruleset names. These prefixes are restricted to Pega Platform internal use and can cause unexpected behavior.
- The maximum length of a Ruleset name is 32 characters. Avoid using spaces in Ruleset names.
- Avoid the use of special characters such as dashes (-), underscores (_), plus signs (+), or quotes (“ ”). The system does not permit the saving of Rules with these attributes.
- For more information about using patterns for the Ruleset version number, see Organizing Rules into Rulesets.

- **Creating a Ruleset and a Ruleset version**
- **Configuring Ruleset version settings**
- **Copying Rulesets**

- Organization data instances
- Organization Unit data instances
- Divisions
- Operators

Creating a Ruleset and a Ruleset version

For easier maintenance and development of your application, create Rulesets so that you can categorize the Rules that your application uses based on usage, purpose, and dependencies. You can also create newer versions of existing Rulesets to keep track of version control. As a result, you can quickly locate and analyze the elements that you change in your application. For example, you can create a Ruleset that stores APIs that your organization provides. When you want to enhance the APIs, create a new Ruleset version, so that you can track and manage the changes over time.

1. In the header of Dev Studio, click **Create** > **SysAdmin** > **Ruleset**.
2. Create a new Ruleset or Ruleset version:
    - To create a new Ruleset, in the **Ruleset Name** field, enter a unique name for your Ruleset.

      The maximum length of a Ruleset name is 32 characters. Avoid using spaces in your Ruleset names.

    - To create a new Ruleset version, in the **Ruleset Name** field, press the Down arrow key, and then select the Ruleset for which you want to create a new version.

    Result:

    The system populates the version and description of your Ruleset.
3. **Optional:** To change the version number of your Ruleset, in the **Version** field, enter a new value.

> **For example:**
>
> Enter `01-01-03`.

By default, for a new Ruleset version, the system enters `01-01-01` as the version number. For existing Rulesets, the system increments the highest version by one patch number. In a version number, the two first digits correspond with the release, the middle digits correspond with the minor version or enhancement, and the final digits correspond with the patch version.

4. **Optional:** To change the description of your Ruleset, in the Description field, enter a new value.

> **For example:**
>
> Enter `SampleRuleset:01-01-03`

By default, the system forms the description by combining the Ruleset name and version number.

5. Click Create and open.

> **What to do next:**
>
> Configure version settings for your Ruleset. For more information, see Configuring Ruleset version settings.

Related tasks

- Creating Branch Rulesets

Related information

- Rule check-in process

Configuring Ruleset version settings

For improved management of your application, define the behavior of your Ruleset and how it interacts with other elements of your application by configuring Ruleset version settings. For example, you can lock a Ruleset to keep it in its original state, or require development approval for checking in Rules into the Ruleset, in order to track and manage changes.

Before you begin:

Create a Ruleset or a Ruleset version. For more information, see Creating a Ruleset and a Ruleset version.

1. Expand the SysAdmin category, and then click RuleSet.
2. In the **Instances of Ruleset** section, select the Ruleset that you want to edit.
3. On the Edit Ruleset form, in the **Validation mode** section, define the Ruleset validation settings:
   - To validate the Ruleset within the application context only, select Application Validation.
   - To validate the Ruleset based on the prerequisites defined in another Ruleset, select Ruleset Validation, and then in the Required Rulesets and versions field, enter the Rulesets that you want to use as the validation prerequisites.
4. **Optional:** To require development approval for checking in Rules for this Ruleset version, select the Approval required checkbox.
   When you require an approval to check in Rules, you minimize the risk of unwanted or accidental changes to the Ruleset.
5. **Optional:** To assist with auditing in the future by setting a date on which you expect the Ruleset version to go into production, in the Effective Start Date field, select a new date.

> **Result:**
>
> The application can later use the Effective Start Date metadata to operate according to Rules as they were on the specific date in the past. Such processing is useful to reconstruct past behavior or apply past policies.

6. **Optional:** To protect the Ruleset from accidental changes, lock the Ruleset:
    a. Click Lock and save.
    b. In the **Lock Ruleset Version** dialog box, enter and then confirm a password.
    c. Click Submit.
7. Save your changes to this Ruleset version by clicking Save in the Ruleset version area.
8. **Optional:** To define more Ruleset versions, click Add a row, and then repeat steps 3 through 7.
9. Click Save.

- **Ruleset validation modes**

Related concepts

- Organizing Rules into classes

Related tasks

- Categorizing a Ruleset
- Deleting a Ruleset
- Moving Rules
- Defining the security of a Ruleset

Ruleset validation modes

Ruleset validation is a Pega Platform™ mechanism for ensuring that Rules that an application references are available on the target system and accessible by the application. Ruleset validation does not affect Rule resolution at run time but Pega Platform applies it only at design time.

For relevant training materials, see Ruleset validation.

Pega Platform provides two types of Ruleset validation:

Application validation (AV) mode

Validates a Ruleset only in the context of the current application. When you configure a Ruleset to use AV mode, Pega Platform determines at design time if Rules have a valid reference, which avoids using Ruleset prerequisites.

The system validates Rulesets quicker when using AV mode than when using the alternative, RV mode.

> **NOTE:**
>
> If you want to use an AV-mode Ruleset in more than one application, first create an additional Application Layer and then use the additional layer as a built-on application for all applications that use the AV mode Ruleset.

Ruleset validation (RV) mode

Validates a Ruleset based on the prerequisites that you define in another Ruleset.

Related tasks

- Configuring Ruleset version settings

Related information

- Validation tool

Copying Rulesets

To speed up application development, reuse resources by copying Rulesets. For example, you can copy a Ruleset if you want to modify only some of the Rules that the Ruleset includes, to increase flexibility of your application. For instance, if you create an application that processes loan requests and mortgage requests, you can copy a Ruleset that includes system-level agreement Rules, and then modify specific Rules to meet your unique business needs.

When you copy a Ruleset, the system copies the Ruleset versions, security, category, and history settings.

1. In the header of Dev Studio, click Configure > Application > Structure > Ruleset Stack.
2. On the Ruleset Stack tab, in the **Current Application** section, click the Ruleset that you want to copy.
3. In the header of the **Edit Ruleset** form, click the Down arrow next on the Save button, and then click Save as.
4. **Optional:** To provide additional information about the Ruleset, in the Short description field, modify the default description.
5. In the Ruleset Name field, enter a new, unique Ruleset name.
6. **Optional:** To change the version number of your Ruleset, in the Version field, enter a new value.

   For example:

   Enter `01-01-03`.

   By default, for a new Ruleset version, the system enters `01-01-01` as the version number. For existing Rulesets, the system increments the highest version by one patch number. In a version number, the two first digits correspond with the release, the middle digits correspond with the minor version or enhancement, and the final digits correspond with the patch version.

7. **Optional:** To change the description of your Ruleset, in the **Description** field, enter a new value.

   > **For example:**
   >
   > Enter `SampleRuleset:01-01-03`

   By default, the system forms the description by combining the Ruleset name and version number.
8. Click **Create and open**.

> **What to do next:**
>
> Modify the copied Ruleset:
>
> - Provide new Ruleset version, validation, and locking settings. For more information, see Configuring Ruleset version settings.
> - Configure further Ruleset security options. For more information, see Defining the security of a Ruleset.

Related concepts

- Defining and changing the Scope of Rules

Related tasks

- Deleting a Ruleset

Defining the security of a Ruleset

Complete the **Security** tab on the Ruleset form to update security information for a Ruleset. For example, you can control who can change Rules in a Ruleset, and prevent potential conflicts when two or more users attempt to change one Rule at the same time.

You can also lock the Ruleset or its versions by requiring developers to enter a password when they do the following:

- Update a Ruleset or update class Rules in a Ruleset.
- Add, copy, or update versions on the **Versions** tab of a Ruleset.

1. In the Records explorer of Dev Studio, click SysAdmin > Ruleset, and then click a Rule to open it.
2. Click the **Security** tab.
3. If you defined a password that a user must match to update the Ruleset, in the **Enter password values** section, in the To update this Ruleset field, enter the password to unlock the Ruleset.
4. To indicate that this Ruleset contains Rules that tailor, override, and extend an application, rather than define an application, in the **Rule management** section, complete the following fields:
   a. To specify that a Ruleset is not intended to be moved to other Pega Platform systems, select Local customization?.
   This setting is advisory only and does not affect the Ruleset functionality.

   > **NOTE:** If you select this option, do not select Use check-out?.

   b. To enable developers to check out this Ruleset when updating it, select Use check-out?.
   As a best practice, select this option to avoid the risk of multiple developers updating a Ruleset and overwriting each other's changes.
   c. **Optional:** To use a flow Rule to manage development and check-in on this Ruleset, in the Work class used for approval process field, enter the work type to which the flow Rule that controls check-ins applies.

   > **NOTE:** Leave this field blank if you did not select Use check-out? in step 4.b. The Work class used for approval process field supports

> advanced Rule management features, which are useful when a large development team is involved or when a development leader wants to review each change made by others on the development team before the Rule is checked in.

For more information about checking in Rulesets, see Configuring the Rule check-in approval process.

d. **Optional:** In the Work flow to use for approvals field, enter the name of the flow Rule that controls Rule check-in for this Ruleset.

> **NOTE:** Leave this field blank if you did not select Use check-out? in step 4.b. The Work flow to use for approvals field supports advanced Rule management features, which are useful when a large development team is involved or when a development leader wants to review each change made by others on the development team before the Rule is checked in.

5. **Optional:** Use the **Define passwords** section to prohibit users from adding or updating versions of this Ruleset, or updating this Ruleset.
Lock the setting by entering passwords in the following fields. When you save the Rule, the field is replaced by a checkbox. If you leave a field blank, no password is required.
    a. In the To Add a Ruleset Version field, enter a password that the user must match before creating (beneath the version array) or copying a version for this Ruleset on the **Versions** tab.
    b. In the To Update a Ruleset Version field, enter a password that the user must match before updating a version on the **Versions** tab.
    c. In the To Update this Ruleset field, enter a password that a user must match when updating this Ruleset. This restriction includes updates to the versions on the **Versions** tab.

NOTE: To unlock the To Add a Ruleset Version and To Update a Ruleset Version settings, clear the checkbox and save the Rule.

To unlock the To Update this Ruleset setting, clear the checkbox, enter the password in the To Update this Ruleset Name field, and save the Rule. Lock this setting to prevent changes to the To Add a Ruleset Version or To Update a Ruleset Version settings.

6. Click Save.

- Deleting a Ruleset
- Organizing Rules into Rulesets

Categorizing a Ruleset

Complete this tab to control the category of this Ruleset and to provide additional information for *Component* Rulesets.

In most cases, accept the default *Standard* Ruleset type.

NOTE: The *Standard* Ruleset type is different from the *Pega-zzzz* standard Ruleset that defines Pega Platform.

1. In the Records explorer of Dev Studio, click SysAdmin > Ruleset, and click a Rule to open it.
2. Click the **Category** tab.
3. In the Ruleset Type field, select the category to which this Ruleset belongs.
   Your selection affects where this Ruleset appears in a requestor's Ruleset list, and how Rule resolution finds the Rules in the Ruleset.
   - Standard – A "normal" Ruleset that can contain all types of Rules.

> **NOTE:** All Rulesets created in Pega Platform 5.3 and earlier belong to this category.

- Component – A restricted Ruleset designed to encapsulate functional or capabilities that operate only on embedded pages. Rules in a component Ruleset are designed and intended for reuse in multiple systems and applications.
- Shared – A restricted Ruleset designed for reuse that operates on top-level pages only. See shared Ruleset for the restrictions.

> **CAUTION:** Choose this value carefully. Although you can change Ruleset types other than *Override* later, Pega Platform does not then check whether existing Rules in the Ruleset meet the restrictions of the *Shared* or *Component* categories.

4. If you chose the *Component* Rule Type in step 3, the **Property to embed in application** section displays. To establish communication between Pega Platform and a component application, complete the following fields:
   a. In the Recommended Name field, enter a `Page` -mode property that is made available to both the primary (calling) application and the component Ruleset.
   This provides communication between the primary application and the component application.
   b. In the Page Name field, enter a Clipboard page name that the component can access to pass values to and from the calling application.
   c. In the Page Description field, enter a description for the Clipboard page.
   d. Click Add a row to add additional properties.
5. **Optional:** To add test Case Rules to the Ruleset, in the Test Automation settings section, select Use this Ruleset to store test Cases.
   For more information, see Creating unit test cases for Rules.

6. Click Save.

- Deleting a Ruleset
- Organizing Rules into Rulesets

Deleting a Ruleset

Use the Ruleset Delete tool to delete an entire Ruleset or a specific version of a Ruleset, or to restore a Ruleset previously deleted with this tool.

Before you begin:

- Do not delete a Ruleset version that contains Rules that are in use at the time.
- Review Access Groups to confirm that they do not reference application Rules to be deleted or the Ruleset version to be deleted.
- Review operator IDs that might reference skill Rules (*Rule-Admin-Skill*) that will be deleted.
- Review listener data instances (such as *Data-Admin-Connect-EmailListener*) to confirm that they do not reference service Rules that will be deleted.
- Check for other Rules or data instances that might reference the deleted Ruleset or version, including:
  - Application Rules (*Rule-Application* Rule Type)
  - Other Ruleset versions (as prerequisites)
- If any checked-out Rules are from the deleted Ruleset version, open each checked-out Rule and delete it with the **Delete Checkout** toolbar icon.

NOTE:

If the Ruleset includes one or more concrete classes derived from the *Work-* base class, and there are instances (work items) in the classes, they are also deleted.

This tool does not delete instances of the *History-Rule* class or *Index-* classes that might reference the Ruleset or version. Such orphaned records do not require much space and are harmless.

1. In the header of Dev Studio, click **Configure** > **System** > **Refactor** > **Rulesets** > **Delete a Ruleset** to start the tool.
2. In the **Original Ruleset** field, select the Ruleset to delete.
3. **Optional:** in the **Ruleset Version** field, select a Ruleset version to delete.
4. Click the **Delete** button to begin processing.
   This process can take several minutes.

   Result:

   The system creates a backup of the Ruleset in the explicit temporary directory of Pega Platform. A status area shows progress and the results of the operation.
5. If errors occur, click the **Total Number of Errors** link in the lower right corner of the display form to see any error messages. You cannot easily access this list after you close the form; print the list so you can research and address the errors.
6. Click **Close**.

Result:

The system saves the deleted Ruleset in a file called "RuleSetName_RuleSetVersion_DELETE.zip" in the Pega Platform temporary directory.

What to do next:

If you delete a Ruleset by accident, enter the Ruleset and version into the **Delete a Ruleset** form, and then click Restore.

- About the Ruleset Maintenance wizard
- Organizing Rules into Rulesets

About the Ruleset Maintenance wizard

The Ruleset Maintenance wizard helps you copy Rulesets and Ruleset versions into new versions, merge several Ruleset versions into one version, move Rules out of one Ruleset into another, or change a Ruleset version's number.

- Use the Move operation to rename a Ruleset: the process moves the Rules from the existing Ruleset into the new Ruleset you specify, and then deletes the source Ruleset.
- For Copy operations, the target Rules, if they exist, must be unlocked and checked in before the operation can be completed.
- For Move operations, the source and target Rules, if they exist, must be unlocked and checked in before the operation can be completed.
- For Change a Version operations, the source Rules should be checked in.

Operations of this wizard depend on the *Pega-RuleRefactoring agents*. If you use this wizard, confirm that the *Pega-RuleRefactoring* agents are enabled on at least one node of your system.

Moving a Ruleset

To move a Ruleset, select an existing Ruleset and specify a target name and, optionally, a version. The wizard processes the source Rules in each Ruleset version in the Ruleset and any non-versioned Rules associated with the Ruleset. The Ruleset and Ruleset Version fields of each Rule are updated to the values specified as the target. The result is that the Rule appears to have been deleted from its source Ruleset and copied into the target Ruleset. The value of the Rule's *pzInsKey* does not change.

The handling of multiple Ruleset versions in the Ruleset depends on whether you specify a target version:

- If you move a Ruleset without specifying a target version, the wizard preserves the structure and contents of the source Ruleset's versions under the target Ruleset.
- If you move a Ruleset and specify a target version, Ruleset versions in the source Ruleset are merged by skimming the Rules to the specified version under the target Ruleset. That is, the highest-versioned instance of each Rule in the source Ruleset versions is moved to the target Ruleset and specified Ruleset version.

For example, if you move a Ruleset containing Rules with Ruleset versions 01-01-01, 01-01-02 and 01-01-03 into a new target Ruleset and specify a version of 01-02-01, the target Ruleset version will contain the highest version of all the source Rules contained in versions 01-01-01, 01-01-02 and 01-01-03 with lower-versioned duplicates dropped. In this example, if a Rule is duplicated in both 01-01-01 and 01-01-02, the Rule will be moved from 01-01-02 to the new Ruleset name with Ruleset version 01-02-01, and the duplicate Rule in 01-01-01 will be dropped.

When you move a Ruleset, the wizard also updates the references to affected Ruleset versions in the following Rules or data instances anywhere in the system, so they reference correctly the versions that exist after the move:

- *Rule-Obj-Class*
- *Rule-Access-Deny-Obj*
- *Rule-Access-Role-Name*
- *Rule-Access-Role-Obj*
- *Rule-Application-Requirement*
- *Rule-Application*
- *Rule-Application-UseCase*
- *Rule-Connect-SOAP*
- *Rule-Declare-Pages*
- *Rule-Ruleset-Name*
- *Rule-Ruleset-Version*

You cannot copy a Ruleset.

Copying or moving a Ruleset version

Copying a Ruleset version performs the same function as the **Save As** toolbar action. The original source Ruleset version does not change. Each target Rule that is created is identical to the original source Rule with the following exceptions:

1. You must specify a Ruleset version for the target Rules that is different than that of the source Rules.
2. The wizard creates a new *pzInsKey* field value for the target Rules using the actual create date/time as one of the values in the construction the new target *pzInsKey*. The create date/time value is the only difference between the data that makes up the source and target *pzInsKey* values.

As with moving a Ruleset, moving a Ruleset version changes the source Rules. The wizard updates the **Ruleset Version** fields of each Rule to the values specified as the target. The result is that the Rule appears to have been deleted from its source Ruleset version, and copied into the target Ruleset version. In a move, the value of the Rule's *pzInsKey* does not change.

Merging Rulesets and Ruleset versions

You can merge multiple source Ruleset versions belonging to the same or different Rulesets. For example, you can "skim" the Ruleset by collecting only the highest version of every Rule in the Ruleset versions into the target, or you can compress the Rulesets down to a lower level (01-01-01) or another designated level. You control the merge by specifying the target version and the order in which the Ruleset versions will be processed.

When you select the Ruleset versions, you establish the order of precedence by the order in which you list them in the wizard interface. The wizard processes the Rules in the order listed, top-to-bottom, collects the first version of every Rule it finds, and drops other duplicate Rules:

- The wizard first processes Rules from the first Ruleset version listed, and then discards duplicate Rules in the following versions in the list.
- Then the wizard processes any remaining Rules in the second Ruleset version in the list, dropping any that are duplicates of what it has already processes.
- Processing continues in the same way down the list of versions.

> **NOTE:** Compress Rulesets with prerequisites beginning with the Ruleset of least dependency. For example, if Ruleset A requires Ruleset B as a prerequisite, and Ruleset B requires Ruleset C as a prerequisite, then you should compress the Ruleset A first, then B, and finally C. If you compress the Rulesets in the order C, B, and A, you risk losing some Rules that you expected to retain.

When you merge Rulesets or Ruleset versions, you have the option of deleting the sources once the merge is complete. However, if you merge Ruleset versions and opt to delete the sources, the source versions are deleted, but not their Ruleset(s).

When you merge Ruleset versions, you have the option of excluding categories of non-versioned Rules. The categories are listd at the bottom of the form. Clear the checkbox to exclude a category.

> **NOTE:** When you select Libraries and Functions from the list of non-versioned Rules, all Functions are merged when the Ruleset name has changed. When the Ruleset name is the same and only the version has changed, Functions are treated like versioned Rules and only patch Functions are merged.

For an example of the merge operation, see the article *How to merge Rulesets using the Ruleset Maintenance wizard*.

Changing a Ruleset version number

You can change the version number for a particular Ruleset version, whether or not the version which you designate as the new number already exists. The target version belongs to the same Ruleset as the source version. All Rules, including non-resolved Rules, from the source Ruleset version become part of the target version. All Rules retain their original *pzInsKey* values, and therefore retain their History.

When the change is complete, the wizard deletes all Rules in the source version.

Rule conflicts

For move, copy, and change operations, you can specify how to handle conflicts if the same Rules are found in both the source and the target. You can choose to overwrite the target Rules with the source Rules, or to not move or copy the source Rules that conflict, leaving the target Rules unchanged. If you choose not to overwrite the Rules, the Summary page of the wizard lists the Rules in the source that were skipped because they were in conflict with the target.

Locked Ruleset versions

The wizard does not change Rules that belong to locked Ruleset versions. If a Ruleset version that will be changed by the wizard operation is locked, you are prompted for the password to unlock the version before proceeding. To copy a Ruleset version the target Rules cannot be locked. The source Rules can be locked, since they are not changed by the copy operation. To move a Ruleset or Ruleset version, both the source and target Rules must be unlocked.

Checked-out Rules

As a best practice, make sure that all Rules are checked in before you move or copy a Ruleset or Ruleset version. Click **Configure** > **Application** > **Development** > **Checked Out Rules**. Complete the parameters and click **Run**. Review the resulting report to confirm that no Rules from that Ruleset version are checked out.

If the wizard encounters Rules to be modified that are checked out, it displays a warning page and provides a report of the checked-out Rules. From the report window you can open the Rules and check in the ones for which you have permission, or save a list of the Rules in a spreadsheet.

You must check in any Rules you have checked out or arrange for any Rules in the report that are checked out by other users to be checked in before you can complete the wizard. If you exit the wizard to arrange for Rules to be checked in, you must restart the wizard from the beginning.

Affected Rules and data instances

The wizard automatically adjusts any references to Ruleset versions in the following Rules or data instances, system wide, to match the versions that exist after a merge or move:

- Ruleset versions (“prerequisite” field)
- Rule-Application (lists on the General tab)
- Organization data instance (listing Ruleset versions here is possible, deprecated)
- Access Group data instances

Prerequisites and preparation

Before running the Ruleset Maintenance wizard, consider the following guidelines:

- Back up your Pega Platform system before performing operations with this wizard.
- You can only copy or move Rulesets or Ruleset versions that are in your Ruleset List.
- Make sure the Rules that will be modified by the operation you specify are unlocked and checked in. As described above, the wizard will warn you if it encounters locked or checked-out Rules during the copy or move, but it is a best practice to make sure that all the Rules you will operate on are unlocked and checked in before running the wizard.

When you are working with Ruleset versions, the Copy missing Ruleset Prerequisites checkbox appears at the bottom of the first form of the wizard. Select this checkbox to have the wizard copy any prerequisite Rulesets that the Ruleset version that results from your process will require.

Starting the wizard

Select the Dev Studio >*System* > *Refactor* >*Rulesets* menu item, then click **Copy/Merge Ruleset** to start the wizard. Then select the operation you wish to execute.

For help in completing the first form, see RuleSet Maintenance wizard.

Recommended next steps

This wizard does not modify references to Rulesets that are deleted or modified. Repair any references to Rules that have been moved by the wizard by replacing references to the source Ruleset with the target or another Ruleset. In particular,

- If a Ruleset to be moved is referenced in an application Rule, be sure to update the references on the application Rule form:

  - Application Rulesets section on the General tab

  - Component and Shared Rulesets section on the General tab

  - Production Rulesets section on the General tab

- Update any references to the old Ruleset in Access Groups, including the local customization section on the Settings tab.

Recovering merged Rulesets

NOTE: Before merging the Rulesets, the tool creates a backup of each Rule named *Bkup_SourceRuleSet.zip* and *Bkup_TargetRuleSet.zip*. If needed, you can use the Import .zip facilities to recover the original Rulesets.

Logging

Each execution of this wizard creates an instance of the *Log-Merge-RuleSet* class, including error messages or other results. The key to this instance is the date and time the wizard was started.

Class Rules do not have a version. Do not delete Class Rules (or other Rules that have no version) if any version of the Ruleset remains in use.

In some organizations, audit or compliance policies might prohibit deleting old Rules, even if they are not in use.

Ruleset Maintenance wizard step 1

The Ruleset Maintenance wizard helps you copy Rulesets and Ruleset versions into new versions, merge several Ruleset versions into one version, move Rules from one version into another, and change a version's number.

Preparation and prerequisites

For Copy operations, the target Rules, if they exist, must be unlocked and checked in before the operation can be completed. Source Rules should be checked in.

For Move operations the source and target Rules, if they exist, must be unlocked and checked in before the operation can be completed.

For Change a Version operations, the source Rules should be checked in.

The wizard uses the full-text search indexes to reduce search time and increase accuracy. If full-text search for Rules not enabled for your system, the process may operate much more slowly, and may not locate all data instances that may need to be updated.

When you are working with Ruleset versions, the Copy missing Ruleset Prerequisites checkbox displays at the bottom of the first form of the wizard. Select this to have the wizard copy any prerequisite Rulesets that the Ruleset version that results from your process will require.

For a checklist of other prerequisites and preparation steps, see About the Ruleset Maintenance wizard.

Completing the form

1. Access the Ruleset wizard landing page by clicking **Configure** > **System** > **Refactor** > **Rulesets**, and click **Copy/Merge Ruleset**.
2. Select one of the four radio button options: Copy, Move, Merge, or Change a Version.

   Copying creates the target Rules and leaves the source Rules unchanged. Moving, merging, and changing create the target Rules and delete the source Rules.

   You can perform all four operations on Ruleset versions. You can only move or merge Rulesets.
3. Select the Ruleset or Ruleset versions to be processed.

   Select the Ruleset or version from the left-hand list, Available Source Ruleset(s), and move it to the right-hand list, Order of precedence during Copy/Move/Merge. You can drag your selection, double-click it, or select it and click the right-facing arrow icon between the lists to move it.
4. Adjust order of precedence, if needed.

   If you are operating on multiple Ruleset versions, they will be processed in the order listed, top-to-bottom. That is, Rules are first processed from the first Ruleset version listed and duplicate Rules in the following versions in the list are dropped. Then any remaining Rules are processed in the second Ruleset version in the list and dropped from the following versions. Processing then continues in the same way down the list of versions. To move a version higher or lower in the list, select it and then click the up or down arrow. You can also click a selection and drag it to a higher or lower position in the list.

5. Specify a Target name and, if wanted, a version.

   Enter the name of the Ruleset or Ruleset version that the selected Rules are to be copied, moved, or merged into; or the new version number for the Ruleset version. If you provide a new name, the wizard creates the target Ruleset. If the target exists, source Rules that also appear in the target are processed according to how you set the Overwrite.option (see the next step).

   If you move a Ruleset and specify a target version, Ruleset versions in the source Ruleset are merged by skimming the Rules to the specified version under the Target Ruleset. That is, the highest versioned instance of each Rule in the source Ruleset versions is moved to the target Ruleset and specified Ruleset version.

   If you move a Ruleset without specifying a target version, the source Ruleset versions are preserved under the target Ruleset.

   If you copy or move a Ruleset version, you must provide a target version. If you copy or move multiple Ruleset versions, they will be merged into the target version following the order in which you have listed them in the Order of precedence... list.

   Note that when you change a Ruleset version number, the source and target Ruleset names must be the same.

6. For Copy, Move, and Change, specify the method to resolve Rule conflicts.

   If you select `Yes`, source Rules that duplicate existing target Rules replace the target Rules in the processed Ruleset. If you select `No`, source Rules that duplicate existing target Rules are not processed, and the target Rules are left unchanged.

7. For Merge, specify whether the source Rulesets or Ruleset versions are to be deleted after the merge is complete. Select `Yes` to delete the sources; `No` to retain them. If you are merging Ruleset versions and opt to delete the source versions, their parent Rulesets are not deleted.

8. When you merge Ruleset versions, a display at the bottom of the form shows categories of non-versioned Rules, and the number of Rules of each type affected by changes to the Ruleset versions you have selected and placed in the second list. You can exclude any category by clearing the checkbox to the left of its name.

   NOTE: When you select Libraries and Functions from the list of non-versioned Rules, all Functions are merged when the Ruleset name has changed. When the Ruleset name is the same and only the version has changed, Functions are treated like versioned Rules.

9. Click Next to continue to the next form in the wizard. See RuleSet Maintenance wizard 2. Click Cancel and then click Done to cancel the wizard.

About the Ruleset Maintenance wizard

Ruleset Maintenance wizard step 2

On this page confirm that the selected operation is as you have specified. Ensure that:

- You have selected Copy, Merge, Move, or Change as intended.
- You have specified the correct target Ruleset or Ruleset version.
- You have specified the correct source Ruleset or Ruleset version(s).
- For Copy, Move, or Change, you have selected the response to Rule conflicts between the source and target as intended: overwrite or not.
- For Merge, you have specified whether to delete the source Rulesets or Ruleset versions when the merge is completed.
- For Ruleset versions, you have specified whether to copy prerequisites the version requires.

The page displays a list of data instances that may be affected by the procedure. Select all entries that you want to update so that they will continue to work with the target Ruleset or Ruleset version.

You have the option of running the wizard as a Background Process: this can be useful when moving or merging Rulesets containing a large number of Rules, as the process may take several minutes to complete. To select this option, click the `Yes` radio button beside the Run as a Background Process label.

The Pega Platform sends an email notifying you when the process is complete. If no email server has been set up for the Pega Platform and you select this option, a message appears stating that an email will be sent once you set up an email server.

You can still run the wizard as a Background Process, but if outbound email is not set up, the Pega Platform cannot send you an email when the process completes. The wizard form provides you with the ID of the process so you can review it at any time, and displays a message at the end of the process: see RuleSet Maintenance Wizard 3.

Depending on the process you selected in the first screen, the continuation button's label will read **Copy**, **Rename Version**, and so on. If you are satisfied with the settings, click the button to begin the process and to continue to the next step. See RuleSet Maintenance wizard 3.

To change any of the settings, Click **<<Back** to return to the previous step.

Click **Cancel** and then click **Done** to cancel the wizard.

About the Ruleset Maintenance wizard

Ruleset Maintenance wizard step 3

If you opted to run the wizard as a Background Process, a message displays on the page stating that your request is being processed and that you will receive an email notification once it has completed.

Place the provided reference number in the Search facility and click the magnifying glass. The facility opens the wizard at its current form so you can check its status.

When the process is complete, this form displays a summary of the changes that have been made, including the number of Rules considered and selected. Depending on the process and the options you selected, up to four links provide access to lists of

- Rules processed
- data instances processed
- Rules in conflict
- processing errors

If there were no data instances to process, errors, or conflicting Rules, the corresponding links do not appear.

Click any link to display the list. In the report that is displayed, you can click any row to review that Rule in detail. To save a copy of a list in spreadsheet form, click **Export Page to Excel** at the top of the list.

After you have closed any reports you have opened, click **Done** to exit the wizard.

About the Ruleset Maintenance wizard

Ruleset Maintenance — Locked Rules

If the wizard finds Rules to be modified in locked Rulesets or locked Ruleset versions, this form displays a list of the locked Rulesets or Ruleset versions with a Password field for each.

- To continue, provide the password to unlock each Ruleset or Ruleset version and click **Next**. The wizard unlocks the Ruleset or Ruleset version, operates on the Rules as specified, and then re-sets the lock.
- If you cannot provide the passwords, click **Cancel** to exit the wizard, arrange to unlock the Rules in Pega Platform™, and then rerun the Ruleset Maintenance wizard from the beginning.

- Click **Back** to return to the previous page.

About the Ruleset Maintenance wizard

Ruleset Maintenance — Checked-Out Rules

If the wizard finds that Rules to be modified are checked out, this page displays a warning. Click **Display** to open a report of the checked-out Rules. From the report window you can open the Rules or click **Export to Excel** to save a list of the Rules in a spreadsheet.

You cannot proceed while Rules to be modified are checked out. Check in any Rules you have checked out and arrange for any Rules listed as checked out by others to be checked in. When all Rules are checked in, click **Next>>** to continue.

If you exit the wizard to arrange for Rules to be checked in, you will need to restart the process from the beginning.

Click **<<Back** to return to an earlier step, or click **Cancel** and then click **Done** to cancel the process and exit the wizard.

About the Ruleset Maintenance wizard

Production Rulesets

To allow users to modify a Rule in production, define the Rule in a production Ruleset.

> **NOTE:** This article replaces the archival article "KB 25289 - How to configure production Rulesets and WorkPools."

Production Rulesets can be listed for applications and for Access Groups. If an application has production Rulesets, you list all of them in the application Rule. In an Access Group, you list the Rulesets that are delegated to the Access Group. Typically, you grant this permission to managers, not to Case workers.

As a best practice, do not include any Work- classes (classes for work objects) in production Rulesets.

Define the modifiable Rule in a regular application Ruleset (which is locked), as well as in a higher-version unlocked production Ruleset. The Rule in the production Ruleset takes precedence at run time since it is a higher-ranked Ruleset.

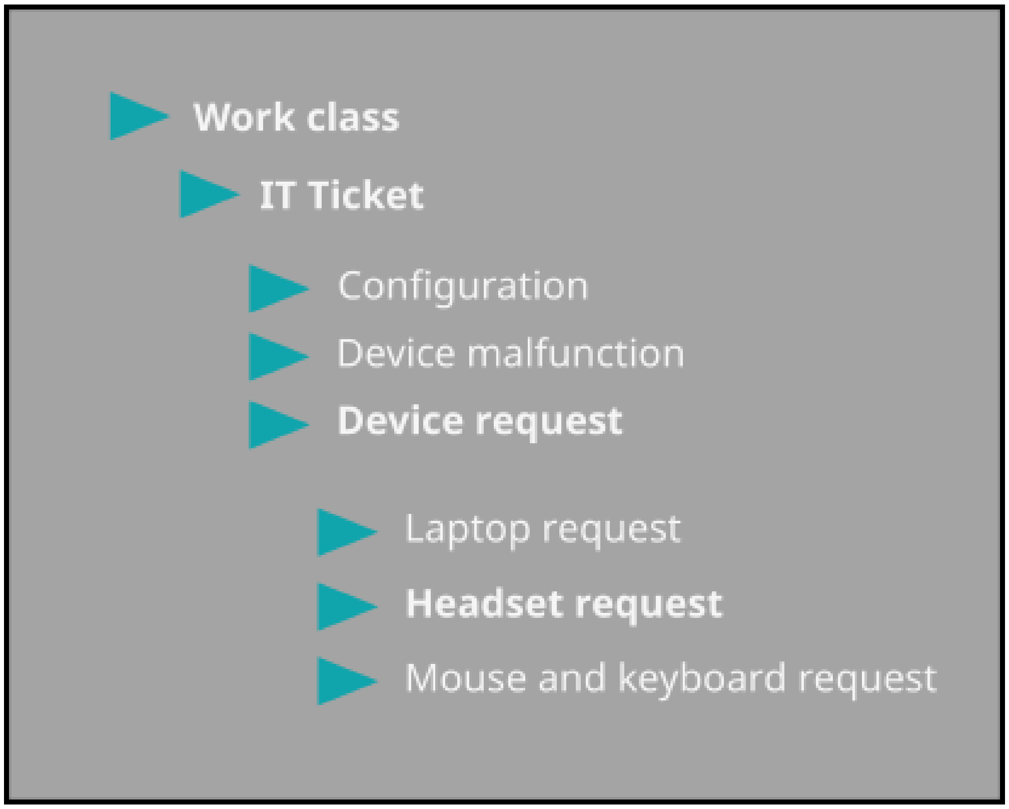
The Ruleset concept does not apply to data instances.

