Low-code application development

20 October 2023

CONTENTS

PEGA

Low-code application development

Turn creating applications into a low-code and user-friendly experience by using the tools that Pega Platform offers. With Pega Platform, you can deliver flexible applications that help your customers reach business goals in dynamically changing scenarios, in an efficient way.

When you develop your application by using the digital solutions that Pega Platform provides, you can approach your project delivery holistically and efficiently at the same time. Implement the following techniques to achieve your results faster:

Planning flexible Microjourneys

When you design your application, focus on a goal that your customers want to achieve. In dynamically changing circumstances, predicting a fixed path through a business process might be difficult and lead to inefficient results. Pega Platform refers to a path that leads to a successful resolution as a Microjourney, and supports creating flexible templates for Microjourneys that can adjust to events that take place on the way. In App Studio, you can clearly visualize the main elements of your Microjourney: the people that are involved in the process, the channels that provide communication with your application, and the data that is required to meet the goal. Consequently, you can think about flexible sets of actions that