- Viewing your application Ruleset list
- Defining a run-time configuration for an Access Group
- Delegating a Rule or Data Type

Organizing Rules into classes

For more efficient management and to improve application quality, organize your Rules into classes. A class groups Rules within an application based on their capacity for reuse.
Pega Platform™ organizes Rule collections into a hierarchy of inheritance that consists of parent and child classes and instances of a class. Each grouping can be reused between Case Types and other applications to significantly reduce development time.

For example, a parent class of the IT Ticket Case Type contains a standardized ticket form with all the properties related to the Case, such as employee details or ticket type, and the approval Process. The Headset request is an example of the specialized version of the generic IT Ticket and is a child class Case Type of the IT Ticket parent class Case Type. The Headset Request Case Type contains the hardware details and the Process for device configuration, and is supplemented by the inherited Rules that are defined for its parent class.



Parent and child classes of Case Types

Class types

In Pega Platform, applications consist of the following class types:

Work class
: The work class collects Rules related to processing a Case, user interface, user forms and Processes.

Integration class
: The integration class collects Rules that describe how the application interacts with other systems, such as an external database or a third-party server.

Data class
: The data class collects Rules that describe the data types in the application, such as an employee data type or a request items data type.

When you create a Rule in App Studio, the system automatically identifies the appropriate class.

For more information, see Creating classes.

Class hierarchy

Class hierarchy describes dependencies between classes to determine which class contains other classes and how developers can reuse Rules in an application. By default, when attempting to identify the Rule to use at run time, Pega Platform starts with the most specific class and expands the search to more general classes.

Pega Platform provides the ultimate base class, which is identified by the keyword *@baseclass* and is the root for all other classes. Specific classes, such as a set of Rules to process a Case Type, inherit Rules from a more general class, such as a class that provides basic functionality for processing Cases.

Class inheritance

Classes can inherit Rules in the following ways:

Pattern inheritance

Pattern inheritance describes the process of automatic reuse of Rules through a class name structure. During Rule resolution, the system looks for Rules from the most specific class to the ultimate base class.

Directed inheritance

Directed inheritance describes the process of Rule reuse from a specified parent class regardless of any defined pattern of inheritance.

For more information, see Understanding class hierarchy and inheritance.

External classes

External classes collect Rules related to using external sources of data. For example, an order management application that handles employee purchases, which sources the product and inventory data from the external system of record. External classes do not follow the basic inheritance process. For more information, see Understanding external classes

- **Understanding class layers**
- **Understanding class hierarchy and inheritance**
- **Understanding external classes**
- **Creating classes**
- **General tab on the Class form**
- **Locking tab on the Class form**
- **Advanced tab on the Class form**
- **External Mapping tab on the Class form**
- **Finding the class of a Rule**
- **About the Rename a Class wizard**
- **About the Delete a Class wizard**

Understanding class layers

To improve the reuse of Rules in your application, understand how Pega Platform™ organizes classes into class layers. Because classes define the applicability, or Scope, of a Rule instance, knowledge of the different layers of classes and the way that they inherit from each other is important as you develop applications.

Framework classes

Framework classes define a common work-processing foundation that you extend and modify as implementation applications for an organization, division, or organizational unit. Framework classes belong to the framework layer.

For example, an enterprise that makes auto loans has an auto loan framework application that contains all of the assets needed for the basic auto loan Process. The enterprise then develops two implementation applications built on the auto loan framework to meet the specific requirements of its two divisions: a commercial business line and a personal line.

Using the New Application wizard, you can build a new implementation application and its implementation classes on an existing framework application. You can reuse some or all of the Case Types and data types of the framework application. In your new implementation application, you can use the Dev Studio landing pages, explorers, Rule forms, and reports to reuse many of the Rules and data objects inherited from the built-on framework layer. For example, while developing a Process, you can select specifications or processes in a framework Ruleset for reuse or customization in the current application.

When you build an implementation application, you can also create a reusable framework application built on the same framework. You can then extend the framework application so that it is usable by other implementations that you might create later. As a best practice, reuse framework Rules and create only specialized Rules in your implementation applications. For example, use report definitions in the framework classes that run with the corresponding implementation classes.

Implementation classes

Implementation classes define the extension, reuse, and specialization of assets in a framework layer to meet the business requirements of an organization, division, or organizational unit. For example, you can build two division-level implementations –

business auto loan and personal auto loan – on an organization's auto loan framework layer.

Implementation classes belong to the implementation layer. Typically, Cases that are related to application processes are instances of Case Type classes that belong only to the implementation layer.

Organization classes

Organization classes contain enterprise-wide data assets and assets that your application reuses for business logic. Data assets include classes and Rules for data stored in the system, and classes and Rules for accessing data in external systems by using connectors. Business logic assets include standard properties, decision tables, and Service-Level Agreements.

For example, the auto loans enterprise might want to reuse, on an enterprise-wide basis, the property for employee serial numbers. By reusing this property, the various applications across the enterprise that the employees use can consistently rely on the same serial number property representing the same employee.

Organization classes belong to the organizational layer. In most configurations, your implementation and framework layers inherit by pattern inheritance from organization classes.

For relevant training materials about a class structure, see an Enterprise Class Structure topic on Pega Academy.

- Configuring applications for reuse
- Creating applications
- Creating applications

Understanding class hierarchy and inheritance

To save development time when creating an application, reuse Rules by organizing them into classes, and then creating dependencies between the classes. When you create a class hierarchy you define which classes contain other classes, and as a result, you define how classes reuse or inherit Rules.

In Pega Platform, you can create parent classes that contain child classes. Any Rules available in a parent class are also available to the child classes. Because of this, Pega organizes classes in the following groups:

- The ultimate base class, which Pega Platform provides and identifies by the keyword *@baseclass*. The ultimate base class is a root for all other classes.
- Base classes that store the Rules that provide basic functionality for processing Cases, such as data elements that record the creator of a Case. The most common base classes include *Work-*, *Data-*, and *History-* classes.
- Classes from other applications, such as industry-specific Pega applications. For example, you can use a generic application for policy administration as a base for customizing versions for different policy types.
- Classes that collect Rules and data elements that are common at the division and organization level, such as a class that stores an approval Process shared across an entire IT department.
- Classes that describe a specific Case Type, such as expense reports or vehicle insurance claims.

The Records Explorer lists child classes under their parent class.

Careful planning and analysis are important when creating new classes, so that you can understand their interrelationships and plan their positions in the standard class hierarchy. Maximizing reuse of standard classes, standard properties, and other standard Rules supports fast development with less effort. You can also provide for greater reuse of classes by creating class groups. A class group data instance unifies a set of classes in the PegaRULES database that share a common name structure and common key structure. All classes that belong to the same class group share one database table. For more information about class groups, see Working with class groups.

Class naming convention

The names of all classes that belong to a common class group start with the class group name, followed by a hyphen. For example, if you define a class group *Work-Object-App-Task*, you can name the child classes in this class group *Work-Object-App-Task-Ordering*, *Work-Object-App-Task-Shipping*, and *Work-Object-App-Task-Billing*.

For more information about class names, see Naming conventions for classes.

Class inheritance

Classes can inherit Rules in the following ways:

Pattern inheritance

A class inherits characteristics automatically through the class name structure. Pega Platform uses a multi-level class hierarchy of Organization (Org), Division (Div), Unit and Class group to organize application assets. During Rule resolution, the system looks for Rules from the bottom of the class hierarchy (the most specific class), to the top, (the ultimate base class). The system can reuse Rules only within one application.

Directed inheritance

A class inherits characteristics directly from a specified parent class, regardless of any defined pattern inheritance. You typically apply directed inheritance to reusing standardPega Platform Rules and Rules from other applications outside the business class hierarchy.

For learning materials about inheritance patterns, see a Rule reuse through inheritance topic on Pega Academy.

For learning materials about classes, see a Classes and class hierarchy topic on Pega Academy.

Related tasks

- Creating Rulesets

- Creating Branch Rulesets
- Categorizing a Ruleset

Understanding external classes

For improved application development, understand how your application uses classes that correspond to tables in an external relational database, rather than to a table or view in the PegaRULES database. By reusing external classes, you can include resources from outside the Pega Platform database in your application.

No inheritance or Rule resolution applies to external classes, even if the class name contains a hyphen. Because each external class has an associated database table *Data-Admin-DB-Table* instance, it cannot be part of a class group.

The following methods can operate on an instance of an external class:

- Obj-Browse
- Obj-Open
- Obj-Open-by-Handle
- Obj-Delete
- Obj-Refresh-and-Lock
- Obj-Save
- Obj-Save-Cancel
- Commit
- Rollback

You cannot create Declare Index Rules that have an external class as the *Applies To* key part. However, *Rule-Declare-Expressions* Rules, *Rule-Obj-Validate* Rules, and *Rule-Declare-Trigger* Rules operate on Clipboard property values for external classes.

External classes do not contain the *@baseclass.pzInsKey* and *@baseclass.pxObjClass* properties, which are present in every internal class. In some cases, no single property in the external class can serve as a unique handle. To support the Obj-Open-by-Handle method with external classes, Pega Platform assembles a substitute string from the key columns of the table.

In certain cases, you can improve the performance of database insert, delete, and update operations for external classes through the optional `batchUpdates` setting in the prconfig.xml file or dynamic system settings.

System messages sometimes refer to external classes as "Obj-External"; however, Pega Platform does not have an Obj-External method.

- Creating classes
- Organizing Rules into Rulesets

Creating classes

For improved development and maintenance of your application, organize the Rules and other objects in your application into classes. By creating classes, you define collections of objects that are available to other classes or to instances of the class. You can reuse classes, and as a result reduce both development time and costs.

1. In the header of Dev Studio, click Create > SysAdmin > Class.
2. In the **Class record configuration** section, in the Label field, enter a short description for the class that you want to create.
3. In the Class name field, enter a unique, descriptive name that clearly conveys the purpose of the class.
   To show the relationship between classes, use a hyphen in the class name in which the first name is a parent class.

   > **For example:**
   >
   > To create a `Jobtitles` class that derives from an HR class, enter `HR-Jobtitles`.

NOTE: A class name can have a maximum length of 56 characters. Avoid using names that start with `Code-`, `System-`, and `History-`, because the system creates these classes automatically. For more information, see Naming conventions for classes.

4. In the **Context** section, select an Application Layer in which to store your class.
5. In the Add to Ruleset list, select a Ruleset in which to store your class.
6. **Optional:** To override the default work item that your application associates with this class, in the Work item to associate field, press the Down arrow key, and then select a work item.
   For more information about default work items, see Setting your current work item.
7. Click Create and open.
8. On the General tab of the class form, select a class type, and then define the class settings for your new class:

Choices	Actions
Define the settings for an abstract class	a. In the Select class type list, select Abstract. b. In the **Settings** section, in the Created in version field, enter the Ruleset version that applies to this class, and then go to step 10. Abstract classes have no instances in a database.
Define the settings for a concrete class	a. In the Select class type list, select Concrete. b. In the **Settings** section, in the Created in version field, enter the Ruleset version that applies to this class, and then go to step 9.

Choices	Actions
	Concrete classes have instances in a database. For more information, see Class keys.

9. Define the class group settings of a concrete class:

Choices	Actions
Create a class group	a. In the This class list, select The class is a class group. **Result:** The Class group field autopopulates with the class name. b. **Optional:** To encrypt the Storage Stream of each instance of this concrete class when saved to the PegaRULES database, and to decrypt the Storage Stream when an instance is retrieved from the database, select the Encrypt BLOB checkbox. For more information, see Encrypting the storage stream (BLOB). c. **Optional:** To define key parts of the class, in the **Keys** section, provide a key name and a caption. For more information about class keys, see Class keys.

Choices	Actions
	d. **Optional:** If you provide only one key, to ensure that instances of this class are unique, and to automatically supply a globally unique value for further new instances, select the Automatically generate a unique ID for records of this type checkbox.
Define the settings for a class that does not belong to a class group	a. In the This class list, select does not belong to a class group. b. **Optional:** To encrypt the Storage Stream of each instance of this concrete class when saved to the PegaRULES database, and to decrypt the Storage Stream when an instance is retrieved from the database, select the Encrypt BLOB checkbox. For more information, see Encrypting the storage stream (BLOB). c. **Optional:** To define the key parts of the class, in the **Keys** section, provide a key name and a caption. For more information about class keys, see Creating classes.
Define the settings for a class that belongs to a class group	a. In the This class list, select belongs to a class group.

Choices	Actions
	b. In the **Class group** field, enter a class group name that you want to include this class. c. **Optional:** To encrypt the Storage Stream of each instance of this concrete class when saved to the PegaRULES database, and to decrypt the Storage Stream when an instance is retrieved from the database, select the **Encrypt BLOB** checkbox. For more information, see Encrypting the storage stream (BLOB).

10. In the **Class inheritance section**, define how the class that you create inherits Rules from superclasses:
    a. **Optional:** To use pattern inheritance before direct inheritance when the system finds Rules at run time, select the **Find by name first (Pattern)** checkbox.
    Pattern inheritance searches in superclasses based on prefixes to class names that end in a hyphen.
    b. In the **Parent Class (Directed)** field, enter the class name of the immediate parent of your new class:

    For concrete classes that are class groups, enter `Work-` or a class that inherits from the *Work-* base class, or enter `History-Work-` or a class that inherits from the *History-Work-* class, if you create a class to hold the history of work items.

If you enter the longest possible prefix for your new class ending in a hyphen, you can select or clear the Find by name first (Pattern) checkbox, with no difference in Rule resolution behavior or results. For example, if you create a *Work-Object-Mortgage* class and the parent class is `Work-Object-`, the system performs the same search regardless of the checkbox setting. Avoid using `System-` and `Code-` base classes, because these classes are reserved.

> **NOTE:** During Rule resolution, the system considers only direct parent classes. For example, class C inherits from class B, and B inherits from A, but C does not inherit from A.

11. Click Save.
12. **Optional:** If you create a concrete class, check the class connection with the database by clicking Test connection.

> **NOTE:** For the test connection to work, ensure that the JDBC JAR file for the database platform is present and included in the classpath, and that the `prconfig.xml` file contains an entry for the database driver. For example, `<env name="database/drivers" value="org.postgresql.Driver"` ↗

> **Result:**
>
> A window that contains the test results opens. The results include whether the class is abstract or concrete, and the database and database table if the class is concrete.

Related concepts

- Organizing Rules into Rulesets

Related tasks

- Creating a Ruleset and a Ruleset version
- Creating Branch Rulesets
- Deleting a Rule

Naming conventions for classes

Unified naming standards for classes help you manage your resources in a logical and efficient way. You can avoid duplicating your content, maximize efficiency, and speed up application development by sharing resources between multiple classes or class layers.

For example, you can create class names that support class inheritance, so that you can reuse resources across different classes. When you name a class `ReviewRequest-LoanRequest`, you indicate that a new class `LoanRequest` inherits resources from the `ReviewRequest` class.

When you name classes, consider the following guidelines:

- Use singular nouns, such as `Customer` or `Address`, to distinguish classes from actions and processes. For better readability, capitalize the first letter of each word in a sequence, for example, `CustomerAddress`.
- Use a name that is a model or abstraction of something real. As a result, you can then create a hierarchy of classes that goes from general to specific. For example, you can create a class named *Feline* that includes classes named *Lion*, *Lynx*, and *Tiger*.
- Do not include words that do not add value, such as Details, Info, or Data. For example, name your class *Cat* and not *CatInfo* or *CatData*.
- Avoid using acronyms and abbreviations that are difficult to decode. The exception are common acronyms, such as HTML.
- Avoid including underscores (_) in class names.

- Pega Platform provides a set of base classes that includes the *Work-*, *History-*, and *Data-* classes. Create new classes only under the organization base class, for example, *UPlusTelco-*.
- The system displays labels for class names on some of the forms, Worklists, reports, and other areas of an application. The system also displays a short description for the class. Therefore, use class names that are self-explanatory and reflect the purpose of the work. Avoid using class names that reflect a status or action, instead of a unit of work.
- Choose class names that support pattern inheritance, as well as directed inheritance. For more information about class inheritance, see Understanding class hierarchy and inheritance.
- To distinguish hierarchy position and inheritance, use class names that visually reflect pattern inheritance. For subclasses, include the names of all parent classes, including your top-level class, separated by dashes. For example, if the parent class *YourCoLoan-Request* includes child classes for mortgage and car loan applications, name the child classes `YourCoLoan-Request-Mortgage` and `YourCoLoan-Request-Auto`.
- Name related objects with consistent prefixes, so that you can quickly locate similar classes in an alphabetical list.
- For class groups, use the same naming conventions as for classes.

- Organizing Rules into Rulesets
- Creating Rulesets
- Understanding class layers
- Creating classes

Class keys

When you define concrete classes, you can add keys to identify the classes. You can add the keys when you create a class, or you can update the keys later.

Every instance of a concrete class includes a key formed from the value of the properties that the class contains. The value is unique within the class or class group.

For example, your system can contain only one instance of the *Data-Admin-Operator-ID* class with an ID value of *SampleID*. Similarly, your system can contain only one HTML fragment Rule named *MyFragment* in the same Ruleset and version.

The properties that form the key might apply to the concrete class itself, or to a parent class. For Rule instances, the system appends the Ruleset and version as part of the internal key.

Pega Platform looks in three places to learn which property values to connect to, and in what order, to form the unique key. The system assembles keys the first time it saves an instance:

- For a concrete class that does not belong to a class group, the system first examines the Keys array of the Class form. If this array is not empty, it forms a key by connecting the values of the properties identified in that array.
- If the Keys array of a class is empty, the system uses class inheritance to locate a superclass for which the Keys array on the Class form is not blank. When the system finds such a class, it connects the values of the properties in that Keys array, and forms the key.
- For a concrete class that does belong to a class group, the system forms a key based on properties in the Keys array of the Class Group form. As a result, the system ignores any information in the Keys array of the Class form.

When you add keys to a class, consider the following factors:

- The system creates both a new class and single value properties of the Text type when you enter the property names into the Keys array, before you save the Class form for the first time. If you want to apply one or more already-defined properties as key properties to a super class of this class, enter the already-defined properties after you save the Class form.
- For concrete classes derived from the *Log-* base class, the property *pxCreateDateTime* is usually the final or only key part. In some high-volume systems, two log events might occur in the same millisecond. If this can occur in your *Log-* class, choose additional properties to ensure uniqueness.

- For concrete classes derived from the *Index-* base class, enter the properties *pxInsIndexedKey*, *pxIndexCount*, and *pxIndexPurpose* in that order. Avoid using any other properties in the key.
- For external classes, use of properties with a *Type* value of *Date*, *Time*, *DateTime*, or *Double* as key parts might introduce rounding or uniqueness issues. Review the data representation on the external database that you use, to confirm that it maps one-to-one with the Pega Platform representation.
- If two or more properties form the key, the order of rows in the Keys array is significant. Consider which order is likely to produce a natural or frequently presented sort order in reporting or searching.

Related concepts

- Understanding class hierarchy and inheritance
- Rule resolution

Related tasks

- Creating Rulesets
- Deleting a Rule

General tab on the Class form

Use the General tab to define the class inheritance and database access.

> **NOTE:** After you save this form, your entries for many fields are permanently set and you cannot change them later. However, the Keys array is an exception. If you save the form with the Keys array blank, you can update the Class form later and specify the Keys array to reference properties that are defined to apply to this class.

How the system forms key values from the properties in the Keys array

Every object saved to the PegaRULES database — that is, every instance of a concrete class — must have a key formed from the value of properties that is unique within the class or class group.

For example, your system cannot contain two instances of the *Data-Admin-Operator-ID* class with an ID value of smith@smith.com. Similarly, your system cannot contain two HTML fragment Rules named MyFragment in the same Ruleset and version.

The properties that form the key may apply to the concrete class itself or to a parent class. (For Rule instances, the Ruleset and version are appended as part of the internal key.)

Pega Platform looks in three places to learn which property values, in what order, to concatenate to form the unique key. The system assembles keys the first time it saves an instance:

- For a concrete class that does not belong to a class group, the system first examines the Keys array of the Class form. If this array is not empty, it forms a key by concatenating values of the properties identified in that array.
- If the Keys array of this class is empty, the system uses class inheritance to locate a superclass (an ancestor) for which the Keys array on this tab of the Class form is not blank. When such as class is found, it concatenates the values of the properties in that Keys array to form the key.
- For a concrete class that does belong to a class group, the system forms a key based on properties in the Keys array of the Class Group form. As a result, any information in the Keys array of the Class form is ignored.

Completing the Settings fields

The appearance of the General tab changes depending on whether you enter a class group, keys, or neither. You cannot enter both a class group and keys. If you enter a class group, Pega Platform uses keys defined in the class group.

When you save the Class form, most of the choices you made become permanent. For example, you cannot later select or clear the Find by name first (Pattern)? checkbox to use, and you cannot change properties entered in the Keys array.

Field	Description
Select	Indicate whether this class is to be abstract or concrete. Both abstract and concrete classes can have subclasses. If this class is to have persistent instances, select `Concrete`. (You cannot select `Concrete` if the class name ends with a hyphen.) Otherwise select `Abstract`.
Created in Version	Identify the Ruleset version number that applies to this class. The Archive tools depend on the version you enter here. The Export Archive tool includes this class instance if you export this Ruleset version or a higher-numbered version. However, version information in class Rules is not used for Rule resolution. You cannot define multiple class Rules with the same class name but different versions.

Field	Description
	Complete this field when first creating a new class (with New or Save as). Alter this field later in rare circumstances only.
This Class	Select one: • `belongs to a class group` to indicate that: (1) the key format of this class is identical to the key format of another class, (2) an existing class group named identically to the other class exists, and (3) you want to store instances of this class in the same table as instances of other member classes. Typically, choose this option for a concrete class derived from the *Work* - base class. • `does not belong to a class group` to indicate that database storage of this class is determined by the class inheritance and database table data instances. • `is a class group` to have Pega Platform create a class group data instance with a name identical to the name of this class, This allows other classes to share a database table associated with instances of this class.

Field	Description
	If this concrete class is derived from the *Work-* base class or the *History-Work-* class, select: • `is a class group` or • `belongs to a class group`. **NOTE:** Concrete classes created in releases prior to PRPC Version 6.2 might not conform to this restriction. As a best practice, choose `belongs to a class group` for any concrete class that is derived (according to pattern inheritance from a class that has This class set to `is a class group`. Otherwise, warning message may appear when you save the Class form. Although possible, avoid creating Database Table mappings for subclasses of a class group that differ from the Database Table for the entire class group. If you create a class is derived from the *Work* - base class, and you selected `is a class group`, review the Description for this class carefully; the Description text is visible to application users. See More about Class Rules.

Field	Description
	For an external class created through the External Data Table wizard, the initial value is `does not belong to a class group`. Do not change this value.
Encrypt BLOB	Select to cause Pega Platform to encrypt the Storage Stream of each instance of this concrete class when saved to the PegaRULES database, and to decrypt the Storage Stream when an instance is retrieved from the database. This checkbox appears only for concrete classes. It affects only instances of exactly this class, not instances of any subclasses of this class. **NOTE:** Determine whether you application is to use this feature early during the development project. You cannot select or clear this checkbox if the database already contains instances of the class. Additional configuration is required to set up this capability. See Storage Stream encryption of selected classes.

About the Class Group field

This field appears only when `is a class group` or `belongs to a class group` is selected in the This class field. Confirm that the class group identified is the one expected.

Field	Description
Class Group	Optional. If you selected `belongs to a class group`, select a class group (an instance of the *Data-Admin-DB-ClassGroup* class) from the list. A class group — a data instance — has the same name as its first or defining container class. The name of the class group you choose must match an initial portion of the class name. Choose this option when defining one or several concrete classes that define types of work items to be stored and processed as part of a single application, or the corresponding class derived from *History-Work-*. If you enter a class group, leave the Keys array empty. In this case, the class group instance defines a common key structure used by all associated classes.

Completing the Keys fields

The keys array appears only for concrete classes that either define a class group or that are do not belong to a class group.

If the Keys array appears, in many cases, you can leave it blank, at least when you first save the Class form:

- To create new `Single Value` properties of type `Text` with this class as the Applies To key part, enter the property names into the Keys array before you first save the Class form. When you save the form, the system creates both the new class and properties.
- To use one or more properties already defined to apply to a superclass of this class as key properties, do not enter them before you first save the Class form. Save the Class form once, and then use SmartPrompt to select the properties that form the key.
- For concrete classes derived from the Log- base class, the property *pxCreateDateTime* is usually the final (or only) key part. In some high-volume systems, two log events may occur in the same millisecond. If this can occur in your Log- class, choose additional properties to ensure uniqueness.
- For concrete classes derived from the Index- base class, enter the properties *pxInsIndexedKey*, *pxIndexCount*, and *pxIndexPurpose* in that order. No other properties can appear in the key.
- For external classes, use of properties with a Type value of `Date`, `Time`, `DateTime`, or `Double` as key parts is permitted, but may introduce rounding or uniqueness issues. Review the data representation on the external database to confirm that it maps one-to-one with the Pega Platform representation.

Field	Description
Keys	If two or more properties are to form the key, the order of rows in this array is significant. Consider which order is likely to produce a natural or frequently presented sort order in reporting or searching. For example, consider a class named Data-County, with a key formed from a county name (such as Suffolk, for Suffolk county)

Field	Description
	and state code (such as MA for Massachusetts). Assuming that reporting requirements present county information within state, a natural order for the two key parts is state followed by county.
Name	Optional. Leave blank if the concrete class is a subclass of a superclass that corresponds to a class group. Also, leave blank if a parent (or higher superclass) of this class is concrete and the key structure of this class is to be the same as the key structure of the parent. Identify the property or properties that together define a unique key for objects of this class. If the property or properties that are to form the key are not yet defined, you can create them automatically the first time you save the Class form by entering a name here. If the properties that are to make up the key are already defined (in a superclass), do not enter them initially. Save the Class form once, then use SmartPrompt to select them as rows in the Keys array, and then resave the Class form. Typically, the key of an object also determines how it is identified when locked. In the unusual case that a different key

Field	Description
	structure is needed, you can define a separate lock string on the Locking tab.
Caption	Optional. For concrete classes derived from *Rule-* or *Data-* for which a *Rule-File-Form* is present, this text appears as field labels in the New dialog box. In other cases, you can enter text to document why you chose this property to be part of the key; the information you enter appears only in this field.
Automatically generate a unique ID for records of this type	This checkbox appears when you have entered one key field. If you want instances of this class to be unique using this one field, and you want the system to automatically supply a globally unique value for new instances, select this checkbox. Selecting this checkbox also changes the key name to pyGUID.

Completing the Rule Instances of this class fields

These four checkboxes appear only for concrete classes derived from the *Rule-* base class, with a few exceptions. They determine which aspects of the full Rule resolution algorithm the system uses to find Rules of the new Rule Type. See Rule resolution.

Field	Description
Allow multiple versions with the same key values? (Rule resolution)	When selected causes the system to allow multiple Rules of this Rule Type to have the same name, distinguished by Ruleset or

Field	Description
	Ruleset version. If this checkbox is selected, at run time Pega Platform uses the Ruleset list part of the Rule resolution algorithm for this class to select a Rule at run time.
Allow selection of Rules based on property values? (circumstance-qualified)	When selected, causes the system to allow Rules of this Rule Type to have circumstance-based qualifications. If this checkbox is selected, at run time Pega Platform uses circumstance-based processing at run time to select a Rule, and these fields appear on the Save As form. See circumstance. This field appears only when the Allow multiple versions field is selected.
Allow Rules that are valid only for a certain period of time? (date range availability)	When selected, causes the system to allow Rules of this Rule Type to have time-based qualifications. If this checkbox is selected, Pega Platform uses time-based processing at run time to select a Rule, and these fields appear on the Save As form. See time-based Rules. This field appears only when the Allow multiple versions field is selected.
Use class-based-inheritance to arrive at the correct Rule to execute?	When selected, causes the system to cause the system to search up the class inheritance structure (using both pattern and directed inheritance) to find Rules of this Rule Type. If this checkbox is not selected, at run time Pega Platform expects to find the Rule in the

Field	Description
	current class only, not by searching through super-classes.

Completing the Class Inheritance fields

Indicate how this class inherits Rules from super classes.

Field	Description
Find by name first (Pattern)?	Select to cause the system to use pattern inheritance before it uses directed inheritance when finding Rules at run time. Pattern inheritance searches superclasses based on prefixes of the class name that end in a hyphen. Clear this checkbox to bypass pattern inheritance. Directed inheritance always occurs for all Rule Types with an Applies To key part. If you selected `belongs to a class group`, this checkbox is selected. After you save a Class form, you cannot change this setting.
Parent class (Directed)	Identify the class that is to be the immediate parent of this class: • If you don't select the Find by name first (Pattern)? checkbox, at run time as part of Rule resolution processing, the system searches the parent class and

Field	Description
	its parent and so on when it cannot find a Rule in the current class. • If you select the Find by name first (Pattern)? checkbox, the system during Rule resolution searches classes with derived class names before it searches the parent class identified in this field. If you selected `is a class group` in the This class field, enter *Work-* or a class that is derived from the *Work-* base class here. (Or, enter History-Work- or a class that is derived from History-Work-, if you create a class to hold the history of work items. Such History-Work- classes are ordinarily created automatically.) If you enter the longest possible prefix of this class ending in a hyphen, you can select or clear the Find by name first (Pattern)? checkbox, with no difference in Rule resolution behavior or results. For example, if this class is named Work-Object-Mortgage and the parent class is to be *Work-Object-*, Pega Platform performs the same search regardless of the checkbox setting. ⚠ **CAUTION:** Do not assume that the "inherits from" relationship is transitive. Because only one Parent class to inherit from (Directed)

Field	Description
	⚠ value is used by a Rule resolution, it is possible that class C inherits from class B, and B inherits from A, but C does not inherit from A. ⓘ **NOTE:** You cannot create a class derived from the *Code-* or *System-* base classes. These base classes are reserved.

Testing the class-to-database table mapping

For a concrete class, you can test the database connection and learn the table structure.

Test Connection	After you save the Class form, click to confirm that the database instance (*Data-Admin-DB-Name*) and database table instance (*Data-Admin-DB-Table*) are working as intended. The test confirms that Pega Platform can access the database and displays the database table that will contain instances of this class. Results appear in a new window, identifying the database and database table that correspond to this concrete class, and whether the class is an internal or external class.

	This button appears only for concrete classes. **NOTE:** For the test connection to work, the JDBC JAR file for the database platform must be present and included in the classpath and the `prconfig.xml` file must contain an entry for the database driver. For example: `<env name="database/drivers" value="org.postgresql.Driver" ↗`

How to include properties in a class as key parts of that class

As you complete the Key Name fields while adding a class, you can only reference existing properties. If a key part for your new class is a property that will apply to the new class, you cannot create the property until after you create and save the class Rule.

Take these steps to work around this circular situation:

1. Save the class Rule with the Keys array empty; no keys defined.
2. Define properties that are to form the keys in the class in the normal way.
3. Open the class Rule again, and enter the key properties into the Keys array of the General tab.
4. Save the class Rule again.

- Organizing Rules into classes

Locking tab on the Class form

Information on this tab is used only for concrete classes that do not belong to a class group.

Select the Allow Locking box to cause the system to lock open instances of the class, that is, instances copied from the database into the Clipboard using the Obj-Open method or similar methods.

For concrete classes that do belong to a class group, leave this tab blank. The Lock field on the Class Group form controls locking for all the classes in the group. See Concepts — Understanding object locking.

Field	Description
Keys	
Key Name	Optional. Leave this array blank in all but rare circumstances. If this field is blank, the Keys properties from the General tab of this form is used to define a lock string. In unusual situations, you can define a non-standard lock for the class by listing here the properties that together define the lock key. The value of these properties (rather than the properties on the General tab) are concatenated to form a lock key for the instance. The resulting value may at run time apply to multiple open objects, locking not only this object but all other open objects (even objects of other classes) with the same completed lock key value.
Key Caption	Optional. Not used.

Field	Description
Locking	
Allow Locking	For concrete classes, select this box except in rare situations. Select to allow the system to lock open instances of the class when the Obj-Open or Obj-Open-By-Handle method specifies locking. However, for concrete classes derived from the *Rule-* base class, leave cleared. To restrict updates to individual Rules to one developer at a time, use Rule checkout (by updating the Ruleset to allow checkout, and updating the Operator ID to enable checkout), not locks. If this class is part of a class group, the Lock field on the class group form supersedes this Allow Locking? value. If selected, a user cannot save or delete an instance within the Scope of this class group unless that user holds a lock on the object. Typically, activities acquire locks when they open an instance to the Clipboard. **NOTE:** If Allow locking? is selected, then when you use the Obj-Open, Obj-Open-by-Handle or Obj-Refresh-and-Lock method in an activity to open an instance within the Scope of this class group (with intent to update and save the

Field	Description
	ⓘ instance), select the Lock checkbox parameter for the method.

- **Object locking**
- Organizing Rules into classes

Object locking

Locks are implemented separately from the locking mechanism supplied by the database software (such as Oracle) that supports the PegaRULES database. Locks are exclusive to one thread, and operate system-wide in a multinode system.

Basics

A requestor can lock an open instance if all of the following are true:

- The instance belongs to a concrete class that has **Allow Locking** selected in the class Rule.

  AND

- The instance has ever been saved and committed to the PegaRULES database.
- The requestor has an Access Role that conveys the privilege needed.

Properties identified on the **Locking** tab of the **Class** Rule form together define the lock identity. (If the **Locking** tab contains no property names, the properties that form a lock definition are the same as those that form the external key that is defined on the **General** tab.) All classes in one class group (which corresponds to one database table) have the same properties in the lock definition.

Viewing locks — Standard reports

To view locks held by your own requestor session, select **Configure** > **Case Management** > **Tools** > **Work Admin** > **My Locks**.

To view all locks held by any requestor, select **Configure** > **Case Management** > **Tools** > **Work Admin All Locks**, and drill down to identify the details of the locked object. Hold the mouse pointer over a row of the drill-down detail to identify the Operator ID of the requestor holding the lock.

These two reports present instances of the *System-Locks* class, corresponding to the `pr_sys_locks` table of the PegaRULES database.

Viewing locks — In an activity or flow

Locks are acquired by activities, using one of three methods: Obj-Open, Obj-Open-by-Handle, or Obj-Refresh-and-Lock. Optionally, each of these methods can store detailed results in a page of class *System-Locks*. If a method failed to acquire a lock because the object is already locked, information on this page identifies the current requestor session that holds the lock. By convention, this page is named `LockInfo` in standard Rules.

As a debugging aid, the standard section *System-Locks.LockInfo* and the standard harness *Work-.LockInfo* can present information from that page to an application user. The LockInfo page contains these properties:

- *pxExpireDateTime*
- *pxLockHandle*
- *pxOwnerID*
- *pxUpdateOperator*
- *pxCreateDateTime*

The lock string

A lock string uniquely identifies a lock. For instances that belong to classes not associated with a class group, the text string consists of the class name concatenated with the key parts of the object (determined on the Keys area of the Basics tab of the Class form, or the Locking tab of the Class form).

For an object that is within the Scope of a class group, the lock string consists of the class group name concatenated with the value of the key properties identified on the Keys tab of the Class Group form. For example, for the Sample Work application, the class group is named *PegaSample* and work items in the *PegaSample-Task* class are identified with a W- prefix. The lock string for work item W-43 is thus:

```
PEGASAMPLE-TASK W-43
```

Expired ("soft") locks

Generally, a thread that acquires a lock retains it until the instance is saved again with a Commit method (which may complete an Obj-Save or Obj-Delete method). However the system automatically expires locks held longer than a specified timeout period. You can change the default timeout interval of 30 minutes to a longer or shorter period. This value is stored in the *Data-Admin-System* data instance.

An expired or "soft" lock remains held by a requestor session until the session releases it or until the requestor session ends, but a soft lock can be stolen — broken and acquired — by another requestor who opens the instance. In that case, any updates by the first user that are not committed are lost.

Releasing locks

Except as described below, a requestor can only release locks held by its own session. A Commit method releases all the locks held by a requestor Thread that were acquired with the ReleaseOnCommit box checked when the object was opened.

When a user signs off, any locks held by that requestor are released.

In contrast, locks are not released when a user closes a browser session by clicking the Windows close box (rather than by logging out). The locks may be needed by another Thread of the same operator, or by another operator or process. Encourage users to log off through a button or link rather than closing the window.

Releasing locks of other sessions

Lock information is held in the memory of each node, rather than in the database, for improved performance. However, even in a multinode system, a requestor can force the release of locks held by any session with the same Operator ID and same user name (*pyUserName*) through the PublicAPI method:

```
LockManager.unlock(StringMap, boolean)
```

This method communicates through the system pulse across all nodes. On each node, the system searches existing requestors (including passivated ones) using the supplied requestor ID. If a session is found, the system checks that the user names (*pyUserName*) also match before attempting to terminate the other session. If a session is found that matches the requestor ID, but the user names do not match, the system generates a SECU007 security alert and lists the corresponding requestor ID and user name in the logs.

NOTE: You cannot release a lock that is held by another user, even if that user is no longer on the system. Instead, wait for the lock to timeout and become “soft” and then steal the lock. Otherwise, the lock is purged after the default timeout interval.

Testing for locks

To test whether a work item is locked, an activity can call the standard activity *Work-.WorkLock*, which tries to lock the work item. If it fails to acquire a lock, the `SuccessInfo` parameter is negative, indicating that the work item is already locked. The `FailureInfo` parameter provides additional detail.

To test whether a lock is held on a page already on your own Clipboard, call the Function Rule *isPageLocked(tools, pagename)*.

To test whether an object other than a work item is locked, use one of several methods in the PublicAPI *LockManagerInterface*.

Debugging

To add detailed lock manager information to the Pega log, set the Logging Level Settings tool with this category to `Debug`:

```
com.pega.pegarules.engine.database.LockManagerImpl
```

To trace locking events from the application, add locking to the list of event types tracked by Tracer:

1. Open **Tracer**.
2. In the **Settings** tab, **Event Types** section, select **Locking**.

Advanced tab on the Class form

The Advanced tab is only meaningful for concrete classes derived from the *Rule-*, *Data-*, or *Work-* class. Use the fields on this tab to control advanced settings such as class deprecation and Ruleset restrictions.

Category

Select a Rule category to control where instances of this class appear in the Application Explorer and Records Explorer. This list is populated by field values with *Rule-Obj-Class* and *pyCategory* as the key parts. You can define additional field values with these key parts to customize Rule categories used in explorers.

Rule form

Use the fields in this section to define the basic Rule form structure for instances of this class:

- Type – Select the *Harness* option. The *Form* option is deprecated.
- Name – Type the name of a harness that contains the sections, layouts, and other components users see when they interact with instances of this class. The applies to key part of the harness must be this class.

Primary Rule assembly

The fields in this section are reserved for custom *Rule-* descendants that use Rules assembly:

- Primary Aspect – Select an interface or aspect. This list is populated by public methods defined in `com.pega.pegarules.pub.runtime.PublicAPI`.
- Primary Implementation Class – Type the name of a Java class that contains the Rules assembly logic for this class.

Secondary Rule assembly

Leave fields in this section blank unless you are creating a custom *Rule-* descendant that requires Rules assembly for embedded Rule Types.

Java wrapper

Fields in this section are reserved for class Rules generated by the Connector and Metadata wizard. When JavaBeans are imported, a Java wrapper class is created that enables the Clipboard to access instances of the JavaBean. The name of this Java wrapper appears in the **Wrapper Name** field.

For more information, review the Pega Community article *Working with the Java Pages Feature*.

Full text search

Select **Exclude this class from search** to exclude instances of this exact class (not descendants) from full text search results. This is helpful in systems where a large number of work objects exist and indexing is enabled. You can eliminate the processing and space required to index these items.

If the class has entries that have already been indexed, you must remove the the index from the file system, then rebuild the search index from scratch to remove those entries from the index.

NOTE: This option applies only to concrete classes derived from the *Work-*, *Rule-*, or *Data-* class. As a best practice, do not check this box for *Rule-* classes.

Deprecate

Check the box in this section to deprecate all instances of this class and block users from creating new instances. As a best practice, add documentation in the **Usage** field of the History tab. You see this documentation in the form of a warning when you open or reference an instance of the deprecated class.

NOTE: Only classes with the *Does not belong to a class group* option selected can be deprecated.

Users are warned when they open the class Rule form or any instance of the class. The warning text is populated by the **Usage** field on the class Rule form's History tab. For example:

The Rule Type Hierarchy is deprecated and should no longer be used. Please use navigation Rules for tree configuration instead.

You cannot create any new instances of a deprecated class. If you save as the class, the **Deprecate this class type** option remains selected.

Deprecated classes in *Pega-* Rulesets are excluded from:

- Search results
- Application Explorer results
- Top-level and right click **Create** menus in Dev Studio explorers
- Lists launched from the Records Explorer

You can find and view a deprecated class by entering the fully qualified class name in the Application Explorer and then selecting the **Definition** right-click option on the class name.

Dedicated table

Check the box in this section to create a dedicated database table for instances of this class. This feature is available only to users with the *@baseclass.pxClassToDbTableOptimization* privilege.

Child classes

Check the box in this section to require all classes derived from this class to reside in the same Ruleset as this class.

Ruleset restrictions

Use the list in this section to control which Rules can be created with *pyClassName* set to this class name. For each row in the list, enter the name of a Ruleset in your application stack. When an operator attempts to create a Rule that applies to this class, the system validates that:

- The Ruleset name specified on the create form resides in this list.
- The Ruleset version specified on the create form has the Ruleset of this class as a prerequisite.

NOTE: You cannot add a Ruleset to the list that is a prerequisite to the Ruleset version of the class itself. For example, if your class resides in the GAMMA Ruleset, and GAMMA depends on the BETA Ruleset, you cannot add BETA to the list.

- Organizing Rules into classes

External Mapping tab on the Class form

The External Mapping tab is used primarily for external classes. External classes are created by the Connector and Metadata wizard or the Database Class Mappings gadget and are linked by a Database Table data instance to a table in a database other than the PegaRULES database.

Typically, the data on this tab is from one of these two tools, and no manual changes are required.

Field	Description
External Table Column Mapping	Optional. Leave blank unless this class is an external class that represents a table in an external database (generated by the Connector and Metadata wizard or the

Field	Description
	Database Class Mappings wizard), or you are replicating scalar embedded properties as described below. Fields in this section map the properties generated for the class Rule to the columns in the external table.
Column Name	For generated class Rules, the name of a column in the external table.
Property Name	For generated class Rules, the name of the property that represents the table column in the Column Name field.

Exposing an embedded Single Value property value into a column

For a concrete internal class, you can use this tab to expose match an embedded `Single Value` property in a database column.

For example, the embedded property *.pyPreferences.pyNavigationPrefs.pyShowTabs* manages whether a user wants to use tabs in his or her developer environment.

If it were important to use this embedded property as report selection criteria, the property must correspond to a column. Use these steps to make a (scalar) embedded property available as a database column.

1. Ask a database administrator to create a new column of the appropriate data type in the PegaRULES database table that corresponds to the *Data-Admin-Operator-ID* table. In this example, the table is `pr_operators`. The database administrator created a text column named TABDISPLAY. (You cannot use the Modify Schema wizard to add this column, because there is no property named TABDISPLAY.)
2. Update the Class Rule for the class *Data-Admin-Operator-ID*. Add a row to the Column Mapping array on this tab. Enter the new column name TABDISPLAY the

Column Name field and the embedded property reference in the Property Name field.

3. Thereafter, the system updates the new column TABDISPLAY from the embedded property *.pyPreferences.pyNavigationPrefs.pyShowTabs.pyTabs* whenever an Operator ID data instance is saved. The property reference *.pyPreferences.pyNavigationPrefs.pyShowTabs.pyTabs* can be used as criteria in reports.

This feature provides a direct but limited alternative to creating an *Index-* class and maintaining the index through a Declare Index Rule, or to replicating the value to a second (top-level) property using a Declare Expression Rule or Declare Trigger Rule.

NOTE:

- In contrast to a Declare Index approach, the embedded property must be a scalar reference, not an aggregate.
- You can use this approach to expose one fixed scalar value within an aggregate property, such as *pyWorkparty("Originator").pyLastName* for the last name of the Originator work party in a work item. In most situations, however, you want to expose all values, not a single value; the technique described here does not suffice.
- Like other schema changes, the database administrator must add the column in every system that contains the report Rules (such as development, test, and production systems).
- You can update Class Rules that belong to foundation Rulesets (for example, the Data-Admin-Operator-ID class).

- Organizing Rules into classes
- Viewing database class mappings

Finding the class of a Rule

You can view XML data to find the class of a Rule. By understanding which class defines a Rule, you can more efficiently use tools such as the Application Explorer and Report Editor.

1. Open a Rule by searching for it or by using the Application Explorer.
2. Click **Actions** > **View XML**.
3. Find the first instance of the *<pxObjClass>* property. This property contains the Rule's object class name. For example, the object class for flow actions is *<pxObjClass>Rule-Obj-FlowAction</pxObjClass>*.

- Viewing the XML of a Rule
- Finding Rules by type

About the Rename a Class wizard

Use the Rename a Class wizard to rename a class and all of its pattern inheritance dependent classes. In addition you can choose to rename associated objects such as work- instances, properties, activities, and flows.

NOTE: The Rename a Class wizard is designed to assist with refactoring applications that are in development. It is not intended to be used on a deployed production application.

The wizard reports any locked Rulesets or Rules that are checked out before renaming anything.

In addition to exact matches to the specified class name to be changed, the wizard identifies references to the current class name in embedded strings, for example in property names or in embedded Java modules. The wizard displays a listing of these "inexact" matches, and you can select which to include to be renamed and which to preserve unchanged.

In particular, use Rename a Class for applications with fewer than a thousand work objects, or select No on the Work Objects page to drop references in the work objects to the new class name. After the class is renamed, work items that referenced the original class can be viewed only as XML.

Starting the wizard

To improve performance, purge all existing work objects from the class before running the Rename a Class wizard.

Select **Configure** > **System** > **Refactor** > **Classes** > **Rename a Class** to start the wizard. You can return to a previous step using the **<<Back** button.

Alternatively, select a Case Type or work pool in the Application Explorer, right-click, and select *Refactor* > *Rename a Class* from the context menu.

No Rules are altered by the wizard until you click **Next** after reviewing additional Rules to be changed on the next-to-last page.

Resuming the wizard

This wizard creates a work item with prefix `pxW-` . To find open wizard work items, select the menu option Dev Studio > **Application** > **Tools** > **All Wizards** .

Prerequisites

You can rename a class if it meets these conditions:

1. The class must be visible in your Access Group.
2. The class may not be one of the standard classes, those in the Pega- Rulesets.
3. The original class cannot have Final availability.
4. All Rules to be modified must be checked in. (If Rules are checked out, check them in and restart the wizard.)

5. All Rules to be modified must belong to unlocked Ruleset versions. You must supply the appropriate password to unlock the locked Rules, or you can opt to skip them for this operation.
6. The new name of the class must not already exist in the system. Class names need to be unique system-wide, rather than unique within a specific Ruleset.
7. You cannot change an abstract class to a concrete class or vice versa. For example, class A- cannot be renamed to B, and B cannot be renamed to A-.

Results

When complete, the renaming wizard changes your system in the following ways:

1. The class is renamed.
2. Direct and pattern inheritance references the new class name. As a result, there will be no effect on Rule resolution.
3. If the option has been selected, *Work-* instances associated with the class are renamed.
4. All properties, activities, flows, and other Rules that referenced the original class now reference the class by the new name.
5. If the class is a work pool or defines an external data table, the *Data-Admin-DB-Table* instance is updated to reflect the new class name.
6. The wizard creates a new *History-* class for the new class, and it will identify that the class has been renamed. The *History-* class for the old class name is deleted.

The old class name is retained until all changes to dependent classes and *Work-* instances are completed. Then the original class is deleted. If the system is stopped while renaming is in progress, you can restart the process by executing the utility from the beginning.

- Understanding class hierarchy and inheritance

About the Delete a Class wizard

You can use the Delete a Class wizard to remove a class, all of its pattern- inheritance dependent classes and associated objects such as properties, activities, instances (including work items, attachments and Assignments).

You can only delete classes that meet the following restrictions:

- The class is not in the basic *Pega-* Rulesets that define the Pega Platform.
- The class is in a Ruleset listed in your Ruleset list.
- The class is not mapped to an external database table.

Starting the wizard

Select **Configure** > **System** > **Refactor** > **Classes** > **Delete a Class** to start the wizard.

You can return to a previous step using the **<<Back** button. No Rules are altered by the wizard until you click **Next>>** in step 2. For instructions on the forms, see:

- Help 1: Using the Delete a Class wizard.
- Help 2: Display Classes to delete
- Help 3: Display results and search for references
- Help 4: Display references
- Help 5: End

Resuming the wizard

This tool creates a work item with prefix `pxW-`. To find open wizard work items, select the menu option Dev Studio > Application > Tools > All Wizards .

NOTE:

Checked out Rules associated with the class will be deleted. Unlike other Rule management wizards, the Delete a Class wizard does not identify and warn of checked out Rules among those to be deleted.

Deleting a class might render an application inoperative. Always back up the Pega Platform system before deleting a class.

This wizard can't delete a Rule that belongs to a locked Ruleset version. Locked Rules will be left in place during the deletion and listed on the summary of undeleted Rules on page 3 of the wizard, Search for References.

The wizard cannot delete a class if it contains one or more Rules that have failed deletion (perhaps because they belong to a locked Ruleset version.) However, all Rules that can be deleted will be when you run the wizard. To complete deleting the class, you must unlock or otherwise make it possible for the undeleted Rules to be deleted and then run the wizard again.

Errors are written to a log. To examine the log, use the Application Explorer to view instances of the *Log-Delete-Class*.

Classes related to deleted classes by directed inheritance are not deleted. If the specified class is a subclass of *Work-*, all instances are deleted prior to the deletion of the class.

Using the Delete a Class wizard - Step 1: Select Class to Delete

Using the Delete a Class wizard —

Step 1: Select Class to Delete

Use the *Class* drop-down menu to select the class you want to delete.

The drop-down includes classes that are contained the Rulesets listed on your Ruleset list. On a large system it may take a while for this list to populate after you click in the field. Expand the drop-down box and make your selection.

After you have made your selection, click the **Next >>** button.

Click <<**Back** to return to the previous step. Click **Cancel** and then click **Done** to cancel the wizard.

About the Delete a Class wizard

Using the Delete a Class wizard - Step 2: Display Classes to Delete

Using the Delete a Class wizard —

Step 2: Display Classes to Delete

This page displays the class and subclasses that will be deleted. The classes and associated objects such as properties, activities, attachments and Assignments will also be deleted.

Confirm whether you want to delete all the classes listed and then choose one of the following actions:

- Click **Next** >> to complete the deletion.
- Click << **Back** to select another class to delete.
- Click **Cancel** to exit from the wizard without deleting any classes.

About the Delete a Class wizard

Using the Delete a Class wizard - Step 3: Display Results

Using the Delete a Class wizard

Step 3: Display results and search for references

This form presents a list of Rules that could not be deleted. The Message column explains why the Rule was not deleted and the actions required to delete it.

If there are undeleted Rules, the class cannot be deleted. However, all other Rules associated with the class have been deleted. To complete deleting the class, modify the Rules listed so that they can be deleted and then run the wizard on the class again.

Select Reference Search

In the Reference Search drop-down, choose one of the following options to set the scope over which the wizard is to search for references to the deleted classes:

- `Don't search for references`
- `Search for references in your Ruleset list.`
- `Search for references in all Rulesets.`

In the next step the system searches for references to deleted objects that may need to be fixed after the deletion is complete. If this search is made over the entire Rulebase on a large system, it can take some time to complete.

When you have reviewed the list of Rules and set Reference Search, click **Next >>** to complete the deletion and see a list of unresolved references.

About the Delete a Class wizard

Using the Delete a Class wizard - Step 4: Display References

Using the Delete a Class wizard —

Step 4: Display References

This form provides a list of Rules and data which might still reference the deleted class. The Value column lists the instance that contains the reference, and the other columns provide the information you need to locate it.

Review this list and correct any invalid references that might interfere with your system.

To review these Rules outside the wizard, click **Export To Excel** to create a spreadsheet of the currently displayed page, or **Export All To Excel** to create a spreadsheet of the entire report.

Click **Finish** to exit the wizard.

About the Delete a Class wizard

Using the Delete a Class wizard - Step 5: Display References

Using the Delete a Class wizard —

Step 5: End

This form appears when all steps are complete.

Click **Finish** to exit the wizard.

About the Delete a Class wizard

Rule resolution

Rule resolution is a search algorithm that the system uses to find the best or most appropriate Rule instance to apply to complete processing. By using automated Rule resolution, you can save time and ensure that you implement resources in a way that is the most efficient in a given scenario.

For example, you can build an application to review loan requests, and then apply different approval criteria for standard loans and mortgages. Rule resolution finds Rules that hold appropriate approval criteria based on a loan type that a user selects in the application. In a different scenario, you can provide an upgraded version of your application to selected users to gather feedback. At run time, because of Rule resolution, users who upgraded the application see a different UI version than the users who use your application version before the upgrade.

For relevant training materials, see the Rule resolution module on Pega Academy.

Benefits of Rule resolution

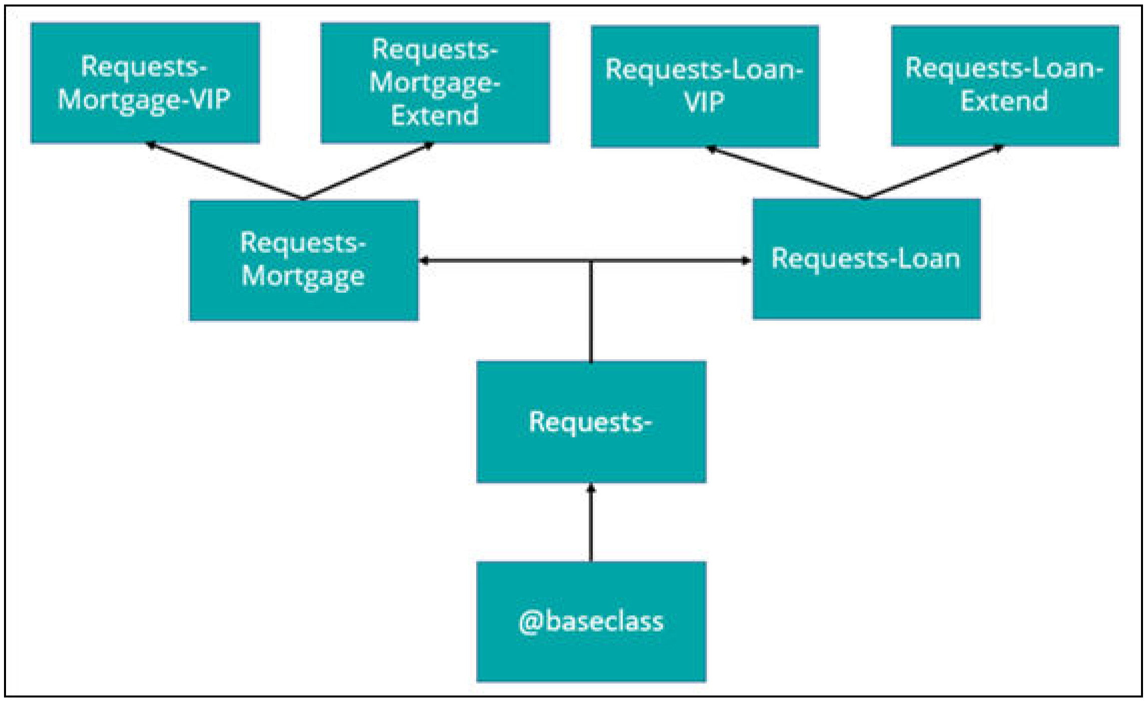
The benefits of Rule resolution include the following advantages:

- Multiple applications and organizations can share Rules. Sharing and reuse are major benefits of goal-oriented and efficient software development.
- More specific Rules that are defined at a higher level can override more general Rules that are defined at a lower level that is closer to the base class. Overriding Rules makes reuse of resources less applicable, but provides flexibility and visibility to exceptions.
- For greater flexibility, Rules can have multiple versions. Rule resolution automatically determines the most appropriate Rule in a given scenario. You do not need to spend time to manually select a Rule.
- One Pega Platform system can host multiple applications, organizations, and versions of one application with less concern for conflicts or interference.
- If multiple applications depend on a common set of Rules, developers can work on the applications independently without interference if the common Rules are locked.

Classes in Rule resolution

Pega Platform defines data in a class hierarchy that follows the object-oriented paradigm. The hierarchy starts with an ultimate class called *@baseclass*. Classes are subdivided into more specialized types of classes. For example, the *Requests-* class can include specialized *Requests-Loan-* and *Requests-Mortgage-* classes. Classes can contain Rules that descend from the *Rule-* base class. Rules can inherit properties from other Rules through class hierarchy. In search for a Rule that is defined in a particular class, the system checks Rules that are defined in the class, as well as Rules defined in the ancestors of the class. The system can hold multiple copies of Rules, which are different, and any one of these copies might be useful at run time, depending upon the situation. Rule resolution is a process by which Pega Platform determines which Rule to apply from the set of candidate Rules. Many different Rules can apply, so Rule resolution checks all the candidate Rules to determine the most accurate way to perform processing.

The following figure shows a sample class hierarchy that starts with a *@baseclass* and then goes from more general to more specialized classes:



Class hierarchy

For example, in a scenario in which you work with an instance of the *UPlusTelco-Auto-ClaimsEntry* class, and the processing requires a *Display* flow, if the system finds the *Display* flow both in *UPlusTelco-* and *UPlusTelco-Auto-* classes, it chooses the latter instance, which is closest to the definition of the class structure of the *UPlusTelco-Auto-ClaimsEntry* instance.

Rule resolution begins by selecting all the possible Rules with a particular name of a particular type, such as activities or When Rules. For example, at run time, a sample scenario requires the *Approval* Service-Level Agreement (SLA) on the *UPlusTelco-Process-Loan* class. The system chooses all the *Approval* SLAs in the *UPlusTelco-Process-Loan* class, or the ancestors of the *UPlusTelco-Process-Loan* class, including Rules in different Rulesets, different versions, and different circumstances. The SLA Rule in question can have different definitions in all these places, or the system might find duplicate definitions in different classes or Rulesets. The goal of Rule resolution is to select just one Rule that is the most accurate to apply to a given situation.

Rule resolution applies to Rule Types in classes that inherit from the abstract *Rule-* base class. The following list includes examples of instances of Rules that derive from the abstract *Rule-* base class:

- Case Types that belong to the *Rule-Obj-CaseType* class
- Properties that belong to the *Rule-Obj-Property* class
- Data Pages that belong to the *Rule-Declare-Pages* class

Rule resolution does not apply to instances of classes that derive from the *Work-*, *Data-*, or any other base class. Those classes typically contain objects to which Pega Platform refers as records. The following list includes examples of records that derive from these abstract base classes:

- Operator IDs that belong to the *Data-Admin-Operator-ID* class
- Email listeners that belong to the *Data-Admin-Connect-EmailListener* class
- The Rule check-in process that belongs to the *Work-RuleCheckIn* class

An in-memory Rule cache helps the Rule resolution process operate faster. If the system finds an instance, or instances, of the Rule in question in the cache, the system accepts the Rule instances in the cache as the candidate Rules and skips multiple steps in the resolution process.

Rule resolution inputs and output

The following list includes the inputs to the Rule resolution process:

- The key parts of a Rule instance that the Rule resolution process targets, such as the *Applies to* class and the name of the Rule.
- The Ruleset list for the user that the system assembles when the user logs in.
- The class hierarchy, which is the structure of classes.

  For more information, see Understanding class hierarchy and inheritance.

- The Access Roles and privileges of the user, determined by the Access Group that the user has.

- Security and access control Rules, such role access to object Rules and privileges.
- Rule availability that determines which Rules are available, blocked, final, withdrawn, or not available.

  For more information, see Setting Rule status and availability.

- In some cases, the value of a circumstance property.

  For more information about circumstancing Rules, see Creating a Rule specialized by circumstance.

The output of the resolution process is the first Rule that matches all the criteria. In some scenarios, the system fails to find a Rule and the process stops. Then, the system can start an exception flow, for example for a Case Type, or return a message about a failure.

Rule resolution process

The following list describes the consecutive steps that the system performs in the Rule resolution process:

1. The system checks the Rules cache.

   By using Rules that are already in the Rules cache, the system avoids additional database lookups.

   For more information about the Rules cache, see the How the Rules cache is populated module on Pega Academy.

   If the system finds the Rule in the cache, the Rule resolution process moves to step 8.

2. The system chooses instances with the correct purpose and puts them in a temporary list.

   The purpose combines all the key parts of a Rule.

For example, for an activity Rule, the key parts include:
- The *Applies to* class that defines the activity.
- The activity name.

In another example, for a field value, the key parts include:
- The *Applies to* class, as above.
- The field name, such as `pyActionPrompt`.
- The field value, such as `ViewHistory`.

3. The system discards all unavailable Rules and removes them from the temporary list.
4. The system discards inapplicable Rulesets and versions.

   The system removes from the list Rules that belong to Rulesets and versions that are not available for the current requestor, which can be a user who is currently logged in, a REST API call, or any factor that triggers Rule processing. For example, if the current user can access the *SampleRuleset:05-01-01* Ruleset version, the system removes Rules that belong to *SampleRuleset:04* or *SampleRuleset:06*.

5. The system discards all candidates that are outside of class inheritance of the current class.

   In the temporary list for Rule resolution, the system only maintains Rules in classes from which the current class descends either by pattern or direct inheritance. For more information about class inheritance, see Understanding class hierarchy and inheritance.

   NOTE: This step is not applicable to Rules that do not have the **Use class-based inheritance to arrive at the correct Rule to execute** checkbox selected in their class definition.

6. The system ranks the remaining Rules in the list by analyzing the following aspects, in the following order:
   a. Class

   b. Ruleset

> **NOTE:** Default Rules that you save to a Branch or privately check out are in their own Ruleset.

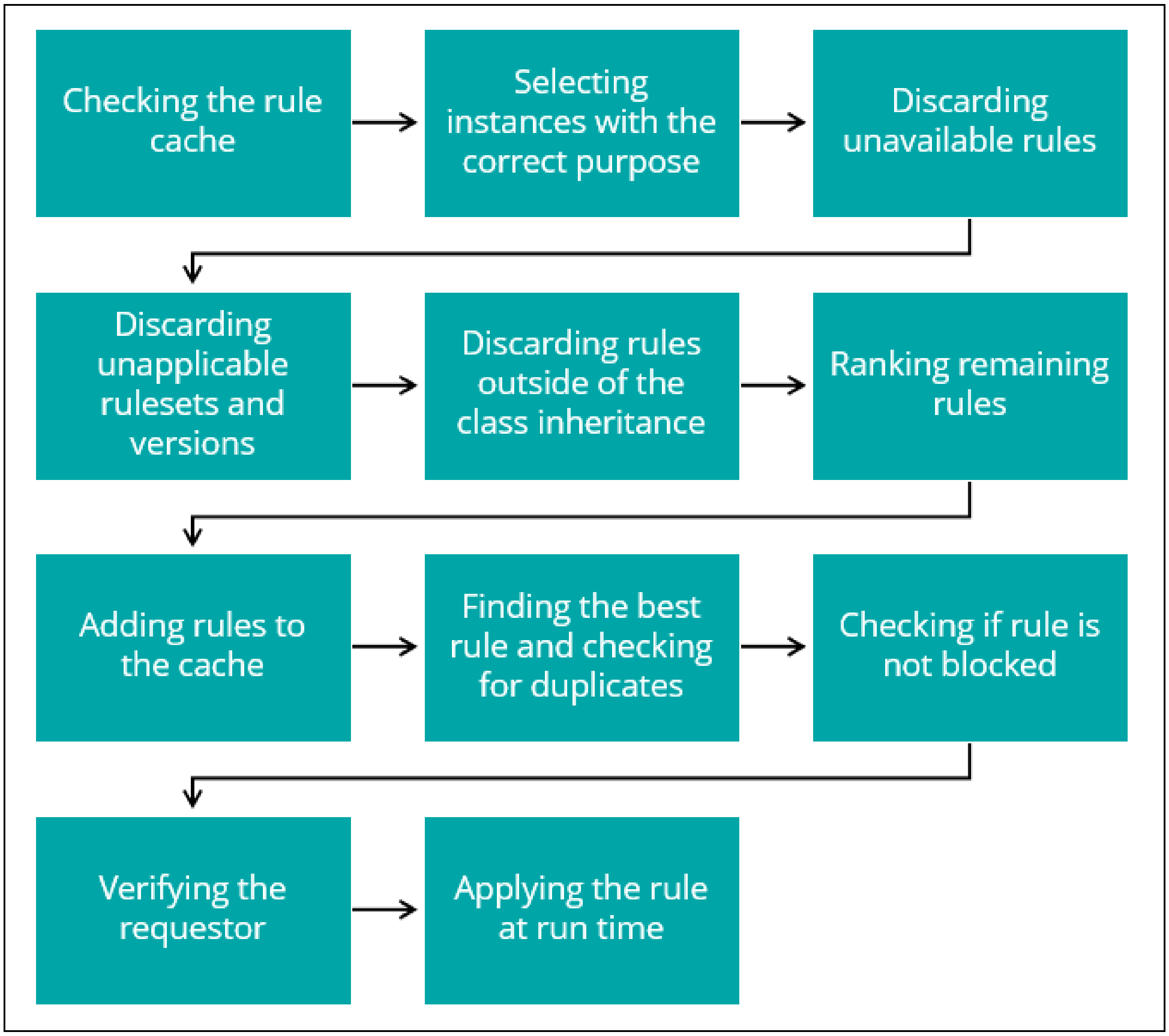
   c. Circumstance
   d. Circumstance date
   e. Version

7. The system adds the Rules that remain in the list to the cache as selectable for use.
8. The system finds the best Rule instance and checks for duplicates.
   The process searches down the list for the first Rule that matches the following criteria:
   - The Rule exactly matches a circumstanced Rule.
   - The Rule has the correct circumstanced date.
   - The Rule is in the correct date/time range.

   When the system finds a Rule that matches these conditions, the process checks whether the next Rule in the list is equally correct. If the next Rule is correct, the process sends a message about duplicate Rules in the system and stops processing. If the Rule resolution process finds no duplicates, the system prepares to use the Rule that matches the listed conditions.

9. The system checks whether Rule availability is not `Blocked`. If the Rule is blocked, the system sends a message that it cannot find an appropriate Rule to use.
10. The system verifies that the requestor is authorized to view the Rule. If the requestor can view the Rule, the system uses the Rule as the output of the Rule resolution process. If not, the system sends a message that it cannot find an appropriate Rule to use.

The following figure shows a diagram that describes consecutive actions that the system performs during Rule resolution. In the figure the Rule resolution process is successful and the system applies a selected Rule at run time:


Checking the rule cache
Selecting instances with the correct purpose
Discarding unavailable rules
Discarding unapplicable rulesets and versions
Discarding rules outside of the class inheritance
Ranking remaining rules
Adding rules to the cache
Finding the best rule and checking for duplicates
Checking if rule is not blocked
Verifying the requestor
Applying the rule at run time

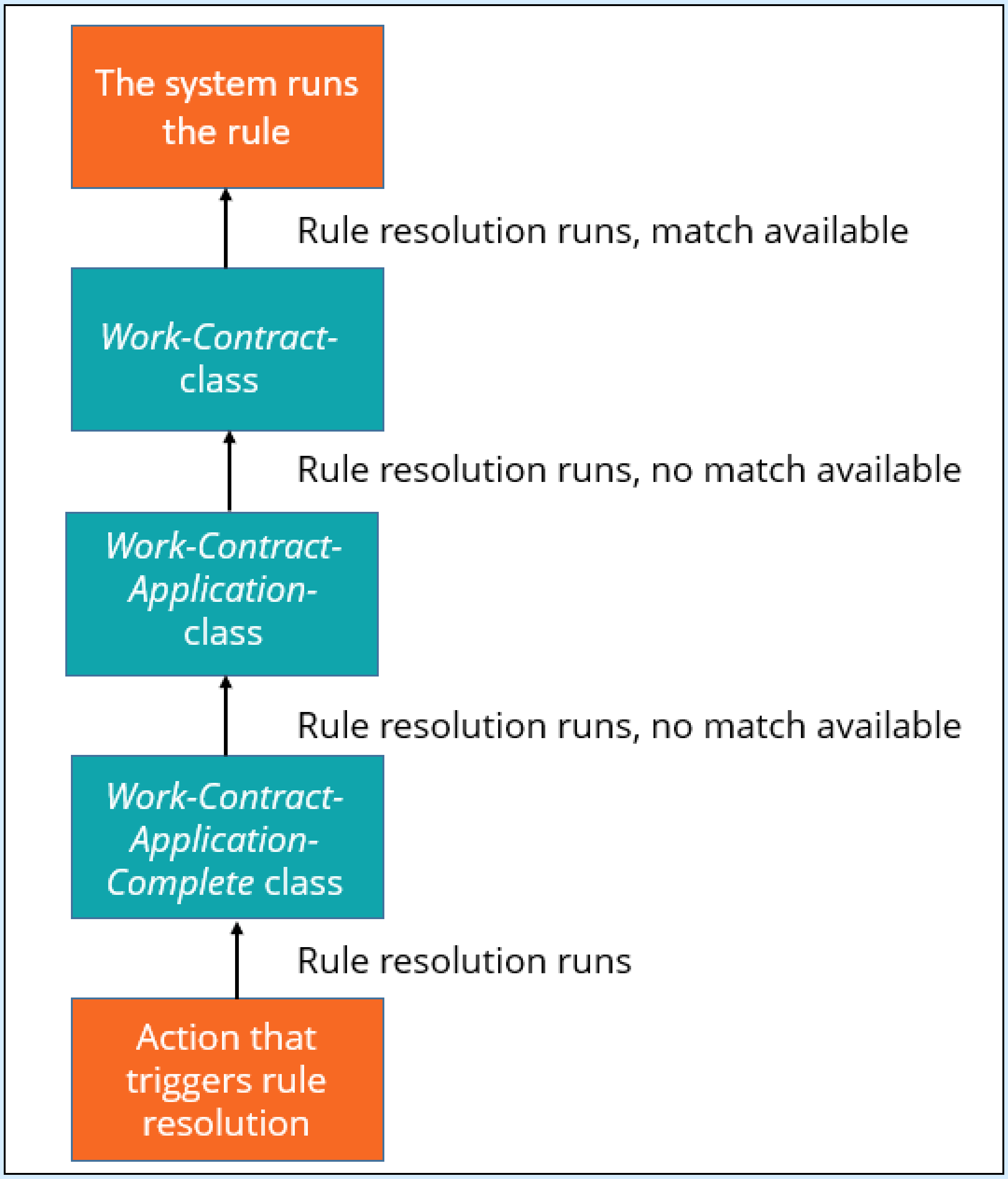
Rule resolution process

For example:

Your use of an application can cause Pega Platform to search for a flow named *Repair* in the *Work-Contract-Application-Complete* class. The system first examines

Rules in the *Work-Contract-Application-Complete* class, and then, if no match is available, searches higher classes in the class hierarchy. Only flows that belong to Rulesets and versions that are present in your Ruleset list are candidates.

The following figure shows a schema of a Rule resolution process that searches through the *Work-Contract-Application-Complete* class, and then moves to classes that are higher in a class hierarchy to find the appropriate Rule:


The system runs the rule
Rule resolution runs, match available
Work-Contract- class
Rule resolution runs, no match available
Work-Contract-Application- class
Rule resolution runs, no match available
Work-Contract-Application-Complete class
Rule resolution runs
Action that triggers rule resolution

Rule resolution process in a class hierarchy

Rule resolution exceptions

Some Rule Types have instances that are not associated with a Ruleset and version, and no Rule resolution occurs when processing requires an instance of these Rules. Additionally, some Rule Types do not support circumstanced processing.

Rule resolution in Branches and withdrawn Rules

When you work in Branches, each Branch has its own Rulesets that the system uses for Rule resolution. However, withdrawn Rules are an exception to this general behavior.

Rules marked as withdrawn in Branch Rulesets, such as deleted UI views, are not withdrawn from the main Ruleset during development. They are withdrawn in the Branch Ruleset. Because the system skips over withdrawn Rules when in different Rulesets, the withdrawn (deleted) Rule can still be resolved in the main Ruleset. When you move the changed Rule to the origin Ruleset by merging the Branch, the system honors the withdrawn Rule and does not resolve it.

For example:

Rule A is available in *TestRS:01-01-01*, but the same *Rule A* is withdrawn in *TestRS:01-01-02*. Rule resolution returns an empty result when you call *Rule A*.

In the next example, Rule resolution logic might return some Rules even though they are withdrawn in a Branch because the Rules are available in the origin Ruleset.

For example:

Rule A is available in *TestRS:01-01-01*. You withdraw *Rule A* and check it into a Branch named *TestingBR*. The withdrawn *Rule A* is now in the automatically generated Ruleset *TestRS_Branch_TestingBR:01-01-01*. The Branch Ruleset is at the top of the stack, but when Rule resolution returns *Rule A*, the system returns *Rule A* from *TestRS:01-01-01*.

- **Ruleset list layers**
- **Ruleset list usage during Rule resolution**
- **Viewing your application Ruleset list**

- **Using pattern inheritance for Rule resolution**

- Creating a Rule specialized by circumstance
- Organizing Rules into Rulesets
- Organizing Rules into classes
- Understanding class hierarchy and inheritance
- Creating Rules

Ruleset list layers

A completed Ruleset list contains sublists of Ruleset versions, arranged in two layers. The following table presents the order of the Ruleset list.

NOTE: Order is significant at the layer, sublist level, and within each sublist.

Layer	Ruleset types	Notes
1	Personal	This Ruleset, at the top of the Ruleset list, holds Rules checked out by the operator. Such Rules are not visible to any requestor except those of operators who have the Allow Rule Checkout? checkbox selected on the Security tab of the Operator ID instance. The name of the Ruleset matches the Operator ID. This single Ruleset has no version and is included implicitly when the system

Layer	Ruleset types	Notes
		assembles the Ruleset list at log-on. You do not need to reference the personal Ruleset on the Access Group form or any other form.
2	Localized	If the application has been localized, Rulesets with names ending in a locale suffix (such as _it_IT for Italian as spoken in Italy, or _fr_CA for French Canadian) display here. Typically, these Rulesets contain only field values. If the application has not been localized, this layer is empty.
3	Mobilized	If the application has been extended to support mobile devices, Rulesets with names ending in the suffix _mobile display in this layer. These Rulesets contain Portals, sections, flow actions, navigation Rules, and controls that are selected by Rule resolution when the HTTP User-Agent value indicates that an HTTP

Layer	Ruleset types	Notes
		request is from a mobile device. If the application has not been mobilized, this layer is empty.
4	Production	Ruleset versions (if any) listed in the Production Rulesets array on the Layout tab of the Access Group form.
5	Application (current)	Ruleset versions listed in the Application Rulesets array on the General tab of the Application Rule form for the current application. A Branch Ruleset (if any) displays immediately above the Ruleset from which it branched. Any shared or component Ruleset versions listed in the Component and shared Rulesets array on the General tab of the Application Rule form for the current application, in the order they display on the tab.

Layer	Ruleset types	Notes
		The system assembles layers 5 and 6 from Ruleset versions listed in the Application Rulesets area of the General tab of any Application Rule that it encounters during log-on when it processes the data instances that contribute to the Rulesets list at login.
6 (multiple)	Application (built-on)	Ruleset versions for built-on applications referenced in the current application Rule. If an application references a lower built-on application, there is a layer for each application in the same relationship as the application Rules, and the contents of the Component and shared Rulesets area of application Rules referenced through the Built on application(s) field are ignored.
7	PegaRULES	Rulesets and versions as listed in the PegaRULES (or PegaDM) application Rule that the application is ultimately built on,

Layer	Ruleset types	Notes
		excluding the Pega-RULES Ruleset version.
8	Pega-RULES:NN-NN	A version of the foundation Rulesets.

- Internationalization and localization

Ruleset list usage during Rule resolution

Each requestor's use of the system continually causes the system to search for a Rule instance, which is known as Rule resolution. This search uses properties from many sources to find the most appropriate Rule for the current need, including class inheritance, security and access control restrictions, and the Ruleset list.

During Rule resolution, the system uses the Ruleset list as it:

- Searches the Rules in the PegaRULES database repeatedly, first using the topmost application on the list, then the next, and so on. The *Pega-RULES* application is searched last.
- Ignores any Rule instances that are contained in a Ruleset on the list but in versions higher than the version specified in the list.
- Treats any partial version number (such as 02-) as a wildcard or mask. For example, matching any Rule instances containing a version that starts with 02-.

Many other factors influence this process, such as:

- Unavailable, final, or blocked Rules
- Time-based Rules
- Circumstances
- Access control (whether the user has authorization to access the Rule)

Viewing your application Ruleset list

An application Ruleset is a collection of Rules that identify the components of an application. You can view a list of your application Rulesets in Dev Studio.

1. Click the **Operator** menu (your name), and select **Profile**. Your operator profile opens.
2. Locate the Application Ruleset section to view your application Ruleset list.

Result:

Your application Ruleset list might include Rules that you do not consider part of the application, or might exclude Rules that you expect. Consider the following criteria:

- The application Ruleset does not include your personal Ruleset, which contains the Rules checked out to you.
- The application Ruleset does not include Rules in Rulesets specified in the Production Rulesets array in your Access Group.
- Rules from Ruleset versions listed in the Requestor Type data instance, Organization data instance, or Division data instance are not included, unless these Rulesets are also in the application Rule.

Using pattern inheritance for Rule resolution

Rule resolution always first looks for a Rule that it needs in the class initially provided.

When the **Find by name first (Pattern)** checkbox (which corresponds to the *Rule-Obj-Class.pyPatternInheritance* property) is selected on the Class Rule form, the system uses pattern inheritance for a class. Pattern inheritance causes the system to next search for a Rule in classes derived from the current class name by truncating, from the right, alphanumeric segments that follow a hyphen.

For example, a request for a Rule in the WalCare-Financial-Credit class causes the system to search through these classes, in the order indicated:

- WalCare-Financial-Credit
- WalCare-Financial-
- WalCare-Financial
- WalCare-
- WalCare

If this checkbox is selected and the Rule is not found, Rule resolution forms a sequence of candidate classes to search by truncating, from the left, portions of the class name that consist only of a hyphen, or consist only of characters other than a hyphen. If no class exists for one of these computed names, Rule resolution continues with the next prefix.

The algorithm

For a class using pattern inheritance, the following is a simplified description of the Rule resolution algorithm:

1. Search each class formed by repeatedly truncating the original class name, stopping if the needed Rule is found.
2. If the pattern search finishes without finding a Rule, go back to the original class and use directed inheritance to find the parent of the original class.
3. If the parent class identified in step 2 was searched during step 1, skip it and skip to step 5.
4. If the parent class was not previously searched, search it. Stop if the needed Rule is found.
5. If not found, determine whether the parent class uses pattern inheritance. If so, repeat this algorithm, starting at step 1, with the parent class.
6. Otherwise, identify the parent of the parent. Continue at step 4.

The ultimate base class *@baseclass* is searched last.

Example

For example, assume your application requests a Rule from *WalCare-Financial-Credit*, which derives from *PegaCare-Financial-Credit*, and WalCare-Financial-Credit uses pattern inheritance.

The system searches for the Rule as follows:

1. Is the Rule in WalCare-Financial-Credit? If not,
2. Is the Rule in WalCare-Financial- ? If not,
3. Is the Rule in WalCare-Financial ? If not,
4. Is the Rule in WalCare- ? If not,
5. Is the Rule in WalCare ? This is the end of the pattern. If the system still cannot find the Rule, it returns to Walcare-Financial-Credit and looks at the parent. If the parent was previously checked because of a similar pattern, it continues with the parent's parent.
6. Is the Rule in PegaCare-Financial-Credit? If not, see if PegaCare-Financial-Credit uses pattern inheritance first. If so, start the sequence again. If not, go next to the parent of PegaCare-Financial-Credit.

Defining and changing the Scope of Rules

Customize, adjust, and define Rules in your application to reflect your business needs and more easily adapt to changes and new features.

Pega Platform™ offers various ways to specialize, modify and manipulate Rules to define the behavior of your application, so that you can address the dynamic business requirements without changing the core logic of your application.

Moving and deleting Rules

Rules can be easily moved or copied between Rulesets and classes to ensure that the localization of the Rule suits the functionality, and the Rule resolution works properly.

With a simple and straightforward process, you can also delete the Rules that are no longer needed for your application. For more information, see Moving Rules.

Setting availability and status of the Rules

The availability and status of a Rule describes how the application behaves during run time. By setting those properties you can ensure that users interact with the correct version of the Rule and that the Rule works properly even if user applies modification and extensions.

By setting Rule availability you control the resolution of the Rule during run time. For example, you can set the Rule status to Final to ensure that the Rule is considered during resolution, but prevent it from being extended or modified by a user because it is part of the core functionality of the application.

The status of the Rule describes how the user can interact with it. For example, you can set the status to Extended, to enable users to override the Rule to provide custom functionality.

For more information, see Setting Rule status and availability.

Defining input parameters of Rules

Defining input parameters allows you to control what kind of information user can provide at run time. For example, you can specify the data format of the input parameters to ensure that users complete a form in the correct way, and make the application displays a list of options to help users provide information quickly and accurately. For more information, see Defining the input parameters of a Rule

Skimming the Rules

In a skimming operation, you copy Rules from a Ruleset into a Ruleset of a higher version to improve the performance of your application. The new Ruleset contains only those Rules that are available for resolution, which minimizes the Rule data that you ship to a different version of your application. For more information, see Rule skimming for higher Ruleset versions

Creating a Rule specialized by a circumstance, Ruleset or class

You can specify the conditions or a time frame for when the Rule resolution is triggered. Specializing Rules by a circumstance makes the application easier to maintain. For example, in a purchase order application you can set a minimum order value that triggers an additional process, such as displaying a form for free shipping. Resolution of the Rules that are a part of a specific Ruleset or a class ensures that users interact with actions that are relevant in a given scenario.

- **Moving Rules**
- **Setting Rule status and availability**
- **Creating a Rule specialized by circumstance**
- **Creating a Rule specialized by a class or Ruleset**
- **Defining the input parameters of a Rule**
- **Defining the pages and classes of a Rule**
- **Rule skimming for higher Ruleset versions**
- **Adding a custom field**

Moving Rules

Adjust the Rule resolution process to your business needs by moving Rules between Rulesets and classes. Moving or copying Rules instead of creating new ones can significantly reduce development time. When you move a Rule, all of the old instances of the Rule are automatically removed from the system, including their references and indexes.

Before you begin:

Ensure that the following requirements are met:

- The Ruleset version or versions containing the Rules that you want to copy are not secured. For more information, see Defining the security of a Ruleset
- The class allows Rules in the target Ruleset.
- Rules that you want to move are checked in. For more information, see Checking in a Rule
- The Rule that you moved does not have the same class, other keys, and Ruleset version as an existing Rule.

1. In the navigation pane of Dev Studio, click **App**.
2. On the **Classes** tab, right-click the class that contains the Rules that you want to move.
3. In the context menu, click **Refactor** > **Move Rules**, and then specify the following filters to define where to move the Rules:
    a. In the Class list, select the destination class to contain the moved Rules.
    b. In the Ruleset list, select the Ruleset to contain the Rules after they are moved. This can be the Ruleset that currently contains the Rule, or a different Ruleset.
    c. In the Version list, select the Ruleset version to contain the Rules after they are moved. This can be the version that currently contains the Rule, or a different version.
4. In the **Rule Type** column, select one or more checkboxes to indicate which Rule Types to move from the list of applicable Rule Types that can be moved.
5. Click **Move**.

Result:

When the process is complete, the form displays the results. Rules that are not moved are marked with a red X. Hover over the red X for more information about why the Rule did not move.

Setting Rule status and availability

Set Rule status and availability to elaborate on the usage information that you provide, and ensure that users interact with the correct version of your Rule at run time. By setting Rule status and availability, you also define application behavior during Rule resolution, and how users can interact with Rules at design time. As a result, you ensure that your application works correctly even if users apply extensions and make modifications.For example, if you need a core functionality of your application to remain unchanged, you can set the status of Rules that build the functionality as final, because users cannot override final Rules.

1. In the navigation pane of Dev Studio, click Records.
2. Expand the category that stores your Rule, and then select a subcategory.

   For example:

   Click Process > Service-Level Agreement.
3. In the header of the form, next to the short description, click the Rule availability label.
   Pega Platform displays Rule availability in brackets.
4. In the Availability list, select an option to control how your application selects the Rule during Rule resolution:
   You cannot change the availability of some Rule Types, such as classes and Rulesets, which are always available to users.
   - To consider the Rule in Rule resolution, select Available.
   - To exclude the Rule from Rule resolution and disable validation logic so that you can continue to edit the Rule even if errors occur, select Not available.

- To consider the Rule during Rule resolution but to stop processing and return an error to the user when the user selects the Rule, select **Blocked**.
- To consider the Rule during Rule resolution but prevent users from extending the Rule by copying it to a different Ruleset unless the Ruleset is a higher version of the current Ruleset, select **Final**.
- To exclude the Rule and all instances with the same name in previous versions of the same major-version Ruleset, from Rule resolution, select **Withdrawn**.

5. In the **Status** list, select an option to indicate how users interact with the Rule:
   - To enable users to provide standard input parameters to this Rule and receive standard return values or results, select **API**.
   - To communicate that Pega Platform no longer supports the Rule, select **Deprecated**.
   - To enable users to override the Rule to provide custom functionality, select **Extension**.
   - To enable users to select this Rule as a starting point to save time when they create a Rule, select **Template**.
6. Click **OK**.
7. Click **Save**.

- **Troubleshooting final Rules**
- **Troubleshooting blocked Rules**
- **Documenting changes in Rule availability**
- **Extension points**

- Base Rules
- Rule resolution

Troubleshooting final Rules

After a Rule is marked as final, no one can create a second Rule with the same visible key in any Ruleset other than the Ruleset that the first Rule belongs to. If the Ruleset version of a final Rule is locked, you cannot create a second Rule with the same visible key in any version of any Ruleset, except a higher version of the Ruleset containing the final Rule.

You cannot override a final Rule except through a higher version in the same Ruleset. Many Rules in the *Pega-** Rulesets are marked Final because correct operation of these Rules is essential to ensure the integrity and security of your system. However, most standard Rules have the Availability field set to Available rather than Final; you can override those Rules in your application.

NOTE: You cannot use circumstances or time-qualified Rules to override a final Rule.

NOTE: You can override a final Rule with a sibling Rule in a language-specific Ruleset.

Conflicts

A final Rule conflict arises if a Rule is marked final but Rules with that same name also exist in other Rulesets. This rare situation can arise when Rulesets are uploaded with the **Import** landing page tab or through other unusual means.

You can find final Rules that have both the same name and other visible key parts by using the Final Conflicts report *Rule-.ListThisRuleFinalCollisions.ALL*. Eliminate any conflicts by deleting one of each conflicting pair until this report is empty. To access this report, in the header of Dev Studio, select **Configure** > **Application** > **Inventory** > **Inventory Reports**. Then, select the **Rule** category, and enter `final` as the search text.

- Setting Rule status and availability
- Extension points
- Creating Rules
- Copying a Rule or data instance

Troubleshooting blocked Rules

Rule resolution processing is halted (with no Rule found) when it encounters blocked Rules. The Rule form colors change from greens to grays for blocked Rules.

Blocked and blocked-by-another

A Rule instance is blocked-by-another if its Availability value is set to Available but a higher-numbered version of this Rule (same name or key, same Ruleset) has the Availability set to Blocked.

Available Rules with that name or key and a different Ruleset may be blocked-by-another as well, if their Ruleset version is appears beneath the Ruleset version of the Blocked Rule on the user's Ruleset list.

When Rule resolution selects a Rule that is blocked, that Rule and all others (same name or key, any Ruleset) are not executable.

To make a Rule available "above" a blocked Rule (that belongs to a secure Ruleset version), choose a higher version number or a Ruleset that appears higher on your (and other users') Ruleset list.

Circumstances

If a Rule has Availability set to Blocked but also has a non-blank Circumstance Property, the blocking affect applies both to that Rule and the base or underlying Rule that has no Circumstance property. A Rule resolution search that meets the Circumstance Property value stops (with no Rule found). The Availability setting in the underlying Rule is not relevant.

However, the converse does not hold. If the Rule with a Circumstance Property has Availability set to Available, and the base Rule has Availability set to Blocked, a Rule request matching the circumstance property and value is successful at finding and using the circumstance-qualified Rule.

Blocked Rules included in ZIP archives

When you create a ZIP archive containing a Ruleset version, any blocked Rules associated with that Ruleset Version are included in the archive (and remain blocked when uploaded into on a destination system).

On a destination system, a blocked Rule can in some cases block a different set of other Rules than it blocked on the source system.

Blocked Rules and skimming

When skimming to a new minor or major Ruleset Version, Blocked Rules are always copied since their purpose is to block all similar Rules regardless of Ruleset name. Blocked Rules can be used, for example, to block another Rule which belongs to a Ruleset name in an underlying Application Layer that will be untouched by the skim process.

- Setting Rule status and availability
- Creating Rules
- Copying a Rule or data instance

Documenting changes in Rule availability

When Rule availability changes to `Withdrawn` or `Blocked`, you can document the change by entering a justification. In addition, developers might document alternate Rules that are intended to replace a blocked or withdrawn Rule. Thanks to that, your application development and refactoring is more transparent.

For example, if a Rule is withdrawn because it was created in error, providing a reason might prevent other developers from changing the Rule availability to `Available`.

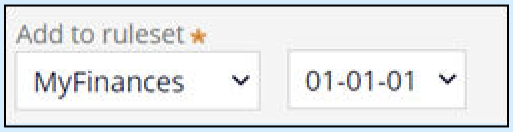
You can also require provision of an alternate Rule to replace a withdrawn Rule. For example, if you withdraw an Activity because it was replaced by a newer Activity, you can provide an alternative Rule that other developers can use instead.

By default, documenting Rule availability changes is optional. You might provide documentation of Rule availability changes by copying and modifying When Rules that Pega Platform™ includes by default. Use the *pyEnableAvailabilityStatusValidation* Rule to request a reason for a Rule's availability change, or the *pyEnableAlternateRuleValidation* Rule to require provision of alternate Rules.

1. In the navigation pane of Dev Studio, click Records.
2. Expand the Decision category, and then click When.
3. In the list of records, open a When Rule that you want to copy:
   - To require developers to provide a reason for Rule withdrawal, click pyEnableAvailabilityStatusValidation.
   - To require developers to provide at least one alternative Rule, click pyEnableAlternateRuleValidation.

> **TIP:** Find the Rule faster by filtering Rules by name.

4. In the Rule form header, click Save as.
5. On the **Save As When** form, in the **Context** section, in the Add to ruleset list, provide your application Ruleset and the Ruleset version.

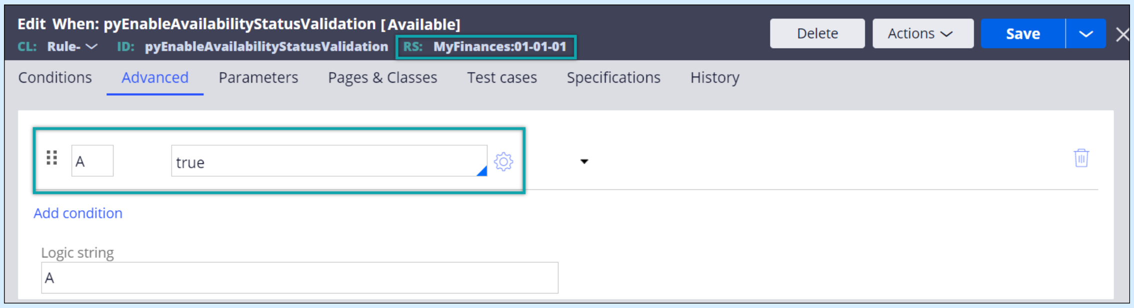
> **For example:**
>
> The following figure shows a sample application Ruleset and version for a My Finances application:
>
> 
>
> Add to ruleset *
> MyFinances | 01-01-01
>
> *Application Ruleset and version*

6. Click **Create and open**.
7. On the **Advanced** tab, in the first row, change the value from `false` to `true`.

> **NOTE:** The configuration applies both to *pyEnableAvailabilityStatusValidation* and *pyEnableAlternateRuleValidation* Rules.

For example:

The following figure shows the configuration for the *pyEnableAvailabilityStatusValidation* Rule. Note that the application Ruleset applies to the My Finances application and not the default Pega Platform Ruleset.


Edit When: pyEnableAvailabilityStatusValidation [Available]
CL: Rule- ID: pyEnableAvailabilityStatusValidation RS: MyFinances:01-01-01

A true

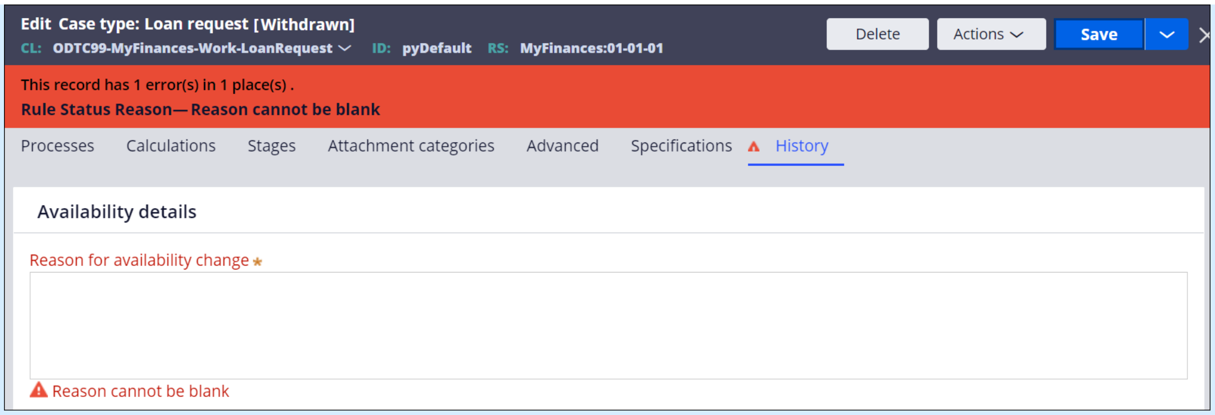
Logic string
A

pyEnableAvailabilityStatusValidation configuration

8. Click **Save**.

Result:

When a developer changes the Rule availability to `Withdrawn` or `Blocked`, based on the When Rules configuration you can make providing either or both a reason or an alternative Rule mandatory. You document Rule availability on the **History** tab of the Rule form. When you try to save a Rule with a `Withdrawn` or `Blocked` availability, the **History** tab displays warnings.
The following figure shows the text field where you can provide reasons for changing the availability of a Case Type Rule to `Withdrawn`:



Edit Case type: Loan request [Withdrawn]
CL: ODTC99-MyFinances-Work-LoanRequest ID: pyDefault RS: MyFinances:01-01-01

This record has 1 error(s) in 1 place(s) .
Rule Status Reason— Reason cannot be blank

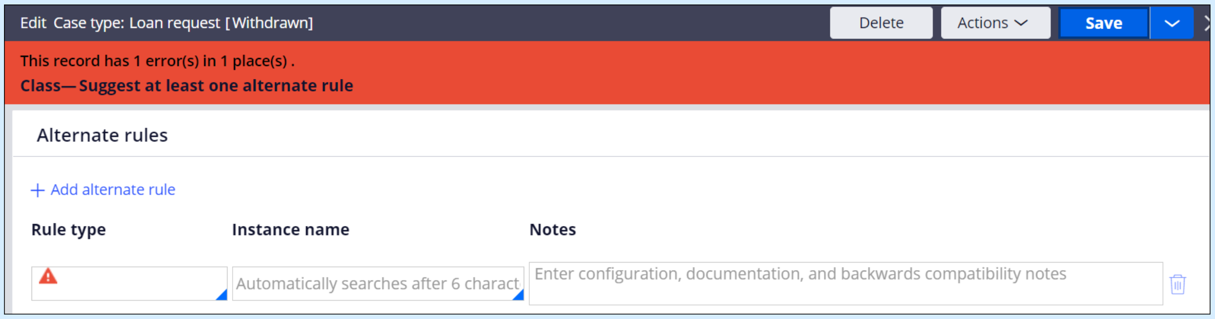
Availability details

Reason for availability change *

Reason cannot be blank

Reason for availability change error

The following figure shows a section where you can provide alternate Rules for a withdrawn Rule:



Edit Case type: Loan request [Withdrawn]

This record has 1 error(s) in 1 place(s) .
Class— Suggest at least one alternate rule

Alternate rules

+ Add alternate rule

Rule type	Instance name	Notes
	Automatically searches after 6 charact	Enter configuration, documentation, and backwards compatibility notes

Alternate Rules error

- Setting Rule status and availability

Extension points

An extension point is an activity or other Rule that is designed and intended to be overridden to meet application needs. Many such Rules are empty stubs that contain no steps. Typically, the overriding Rule is in a derived class.

The Pega Platform itself includes a few dozen extension points. For example, the empty activity *Code-Security.ApplicationProfileSetup* is executed when an operator logs on. If your application requires special initialization processing for authenticated operators, you can override this application (in your application's Ruleset) to perform such processing.

Extension points in frameworks

An extension point is a Rule (typically a blank or empty Rule) that is marked as an Extension in the Status field on the **New** or **Save As** dialog box. The corresponding property *Rule-.pyMethodStatus* does not appear on the Rule form, does not affect Rule resolution or execution of the Rule, and cannot be changed.

For example, a flow in a framework may include an Integrator shape that connects to a customer master file database. In the framework Ruleset, this can be an empty activity with an activity type of Connect, marked as an extension. When Rulesets are an implementation of that framework, the developers must override the empty activity with a functioning interface to their customer master file.

- Setting Rule status and availability

Creating a Rule specialized by circumstance

Create a Rule specialized by circumstance to provide a variant of the Rule that your application triggers only conditionally under specified conditions. By creating specialized or circumstance Rules, you address dynamic business requirements without changing the core logic of your application.

For example, in an application to review loan requests, additional processing can start only when the request amount exceeds a value that you specify. You can also specify a time frame when the Rule is active and available for Rule resolution. When you circumstance Rules instead of applying other solutions, such as decision tables, you create an application that is easier to maintain and edit if you need any adjustments in the future.

Before you begin:

If you want to circumstance a Rule by using a set of conditions, create a circumstance template and definition. For more information, see Creating a circumstance template and Creating a circumstance definition.

NOTE: You cannot create a new Agent Rule by creating a Rule specialized by circumstance. Pega strongly encourages clients to complete activities run as background processes using a Queue Processor or Job Scheduler instead of Agents. For more information, see Using job schedulers and queue processors instead of legacy Rule-type-agents.

For relevant training materials, see a Circumstancing case processing module on Pega Academy.

1. In the navigation pane of Dev Studio, click Records.
2. Expand the category that you want to open, and then open the subcategory that holds the Rule that you want to circumstance.

   For example:

   Expand Process, and then click Flow.
3. In the list of instances, click the Rule that you want to open.
4. In the Rule form header, open the new Rule form:
   - If the Rule is checked out, click the Down arrow next to the Save button, and then click Specialize by circumstance.
   - If the Rule is checked in, click the Down arrow next to the Save as button, and then click Specialize by circumstance.
5. **Optional:** To provide information about the purpose of the new Rule, in the Label field, enter a new short description.
   The identifier remains the same, even if you provide a new label.
6. Define circumstance conditions:

Choices	Actions
Circumstance the Rule by a set of conditions	a. In the **CIRCUMSTANCE BY** section, select Template.

Choices	Actions
	b. In the **Template** field, enter the template that stores properties to evaluate. c. In the **Definition** field, enter a definition that references properties from the template. At run time, your application resolves the Rule if conditions from the definition evaluate to true.
Circumstance the Rule by property	a. In the **CIRCUMSTANCE BY** section, select **Property and Date**. b. In the **Property** field, enter a property to evaluate at run time. **For example:** Enter `LoanAmount`. c. In the **Value** field, enter a value to evaluate against the property. **For example:** Enter `5000`.
Circumstance the Rule by date	a. In the **CIRCUMSTANCE BY** section, select **Property and Date**. b. In the **Date property** field, enter a property that stores a date to evaluate at run time. c. **Optional:** To specify an exact starting time when an application

Choices	Actions
	can resolve the Rule, in the **Start Date** field, select the date. d. **Optional:** To specify an exact time after which an application omits the Rule in Rule resolution, in the **End Date** field, select the date. If you leave a date property blank, your application evaluates the specified time period against the current system date. If the Rule is active during the specified time interval only, then at all other times, the base Rule is active. During Rule resolution, if two or more date circumstance versions are candidates at the current time and date, the application selects the candidates with the nearest end date. If multiple remaining candidate Rules have the same end date, the application picks the candidate with the most recent start date. For more information about possible date combinations and the business-logic requirements that they meet, see Date combinations and business requirements.

TIP: In specific business scenarios, you can also circumstance Rules by a combination of property and date.

7. **Optional:** To override the default work item that your application associates with this development change, in the **Current work item** section, in the **Work item to associate** field, select a work item.
   For more information about your default work item, see Setting your current work item.
8. Update the values in the **Context** and **Add to Ruleset** lists as appropriate.
9. Click **Create and open**.

Result:

At run time, an application resolves the Rule when the conditions that you specified evaluate to true. If the conditions evaluate to false, the application resolves the base form of the Rule. For more information, see Base Rules.

What to do next:

If you skim circumstanced Rules, and you want a skimmed Rule to be recognized as a base Rule during Rule resolution, select the **Base Rule** check box in the Rule availability dialog box. For more information, see Base Rules.

Skimming ignores the **Base Rule** check box and moves Rules from lower to higher Ruleset versions regardless of the **Base Rule** setting and circumstancing. For more information, see Rule skimming for higher Ruleset versions.

- **Creating a circumstance template**
- **Creating a circumstance definition**
- **Controlling the order of circumstance definitions**

- **Redirected Rule**
- **Base Rules**
- **Finding Rules by circumstance**
- **Specializing a Case Type**
- **Rule resolution exceptions**

- Setting Rule status and availability
- Creating Rules
- Circumstance Rule examples

Creating a circumstance template

Create a circumstance template to define a set of conditions that your application evaluates at run time to determine whether a Rule is available for Rule resolution. By creating circumstance templates, you deliver a flexible application that starts relevant processing under specified conditions, without implementing complex and advanced business-logic solutions.For example, in an application to review loan requests, you can define a circumstance template with properties that hold a loan amount, a customer income, and an account type that the customer has. At run time, the application starts relevant processing when the values of the properties meet specified conditions.

Creating a circumstance template is a first step in circumstancing a Rule. As the template holds only properties, you also need conditions to evaluate against the values of the properties.

1. In the header of Dev Studio, click Create > Technical > Circumstance Template.
2. On the **Circumstance Template Record Configuration** form, enter values in the fields to define the context of the template:
   a. In the Label field, enter text that describes the purpose of the circumstance template.

b. **Optional:** To change the default identifier for the circumstance template, click **Edit**, and then provide a unique value in the **Identifier** field.
c. In the **Context** section, select the application to store the template.
d. In the **Apply to** field, press the Down arrow key and select the class that defines the Scope of the circumstance template.
e. In the **Add to Ruleset** field, select the name and version of a Ruleset that stores the circumstance template.
f. **Optional:** To override the default work item that your application associates with this development change, in the **Work item to associate** field, press the Down arrow key, and then select a work item.
For more information about your default work item, see Setting your current work item.

3. Click **Create and open**.
4. On the **Template** tab, in the **Property** field, press the Down arrow key, and then select the name of a property that you want to evaluate at run time.

For example:

Select LoanAmount.

5. In the **Label** field, enter the name that corresponds with the property in a circumstance definition.
When you create a circumstance definition, which has a table layout, each row header displays a property label.
6. **Optional:** To add more properties, click **Add a row**, and then repeat steps 4 through 5.
7. Click **Save**.

What to do next:

Create a circumstance definition that stores conditions to evaluate against the circumstance template. For more information, see Creating a circumstance definition.

Related concepts

- Defining and changing the Scope of Rules
- Building logic and calculating values in your application

Related tasks

- Setting Rule status and availability
- Creating Rules

Creating a circumstance definition

Begin relevant processing in your application only under specified conditions by creating a circumstance definition. When you implement circumstancing in your application, you provide flexible solutions without defining complex logic. As a result, you increase efficiency and design an application that is easier to maintain in the future if your business requirements change.For example, in a banking application, if your circumstance template includes a `LoanAmount` property, and the definition includes the value `5000` with an operator `>`, the application resolves the circumstance Rule only when the loan request amount is greater than 5000.

Before you begin:

Create a circumstance template that stores properties to evaluate by the circumstance definition. For more information, see Creating a circumstance template.

1. In the header of Dev Studio, click Create > Technical > Circumstance Definition.
2. On the **Create Circumstance Definition** form, enter values in the fields to define the context of the definition:
    a. In the Label field, enter text that describes the purpose of the circumstance definition.

   b. **Optional:** To change the default identifier for the circumstance definition, click Edit, and then in the Identifier field, provide a unique value.
   c. In the Template Name field, press the Down arrow key, and then select the circumstance template that your circumstance definition implements.
   d. In the **Context** section, select the application to store the template.
   e. In the Apply to field, press the Down arrow key and select the class that defines the Scope of the circumstance definition.
   f. In the Add to Ruleset field, select the name and version of a Ruleset that stores the circumstance definition.
   g. **Optional:** To override the default work item that your application associates with this development change, in the Work item to associate field, press the Down arrow key, and then select a work item.
      For more information about your default work item, see Setting your current work item.
3. Click Create and open.
4. On the Definition tab, click the column header to choose which operations the column supports:
   - For columns that evaluate properties from the template against a single value, select an operator from the Use Operator list, for example >.
   - For columns that evaluate properties from the template against a range of values, select the Use Range checkbox, and then choose operators from the Starting Range and End Range lists, for example >= and <=.
5. Click Save.
6. Click a cell in a table that you want to define, and then enter a value or an Expression that evaluates properties in the template.

   For example:

   To evaluate loan amounts greater than 5000, use a single operator >, and then enter `5000`.
7. **Optional:** To provide more values for evaluation, add rows to the table:

- To add a row before the current cell, on the toolbar, click **Insert Row Before**, and then repeat step 6.
- To add a row after the current cell, on the toolbar, click **Insert Row After**, and then repeat step 6.

> **TIP:** As your application evaluates the table from top to bottom, to save time and resources, place the most likely outcomes at the top of the table.

8. **Optional:** To ensure that your application can process the table, check the table for conflicts by clicking **Show conflicts** on the toolbar.

   > **Result:**
   >
   > A warning icon appears in rows that are unreachable or empty.

9. Click **Save**.

> **Result:**
>
> At run time, an application evaluates the values from the circumstance definition against the properties in the circumstance template and resolves the Rule only when the values evaluate to true.

- **Date circumstance Rule**
- **Circumstance Rule examples**
- **Circumstance definitions**
- **Date combinations and business requirements**

Related concepts

- Defining and changing the Scope of Rules
- Building logic and calculating values in your application

Related tasks

- Setting Rule status and availability
- Creating Rules

Date circumstance Rule

A date circumstance Rule is a circumstance that is only resolved during a specified range of time.

When a date circumstance is applied, the value of the date property and the specified start and end date determine the time period during which the Rule will be active.

Restrictions

- Date circumstances can only be created for supported Rule Types. This is controlled by the Allow Rules that are valid only for a certain period of time checkbox on the Class form.
- You cannot override a final Rule with a date circumstance Rule.
- When date circumstance Rules are used in a multinode system, be sure to synchronize the internal clocks of all the server nodes in the cluster. Clock differences of less than a few seconds may lead to incorrect application results. Most operating systems offer facilities for such synchronization.

- Setting Rule status and availability
- Creating a Rule specialized by circumstance
- Circumstance Rule examples

Using more than one time-qualified Rule

Your application can include multiple date range circumstances for the same base Rule with overlapping (but not identical) date and time intervals. At run-time, Rule resolution processing finds all the time-qualified Rules with an interval that includes the current date and time. It then selects the best Rule to run based on the following tests:

1. Examine the end dates on each candidate time-qualified Rule. Choose the Rule or Rules that have the nearest end date, discarding others.
2. If only one candidate remains, that Rule is the result of this phase of Rule resolution processing.
3. If two or more candidates remain, the one with the most recent start date is selected.

- Date circumstance Rule

Contrasting time-qualified Rules and historical processing

Pega Platform offers two separate features that can cause processing to be dependent on a date or time value. Which of these to apply to best meet an application need depends on the nature of the Rules affected and the requirements and environment of the application:

- Date circumstance Rules
- Historical processing capability

How circumstance Rules evaluate time

Date circumstance Rules are specialized versions of a base Rule that only execute when certain time conditions are met.

Based on the combination of specified date property and the time interval, a Rule can be resolved in one of the following ways:

Result	Date property	Start Date	End Date
Rule to be effective only if the value of the specified date property occurs within a date range	Select	Select	Select
Rule to be effective only if the value of the specified date property occurs after a certain date	Select	Select	
Rule to be effective only within a date range		Select	Select
Rule to be effective only after a certain date		Select	

How historical processing evaluates time

The historical processing capability applies to an entire Ruleset version, not to a Rule. This capability is unrelated to the concept of circumstancing and base Rules.

With careful advanced design, this feature allows an application (for the current requestor) to operate according to Rules as they were on a specific past date. Such processing is useful to reconstruct past behavior or apply past policies.

- Creating a Rule specialized by circumstance

Circumstance Rule examples

Assume that you have different pricing levels for your customers. You first define a base pricing Rule for all customers. Then you qualify the base Rule by creating circumstanced Rules for customers at different buying levels. The property .CustomerType is part of the customer order and has values of "Silver" and "Gold". In this example, a customer has purchased a $100 item. Using the property and values, you create circumstanced instances of the base Rules as shown here:

- Base — if .CustomerType="none", then Price =$100
- Circumstance 1 — if .CustomerType = “Gold”, then price = $100 - 25%
- Circumstance 2 — if .CustomerType =“Silver”, then price = $100 - 10%

When the system processes the order, the value of that property dictates which Rule is run and thereby determines the discount (if any) the customer receives.

Properties in a circumstance

The example above used the single value property.CustomerType to qualify the base pricing Rule. Specifying a property value in a circumstance is only supported by certain Rule Types; refer to the **Allow Selection based on Property Rules?** checkbox selected on the Class form defining the Rule Type.

You can also specify a date property along with the time period against which the value of the date property is evaluated.

For example, an annuity is an investment vehicle providing scheduled payments for many years (or in some cases for the life of the investor). A work item supporting annuity processing needs that an expiration date (say ExpireDate), occurs between December 15, 2015 and December 31, 2015. If a few processing aspects of the work item depend on this value, they can be derived from a base Rule by specifying a Date property of .ExpireDate and the start date and end date as 12/31/2015 and 12/31/2015.

A single Rule can contain both a circumstance property and a date circumstance property. For the circumstance property value, an exact match is required. For the date circumstance, when a date property is specified its value must occur after the specified start date and before the end date. If a date property is not specified, the system time at the time of processing should occur after the specified start date and before the end date.

NOTE: If two base Rules with the same **Apply to** key part and family name both have one or more associated property circumstanced Rules, the same circumstance property must be used. For example, if activity MyClass.Alpha has an associated circumstanced Rule using property .State, then another activity MyClass.Alpha cannot have a circumstance Rule with any property other than .State.

Dates in a circumstance

A date circumstance defines a period of time where a version is eligible for selection during Rule resolution.

For example, assume that the normal Service-Level Agreement for a retail operation allows four days as the goal time. Management might decide (to accommodate an extraordinary volume crunch in December or January) to create a temporary Rule with six days as the goal time. The Start Date of the circumstanced by date Rule can be set to December 1 of a year, and the End Date to January 31. No other changes to the application are necessary; Assignments created anytime during those two months have the longer intervals.

Multiple properties in a circumstance

You can use multiple properties to circumstance a Rule by more than one feature. For example, an insurance company may be required to collect, for every state, claims that exceed a specific amount and the method by which the claims were conveyed (letter, phone, email, and so on). Another example may be the need for a Rule that is specialized by an age range and not an absolute value.

If you use multiple properties in a circumstance, you select the following records:

- Circumstance Templates (Rule-Circumstance-Template Rule Type)
- Circumstance Definitions (Rule-Circumstance-Definition Rule Type)

NOTE:

- For decision tables, you can redirect one circumstanced Rule to another circumstanced Rule (a peer with the same base Rule), to reduce the

number of distinct Rules that you need to maintain. Select the **Redirect this Rule** checkbox on the **Results** tab.
- A Rule cannot be a circumstance if another Rule of the same type and name exists in a different class, even if the classes are unrelated and have no inheritance relationship, and even if the conflicting Rules are set to withdrawn Rule. This is because the Applies To class of a Rule is not taken into account when looking for conflicting circumstance Rule instances, only the name.

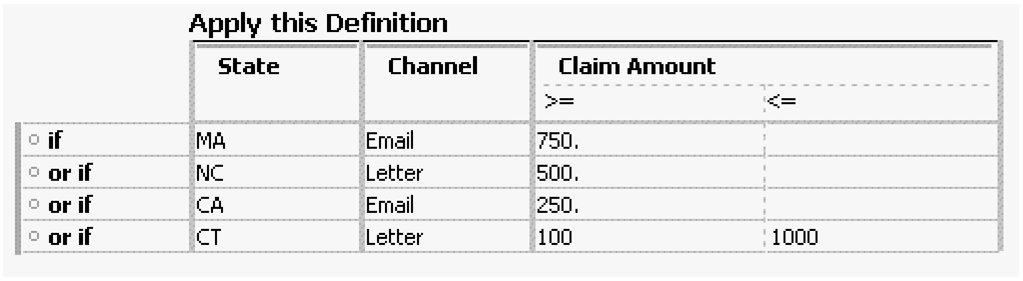
- Creating a Rule specialized by circumstance
- Finding Rules by circumstance
- Rule resolution
- Circumstance definitions

Circumstance definitions

A circumstance is an optional qualification available for supported rule types and is built upon the base rule.

Using circumstances in your application allows you to easily support a variety of use cases. For example, you might want a different data transform to execute depending on a customer's geographic location (for example, when `.StateCode` = "MA"). Instead of maintaining the logic for all cases in one large rule, start with a base rule and extend (or specialize) it as needed. At runtime, the Pega Platform automatically selects the correct version to execute as part of its rule resolution process.

The following figure illustrates example logic for evaluating geographic location and other properties in a case.



	Apply this Definition			
	State	Channel	Claim Amount	
			>=	<=
if	MA	Email	750.	
or if	NC	Letter	500.	
or if	CA	Email	250.	
or if	CT	Letter	100	1000

You can create a circumstance of a base rule by a single value property, template (multiple properties), or date.

NOTE: If you have upgraded to Pega 7.2, all date circumstance rules will be upgraded automatically and can be modified, except the rules circumstanced by date range and as-of date. These rules continue to run without issues but must be manually upgraded to use the new date circumstancing options.

- Creating a circumstance definition

Date combinations and business requirements

The following table lists the possible date combinations and the business-logic requirements they meet when you circumstance by date.

Business requirement	Specify date property	Specify start date	Specify end date
Rule to be effective only if the value of the specified date property occurs within a date range	Yes	Yes	Yes
Rule to be effective only if the value of the specified date property occurs after a certain date	Yes	Yes	No
Rule to be effective only within a date range	No	Yes	Yes

Business requirement	Specify date property	Specify start date	Specify end date
Rule to be effective only after a certain date	No	Yes	No

- Creating a Rule specialized by circumstance

Controlling the order of circumstance definitions

Provide relevant processing in scenarios when multiple conditions evaluate to true, to ensure that users of your application interact with the intended data and application behavior. By defining the priority of circumstance definitions, you determine how your application behaves when more than one definition meets specified conditions.Consider a situation in which a circumstance template includes a property that holds a loan amount. Two circumstance definitions correspond with the template; one definition that evaluates the property against values greater than 4,000, and a second definition that evaluates a property against values greater than 7,000. A customer creates a loan request for 10,000. Because both circumstance definitions evaluate to true, you can select which definition you want your application to use.

Before you begin:

Define a circumstance template that stores properties and at least two circumstance definitions that store values to evaluate against the template. For more information, see Creating a circumstance template and Creating a circumstance definition.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Technical** category, and then click **Circumstance Template**.
3. In the list of circumstance templates instances, click the template that you want to open.
4. Click the **Definitions** tab.

Result:

The **All Definitions** section displays all circumstance definitions that correspond to the open circumstance template.

5. In the **Priority Definitions** section, in the field in row **1**, enter the circumstance definition that you want your application to run first.
6. To add more circumstance definitions, click Add a row, and then in the row that appears, enter the definition that you want your application to run next.

TIP: You can also drag a field to a different position in the list to change its run-time priority.

7. To check the correctness of your circumstance definitions, on the toolbar, click Show Conflicts.

Result:

A read-only decision table opens in a separate window that displays all the definition values for all the circumstance definition Rules in the specified order. A warning icon appears next to the rows of the table that are unreachable. A warning message also appears when you attempt to save a form that contains conflicts. Even though the tables might not have any conflicts individually, conflicts might generate when the system assembles the definitions as a single table at run time.

8. Click Save.

Related concepts

- Building logic and calculating values in your application

Related tasks

- Creating Rules

Redirected Rule

A redirected Rule is a circumstance Rule that is configured to reference another circumstance Rule for the same base Rule. Redirection is available for Decision Tables (*Rule-Declare-DecisionTable*).

At run time, if Rule resolution initially selects a redirected Rule, the system accesses the target of the redirection, and that second Rule runs. (If the target is not found, the base Rule runs.) The system ignores all fields on the redirected Rule form, except for the redirection details and the Rule name.

Benefits

Redirection can simplify maintenance of a set of circumstance Rules. For example, an application might include a decision table named SalesTax that computes state sales taxes for a state and is circumstanced by state code, such as VA for Virginia.

Your application needs one base Rule and 50 circumstance-qualified Rules, one for each state code. However, if the tax structure of several states (such as Alaska, Delaware, Florida, Montana, and New Hampshire) is identical, you can:

- Create a circumstance Rule for Alaska (with AK as the circumstance value) that defines the Alaska state sales tax.
- Create circumstance Rules for Delaware, Florida, Montana, and New Hampshire which are marked to redirect to the AK Rule.

Redirection does not reduce the number of circumstance Rules you need to create, but it can reduce your maintenance effort when several of them operate alike (that is, are functionally equivalent).

NOTE:

- Redirection is possible only for single value circumstancing and not for template circumstancing.
- The target of a redirected Rule can be another redirected Rule.
- You cannot redirect a Rule to itself. You should avoid creating a circular set of redirections (A to B and B to A), as doing so causes an exception at run time.

- Creating a redirected Rule
- Circumstance Rule examples
- Creating a Rule specialized by circumstance

Creating a redirected Rule

To create redirected Rules, which are circumstance Rules that are configured to reference other circumstance Rules for the same base Rules, complete the following tasks.

1. Create a base Rule for the decision table.
2. Using the **Save As** toolbar operation, create each circumstanced Rule that provides distinct results.
3. Create additional circumstance Rules for other values of the Rule.
4. Open the circumstance Rule that you want to redirect to another circumstanced Rule.
5. On the **Results** tab of the circumstance version of the decision table that you want to redirect to the target Rule, select the **Redirect this Rule** field.
6. From the **Circumstance Value** list, select the circumstance value of the target Rule.

- Decision tables
- Creating a Rule specialized by circumstance

Base Rules

A base Rule is the fallback Rule selected by Rule resolution when no other circumstance version's criteria is met. Base Rules have no circumstance qualification.

A base Rule must exist for every circumstance Rule.

Restrictions

- Circumstance Rules are valid only when certain qualifications are met. You cannot delete a base Rule when a circumstance Rule with the same key exists; the circumstance version provides a fallback.
- For Rules with an **Apply to** key part, the base Rule can have an **Apply to** class that is a parent or ancestor class of the **Apply to** class of the circumstance Rule.
- You cannot check out a Rule that is a base Rule when a related circumstance Rule is checked out.
- You cannot check out the base Rule of a date circumstance Rule at a time between the start and end date and time.
- If a date circumstance stream Rule contains JSP tags (rather than directives), the base Rule must also contain JSP tags rather than directives. Conversely, if the date-circumstanced Rule contains directives, the base Rule must contain directives.
- In releases before PRPC Version 5.2, circumstance Rules with a Ruleset version number lower than the Ruleset version of a base Rule were ignored (never selected) during Rule resolution. Starting with PRPC Version 5.2, this is not the default Rule resolution behavior. You can copy a base Rule to a higher version without the need to also copy each of the (possibly many) circumstanced Rules associated with the base Rule into that higher version. While not recommended, you can revert to previous behavior by selecting the **Base Rule** checkbox on the form produced by the Availability label on the Rule toolbar. The **Base Rule** checkbox only impacts Rule resolution. Other tools, such as skimming a Ruleset, do not honor the checkbox.

- Creating a Rule specialized by circumstance
- Setting Rule status and availability

- Circumstance Rule examples

Finding Rules by circumstance

To verify if an existing Rule is either a base Rule or a circumstance, open it in Pega Platform™ and inspect the form header. Click the **Circumstanced** link to see circumstance values or an indication that the Rule is a base version. The link does not appear if the existing Rule is neither a base nor a circumstanced version.

To see all available circumstances for a given base Rule, navigate to the Rule in the Application Explorer. Use the expand icon next to the name of the base Rule to display each of the circumstanced instances and their values.

Select **Configure** > **Case Management** > **Tools** > **Find Rules** > **Find by Circumstance** to create a report comprising single-property and multiple-property circumstance Rules. You can filter the report by searching on the following default circumstance properties: U.S. State Codes, Channels, and Customer Level. You can also search for circumstance Rules by entering text used in key parts (**Apply to** or **Identifier**) in the **Name Contains** field.

For more information on how to report on circumstance Rules, or how to add or change the circumstance property columns for the report, see the Pega Community article *How to find Rules with a specific circumstance*.

Use the *Data-Circumstance-Duplicates.CircumstanceMultiples.ALL* report to identify single circumstance Rules that incorrectly use two or more different properties. Such conflicts can arise when Rules are imported into a Ruleset that already contains some circumstanced Rules. To access this report, select **Configure** > **Application** > **Inventory** > **Inventory Reports**. Enter `Rule` as the Category and circumstance as the search text.

- Creating a Rule specialized by circumstance
- Setting Rule status and availability
- Circumstance Rule examples

Specializing a Case Type

Specialize a Case Type to support variations in the way that Cases are processed in your application. Each version that you create is uniquely identified by a circumstance.

> **TIP:** Specialized versions are not automatically added to the parent Case Types of your base Case Type. Open the Case Type form for each parent Case Type to add your specialized version as a child.

1. In the navigation pane of App Studio, click **Cases**, and then click the Case Type that you want to open.
2. On the **Settings** tab, click **Specialization**.
3. Press the Down Arrow key in the autocomplete field and select the name of a single-value property.

   > **NOTE:** Case Types support specialization by a single-value property. The property that you specify is used for this version and all future, specialized versions that you create. This means that the autocomplete field is hidden the next time that you follow this procedure.

4. Click **+ Add specialization**.
5. Define a property value for the circumstance by using one of the following fields:

   - Text input — Enter a string value without quotes that is consistent with the property type.

     For example, enter 100 for a property that stores integer error codes.

   - Drop-down list — Select a value that maps to an entry from the property's configured table type.

For example, select XL when the property is configured as a local list of clothing sizes.

6. Click **Save**.

Result:

The values that you provide are used to create a specialized version of the base Case Type. To view your specialized Case Types in the Case Type Explorer, expand the name of a base Case Type.

Rule resolution exceptions

Understanding Rule resolution of circumstance Rules across Rulesets can help you troubleshoot unexpected Rule resolution results.

Privately edited and branched Rules

When privately edited or branched Rules are matched, circumstance Rules do not work because they are in a different Ruleset. In the following example, line 1 is a privately edited Rule and line 2 is a branched Rule. If either of the Rules is matched, the circumstance Rules are ignored.

```
1. MyRule in myusername@ ***
2. MyRule in MyRuleset_branch_BranchName Base Rule ***
3. MyRule in MyRuleset 01-01-05 Circumstance by Property: .pyLabel = Green
4. MyRule in MyRuleset 01-01-05 Circumstance by Property: .pyLabel = Yellow
5. MyRule in MyRuleset 01-01-05 Base Rule
6. MyRule in MyRuleset 01-01-04
7. MyRule in MyRuleset 01-01-03 Circumstance by Property: .pyLabel = Yellow
8. MyRule in MyRuleset 01-01-02 Circumstance by Property: .pyLabel = Green
9. MyRule in MyRuleset 01-01-02 Circumstance by Property: .pyLabel = Red
```

```
10. MyRule in MyRuleset 01-01-01 Circumstance by Property: .pyLabel = Green
11. MyRule in MyRuleset 01-01-01
```

Circumstance Rules outside of the matched Ruleset

Circumstance Rules are honored only if they are in the same Ruleset as the matched non-circumstance Rule. For example, if you have a Rule stack as shown in the following example, and line 3 is the matched Rule, only the circumstance Rules for green and yellow are honored. All Rules below line 3 are discarded even though their circumstancing matches.

```
1. MyRule in MyRuleset 01-01-01 Circumstance by Property: .pyLabel = Green
2. MyRule in MyRuleset 01-01-01 Circumstance by Property: .pyLabel = Yellow
3. MyRule in MyRuleset 01-01-01
4. MyRule in BusinessRules 01-01-03 Circumstance by Property: .pyLabel = Yellow
5. MyRule in BusinessRules 01-01-02 Circumstance by Property: .pyLabel = Green
6. MyRule in BusinessRules 01-01-02 Circumstance by Property: .pyLabel = Red
7. MyRule in BusinessRules 01-01-01 Circumstance by Property: .pyLabel = Green
8. MyRule in BusinessRules 01-01-01
```

- Rule resolution

Creating a Rule specialized by a class or Ruleset

Provide a version of a Rule that your application triggers only during resolution of Rules that belong to a specified class or Ruleset. When you define the conditions for a Rule resolution, you ensure that users interact with actions and the application behavior that are relevant in a given scenario. You also save time and resources because you promote reuse across your application.

Consider a scenario in which Vehicle-Insurance and Loan-Request classes in your application are outside a class hierarchy and have no shared Rulesets, but you want to apply a Service-Level Agreement (SLA) Rule from the Loan-Request class to the Vehicle-

Insurance class. You can specialize the SLA Rule so that your application can resolve it during processing of the Vehicle-Insurance class.

Specializing Rules by class or Ruleset saves time and improves consistency in your application, because when you specialize a Rule, the system does not create a new Rule instance with a new identifier but only references a Rule that already exists. As a result, when you edit a Rule, your changes apply in classes and Rulesets that reference that Rule, and you avoid editing multiple copies of the Rule.

1. In the navigation pane of Dev Studio, click Records.
2. Expand the category that you want to open, and then open the subcategory that holds the Rule that you want to circumstance.

   > **For example:**
   >
   > Expand Process, and then click Flow.

3. In the list of instances, open the Rule.
4. In the Rule form header, click the Down arrow next to the Save button, and then click Specialize by class or Ruleset.
5. **Optional:** To provide more information about the purpose of a new Rule, in the Label field, enter a new short description.
   The identifier remains the same even if you provide a new label.
6. In the **Context** section, define a new location with which to reference your Rule:
   a. In the list of built-on applications, select an Application Layer for the Rule.
   b. In the Apply to field, enter a class that you want to associate with the Rule.
   c. In the Add to Ruleset list, select a Ruleset and a Ruleset version to associate with the Rule.
7. **Optional:** To override the default work item that your application associates with this development change, press the Down arrow key in the Work item to associate field, and then select a work item.
   For more information about your default work item, see Setting your current work item.

8. Click **Create and open**.

Result:

The system resolves the Rule during Rule resolution of the specified class and Ruleset.

Related concepts

- Rule resolution
- Building logic and calculating values in your application

Related tasks

- Copying a Rule or data instance

Defining the input parameters of a Rule

Control the type of information that users can pass to a Rule by defining input parameters. By referencing these input parameters in your Rule logic, you can use run-time data to make decisions.

For example, you can define income, debt, and expense information as input parameters to a Rule in a life insurance application. At run time, the Rule calculates a debt-to-income ratio to determine which policy plan meets the customer's lifestyle.

NOTE: Some Rules do not support parameters or support only a subset of the following options. The names of fields vary, based on the type of Rule that you update.

1. In the navigation pane of Dev Studio, click **Records**.
2. Open the form for a Rule that supports parameters.

> **For example:**
>
> To edit an activity, in the **Records** section, expand the **Technical** category, click **Activity**, and then, in the list of instances, open the activity that you want to configure.

3. Click the **Parameters** tab.
4. Define an input parameter that your Rule supports:
    a. Click the **Add item** icon.
    b. In the **Name** field, enter a unique identifier.
    c. In the **Description** field, enter text that describes how your Rule logic processes the input parameter.
    d. In the **Data Type** list, select a format for the input parameter.
    e. **Optional:** To require a non-null value for the input parameter, in the **Required?** list, select **Yes**.

    > **NOTE:** Do not make Boolean input parameters required because your application might interpret a false value as a null or blank value.

    f. **Optional:** To set the initial value of the input parameter, enter a value in the **Default Value** field that corresponds to the format that you select in the **Data Type** list.
5. If the input parameter has the **Data Type** field set to **Page Name**, on the **Pages & Classes** tab, add a corresponding page name and class.
   For more information, see Defining the pages and classes of a Rule.
6. On other tabs of the Rule form, update your Rule logic to reference the input parameter by using the following notation: `param.[`*input parameter name*`]`
7. To add another input parameter, click **Add item**, and then repeat steps 4 through 6.
8. Click **Save**.

What to do next:

You can test your changes by running the Rule and providing values for the input parameters.

- Defining the pages and classes of a Rule
- Defining the local variables for an activity

Prompting users for parameter values

You can prompt users for values when they run a Rule that has input parameters. By displaying a list of options instead of a blank text box, you can help users provide information quickly and accurately.

NOTE: Some Rules do not support parameters or support only a subset of the options below. Names of fields may vary, based on the type of Rule that you are updating.

1. Defining the input parameters of a Rule.
2. In the header of the Rule form, click **Actions** > **View references** to display a list of Rules that call your Rule.
3. Inspect the list to ensure that your Rule runs in response to an action that a user performs, such as a flow action, because parameter prompting does not occur when a Rule runs programmatically.
4. Click the **Parameters** tab.
5. In the **SmartPrompt type** field, enter the first key part, which is typically the class, of the input parameter.

For example, you can enter `Rule-Message` to display a list of messages that are available in the run-time class path.

6. To refine the options in the list, enter the second key part of the input parameter in the **Validate as** field.

   For example, you can enter `pyCaption` when the **SmartPrompt type** field is set to `Rule-Obj-FieldValue` to display only field values with a field name that contains "pyCaption".

   You can also reference a property in this field.

7. Click **Save**.

You can test your changes by running your Rule.

- Accelerating Case processing with Pega GenAI, optional actions, and collaborative tools
- Defining the input parameters of a Rule

Defining the pages and classes of a Rule

Many types of Rules can access or update information on various Clipboard Pages when they run. Most of these Rule Types include a Pages & Classes tab that you can use to provide important information about what your Clipboard Pages will look like during execution.

- Setting up Rule prompting during when you configure forms. For example, if you specified that a page belongs to the Work-class in the Pages & Classes tab of an activity, when the page is used as a step page, prompting shows results from the Work- class.
- Configuring Rule validation. The information specified in the Pages and Classes tab defines how Rule references are validated during Rule validation.

- Applying the default class for list elements. When you define a list or group page on the Pages and Classes tab, the system automatically sets the class of those elements when new entries are added.

Guidelines

To complete a Pages & Classes tab, follow these guidelines:

- Leave this tab blank if the other tabs of the Rule do not reference any pages.
- Complete this tab to describe the Clipboard context expected at run time.
- Page names are case-sensitive, so be sure to match the case of the page name.
- Enter the name of each page referenced explicitly on the other tabs of the Rule form. For Rules that contain HTML or XML text, such as Correspondence or HTML Rules , the reference can be inside a JSP tag or directive.
- To set the class of the Primary Page, add a row that has no value for the page name and the desired class name.
- If you use the keyword Top on other tabs of the Rule form, such as in embedded properties, add a row with the keyword Top in the **Page Name** column and the class of the top level page in the **Class** column. You frequently do this when the name of the top-level page name varies in different settings.
- For an embedded page enter the full page name. You can also enter the full page name starting with the top-level page name or with the keyword Top. For List and Group arrays, use trailing (). For example:
    - `myPage.ListA().PageA`
    - `myPage.ListA().listB()`

  Page names must be absolute property references. Relative references, such as `.myEmbeddedPage` are not supported. Instead, use the keywords `Primary` or `StepPage` to make the path absolute. For example:

    - `Primary.myEmbeddedPage`
    - `StepPage.myEmbeddedPage`

- You do not need to list all the Clipboard Pages expected to be on the Clipboard, only those that are referenced in this Rule.
- For HTML Rules, XML Stream Rules, and other stream Rules containing JSP tags, list all the pages identified in the HTML text, including system pages (`pxRequestor`, `pxProcess`, `pxThread`, and their embedded pages, if mentioned).
- You cannot use the keywords `$ANY` or `$CLASS` in the Class column.
- Do not use the keywords $ANY, $CLASS, or $None in the **Class** column; these keywords are deprecated.
- If the **Class** column is blank, the Applies To class of the Rule is used by default.
- For Rules that run in the context of a Primary Page, you can leave the page name blank for greater flexibility. (The class of the Primary Page matches the Applies To key part of the Rule.)
- You must have classes for Data Pages defined on the **Pages & Classes** tab. When you add a parameterized Data Pages to the tab, do not use parameters or brackets.

NOTE:

- The context at run time often differs from the context at design-time.
- You can enter rows in the list in any order.
- You cannot use the Top keyword in the Pages & Classes tab of a list view or summary view Rule.
- Using the Top keyword in a Data Transform does not reference the top-level page.
- You must list embedded pages in the Pages & Classes tab if the class of the embedded page varies from the class defined for the property at run time.

Run-time processing

The pages that are present when a Rule is running might vary from execution to execution. The system neither requires nor checks that all the pages listed are present each time it runs the Rule. It only determines it can find the properties needed to perform the current execution.

In some situations, the class of a page may vary from execution to execution. If all the properties referenced are available in a common parent class, you can list the parent on this tab, even though the page belongs to a subclass.

For example, an activity may operate on a page named Customer that, in some situations, has the class Data-Account-Savings and in other cases has the class Data-Account-Checking. If all the properties that the activity needs — for example CurrentBalance and AccountOpenDate — are defined in the Data-Account- class, you can list Data-Account- in the Class column.

However, if the activity needs to access both checking and savings data at once, the activity needs to work with two pages — one of each class — with distinct page names.

Identifying pages in your application

Managing the number and names of Clipboard Pages referenced in your application is important. Using too many pages or omitting to remove the pages when no longer needed can add to the run time memory demand of your applications. Misspelling page names or having two copies of one page (with different names) can also hurt performance or make testing difficult.

Use the Data Designer **Usage** tab to see the pages that use a data type.

Special cases

Certain Rule forms offer more capabilities for greater power and flexibility:

- This functionality is not supported. Use a Data Page instead. For three Rule Types — Declare Expression, constraints, and When Condition Rules — you can define page names that start with the keyword `locate` and that use an activity to search or assemble the page needed. These pages are known as locatable pages.
- For six Rule Types — including activities and Correspondence — pages with names that start with the keyword `prompt` are known as indirect pages. This facility allows a calling activity to share multiple pages with a called activity, for example. When the called activity references a property on a page named promptALPHA, the system searches the Clipboard for a page named ALPHA.

> **NOTE:** This feature has been deprecated.

- Using the Clipboard tool

Page names and reserved pages

A page is a data structure that holds name-value pairs.

Most pages are named; the names of pages can arise from any of the following sources:

- Literal values entered into a Rule such as a parameter to the Page-New method.
- System-created pages with reserved names and purposes, such as the requestor page.
- Names used by convention in a collection of Rules designed to operate together.
- Property names (for embedded pages).

Valid page names

A page name can contain only letters, digits, and the underscore character. Start the page name with a letter.

Several keywords identify elements in specific operations. Using keywords as a page name is not recommended: `param`, `local`, `locate` (unsupported), `primary`, `steppage`, `mysteppage`, `top`, `parent`, or `current`.

Do not use any of the following names or patterns as a page name, where an asterisk indicates one or more characters: `pxNode`, `pxRequestor`, `pxThread`, `LISTVIEW_*`, `locate*` `prompt*`, `$**`, `px*`, `pz*`, or `py*`.

Special page name prefixes

You cannot use certain prefixes and keywords as page names, as they have a special meaning:

- `local` – Not a page, a keyword indicating that the name that follows the word "local" and the period is a local variable, not a property.
- `locate` – A prefix to a page name (in lowercase), indicates that this page is retrieved or located at run time through an activity of type `Locate`. This prefix is unsupported. Locatable pages have been replaced by Data Pages.
- `mode` – Identifies an indirect page reference.
- `param` – Identifies the parameter page for an activity. This page structure is not part of the Clipboard.
- `parent` – In the context of an embedded page, identifies the immediately containing page to which the embedded page belongs. This keyword might appear more than once in a property reference. You keyword is used in Declare Index Rules, decision tables, decision trees, and map values. You can also use the `PARENT` keyword in the **Step page** field in activities. You cannot use this keyword in Data Transforms.
- `prompt` – As a prefix portion of a page name, not a keyword, identifies an indirect page reference in a Call or Branch.
- `primary` – In the context of an activity, refers to the Primary Page of that activity.
- `top` – In the context of an embedded page, identifies the top-level page to which the current embedded page belongs.

The prefixes `D_` and `Declare_` in a page name indicates a Data Page, a page that is created and updated only by Data Page Rules. The properties on this page are "read-only" to normal requestor processing. To avoid confusion with such pages, do not create pages named Declare_zzzzz or D_zzzzz in other processing.

Except as noted, Case is not significant in keywords: Param.Country, PARAM.Country and param.Country all reference the same value.

Page names used by convention

Many standard activities and other standard Rules use these page names by convention. To utilize these standard Rules, follow these naming conventions in the Rules you create.

Page Name	Description
`NewAssignPage`	When a single Assignment object is open on the Clipboard, the page is named `NewAssignPage`. If your application involves multiple work items (or multiple Assignments) on the Clipboard, you can use other names.
`pyCoverPage`	By convention in standard flow processing activities, when a single cover work item is opened onto the Clipboard, the page is named `pyCoverPage`.
`pyOutput`	Created by Process API activities.
`pyWorkPage`	By convention in standard flow processing activities, when a single (non-cover) work item is opened onto the Clipboard, the page is named `pyWorkPage`.
`RuleToRun`	Created during unit testing of a Rule with the Run toolbar button. The class of this page

Page Name	Description
	matches the Rule Type. See Unit testing individual Rules.

System-created pages

These top-level pages are present in the Clipboard of every authenticated requestor:

Page Name	Description
Process page (`pxProcess`)	A named page of class *Code-Pega-Process* containing system-wide information, such as information from the *Data-Admin-System* instance.
Requestor page (`pxRequestor`)	A named page of class *Code-Pega-Requestor*. Created at log in and contains information about your Access Roles, Ruleset list, and TCP and HTTP protocol parameters.
Thread page (`pxThread`)	A named page of class *Code-Pega-Thread*, identifies a named context of Clipboard Pages. The first Thread for a requestor is named `STANDARD`.
`AccessGroup`	Contains information from the requestor's current Access Group *(Data-Admin-Operator-AccessGroup* class). This page does not exist for guest (unauthenticated) users.
`Org`	Contains information from the requestor's organization (*Data-Admin-Organization* class). This page does not exist for guest (unauthenticated) users. This page does not exist for guest (unauthenticated) users.

Page Name	Description
`OrgDivision`	Contains information from the requestor's organization (*Data-Admin-Organization* class). This page does not exist for guest (unauthenticated) users.
`OperatorID`	Contains information from the requestor's Operator ID (*Data-Admin-Operator-ID* class). This page does not exist for guest (unauthenticated) users.

Other

Page Name	Description
`pyQueryResultPage`	Produced by execution of a list Rule (*Rule-Obj-List* Rule Type).

- Indirect pages
- Properties
- Clipboard tool
- Using the Clipboard tool

Rule skimming for higher Ruleset versions

Skimming is an operation of copying Rules from your Rulesets into a Ruleset of a higher version. Skimming improves the performance of your application because the system filters out Rules that are unavailable for Rule resolution. Because the new Ruleset contains only the highest Rule versions, skimming simplifies Rule resolution and minimizes the Rule data that you ship to a different version of your application.

Skimming collects the highest version of every Rule in the Ruleset and copies the Rules to a new major or minor version of that Ruleset on the same system. Rule availability and the type of skimming determine which Rules the system carries to a higher Ruleset version.

You can select between a major skim and a minor skim. The skim type corresponds with the digits in a Ruleset version number. The two first digits define the major version, and the two middle digits define the minor version. For example, in the Ruleset version `01-02-10`, `01` indicates the major version, and `02` corresponds with the minor version. The major Ruleset version typically specifies a release of your application or other significant changes, while the minor version relates to an enhancement or a less extensive change in your application. When you select a skim type, consider changes that you want to merge in the higher Ruleset version. The final two digits correspond with patch releases and do not have a separate skim type associated with them. Skimming copies Rules so that all your Rules in lower Ruleset versions remain unchanged. During skimming, the system omits Rules in the major versions that are lower than the major version that you specify. For example, if you opt to skim `02-05-02` into `03-01-01`, the system ignores any Rules in the version `01-XX-XX`.

The following table displays which Rules you can include in a higher Ruleset version by using either the major or minor skimming type:

Skim type/ Rule availability	Available	Not Available	Final	Withdrawn	Blocked
Major	Yes	No	Yes	No	Yes
Minor	Yes	No	Yes	Yes	Yes

NOTE: The default system behavior is to reindex the *Declare Index* Rule while skimming, which might cause significant performance issues. To avoid this, consider placing the Rule in a different Ruleset.

The update history of the new skimmed Rule contains only one instance that reflects the date and time of the Rule creation based on the skim operation. The history of the source Rule is available and remains unchanged.

For relevant learning materials, see an Application versioning module on Pega Academy.

Related concepts

- Organizing Rules into Rulesets

Related tasks

- Creating Rulesets
- Defining the security of a Ruleset
- Deleting a Ruleset

Creating higher Ruleset versions by skimming Rules

Improve the performance of your application and simplify the Rule resolution by skimming Rules to create higher version Rulesets. Skimming filters out Rules that are not available for Rule resolution and makes Rule management more convenient.

You can select between two types of skimming, a minor skim and a major skim. The minor skim moves Rules to a higher minor version of your application, for example, after you implement new enhancements. You can check the minor Ruleset version by analyzing the middle two digits of the Ruleset version. An example of a minor skim is skimming Rules in `01-05-01` through `01-09-25` into the `01-10-01` Ruleset versions. The major skim moves Rules into a higher major version of your application, such as a new release. The two first digits in the Ruleset version correspond with the major version. A major skim is skimming Rules `01-05-01` through `01-09-25` into the `02-01-01` Ruleset versions.

Before you begin:

- Check in all the Rules that you want to skim. For more information, see Checking in a Rule.

- To ensure that users cannot edit lower versions of the Ruleset after skimming, secure the lower Ruleset versions. For more information, see Defining the security of a Ruleset.

During skimming, the system copies Rules to a new Ruleset with a higher version number. The Rules in the lower versions remain unchanged in your system.

1. In the header of Dev Studio, click Configure > System > Refactor > Rulesets.
2. On the Rulesets tab, in the **Refactor Rulesets Utilities** section, click Skim a Ruleset.
3. In the **Skim to create a higher version** window, define the skimming settings:

Choices	Actions
Perform a major skim	a. Select the Major Version Ruleset Skim radio button. b. In the Ruleset list, select a Ruleset that you want to skim. c. In the From Major Version list, select an existing major version that you want to skim to a higher version. d. In the To New Version field, enter a Ruleset version that you want to create with skimming. Ensure that the new version relates to the skimming type. **For example:** If you want to perform a major skim on the `01-09-25` version, enter `02-01-01`.

Choices	Actions
	e. Click **Skim**.
Perform a minor skim	a. Select the **Minor Version Ruleset Skim** radio button. b. In the **Ruleset** list, select a Ruleset that you want to skim. c. In the **Starting Version** and **Ending Version** lists, select a version range of the Ruleset that you want to skim. **For example:** Select **01-05-01** as a start version and **01-09-25** as an end version. d. In the **To New Version** field, enter a Ruleset version that you want to create with skimming. Ensure that the new version relates to the skimming type. **For example:** If you want to perform a minor skim on the `01-05-01` to `01-09-25` version range, enter `01-10-01`. e. Click **Skim**.

Result:

The window displays the progress and the results of the skim after the process is complete.

4. If any errors occur, click the **Total Errors** link.
   You can only access the list of errors from the **Skim to create a higher version** window.

What to do next:

- Research and resolve errors that occurred during the skim operation.
- Update Access Groups or application Rules to make the new major version available to appropriate users. For more information, see Learning about Access Groups.
- To ensure that users can edit only the highest Ruleset version, secure lower Ruleset versions. For more information, see Defining the security of a Ruleset.
- To decrease the number of Rules in your system and speed up Rule resolution, delete Rules in the lower major versions. For more information, see Deleting a Ruleset.

  NOTE: Ensure that your organization accepts deleting Rules that are no longer in use.

- If you skim circumstanced Rules, and you want a skimmed Rule to be recognized as a base Rule during Rule resolution, select the **Base Rule** check box in the Rule availability dialog box. For more information, see Base Rules.

Skimming ignores the **Base Rule** check box and moves Rules from lower to higher Ruleset versions regardless of the **Base Rule** setting and circumstancing. For more information about circumstancing, see Creating a Rule specialized by circumstance.

- Organizing Rules into Rulesets
- Creating Rulesets

Adding a custom field

Associating custom fields with Rules provides a flexible way to supplement your application with metadata, such as a change order number or log file attachment.

To add a custom field, do the following.

1. Click the History tab of a Rule form.
2. Click **Add Field** to open the **Add custom fields** dialog.
3. Choose a name from the list of properties to populate the Name field. Alternatively, enter a string (letters and digits only) directly in this field to create a new property with the specified name.
4. If you entered a string in the Name field, select a Type for the property. For existing properties, this field is populated and read-only.
5. From the **Ruleset version** drop-down box, select the appropriate Ruleset version.
    - For properties of Type File:
        - Enter a brief description of the file in the Description field.
        - Select the **New** or **Existing** radio button to indicate how you will provide the file.
        - Click **Browse** to upload a file from your local system. The file you specify becomes an instance of *Rule-File-Binary*.
        - Click the **Pick a file** button to select a file previously linked to this property.
    - For properties of Type Text:

    - Enter text to pair with the property in the Value field.
    - Alternatively, click the **Open** icon to select a value previously applied to this property.
6. Click **Submit** to close the dialog.
7. Click **Save** in the Rule form header to persist your changes. The Custom Fields list is automatically updated by the Rule form.

- Viewing Rule history

Preparing for modifying Rules

Adjust your application to your unique business scenario by modifying existing Rules. As a result, you save development resources, and deliver software that precisely meets your business requirements. You can also delegate a Rule so that business stakeholders can perform needed changes.

Before you modify a Rule, you can check out the Rule to ensure that no other developer modifies the same Rule at the same time, if your Ruleset uses the Rule checkout. If your business requirements change, you can revert to the previous state of the Rule, or delete the instance of the Rule.

Consider the following actions when you modify your Rules:

Checking out a Rule
: Checking out Rules prevent other developers from editing the same Rule simultaneously. Consequently, you avoid errors and duplicating work.

Delegating a Rule
: Rule delegation allows business members of your organization to modify application logic. As a result, application development is faster, and Rules in your application can illustrate relevant business processes more precisely.

Restoring a Rule
: Bringing back an earlier state of a Rule saves time when you need to reimplement a previous version of the Rule as a result of business requirements or technical issues.

Documenting a Rule

Adding usage and historical information to a Rule makes your application more transparent, and other developers can make more informed decisions while working with a specific Rule.

Deleting a Rule

Deleting a Rule that is no longer required saves resources and helps you avoid erroneous use of irrelevant rules in your application.

> **NOTE:** When you delete a Rule, you delete the current instance of the Rule. The other instances, for example in built-on applications, remain intact. To prevent other developers from using a Rule, you might consider changing rule availability. For more information, see Setting Rule status and availability.

- **Checking out a Rule**
- **Restoring the earlier state of a Rule**
- **Creating guidance for developers**
- **Delegating a Rule or Data Type**
- **Deleting a Rule**

Checking out a Rule

To avoid accidental Rule changes or conflicts that might result from multiple developers working on the same Rule, perform a check out, so that you can lock the Rule and safely make changes to it. By checking a Rule out before editing, you avoid unwanted Rule changes, and as a result save time and maintain a better quality application.

You can only check out Rules that belong to an unlocked Ruleset. If you want to edit a Rule that belongs to a locked Ruleset, perform a private edit. For more information, see Performing a private edit.

NOTE: When you make updates to your application in App Studio, Pega Platform automatically manages the check-in and checkout process.

1. In the navigation pane of Dev Studio, click **Records**, expand the category that contains the Rule that you want to edit, and then click the Rule.
2. In the header of the Rule form, click **Check out**.
   Your application places a copy of the original or base Rule in your personal Ruleset. No other user can check out this Rule until you check in your changes.
3. Modify the Rule as appropriate.
4. Click **Save**.
   Your application saves your changes into the checked-out version of the Rule. These changes are visible only to you. When you run Rules in your application, they resolve to only your checked-out version.

What to do next:

Make your changes available to other developers of your application by checking in the Rules that you edit. For more information, see Checking in a Rule.

NOTE: You do not have to check in your changes immediately. You can log out and return to a checked-out Rule later or click **Discard** to remove the Rule from your personal Ruleset.

- **Standard Rule checkout**
- **Performing a private edit**
- **Checking out a Rule to a Branch**
- **Viewing your checkouts and accessing bulk actions**

- **Rule checkout notes**
- **Rule checkout tips**
- **Personal Ruleset**
- **Checking in a Rule**
- Rule check-in process

Standard Rule checkout

The **Check out** button appears for standard checkouts when all the following criteria are met:

- The Rule belongs to a Ruleset that has the checkouts enabled.
- For this operator, the **Allow Rule check out** checkbox on the **Security** tab of the Operator ID form is selected.
- The Rule is not locked by another user or in a locked Ruleset. A locked Rule displays a padlock next to the **Check out** button. Click this icon to see the reason it is locked.

Other operators can use **Check out to Branch** or **Private edit** until you release the lock.

- Checking out a Rule

Performing a private edit

To ensure that your application meets the unique needs of your customers, you can modify a Rule that is unavailable for regular checkout by performing a private edit. Check out Rules as private edits to edit Rules that belong to locked Rulesets.

Private edits are useful for quick debugging without interrupting other team members, because during a private edit other system architects can edit a Rule at the same time.

Before you begin:

- Ensure that you have the *pxAllowPrivateCheckout* privilege, which is typically granted through a role on your Access Group. The standard Access Role *PegaRULES:SysAdm4* provides this privilege.
- Ensure that the Rule is not available for regular checkout, usually because the Rule is already checked out to another user or because its Ruleset version is locked.
- Ensure that the Rule belongs to a Ruleset that has the checkouts enabled.
- Ensure that your Operator ID is allowed to check out Rules. For more information, see Defining security information for an operator.

When you perform a private edit, you edit a Rule in an isolated sandbox, in which you can make and then test your changes. As a result, you maintain a good quality application and avoid creating errors, because during a private edit the original Ruleset that contains your Rule remains unchanged.

1. In the navigation pane of Dev Studio, click **Records**, expand the category that contains the Rule that you want to edit, and then click the Rule.
2. In the Rule form, click **Private edit**.
3. In the dialog box that appears, click **Continue with private edit**.
   The system saves a copy of the Rule in your personal Ruleset. Rules in your personal Ruleset do not affect other users' run-time experiences, or the actions that they can perform on this Rule.
4. Edit the Rule.
5. Click **Save**.
   Your application saves your changes into the checked-out version of the Rule. These changes are visible only to you. When you run Rules in your application, they resolve to only your checked-out version.

What to do next:

Make your changes available to other developers of your application by checking in the Rules that you edit. For more information, see Checking in a Rule.

- Checking out a Rule to a Branch

Checking out a Rule to a Branch

You can check out a Rule to a Branch so that you can make changes to a Rule and then save it in the Branch Ruleset. For information about Rule resolution for Rules saved to a Branch Ruleset, see Rule resolution exceptions.

Because you cannot make changes to Rules in a locked Ruleset, you can check out a Rule to a Branch in your application, make changes, and then check the Rule in to your Branch. You can also perform a private checkout.

1. In the Rule form toolbar click **Check out to Branch**, if available. Otherwise, click the arrow on the **Private edit** button and select **Check out to Branch**.
2. From the **Branch** list, select a Branch and click **Check out to Branch**.
   The system saves a copy of the Rule to the selected Branch Ruleset and then checks it out.
3. Modify the Rule as appropriate.
4. Click **Save**.
5. Check in the Rule. For more information, see Checking in a Rule.

   NOTE: You do not have to check in your changes immediately. You can log out and return to a checked-out Rule later or click **Discard** to remove the Rule from your personal Ruleset.

   If you discard the checked-out version, the original copy of the Rule remains in the Branch Ruleset.

- Performing a private edit

Viewing your checkouts and accessing bulk actions

Click the **Checkouts** icon in the Dev Studio header to open a list of Rules checked out by your operator. Rules are organized by type, name, and Applies To class.

In the Private overlay, you can perform the following actions:

- Click any Rule Type to see only checked-out Rules of that specific type.
- Click any Rule name to open the Rule.
- Expand any row to see all versions of the checked-out Rule.
- Click the Type or Name columns to sort the list.
- Use the filter icon in the Type and Name columns to apply more specific filter criteria.

- Open, delete, or check out multiple Rules at once using bulk actions by clicking the **Bulk actions** link. A list of Rules checked out by your operator appears in a new window.

- Standard Rule checkout
- Checking in Rules in bulk

Discarding checked out Rules in bulk

Maintain the best quality application by discarding checked out Rules that are irrelevant to your development projects. To save time, you can discard multiple Rules at once by performing a bulk action.

> **Before you begin:**
>
> Check out Rules that you want to edit. For more information, see Checking out a Rule.

1. In the header of Dev Studio, click the Checked out records icon.
2. In the **Checked out records** window, click Bulk actions.
3. In the window that appears, select the checkbox next to one or more Rules that you want to discard.
4. In the list at the bottom of the window next to the Start button, select Discard.
5. Click Start.

   Result:

   The **Status** column shows the results of your bulk discard.

- Viewing your checkouts and accessing bulk actions

Opening checkouts in bulk

To open checkouts in bulk, complete the following steps:

1. Click the Checked out records ✓ icon in the Dev Studio header and then click Bulk actions.
2. Select the checkbox next to one or more Rules.
3. Select Open from the list at the bottom of the window next to the Start button.
4. Click Start.
5. Wait for the lock icon to appear in the Status column next to each selected Rule name.
6. Review each opened checkout. If you do not have checkouts enabled, open the Recent Explorer to access each checkout.

- Viewing your checkouts and accessing bulk actions

Rule checkout notes

Note the following information when you are checking out Rules:

- A checked-in Rule version can have validation errors, which can occur when Ruleset prerequisites change or a referencing Rule is relocated. When you perform a standard checkout, the validation errors appear in the Rule form header. You must resolve all errors and save before you can check in any new changes.
- You cannot check out a base Rule when you or another developer has a circumstance version checked out.
- You cannot check out the following Rule Types:
    - Application
    - Class
    - Ruleset version
    - Library
    - Agent
    - Application settings

- Standard Rule checkout

Rule checkout tips

You can use several features to determine that you have checked out all the Rules in your application You can also use other features to undo changes and to override the default check-in procedure.

Refer to the following list for tips about Rule checkout.

- Click the **Checked out records** ✓ icon in the Dev Studio header to view all the Rules that you have checked out.
- Use the Application Development landing page to view all checked-out Rules. Rules checked out using the **Private edit** option do not appear in this list.
- Use the restore feature to undo changes made by a checked-in version.
- Use the *Rule-.SetApprovalDependency* activity to override the default check-in procedure.
- Use the lock icon to see who has a specific Rule instance checked out.

- Select the **View version** > **Checked out from [version number]** option in the **Actions** menu to view the base version of the Rule that you checked out.

- Standard Rule checkout
- Viewing your checkouts and accessing bulk actions
- Restoring the earlier state of a Rule

Personal Ruleset

Users who have the **Allow Rule check out** option enabled for their operator IDs can place copies of Rule instances into a personal Ruleset. The name of this Ruleset is the Operator ID, such as john_sawyer@abc.com. The personal Ruleset is sometimes called the private Ruleset.

Personal Rulesets are used in with the checkout and check in features. The system creates a personal Ruleset automatically when needed. No instance of the *Rule-RuleSet-Name* class exists for this Ruleset. Rules in a personal Ruleset are never run by other users.

On the requestor page, the standard property *pxRequestor.pxSecuritySnapshot.pxPersonalRuleSetName* identifies the personal Ruleset. If this property is blank, the requestor cannot check out Rules.

A memory cache contains Rules in your personal Ruleset. The maximum size of this cache (for all developers combined) is determined by elements in the `fua` node of the `prconfig.xml` file or by dynamic system settings.

- Learning about Access Groups

Checking in a Rule

To make your changes available to your development team after you check out and edit a Rule, check in the Rule. When you check in edited or updated Rules, you contribute to the efficient development of your application and avoid errors that might occur when other developers work on the same Rule.

Before you begin:

Check out, and then update a Rule. For more information, see Checking out a Rule.

> **CAUTION:** Use caution when you check in Rules after you perform a private edit, because you may overwrite changes from other developers. For more information about private check out, see Performing a private edit.

1. In the navigation pane of Dev Studio, click Records, expand the category that contains the Rule that you want to check in, and then click the Rule.
2. In the header of the form, click Check in.
3. If you perform a check in after a private edit, in the **Check In** window, in the **Destination** section, specify where you want to save the Rule:
   - To save a Rule to an unlocked Ruleset, select Ruleset, and then specify a Ruleset and a Ruleset version.
   - To save a Rule to a Branch, select Branch, and then specify the Branch.

   > **NOTE:** Checking in a private edit to a Branch could result in loss of work if another user checks in changes to the same Rule. You receive a warning message if your check-in might result in lost work.

4. In the Check-in comments field, enter a comment that describes your changes to the Rule.
5. **Optional:** To override the default work item that your application associates with this development change, in the **Current work item** section, in the Work item to associate field, select a work item.

For more information about your default work item, see Setting your current work item.

6. Click **Check in**.

Result:

If your edits do not require approval, your changes immediately affect Rule resolution and run-time behavior for all users.

- **Configuring the Rule check-in approval process**
- **Rule check-in process**
- **Checking in Rules in bulk**

- Checking out a Rule to a Branch
- Checking out a Rule

Configuring the Rule check-in approval process

To turn on controlled check-in for a Ruleset, complete the following tasks:

1. Enable checkout features for the Ruleset by selecting the **Use check-out?** box on the **Security** tab of the Ruleset form.
2. Enable the checkout feature for each development team member by selecting the **Allow Rule checkout** box on the **Security** tab of each member's Operator ID form.
3. On the **Security** tab of the Ruleset form, specify both key parts of the flow that is to control the Rule check-in approval process. Specify the flow's work class in the Work class used for approval process field and the name of the flow in the Work flow to use for approvals field. Your system includes a standard work type *Work-RuleCheckIn*, standard properties and activities, and a standard flow named *ApproveRuleChanges* to use for this feature. You can use the standard Rules as is, or copy and adapt them.

4. Review (and override if desired) the standard decision tree named *Work-RuleCheckIn.FindReviewers* to identify the Work Queues used in this flow. If overriding the standard Rule, create the new decision tree in the Ruleset and version being developed by the team.
5. Update the Access Group associated with those reviewers you want participating in the Rule check-in approval process to include the standard *Work-RuleCheckIn* work pool in the Access Group's Work Pools list.
6. Create a Work Queue named `ReviewChanges@ org.com` or others identified in the decision tree. Update the Operator ID data instances for each reviewer to allow them to access the Work Queues.
7. Associate the standard Access Role `PegaRULES:SysAdm4` (or another role with the privilege *Rule-.UpdatePrivateRuleSets*) with each Operator ID data instance who will approve check-ins.
8. Update the standard Ruleset *CheckInCandidates* (version 01-01-01) to identify an appropriate prerequisite Ruleset.
9. Add this Ruleset (version 01-01-01) to the Access Group of those developers who are to approve check-ins.
10. For each version you want to enable with this process, select `Yes` in the **Approval Required** field on the **Versions** tab of the Ruleset form.

Rule check-in process

The optional Rule checkin facility allows you to use a Pega Platform™ flow Rule to manage change to your application. Use this facility to coordinate the efforts of large development teams or to control changes to sensitive applications.

See Configuring the Rule check-in approval process for information about setting up controlled check-in for a Ruleset.

When this facility is in force for a Ruleset version, the flow starts when a developer begins Rule check in.

The flow creates a work item that is routed to a Work Queue, waiting for an approver to review.

The approver might approve the check-in (which completes the check in), reject it (which deletes the changed Rule), or send it back to the developer for further work.

Assignments corresponding to Rules awaiting evaluation are held in the `ReviewChanges@org.com` Work Queue.

Multiple Rules can be associated with one work item. The flow Rule notifies affected parties by email about evaluation results.

When uploading approved Rules, select the following Import Options: Compile Libraries, Overwrite Existing Rules, Overwrite Existing Data.

You can enhance this process to meet local requirements in various ways. For example, certain empty (dummy) activities are called by this flow; your application can override these with desired processing.

Checking in Rules in bulk

Save time and improve your management of the Rules that you edit in your application by performing a bulk check-in. When you perform bulk actions, you can view all of the Rules that are currently checked out in your application, and then you can check in several Rules at once.

Bulk check in supports only Rules that you check out through the standard checkout procedure. For information about checking in Rules after a private checkout, see Performing a private edit.

1. In the header of Dev Studio, click the Checked out records icon.
2. In the **Checked out records** window, click Bulk actions.
3. In the window that appears, select the checkbox next to one or more Rules that you want to check in.
4. In the Check-in comments field, enter a brief description of your changes.
5. Select CheckIn from the list next to the Start button at the bottom of the window.

6. **Optional:** To apply one prefix to all comments, in the **Bulk Prefix** field, enter the text, and then click **Insert**.
7. **Optional:** To override the default work item that your application associates with this development change, in the **Current work item** section, in the **Work item to associate** field, select a work item.
   For more information about your default work item, see Setting your current work item.
8. Click Start.

   Result:

   The **Status** column shows the results of your bulk check in.

- Viewing your checkouts and accessing bulk actions

Restoring the earlier state of a Rule

During application development, you can undo recent changes to a Rule and replace the current Rule with a previous copy, even if another developer created the newer copies.

Before you begin:

Ensure that you have the *@baseclass.ToolbarFull* privilege.

When you restore a Rule, consider the following circumstances:

- You do not need to use the **Restore** operation to fall back to a Rule instance that is in a lower-numbered version. Use the **Delete** toolbar button to delete the current higher-numbered version or versions that mask or override a lower-numbered version.
- If you want to change the lower numbered version but the version is locked, use **Save As**, not **Restore**, to copy — promote — an older version to the current unlocked higher version.

1. Open the current version of the Rule.
2. Click the **History** tab.
3. Click the **View full history** button to view a list of snapshots in a new window.
4. Details display that contains the state you want. Click the Rule history button to open the older, historical state of this Rule instance. The padlock image indicates that this is not the current Rule, not that the Rule is checked out to you.
5. Click the **Restore** button for the state that you want to make current.

- Recovering a deleted Rule

Creating guidance for developers

Create internal documentation to provide guidance to the developers who implement or extend your application.

Use the following techniques to create internal documentation:

- Creating project documents for stakeholders in Dev Studio
- Creating a guide for application developers and administrators

Adding usage information to a Rule

Add usage information to a Rule to help developers decide whether they can reuse the Rule in their features or applications.

As a best practice, provide usage information when you create a Rule.

1. Open a Rule by searching for it or by using the Application Explorer.
2. Click the **History** tab.
3. In the **Documentation** section, enter text in the **Description** field that describes the purpose of your Rule.
4. In the **Usage** field, enter text that helps developers understand how to use this Rule.

   For example, you can describe the expected inputs and outputs.

5. Click **Save**.

- Adding historical information to a Rule
- Setting Rule status and availability
- Adding a custom field

Adding historical information to a Rule

Describe the recent changes to a Rule in the Rule history to help developers trace your implementation and compare versions more quickly.

1. Open a Rule by searching for it or by using the Application Explorer.
2. On the **History** tab, click **View full history**.
3. Click **History For All Versions**.
4. In the **Add Memo** field, enter text that describes the differences between this version of the Rule and the prior version.
5. Click **Add Memo**.

- Viewing Rule history
- Comparing Rule versions

Delegating a Rule or Data Type

Enable your business line to configure simple application logic without involvement from IT by delegating a Rule or Data Type.

For example, you can delegate a decision table to a business line manager. The line manager views the decision logic in a familiar environment, such as the Case Manager Portal, and can update it as a stand-alone item without knowing all of the related technical details.

NOTE:

- For Constellation-based applications, consider Configuration Sets and CRUD operations before resorting to rule delegation because they are easier to configure and maintain. For more information, see Creating a Configuration Set and Introducing CRUD operations.
- Standard Pega Platform applications support Rule delegation out of the box. To enable Rule delegation in applications that you build on Constellation, see Delegating a Rule or Data Type in Constellation.

You can delegate a paragraph, decision table, Data Type (configured for local storage), map value, Correspondence, or Service-Level Agreement (SLA), including circumstance types.

1. Open the Rule or Data Type that you want to delegate.
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 that is being delegated.
   - **Manage content** – Users can modify the content of decision tables, map values, paragraphs, and Correspondence Rules.
   - **Manage content and modify table** – For map values, users can add, modify, or delete rows and columns in the map value table in addition to modifying contents.
   - **Manage Data Records** – For Data Types, users can add, modify, or delete data entries for the Data Type.
   - **Manage SLA** – For SLAs, users can configure goals, deadlines, and escalation actions.

   NOTE: You cannot delegate SLAs on which service levels are calculated by the value of a property.

> **NOTE:** If you later modify a delegated SLA in Dev Studio to have service levels calculated by the value of a property, users cannot modify the **Days** or **hh:mm** fields or select the **Calculate using business days** checkbox for the **Goal**, **Deadline**, or **Passed Deadline** sections.

> **NOTE:** When users modify SLAs in the Case Manager, they can select only **Notify Manager** or **Notify Assignee** as escalation actions. If you change the escalation action in Dev Studio to anything other than these actions, users cannot modify escalation actions.

4. In the **Delegate to Access Group** drop-down list, press the Down arrow and select the Access Group to which you want to delegate the Rule or Data Type.
5. In the **Title** field, enter a title for the delegated Rule or Data Type.
6. In the **Detailed Description** field, enter detailed information about the delegated Rule or Data Type. This text is displayed to the user.
   Provide information about how the delegated Rule or Data Type will affect the application. For example, "The logic in this delegated decision table determines whether a travel request required additional manager approval. Changing the logic in this decision table may affect all subsequent travel requests submitted in this application."
7. Click **Delegate**.

> **Result:**
>
> Users can access delegated Rules and Data Types from the Case Manager Portal. Users can also access Data Types in App Studio.

- **Delegating a Rule or Data Type in Constellation**
- **Delegating a data type**

- **Prerequisites for delegating a Rule or data type**
- **Best practices for delegating a Rule or data type**
- **Modifying a delegated record or data type**
- **Configuring Ruleset update behavior**

- Modifying a delegated record or data type
- Creating a data object

Delegating a Rule or Data Type in Constellation

Enable your business line to configure simple application logic without involvement from IT by delegating a Rule or Data Type in your Constellation application.

For example, you can delegate a decision table to a business line manager. The line manager views the decision logic in a familiar environment, such as the Case Manager Portal, and can update it as a stand-alone item without knowing all the related technical details.

You enable Rule delegation in your Constellation application by adding a Theme Cosmos page to the Constellation application navigation pane.

NOTE: When building applications in Constellation, prioritize the following alternatives to rule delegation:

- Configuration Sets - provide a centralized, consistent way to manage application settings. For more information, see Creating a Configuration Set.
- CRUD operations - ensure that only authorized users can perform create, read, update, and delete operations, giving you much tighter control over data access. For more information, see Introducing CRUD operations.

Before you begin:

Prepare the applications that you want to integrate:

- Ensure that both the traditional application and the Constellation application have the same name but different version numbers. For more information, see Duplicating the application rule.
- Create an activity that opens a harness in a new browser tab to use for the integration. For more information, see Creating an openHarness activity.

- Modifying a delegated record or data type
- Creating a data object

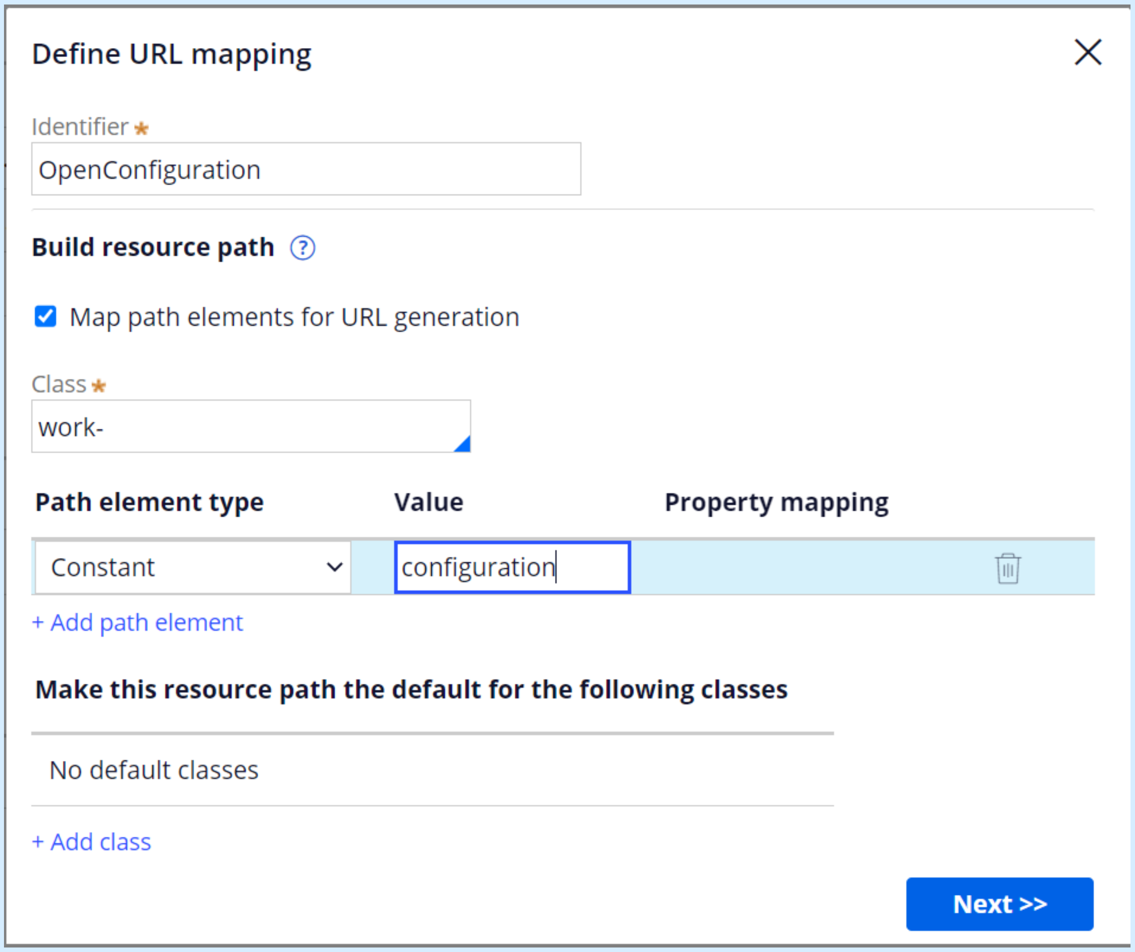
Configuring URL mapping for the Configuration landing page

Create an alias for the Configuration landing page, which contains delegated Rules and data types. You configure this mapping in the Theme Cosmos application so that you can access the landing page in your Constellation application.

1. Log in to your Theme Cosmos application.
2. In the navigation pane of Dev Studio,click Records.
3. Expand the Technical category, and then click URL Mappings.
4. In the list of URL mapping Rule instances, open the top-most Rule for your application: *pyDefault*.
5. In the **Landing pages** section, click Add URL alias.
6. In the **Define URL mapping** dialog box, configure the URL path:
    a. In the Identifier field, enter `OpenConfiguration`.
    b. In the Class field, enter `Work-`.
    c. In the Path element type list, select Constant.
    d. In the Value field, enter `configuration`.
    e. Click Next.

Result:

The following figure shows the URL mapping settings for the *OpenConfiguration* URL alias:


Define URL mapping

Identifier *
OpenConfiguration

Build resource path

☑ Map path elements for URL generation

Class *
work-

Path element type	Value	Property mapping
Constant	configuration	

+ Add path element

Make this resource path the default for the following classes

No default classes

+ Add class

The settings for the new OpenConfiguration URL mapping

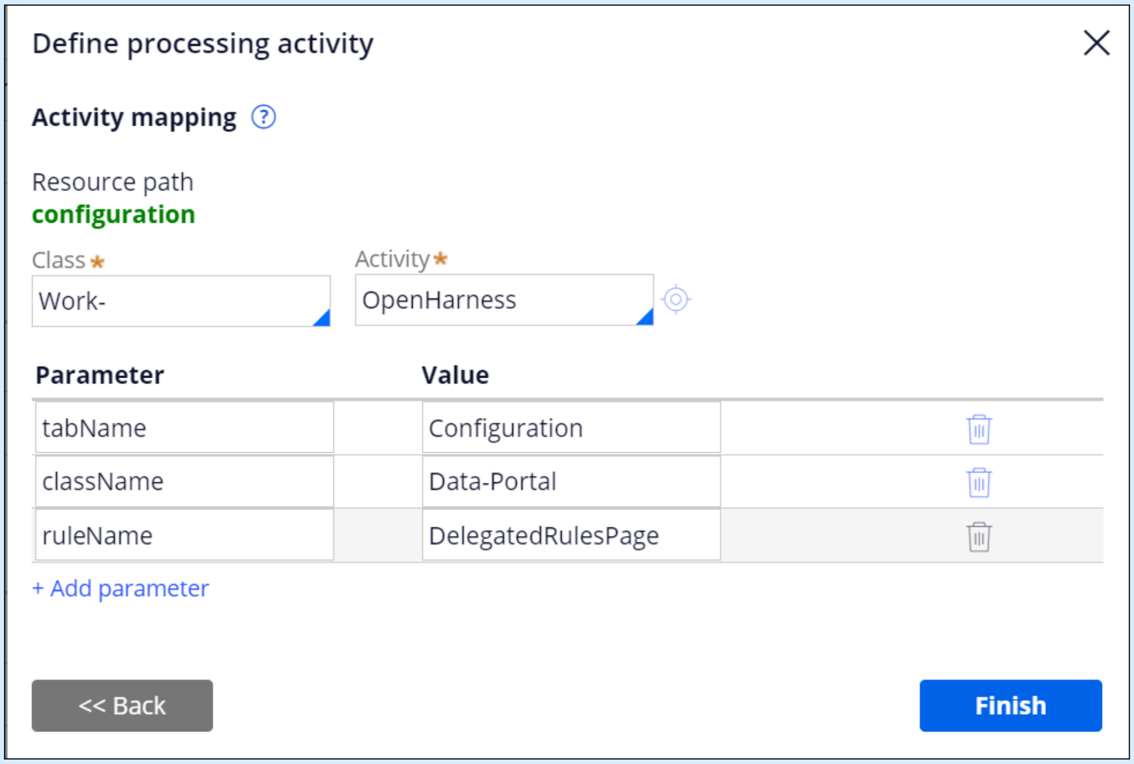
7. In the **Define processing activity** dialog box, configure the activity that starts when the user opens the URL:
   a. In the Class field, enter `Work-`.
   b. In the Activity field, enter `openHarness`.

> **NOTE:** Create an *openHarness* activity before configuring URL mapping. For more information, see Creating an openHarness activity

c. Click **Add parameter**, and then in the **Parameter** field, enter `tabName`.
d. In the **Value** field, enter `Configuration`.
e. Click **Add parameter**, and then in the **Parameter** field, enter `className`.
f. In the **Value** field, enter `Data-Portal`.
g. Click **Add parameter**, and then in the **Parameter** field, enter `ruleName`.
h. In the **Value** field, enter `DelegatedRulesPage`.

Result:

The following figure shows the activity mapping for the *OpenConfiguration* URL alias:



Define processing activity

Activity mapping

Resource path
configuration

Class: Work-
Activity: OpenHarness

Parameter	Value
tabName	Configuration
className	Data-Portal
ruleName	DelegatedRulesPage

The settings for the OpenConfiguration activity mapping

8. Click **Finish**.

Delegating a Rule or data type

Enable your business line to configure simple application logic without involvement from IT by delegating a Rule or data type.

For example, you can delegate a decision table to a business line manager. The line manager views the decision logic in a familiar environment, such as the Case Manager Portal, and can update it as a stand-alone item without knowing all of the related technical details.

You can delegate a paragraph, decision table, data type (configured for local storage), map value, Correspondence, or Service-Level Agreement (SLA), including circumstance types.

1. Open the Rule or data type that you want to delegate.
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 that is being delegated.
   - **Manage content** – Users can modify the content of decision tables, map values, paragraphs, and Correspondence Rules.
   - **Manage content and modify table** – For map values, users can add, modify, or delete rows and columns in the map value table in addition to modifying contents.
   - **Manage Data Records** – For data types, users can add, modify, or delete data entries for the data type.
   - **Manage SLA** – For SLAs, users can configure goals, deadlines, and escalation actions.

   NOTE: You cannot delegate SLAs on which service levels are calculated by the value of a property.

> **NOTE:** If you later modify a delegated SLA in Dev Studio to have service levels calculated by the value of a property, users cannot modify the **Days** or **hh:mm** fields or select the **Calculate using business days** checkbox for the **Goal**, **Deadline**, or **Passed Deadline** sections.

> **NOTE:** When users modify SLAs in the Case Manager, they can select only **Notify Manager** or **Notify Assignee** as escalation actions. If you change the escalation action in Dev Studio to anything other than these actions, users cannot modify escalation actions.

4. In the **Delegate to Access Group** drop-down list, press the Down arrow and select the Access Group to which you want to delegate the Rule or data type.
5. In the **Title** field, enter a title for the delegated Rule or data type.
6. In the **Detailed Description** field, enter detailed information about the delegated Rule or data type. This text is displayed to the user.
   Provide information about how the delegated Rule or data type will affect the application. For example, "The logic in this delegated decision table determines whether a travel request required additional manager approval. Changing the logic in this decision table may affect all subsequent travel requests submitted in this application."
7. Click **Delegate**.

Result:

Users can access delegated Rules and data types from the Case Manager Portal. Users can also access data types in App Studio.

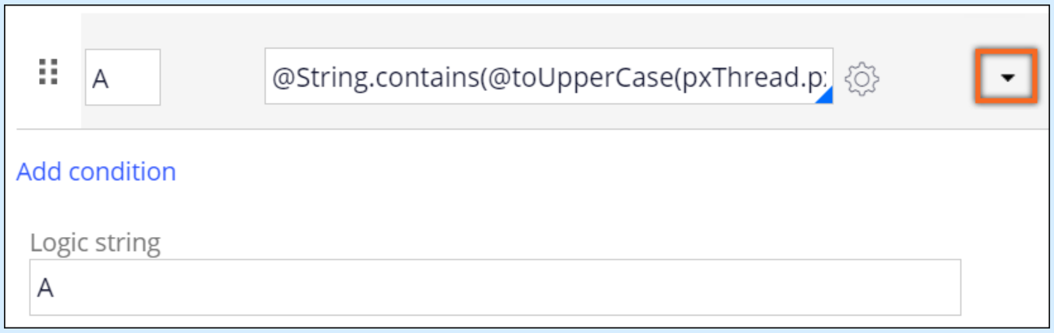
- Modifying a delegated record or data type
- Creating a data object

Overriding the pyIsAutoGenThread

Ensure that the mapped URL address of the target traditional landing page opens correctly in a Portal by adjusting the *pyIsAutoGenThread* When Rule.

1. In the traditional UI architecture application, in the header of Dev Studio, search for and open the *pyIsAutoGenThread* When Rule.
2. On the **Advanced** tab of the Rule form, define the new logic for the When Rule:
    a. To the right of the row **A** of the condition, click the **Configure advanced conditions here** arrow.
    b. In the list of logic structures, select **[expression evaluates to true]**.
    c. In row A, enter `@String.contains(@toUpperCase(pxThread.pxThreadName),@toUpperCase("`

For example:



A

@String.contains(@toUpperCase(pxThread.p:

Add condition

Logic string

A

When Rule configuration and the advanced conditions arrow

3. On the Rule form, click **Save as**.
4. On the **New** tab, ensure that the context points to the application Ruleset and the *@base* Class, and then click **Create and open**.

Adding the Configuration landing page to the navigation pane

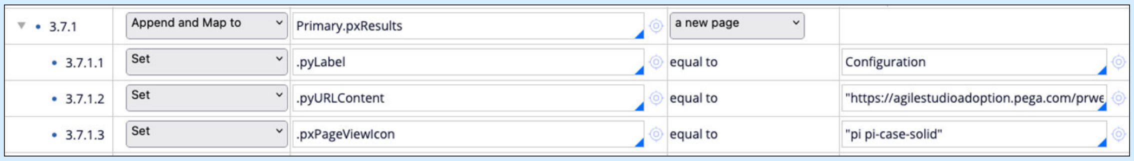
Enable users of your Constellation application to access delegated Rules and data types by adding the Configuration landing page to the navigation pane.

1. Open the Constellation application.
2. In the header of Dev Studio, search for and open the *pyPopulateAdditionalNavPages* Data Transform.
3. **Optional:** To limit the visibility of the Configuration landing page to a specific group of users, add a When Condition step to the Data Transform:
   a. Create a *HasDelegatedRules* When Condition that applies to the *Code-Pega-List* class.
      For more information, see Creating a When Rule.
   b. On the **Pages & Classes** tab of the When Condition, in the **Page name** field, enter `DelegatedRules`, and in the **Class** field, enter `System-User-MyRules`.
   c. On the **Conditions** tab of the When Condition, set the following condition: `@(Pega-RULES:Utilities).SizeOfPropertyList(D_pzFilteredDelegations.pyInsta` `> 0`.
   d. Back on the **Definition** tab of the *pyPopulateAdditionalNavPages* Data Transform, click **Add a row**.
   e. In the **Action** list, select **When**.
   f. In the **Target** field, enter `HasDelegatedRules`.
4. On the **Definition** tab, click **Add a row**, and then define the parent row for the logic that appends the Theme Cosmos page to the navigation menu:
   a. In the **Action** list, select **Append and Map to**.
   b. In the **Target** field, enter `Primary.pxResults`.
   c. In the **Relation** list, select **a new page**.
5. Under the **Append and Map to** action, add and configure the first child action that defines the label for the Theme Cosmos page:
   a. In the **Action** list, select **Set**.
   b. In the **Target** field, enter `.pyLabel`.
   c. In the **Relation** list, select **equal to**.
   d. In the **Source** field, enter `"Configuration"`.
6. Copy the URL of the Theme Cosmos application:

   a. In the header of Dev Studio, click the name of the application, and then click Definition.
   b. In the **Application URL alias** section, select and then copy the application URL to the Clipboard.
7. On the Rule form of the *pyPopulateAdditionalNavPages* Data Transform, under the Append and Map to action, click Add a row, and then configure the second child action that defines the URL address for the Theme Cosmos page:
   a. In the Action list, select Set.
   b. In the Target field, enter `.pyURLContent`.
   c. In the Relation list, select equal to.
   d. In the Source field, enter the application URL that you copied in step 6 and the URL path that you mapped in steps 1 through 8.
8. Under the Append and Map to action, click Add a row, and then configure the third child action that defines the menu icon for the Theme Cosmos page:
   a. In the Action list, select Set.
   b. In the Target field, enter `.pxPageViewIcon`.
   c. In the Relation field, list equal to.
   d. In the Source field, enter `"pi pi-case-solid"`.

Result:

The following figure shows the configuration for the *pyPopulateAdditionalNavPages* Data Transform without an additional When Condition that limits the visibility of the Configuration landing page to a specific group of users:



3.7.1	Append and Map to	Primary.pxResults	a new page	
3.7.1.1	Set	.pyLabel	equal to	Configuration
3.7.1.2	Set	.pyURLContent	equal to	"https://agilestudioadoption.pega.com/prwe
3.7.1.3	Set	.pxPageViewIcon	equal to	"pi pi-case-solid"

Configuration of the pyPopulateAdditionalNavPages Data Transform

Result:

Users of your Constellation application can access delegated Rules and data types from the Case Manager Portal. Users can also access data types in App Studio.

Delegating a data type

You can delegate a data type so that other users can modify records in local data storage tables.

1. Click **Turn editing on**.
2. In the navigation pane, click **Data** to view the list of current data types in your application.
3. Select the data type that you want to delegate.
4. On the **Settings** tab, select the **Allow selected users to modify records** checkbox.
5. Select the role of the users that you want to let modify records.
6. Click **Done**.

- Modifying a delegated record or data type
- Creating a data object

Prerequisites for delegating a Rule or data type

Before delegating a Rule or data type, ensure that you certain requirements, such as having the appropriate permissions.

Ensure that the following prerequisites are met:

- The Rule or data type you are delegating is not a checkout.
- The Rule you are delegating is not in a Pega Ruleset.
- You have the *pxCanDelegateRules* privilege, which grants you the ability to delegate Rules and data types that are visible to users outside of your Access Group.
- The Rules you are delegating are in an unlocked Ruleset accessible to end users.

- Delegating a Rule or Data Type

Best practices for delegating a Rule or data type

To successfully delegate a Rule or data type to less-technical business users, you should follow certain best practices.

When you delegate Rules or data types, you should:

- Choose titles and descriptions that are meaningful in a business context.
- Isolate delegated Rules in a separate, unlocked Ruleset that does not require checkins or checkouts. This insulates the rest of your application, which typically resides in locked Rulesets, from frequent changes made by managers to other delegates.
- Provide appropriate training and documentation to line managers.
- If you delegate a Rule or data type and then rename the class, you must re-delegate the class.

- Delegating a Rule or Data Type

Modifying a delegated record or data type

You can modify records or data types that are delegated to you by developers. By using delegation, you can change the functionality of your application without involving additional staff members, such as database administrators and information technology professionals. For example, you can modify the body of an email template or delete an entry from a list of customer addresses.

1. In Case Manager the left navigation pane, click **Configuration**.
2. To filter items by keywords, enter text in the search field.
3. Click **Edit** to modify a delegated item.
   The type of information you can modify depends on the type of item and the options selected at the time of delegation.

4. Click **Check Out** for a checked-in item to create a copy of the record in your personal Ruleset. No one else can check out a record while it is checked out. The **Check Out** button appears when all of the following conditions are true:

   - The record belongs to a Ruleset that has checkouts enabled.

   - For the operator, the **Allow Rule check out** checkbox (on the Security tab of the Operator ID form) is selected.

   - The record is not locked by another user or in a locked Ruleset.

5. Click **Check In** for a checkout in your personal Ruleset to copy the modified instance over the original instance in the public Ruleset. This action also releases the lock so other operators can use the **Check out** action.
6. Click **Discard** to delete a checked-out record from your personal Ruleset. The original, unchecked-out record is not affected.
7. Click **Save**.

- Delegating a data type

Configuring Ruleset update behavior

You can configure when users see new Ruleset lists due to changes of Access Groups or applications: immediately, at the start of the next thread, or at the next session.

By default, a session's Ruleset list changes immediately when the session's Access Group or application changes. You can configure your server to change the Ruleset list at the start of the next thread or session.

1. Configure the *prconfig/Authorization/RSLUpdateBehavior/default* dynamic system setting (DSS) with the owning Ruleset *Pega-Engine*and a value that reflects how you want the Ruleset to update:
   - `Immediate`
     - The default setting.

        - Ruleset lists are updated as application and Access Group changes are made.
    - `Threadset`
        - Ruleset lists are updated from application and Access Group changes at thread creation boundaries and stay the same for the entire lifetime of the thread.
        - Thread creation boundaries include thread creation and thread switch events.
    - `Fixedreq`
        - Snapshots of the Ruleset lists are taken for all available applications at login.
        - Fixed throughout the lifetime of the session.
        - If a requestor with the Ruleset list update behavior that is set to Fixedreq spawns a child requestor, the child requestor gets the most recent context using their parent's Access Group as a key.

   For more information, see Creating a dynamic system setting.
2. Restart the server.

- Configuring dynamic system settings
- Changing node settings by modifying the prconfig.xml file

Deleting a Rule

Save disk space by deleting Rules that are no longer relevant in your application. For example, during application development, you create a field property to capture a phone number, but later you need to capture multiple phone numbers. You create a list property, and then delete the field property. As a result, you improve the extensibility and user understanding of your application by providing only the Rules that your application requires.

NOTE: After you ship a Ruleset, withdraw or block a Rule instead of deleting it. You cannot search for deleted Rules.

When you develop a Process in App Studio, your application might create additional Rules, such as views, that the final Process does not require to function correctly. To ensure that you deliver only the minimum number of Rules that your application requires, remove such unnecessary Rules.

The following conditions can prevent you from deleting a Rule:

- You cannot delete a Rule that belongs to a locked Ruleset version. However, in most situations, you can create a blocked or withdrawn Rule in your application that masks a Rule (makes invisible to Rule resolution), that you no longer need in your application.
- You cannot delete standard Rules because they are part of the Pega Platform™ product. However, you can override many standard Rules.
- You can delete a Rule or data instance only in the following circumstances:
    - If an Access of Role to Object Rule that is associated with your Access Role allows you to delete Rules or data instances.
    - If no Access Deny Rules that are associated with your Access Role prevent you from deleting Rules or data instances.
- You cannot delete a Rule in which the Circumstance or Start Time fields are blank if your system contains other Rules with identical keys that are circumstance-qualified or time-qualified. Delete the qualified Rules first, and then delete the unqualified Rule.
- You cannot delete an *Operator ID* if the operator has any checked-out Rules. Before you delete an Operator ID, the operator needs to sign in, and delete or check in all Rules in the personal Ruleset.
- You cannot delete a concrete class that contains instances.
- You cannot delete a class, either concrete or abstract, if the system contains Rules with the same class name as the Applies To class. You receive a notification with a list of the Rules that you need to delete before you delete the class Rule.
- You cannot delete a Ruleset version Rule that identifies a non-empty collection of Rules. First, you need to delete each of the Rules in the version.
- You cannot delete a Ruleset for which a Ruleset version exists. Delete each version first.

1. In the navigation pane of Dev Studio, click Records.
2. In the Records explorer, expand the category of the Rule that you want to delete, and then click its Rule Type.

   > **For example:**
   >
   > If you want to delete an activity, expand the Technical category, and then click Activity.

3. In the list of Rule instances, click the Rule that you want to delete.
4. Click Delete.
5. In the **Delete** dialog box, describe the reason for deleting the Rule, and then click Delete.
6. **Optional:** To undo your changes while you still have the Rule open, click Restore.

> **What to do next:**
>
> Recover a specific version of a deleted Rule. For more information, see Recovering a deleted Rule.

- **Recovering a deleted Rule**

- Performing a private edit
- Copying a Rule or data instance

Recovering a deleted Rule

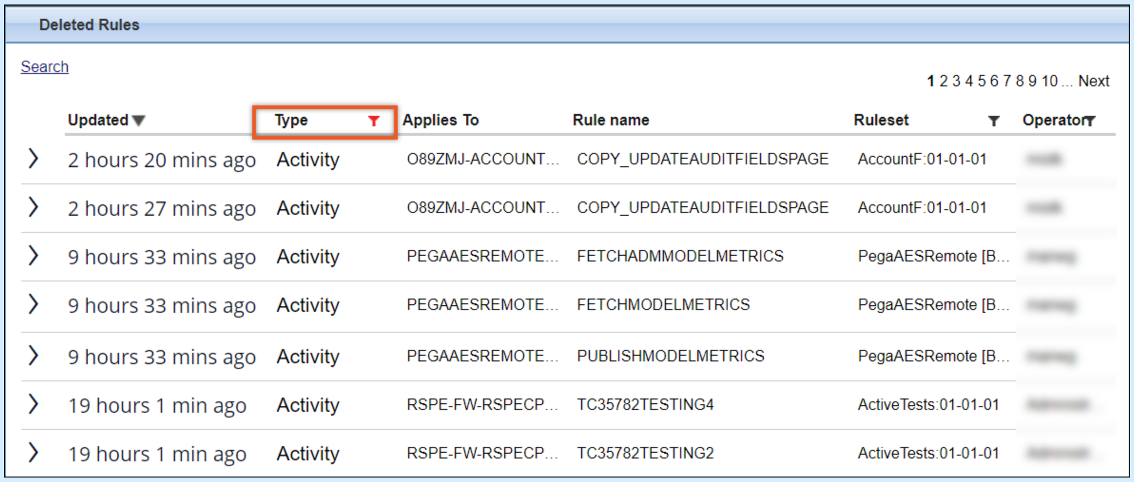
Facilitate application development by recovering one of the most recent versions of a deleted Rule. For example, if you accidentally delete a property, or delete a Process Rule that seems irrelevant, but you later realize that you need the Process to complete the Case Lifecycle.

NOTE: Recovering a deleted Rule is a different operation than restoring a Rule immediately after deleting it. By recovering a Rule, you can roll back the Rule to a specific version.

1. In the header of Dev Studio, click Configure > Application > Development > Recent Actions.
2. Click View Deleted Rules.
3. Filter the relevant columns to narrow the results.

For example:

Filter the **Type** column to find removed activities, as shown in the following figure:



Deleted Rules

Search

1 2 3 4 5 6 7 8 9 10 ... Next

Updated	Type	Applies To	Rule name	Ruleset	Operator
2 hours 20 mins ago	Activity	O89ZMJ-ACCOUNT...	COPY_UPDATEAUDITFIELDSPAGE	AccountF:01-01-01	[redacted]
2 hours 27 mins ago	Activity	O89ZMJ-ACCOUNT...	COPY_UPDATEAUDITFIELDSPAGE	AccountF:01-01-01	[redacted]
9 hours 33 mins ago	Activity	PEGAAESREMOTE...	FETCHADMMODELMETRICS	PegaAESRemote [B...	[redacted]
9 hours 33 mins ago	Activity	PEGAAESREMOTE...	FETCHMODELMETRICS	PegaAESRemote [B...	[redacted]
9 hours 33 mins ago	Activity	PEGAAESREMOTE...	PUBLISHMODELMETRICS	PegaAESRemote [B...	[redacted]
19 hours 1 min ago	Activity	RSPE-FW-RSPECP...	TC35782TESTING4	ActiveTests:01-01-01	[redacted]
19 hours 1 min ago	Activity	RSPE-FW-RSPECP...	TC35782TESTING2	ActiveTests:01-01-01	[redacted]

List of deleted activities

4. Find the Rule that you want to recover, and then click the value in the **Rule name** column to open the Rule.
5. Click Restore.
6. Click Save.

- Restoring the earlier state of a Rule

Exploring Rules in your application

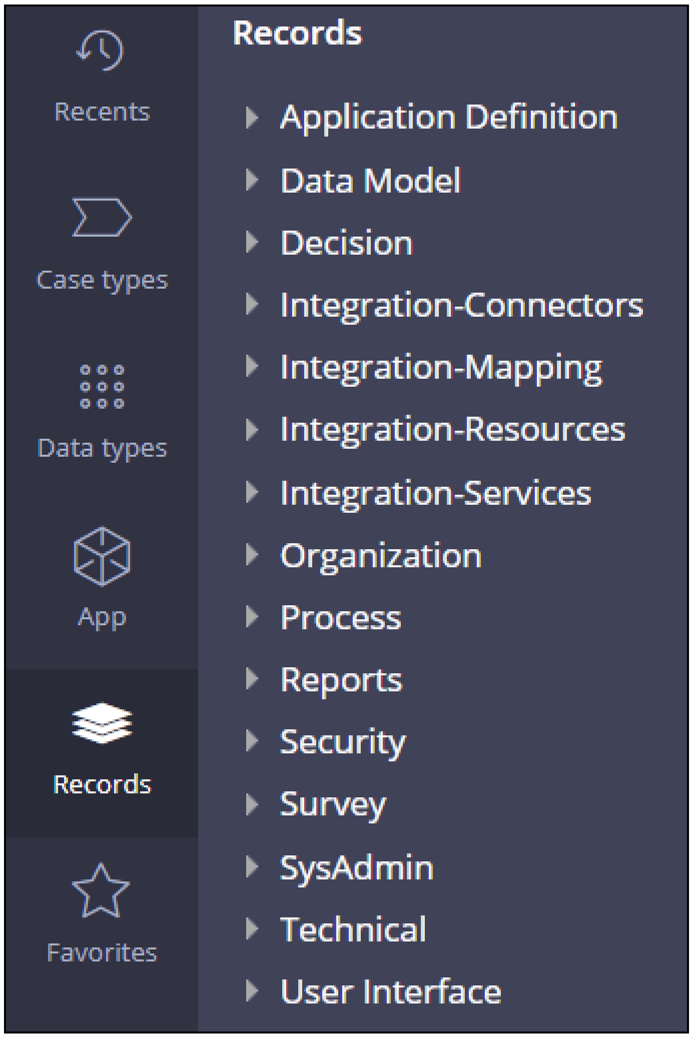
Ensure that your application includes all the features that your business processes require by exploring Rules that define these features. Exploring Rules before you define new elements of your application also helps you avoid duplicating resources, and as a result, save time and development costs.

By exploring Rules, you can find different records that define your application without knowing their exact name or location in your instance of Pega Platform™.

You can explore Rules by applying multiple criteria, such as type, name, or class.

Records Explorer

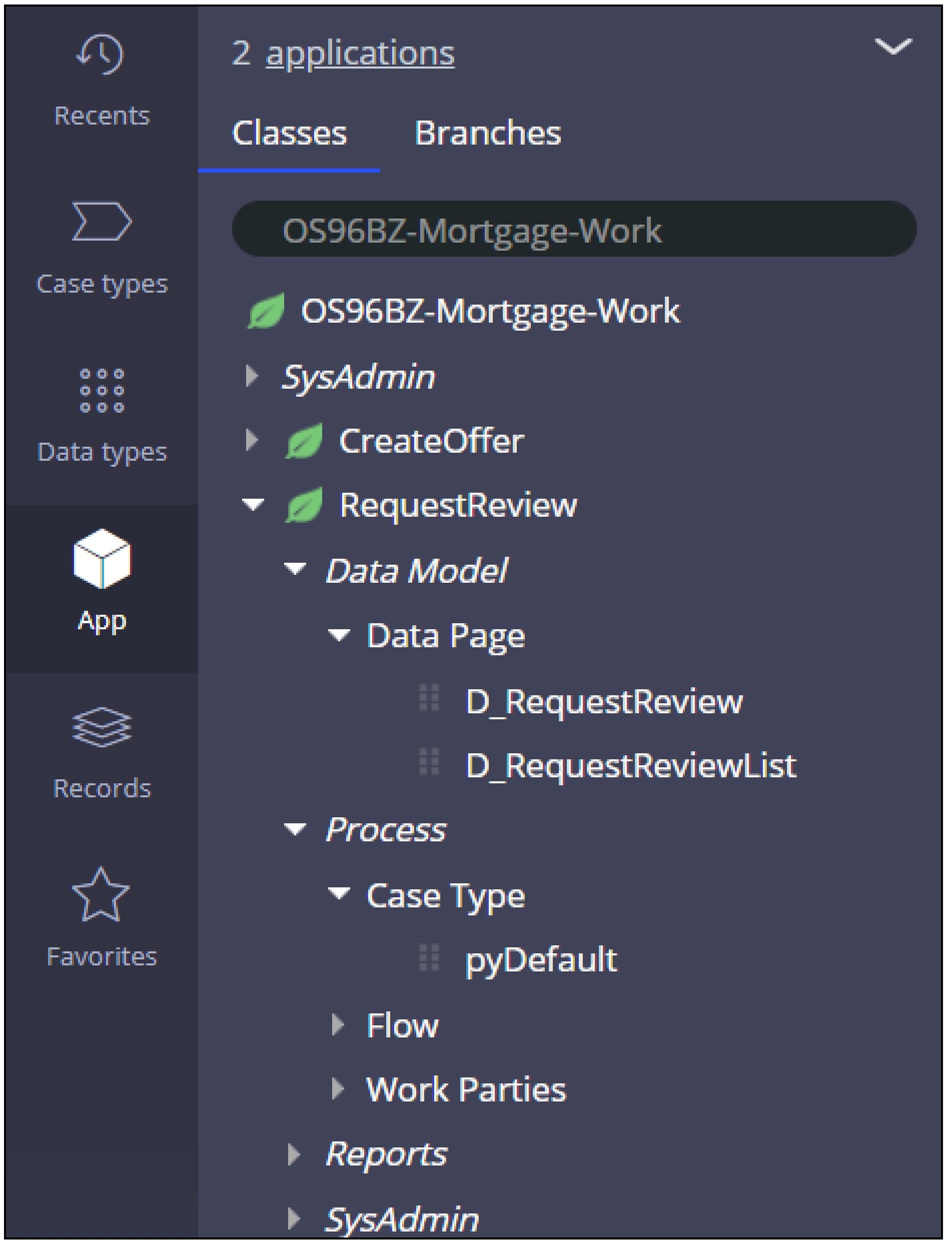
By using the Records Explorer, you can find Rules based on their type, as shown in the following figure:



Records Explorer

App Explorer

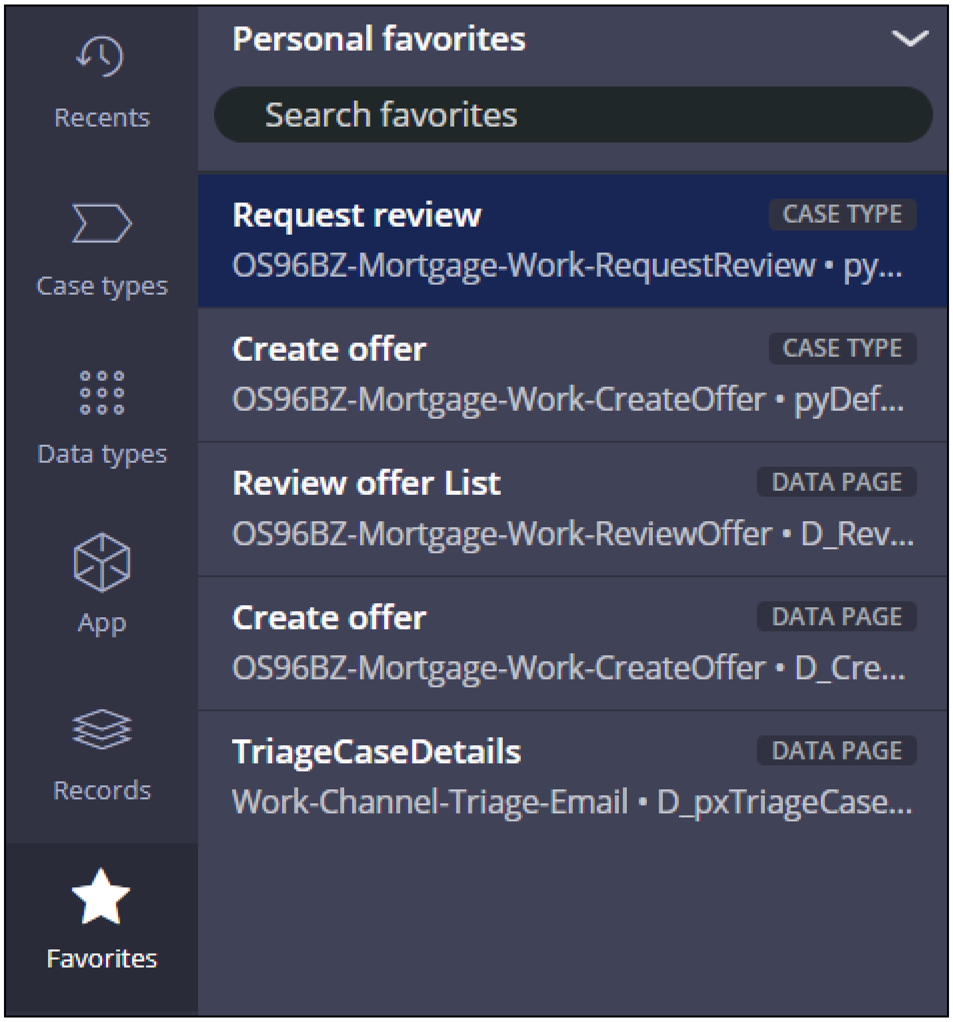
By exploring Rules by class, you can better understand inheritance patterns, and, as a result, you can manage your resources more efficiently by reusing them between classes. The following figure shows exploring Rules by class:



Exploring classes

Additional actions

For more convenient and better-informed application development, you can perform various actions on Rules that you find, such as bookmarking a Rule, viewing Rule versions and history, or viewing the XML of a Rule for more advanced purposes. The following figure shows a list of bookmarked Rules:



Request review CASE TYPE
OS96BZ-Mortgage-Work-RequestReview • py...

Create offer CASE TYPE
OS96BZ-Mortgage-Work-CreateOffer • pyDef...

Review offer List DATA PAGE
OS96BZ-Mortgage-Work-ReviewOffer • D_Rev...

Create offer DATA PAGE
OS96BZ-Mortgage-Work-CreateOffer • D_Cre...

TriageCaseDetails DATA PAGE
Work-Channel-Triage-Email • D_pxTriageCase...

Favorites

Learn more about finding Rules by exploring the following articles:

- **Finding Rules by type**
- **Finding Rules by class**
- **Finding Rules by name**
- **Finding Rules by dependency**
- **Rule versions review and comparison**
- **Comparing Rules by system**

- **Bookmarking a Rule**
- **Searching for a Rule**
- **Reporting on Rules**
- **Viewing Rule history**
- **Viewing Rule versions**
- **Viewing the generated Java of a Rule**
- **Viewing the XML of a Rule**
- **Replacing a string in Rulesets**
- **Finding deprecated Rules in sections**

Finding Rules by type

You can use the Records Explorer to find all instances of a specific class, or Rule Type. By using the Records Explorer instead of searching, you can quickly find a Rule when you know the Rule Type but do not know the Rule name.

Records Explorer also displays additional information about Rules, such as a user who commited the last Rule update.

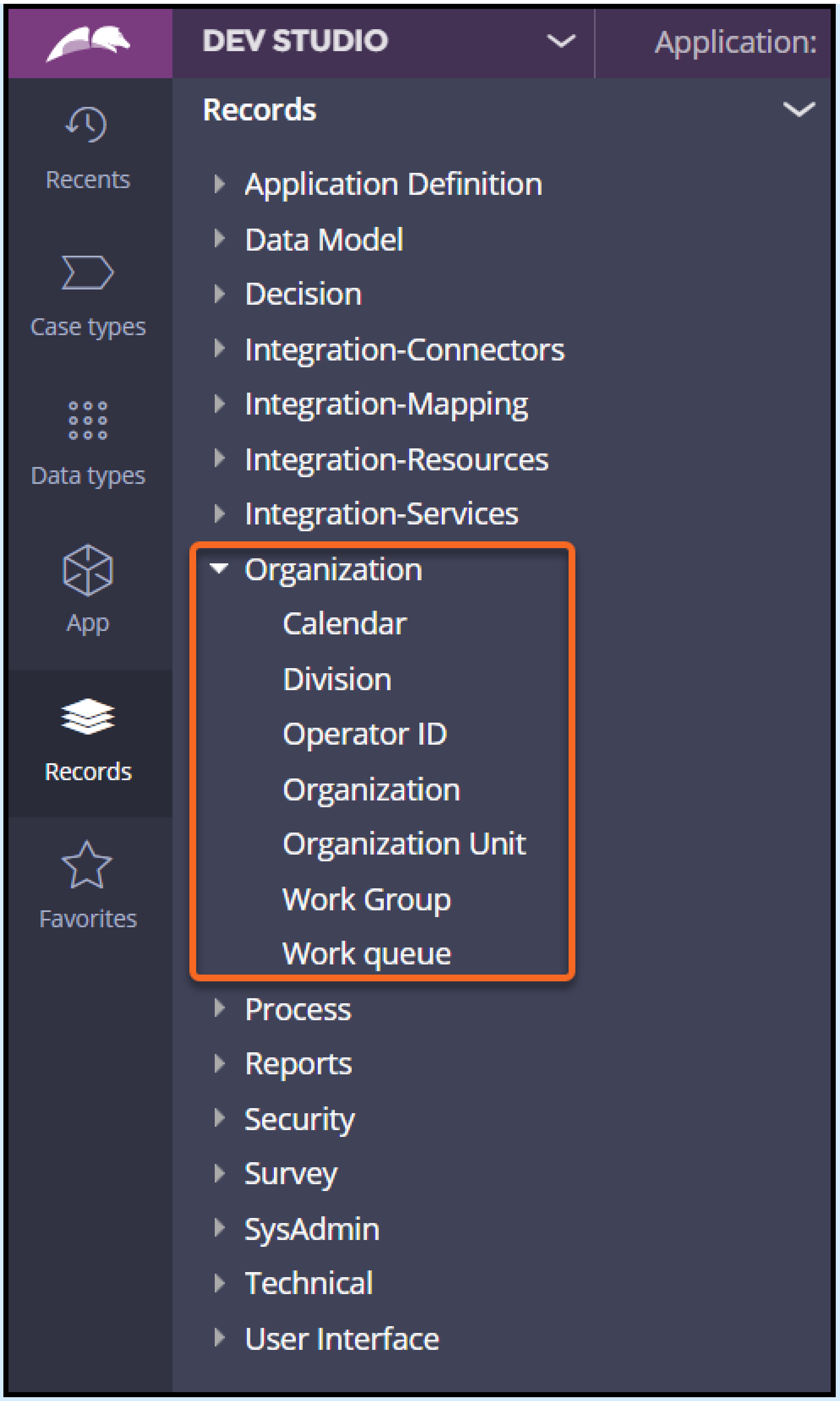
Before you begin:

To view Rules in the **Security** category, ensure that your Access Role has the *pzCanManageSecurityPolicies* privilege. For more information, see Granting privileges to an Access Role.

1. In the navigation pane of Dev Studio, click Records.
2. Browse the different Rule Types by expanding a Rule category.

For example:

The following figure shows Rule Types in the **Organization** category:



Records

- Application Definition
- Data Model
- Decision
- Integration-Connectors
- Integration-Mapping
- Integration-Resources
- Integration-Services
- Organization
  - Calendar
  - Division
  - Operator ID
  - Organization
  - Organization Unit
  - Work Group
  - Work queue
- Process
- Reports
- Security
- Survey
- SysAdmin
- Technical
- User Interface

Rule Types in the Organization category

3. Click a Rule Type whose instances you want to explore.

> **For example:**
>
> Click Operator ID.

4. **Optional:** To find the Rule more quickly, review, sort, or filter the results by clicking the Click to filter icon by the parameter that you want to apply.
5. Open the Rule by clicking it.

- Creating Rules

Finding Rules by class

You can use the Application Explorer to find Rules that share the same class. By using the Application Explorer instead of searching, you can quickly find Rules that are in your application or inherited by your application, without having to know their names.

1. In the navigation pane of Dev Studio, click App.
2. **Optional:** To find Rules in a specific built-on application, change the Scope of the Application Explorer.
   a. Click applications.
   b. Select the checkbox next to an application name.
   c. Click Apply.
3. On the Classes tab, type one or more characters to display a list of matching class names.
4. Click a class name to browse its Rules and class structure.

> **TIP:** To reset the Application Explorer to display Rules in your team, clear the class name.

5. **Optional:** To bookmark this class, hover on the class name, and then click the Pin icon.

6. Expand subclasses and Rule Types, until you find the Rule of interest.
7. Click the Rule to open it.

- Understanding class hierarchy and inheritance
- Finding the class of a Rule

Finding Rules by name

To find sibling Rules, which are Rules with the same name (key parts) but different Rulesets or versions, do the following..

1. Open the Rule form.
2. Select **Actions** > **View Siblings** to search the entire PegaRULES database for Rules that share a second key part with the currently displayed Rule. The results are displayed in a new window.

What to do next:

The Sibling Rules window shows all the Rules that share the same second key part as the current Rule. Not all of these are true siblings. This display is most useful for Rule Types with two key parts that have an Applies To class as the first key part.

The window might include Rules in Rulesets you cannot access, Rules that are unavailable, and circumstanced.

Rows in the display correspond to Applies To values. Numbers in brackets show how many distinct Rules (counting siblings, versions, circumstances, and so on) are in each class.

Columns identify Rulesets and versions. Each cell that contains an icon indicates that a Rule of that name and Applies To class exists, with an icon describing the Rule availability.

You can do any of the following actions:

- Click the Circumstanced link, which appears for a Rule with circumstance verions, to display the circumstance property and value.
- Select a value in the Applies To field to change its value and click Get Related Rules to view Rules with the same second key part in other classes.
- Click Legend to view a legend for this window.
- Click Details to display information about the sibling Rules.

Finding Rules by dependency

Save development time and find Rules by dependency with the Referencing Rules tool. Instead of conducting a full-content search, you can quickly locate a Rule in context by tracing the relationship between the Rules in your application.

For example, before you remove steps from a Data Transform, you can use the Referencing Rules tool to find the Rules that the removal affects.

NOTE: Some Rules do not support the Referencing Rules tool, because their references are determined at run time.

1. Open a Rule whose dependencies you want to check.
2. Click Actions > View references.
3. Review the list of Rules in the Referencing Rules and Referenced Rules tabs to understand the relationships between the Rules.
4. **Optional:** To open any of the Rules listed in the tabs in their own, new tab, click the Edit icon.
5. **Optional:** To filter the results, on the Referencing Rules tab, click the Settings icon, and then choose any of the following options:
    - To narrow the search results to the Rules used only in the application, in the Rule Sets list, select Application Only.

- To include Rules with the same name in the same class path, select the **Show overrides** checkbox.
- To include other versions of the Rule, select the **Show Versions and Circumstances** checkbox.

Finding Rules by custom field

The Find Rules by Custom Field wizard helps you generate a tailored report of Rules in the system with custom fields defined.

> **NOTE:** Only Decision Rules are supported by this wizard by default. You can include additional Rule Types in search results by overriding the standard list view Rule *Rule-Obj-Class.getCategorizedRules*.

1. In the header of Dev Studio, click **Configure** > **Case Management** > **Tools** > **Find Rules By Custom Field**.
2. Filter the search results based on the value of a custom field.
3. Choose which property values to include in the report that the wizard generates.
4. Review the report.

Selecting a Rule in the Find Rules by Custom Field wizard

Select the Rules you want to include in the search by performing one of the following actions.

- Select the checkbox next to an individual Rule Type to include it in search results.
- Select the checkbox next to a Rule category to include a group of Rule Types in search results.

		Next: Setting filters

- Finding Rules by custom field

Find Rules by Custom Field Wizard - Step 2

Setting filters in the Find Rules by Custom Field wizard.

Before you begin:

Enter criteria in this step to filter results based on the value of a custom field. For example, you can find all Rules with an applies to class name that starts with Rule-Obj or a Ruleset name that contains the string customer.

1. Select a label from the **Field** menu of the custom field to evaluate. This list is populated by values in the pr_index_customfields database table. If you do not see a custom field in the list, it must be exposed as a column.
2. Select a comparison technique from the **Condition** menu.
3. Enter a string in the **Value** field to compare it with the value of the custom field selected in the **Field** menu.
4. Add rows to define additional criteria or delete rows to remove them.
5. In the Filter Logic (Advanced) section, specify the search criteria order of operations. Each row in the Filter by Custom Fields section is labeled by a letter. Enter a logical string using these letters, parentheses, and keywords AND, OR, NOT; for example, (A AND B) OR C. If no logical string is specified, the system uses AND to join all search criteria together.

What to do next:

Previous: Selecting a Rule in the Find Rules by Custom Field wizard		Next: Selecting fields

- Optimizing database properties

Find Rules by Custom Field Wizard - Step 3

Selecting fields in the Find Rules by Custom Field wizard

Include additional properties in the generated report. The selections you make here only augment the returned results; they do not filter them.

Select the checkbox next to the property names to include as columns in the final report. You must select at least one property. As a best practice, include the Rule Type property to help organize results.

Previous: Setting filters		Next: Viewing the report

Find Rules by Custom Field Wizard - Step 4

Viewing the report in the Find Rules by Custom Field wizard

Review the list of Rules matching the custom field search criteria. You can expand the report to look at individual items or view the full results offline in Excel.

Results are limited to 1000. Use more specific filter criteria to help isolate results.

Previous: Selecting fields		

Rule versions review and comparison

Inspecting and comparing Rule versions helps you resolve merge conflicts, review and debug code, and audit changes that a developer made while the Rule was checked out.For example, you can compare the Rule that you currently work on with previously checked out versions of the Rule, branched versions, or the same Rule in other Ruleset versions.

Depending on your needs, you can inspect and compare Rules in Pega Platform in the following ways:

Viewing changes to a specific Rule on the History **tab of the Rule**

View changes to a Rule by reviewing different versions of the Rule.

For more information, see Viewing changes to a specific Rule.

Viewing versions of a Rule on the Actions **menu on a Rule form**

View differences between the current Rule and a selected version and modify your version of the Rule.

For more information, see Comparing Rule versions.

Comparing Rules and Rule versions with the Compare Rules tool

Compare two different Rules or two versions of a Rule. The tool displays results in a user- friendly, color-coded way. You also can use the tool to export the results to a Microsoft Excel spreadsheet.
You can compare Rules of the following types:

- Activity
- Data Transform
- Function
- Validate
- Other technical Rules that are text, such as the text files, JSP, and HTML Rules.

For more information, see Comparing Rules and Rule versions with the Compare Rules tool.

Viewing changes to a specific Rule

Get quick and full insight into the changes to a Rule by reviewing different versions of the Rule on the **History** tab of the Rule.

1. Open the Rule that you want to compare.

   For more information, see Exploring Rules in your application.

2. On the Rule form, click the **History** tab.
3. In the **History details** section, click **View full history**.
4. In the new window, click **History for All Versions** to display all versions of the Rule.
5. In the list that appears, select the **Compare** checkbox for each version of the Rule that you want to compare.

   > **NOTE:** You can compare only historic versions of the Rule. You cannot compare the current version of the Rule with a historic version unless your current Rule is checked out. If the current Rule is checked out, the **Compare With Checked Out** button appears.

6. If the current Rule is checked out, to compare a Rule version with the checked out version, select a version, and then click **Compare With Checked Out**.
7. In the **Differences Found** report, select a Rule to view the differences between the Rule versions.

Comparing Rule versions

Compare Rule versions to resolve merge conflicts, debug code, and audit changes that application developers made during Rule check out. You can compare a Rule with versions of the Rule from before the Rule check out, with the branched versions, or with the same Rule in other Ruleset versions.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the category that stores your Rule, and then select a subcategory.

> **For example:**
>
> Click Process, and then click Service-Level Agreement.

3. In the list of Rules, click the name of the Rule for which you want to run the comparison.
4. In the header of the Rule form, click Actions.
5. Click View versions, and then select the version of the Rule that you want to compare with the current Rule.
6. View the list of differences between the current Rule and the compared version. For the current Rule, each list entry shows:
   - The type of changes.
   - The name of the changed page or property.
   - For modifications, the former and current values.
7. If you review Rules properties that store code, use a compare tool that provides a detailed view of changes between versions.
   a. Click Compare text on a difference that you want to examine to view all the changes between versions.

   > **TIP:** You can move to a specific change by clicking the drop-down arrow, and navigate between multiple changes by clicking the arrows.

   b. **Optional:** To make changes to the current version of the file, you can edit, add, or remove text.
   c. **Optional:** To replace a change in the currently edited Rule with the version of the Rule that you use in comparison, click the Revert chunk icon next to the change.
   d. **Optional:** To replace all your changes with the version of the Rule that you are comparing to, click the Undo all changes icon.
   e. After you review the changes, click Done.

8. Click the entry on the list to highlight the tab and element with the change on the current Rule form.
9. **Optional:** To see additional information about page and property changes, expand a list entry.
10. If you make changes to the current Rule in this view, click Save changes to compare your revisions.

> **NOTE:** If the Save changes option is not visible, it means that you do not have permission to make changes to the Rule. To make and save changes to the Rule, exit the Compare view and either check out the Rule or save it into a Branch.

11. **Optional:** To compare the current Rule with another version, select the other version from the list in the Compare Rule header. The view is refreshed with the revised change information.
12. Click Exit compare to return to the Rule form.

Comparing Rules and Rule versions with the Compare Rules tool

Apply the Compare Rules tool to two different Rules or two versions of the same Rule to identify and review changes between Rules. The side-by-side comparison lets you quickly find the additions, deletions, or value changes that have occurred between two versions of a Rule.

1. In the header of Dev Studio, click Configure > System > Refactor > Rules.
2. Click Compare Rules.
3. In the Instance #1 section, specify the first Rule by providing the following information:
    a. In the Rule Name field, enter the name of the Rule that you want to compare.
    b. In the Rule Type field, click the displayed Rule Type and select a value from the list.

    c. In the **Applies To** field, select the class that contains the Rule.
    d. In the **Ruleset** field, select the Ruleset that contains the Rule.
    e. In the **Version** field, select the Ruleset version that contains the Rule.
    f. Click **Find**.
4. In the list of Rules that appears, mark the **Select** field of the first Rule that you want to compare.
5. In the **Instance #2** section of the new panel, select the second Rule to compare by performing one of the following actions:
    - If the second Rule instance that you want to compare is in the list, to select it, click the button beside the Rule name.
    - If the second Rule instance that you want to compare is not in the list, in the new panel, repeat actions from step 3, and then, in the list of Rules that appears, mark the **Select** field of the second Rule that you want to compare.
6. Click **Compare >>** to generate the **Differences Found** report.
7. In a selected column of the report, click a row entry that has a change to display the **Comparison of Rules** form.
    The form displays the first selected Rule in the left column and the second Rule in the right column.

    The differences are color-coded:

    Yellow
    : The values have changed in the two selected instances.

    Green
    : At least one property value in the second Rule is not present in the first Rule. The first selected Rule is presumed to be the source instance. Consequently, the changes highlighted in green are considered additions.

    Red
    : At least one property with values in the first Rule is not present in the second Rule. The first selected Rule is presumed to be the source instance. Consequently, these changes are deletions.

If the screen displays code that includes more than one change, you can move among changes by clicking the left and right arrow keys at the top of the screen. You can also use the search field to locate a specific text string.

Browse the changes by using the scroll bars at the right of each column. Each scroll bar controls both columns, which move together to maintain the relationship between the material displayed from each Rule instance.

8. **Optional:** To save the comparison results to a Microsoft Excel spreadsheet, click Export to Excel.

Comparing Rules by system

You can use the Rulebase Compare wizard to identify differences in the Rules present on two Pega Platform systems. For example, you can use the tool to confirm the success of a product migration between systems, or to identify the changes needed to synchronize two systems.

Starting the tool

Select Configure > System > Refactor > Rulesets > Rulebase Compare to start the wizard. For instructions on the form, see Using the Rulebase Compare tool.

You specify the target system and the Ruleset or a *Rule-Admin-Product* Rule to compare. The same Ruleset or *Rule-Admin-Product* must be on both systems. The comparison processing is performed on the system on which you are running the Rulebase Compare wizard.

Using the Rulebase Compare wizard — Step 1: Enter System Info

Use this wizard form to specify the target system for the Rule base comparison.

Complete the following fields to specify the system you want to compare to your current system.

Field	Description
Select a System	If you previously saved a target server definition, select it from the drop-down.
Host Name	Target server host name.
Port	The TCPI/IP port number for SOAP connections on the target server. Often this is port 80.
Context Root	Target server context root name for the Pega Platform application.
HTTPS?	Select if the port requires an HTTPS (Secure Socket Layer) connection.
Save	Click to save this system specification for future use. Saved specifications appear in the Select a System drop-down the next time you use this wizard.
Username	Enter a valid operator ID for the target server.
Password	Enter the user password for the operator ID you specified.
Next >>	Click to continue to the next step of the wizard.
Cancel	Click to exit the wizard.

Using the Rulebase Compare wizard — Step 2: Select Instances

Use this wizard form to specify the Ruleset or product that defines the set of Rules you wish to compare.

NOTE: If you are comparing products, make sure both product Rules you are comparing include all the Rulesets that you want to compare.

Field	Description
Specify Collection Method	Select the type of Rule collection you want to compare: `Ruleset` or `Rule-Admin-Product.`
Select Rule Admin Product Parameters	If you selected `Rule-Admin-Product` as the Collection Method, complete the following fields:
Name	Select the product Rule on the current system that you want to compare with the target system.
Version	Select the version of that `Rule Admin Product` that you want to compare.
Select Ruleset Parameters	If you selected `Ruleset` as the Collection Method, complete the following fields:
	Click the insert row icon to add a row of Ruleset parameters. Add a row for each Ruleset you want to include in the comparison.
Ruleset Name	Select the name of the Ruleset on the current system that you want to compare to the target system. Choose a Ruleset that is present in both systems.
Minimum Version, Maximum Version	Select the minimum and maximum Ruleset versions of the Ruleset you have selected to compare with the target system. The version range must match Ruleset versions on the

Field	Description
	target system. If you do not specify a range, all versions are assumed.

Using the Rulebase Compare wizard - Step 3: Display Summary

The Parameter Review form displays your target system specification and the Rulesets or product Rule you have chosen to compare.

Review this information to make sure you are performing the comparison you intend to.

Using the Rulebase Compare wizard — Step 4: Display Report

The Synchronization Report lists the Rules, Rulesets, and Ruleset versions in which differences were found between the source and target systems. For each Rule the wizard identifies the action to be performed on the target system to make it match the source system.

Results

The wizard identifies the following types of actions:

- `Add` – Rules that appear on the source system but that are missing on the target system. Add these Rules to the target system to match the source system. The system determines additions and deletions by comparing values of *pzInsKey*, the permanent handle of a Rule.
- `Delete` – Rules that are not on the source system but that are found on the target system. Delete these Rules from the target system to match the source system.
- `Update` – Rules that have a different time stamp (*pxUpdateDateTime*) on the source system than on the target system. The wizard only identifies a difference in the update times between the systems for these Rules; the report does not indicate which is more recent. Compare these Rules on the two systems to

determine whether you want to preserve the change in the most recently updated Rule or use the older Rule without the change.

Do not assume that Rules with the most recent *pxUpdateDateTime* values correspond to the highest version. Rules in any version can be updated at any time.

Click **Export to Excel** to save this listing in a spreadsheet.

Bookmarking a Rule

You can use favorites to bookmark a Rule so that you can quickly find it later. You can create favorites that are visible to your operator exclusively, another operator, or all operators within a specified Access Group.

> **Before you begin:**
>
> - Make sure that the Rule you are adding as a favorite is not a checkout.
> - Make sure that you have opened the correct version of the Rule to add as a favorite.
>
>   For example, a Rule can have a base version and 50 circumstanced versions, one for each state in the United States. You can add the *Vermont* instance as a favorite to Doug Smith and the *California* instance as a favorite to Belle Cruise, respectively.

1. In the **Actions** menu of the Rule form header, click **Add to Favorites**.
2. Enter text in the **Label** field that distinguishes this favorite from other delegated Rules.
   The value you specify appears in the Favorite Explorer and the operator menu in Dev Studio.
3. In the **Add to** drop-down list, indicate who can access this favorite:
   - **My Personal**: Visible to your operator only.
   - **My Access Group**: Visible to all operators in your current Access Group.

    - **Other Access Groups:** Visible to operators in the Access Group you specify.
    - **Other User Personal:** Visible to the specified operator ID only.
4. Select an option to indicate how the system finds this favorite:
    - **Open the highest version:** Rule resolution is used to find and open the delegated Rule.
    - **Always open this version:** The exact version used to create the favorite is opened, even if other instances exist in a higher Ruleset version or class.

   These radio buttons appear when a Rule supports multiple versions.
5. Click **Submit**.

What to do next:

While you can no longer delegate Rules by using favorites, you can use the pyCMDelegatedRules harness in the Case Manager Portal to view Rules that were previously delegated using the **Add to favorites** action.

Searching for a Rule

Dev Studio provides full-text search for non-deprecated Rules, data objects and the help system. You can search help system topics at any time. Searching for Rules or data depends on system-maintained index files.

To start a search perform one of the following actions:

- Enter one or more terms (keywords) into the search text field and press **Enter**.
- Leave the search text field empty and press **Enter** to see a list of Rules recently updated by your operator.
- Click the **Tick** icon next to the search text field to see a list of Rules that are checked out by your operator.

What to do next:

- Use the following drop-down lists to control how results are displayed:

Menu	**Option**	**Result**
Result type	*Rules*	Only return instances of classes derived from the *Rule–* base class.
Data	Only return instances of classes derived from the *Data–* base class.	
Rules and Data		Return results for all record types.
Help Topics		Only return topics available in the Pega Platform Help System.
Search Type	*Name*	Evaluate the *pyRuleName* property for *Rule–* instances, the *pxInsName* for *Data–* instances, and Help topic titles when looking for a match.
All Content		In addition to record names and help topic titles, evaluate record data (XML) and bodies of help topics when looking for match.
Match Criteria	*Contains*	Return results based on words that match any part of for the entered term(s), including in the middle of words.

Menu	Option	Result
Starts With	Return results based on words that begin with the entered term(s).	
Exact Match		Return results based on words that exactly match the entered term(s).
Scope	*My Current Application*	Only evaluate records in your current application when looking for a match.
All Applications		Evaluate records in your entire application stack when looking for a match.

NOTE:

By default, matches include both Rules and data instances.

Changes to filtering options are remembered only for the duration of the session.

The Search tool does not search work items.

Rule–, *Data–* and other matches are ranked by Elasticsearch's default relevancy ranking, based on factors such as the number of times that the search term appears in the object.

- Perform advanced searches:

The search capability is implemented using robust, fault-tolerant Elasticsearch technology for use in distributed environments. Elasticsearch will detect and remove failed nodes in a cluster, automatically replicating and reorganizing nodes as needed.

> **NOTE:** To take advantage of Elasticsearch distributed architecture, multiple nodes should be configured to host Elasticsearch index files. See Full text search.

The search terms entered are converted into the appropriate Elasticsearch query syntax based on the search method chosen. You can enter more advanced Elasticsearch query syntax to perform complex searches.

The following general Rules apply to searches:

- Elasticsearch uses the (`*`) asterisk as a wildcard to match zero or more characters, and the (`?`) question mark as a wildcard to match any single character. The `*` character is automatically added before and after search terms when you use the Starts With (term `*`) or Contains (`*` term `*`) search method , and you do not have to manually enter them.
- To search for a Rule based on its Applies To class and name, select the `All Content` option for Search Type (since *pyRuleName* does not contain the class name), and enter a search such as `class-name*rule-name` or `class-name`. For example, to see the version of the *pyLabel* property within the *Work-* class, search for `Work-*pyLabel`.
- If you enter two or more words, they are treated by default as a single phrase during search.

- Your search string may contain characters - _ % or @. Your search string may contain a ! but escaped with a preceding \ character. Searches with any of these characters are exact match only. Searches involving other special characters such as ([{ }]} ^ ~ * # : \ | are not supported.

- When a search string is enclosed in quotes, the selected search method is ignored, and an Exact Match is always performed using the exact text within quotes.

- Use the special syntax `property-name:value` to search for instances with a specific property value. Set the Search Type option to `Content` for this type of search. Enter an exact property name, including capitalization. The searched value cannot contain any special characters. Instances whose value starts with the value entered are matched. For example, `pxObjClass:Rule-Obj-Flow` returns a list of all *Rule-Obj-Flow* and *Rule-Obj-FlowAction* Rules.
- You can enter more advanced Elasticsearch queries. For information about the Elasticsearch query syntax, go to the Elasticsearch website at elastic.co.

  When entering advanced syntax, select **Exact Match** as the **Search Method**. For example, you can use the Boolean operators `AND`, `OR`, and `NOT` to search for combinations of multiple terms and phrases. Capitalize the Boolean operators to distinguish them from the terms or phrases being combined. You can use parentheses to group conditions. See the examples below.

Entered text	**Search method**	**Syntax sent to Elasticsearch**	**Match returned if**
`abc`	`Exact Match`	`abc`	Exact word `abc` exists in searched text.

Entered text	Search method	Syntax sent to Elasticsearch	Match returned if
`abc`	`Starts With`	`abc*`	One or more words beginning with `abc` exist in searched text.
`abc`	`Contains`	`*abc*`	One or more words containing `abc` exist in searched text.
`abc def`	`Exact Match`	`abc def`	Exact phrase `abc def` exists in searched text.
`abc def`	`Starts With`	`abc def*`	One or more phrases starting with the exact word `abc` followed by a word starting with `def` exists in searched text.
`abc def`	`Contains`	`*abc def*`	The string `abc def` exists anywhere in searched text.
`“abc:01-01-01”`	`Any`	`abc:01-01-01`	Exact word `abc:01-01-01` exists in searched text.

Entered text	Search method	Syntax sent to Elasticsearch	Match returned if
`abc\:def`	`Exact Match`	`abc\:def`	Exact word `abc:01-01-01` exists in searched text.
`abc\:def`	`Starts With`	`abc\:def*`	One or more words beginning with `abc:def` exist in searched text.
`abc\:def`	`Contains`	`*abc\:def*`	One or more words containing `abc:def` exists in searched text.
abc OR def	`Exact Match`	`abc OR def`	Either the exact word `abc` or the exact word `def` exists in searched text.
`abc* OR *def*`	`Exact Match`	`abc* OR *def*`	Either one or more words beginning with `abc` or one or more words containing `def` exists in searched text.

Entered text	Search method	Syntax sent to Elasticsearch	Match returned if
`abc AND def`	`Exact Match`	`abc AND def`	Both the exact word `abc` and the exact word `def` exists in searched text.
`abc AND def`	`Exact Match`	`abc* AND def*`	Both one or more words beginning with `abc` and one or more words beginning with `def` exists in searched text.
`(+abc* *def*) NOT ghi*`	`Any`	`(+abc* *def*) NOT ghi*`	One or more words starting with `abc` exists in searched text, the string `def` may also exist anywhere within the searched text, and no words starting with `ghi` exist in the searched text.

Reporting on Rules

You can report on Rules of more than one type and Rules that are checked out. For reporting purposes, searching all instances of the classes derived from the *Rule-* base class, or even all instances of the *Rule-Service-* class, can impact performance. To provide more efficient reporting, you can create report definitions that query the database view that corresponds to a particular class.

- To report on Rules of more than one type, create a report definition Rule with *Data-Rule-Summary* as the Apply to class. The properties that you use for selection criteria must be stored as exposed columns of the PegaRULES database view that corresponds to the *Data-Rule-Summary* class.
- To report on Rules that are checked out, create a report definition Rule with *Data-Rule-Locking* as the Apply to class.

- Viewing and running inventory reports

Viewing the heat map

You can view a heat map that shows in the number of Rules in each category, broken down by Rule Type. The size of each box reflects the number of Rules of a single Rule Type. Complete the following steps.

1. In Dev Studio, click **Configure** > **Application** > **Inventory** > **Heat Map**.
2. In the **Shaded by** list, select an option to graphically shade the heat map by number of recently updated Rules, number of warnings, or number of Rules checked out per category. The darker the shade, the higher the number of the filtered type that is found in that Rule Type.
3. Click the **Filter** button to add or remove Rulesets and Rule categories from the heat map.
4. Left-click a Rule Type on the heat map top open a report that displays all Rules of the selected type in the selected type in the application.
5. Right-click a Rule Type on the heat map to open a report that displays all Rules of the selected type in the application that match the filter selected for the recently

updated Rules, the number of warnings, and the number of Rules checked out per category.

- Managing application inventory

Viewing and running inventory reports

Inventory reports are not associated with a specific application's work items, Assignments, or processing. You can run reports for which the Applies To key part is a class derived from the *System-*, *Data-*, or *Rule-* base classes.

1. To view inventory reports, click **Configure** > **Application** > **Inventory** > **Inventory Reports**.
2. Click the report name to open its Rule form, and specify the appropriate information.
3. To run the report, click **Run** from the **Actions** menu.

- Reporting on Rules

Reviewing exposed properties

To enhance record selection operations for reporting, certain single value properties are stored as exposed columns in an application. You can review which properties are exposed in your appliction.

You can use exposed properties to search for archived Cases.

1. Click **Configure** > **Data Model** > **Classes & Properties** > **Database Class Mappings**.
2. Locate your class in the **Database Class Mappings** table.
3. In your class's row, click the number in the **Exposed/Mapped** column to view exposed properties for that class.

- Insights and Reporting

Viewing Rule history

You can view the saved history of a Rule to see when it was changed and by whom. You can also compare the current version with a previous version or restore a previous version of a Rule for testing purposes, or if the current version is faulty.

1. Open a Rule form, click the **History** tab, and then click **View full history**.
2. Click the **Pencil** icon next to a history record to see a previous version this Rule instance.
3. **Optional:** To make an older copy of the Rule become the current copy, click **Restore** on the Rule form. For more information, see Restoring the earlier state of a Rule.
4. **Optional:** To compare Rule versions, select any two versions of the Rule from the table and click **Compare**. For more information, see Comparing Rule versions on the History tab.
5. Click **History For All Versions** to see the history of all versions of this Rule with the same name, Ruleset, and circumstance qualification, if any. To return to the original display that show showing only the history of this open Rule, click **History of Version NN-NN-NN**.

> **NOTE:** The **History For All Versions** button is not displayed for class Rules (*Rule-Obj-Class* Rule Type) and a few other Rule Types. Class Rules have an associated version field, but do not belong to a single specific Ruleset version.

Related tasks

- Viewing Rule history

Viewing Rule versions

View Rule versions by selecting Actions > View versions. A dialog box appears and displays the versions of the Rule, their availability, and their circumstance. You can click an item to open the Rule form.

You cannot view versions on Rules that are not versioned, such as applications, classes, and Rulesets.

Viewing the generated Java of a Rule

After you save a Rule form, you can view an approximation of the generated Java to be created by a Rule instance by selecting Actions > View Java.

You can view Java if you have the *@baseclass.ToolbarFull* privilege, and you can view Java for certain Rules, including activities, When Rules, and stream Rules.

Viewing the XML of a Rule

If you have the *@baseclass.ToolbarFull* privilege, you can view the XML document that comprises the form that displays a Rule instance by selecting Actions > View XML.

You can view the XML document to assist in debugging so that you can identify the property names and the handle for each property that is referenced in the form.

NOTE: If you have modified and saved the Rule or data form, this information might be out of date. To see updated information, close the form, open the same Rule or data type again, and select Actions > View XML again.

- Finding the class of a Rule

Replacing a string in Rulesets

Use the Search/Replace a String wizard to search for and replace a string wherever it appears in Rules in the entire PegaRULES database.

You can search and replace the string in:

- All Rulesets in the system
- The Rulesets included in your Ruleset list.
- A single Ruleset or set of selected Rulesets

The wizard reports any locked Rulesets or Rules that are checked out before performing the replacement, and any class names that might be affected. A list of qualifying Rules appears for your review before any Rules are altered.

The wizard uses the full-text search to reduce search time and increase accuracy. If full-text search for Rules not enabled for your system, and you do not limit the scope of the process to certain Rulesets, the process can take many minutes to complete.

Use this powerful tool with care. Rules updated by this wizard are not automatically revalidated and may become invalid as a result of string replacement. As a precaution, export the Rulesets in to a .zip archive or back up the entire PegaRULES database before using this tool. The .zip archive can serve as a backup if the results of the wizard are not satisfactory.

Select **Configure** > **System** > **Refactor** > **Rules** > **Search/Replace a String** to start the tool.

In changing a string, the utility looks for the specified string anywhere in any value within any Rule within the search scope, and gives you the option to select the Rules you want to modify. To specify the string to replace for the selected Rules, it will replace the original string with the replacement string.

Strings are replaced only in Rules, not in work items, Assignments, attachments, or other objects. If string replacement changes Rules in subclasses of *Work-* or any of its properties, the subsequent processing of those work items in those classes may encounter problems.

Finding deprecated Rules in sections

Find sections that use deprecated Rules by using an application upgrade utility. Once found, these legacy Rules can be updated to newer Rules.

Your application might contain sections that use legacy Rules or images. Legacy Rules and images have been deprecated or upgraded with newer Rules. Update or refactor legacy Rules to use the latest Pega Platform™ Rules. You can search for legacy Rules by using the Application Upgrade Utilities.

NOTE: Images are found using index references and the results are compared using Rules names. This means that even after you fix an image in a section, the Rule might still be listed in the report.

1. Click **Configure** > **System** > **Release** > **Upgrade** > **Upgrade tools**.
2. Click the **List Rules using legacy Rules** link.
3. From the list of Rules, select a Rule to open it. From here you can upgrade or refactor the Rule.

- Moving applications between systems
- Importing the legacy Ruleset from the Resource Kit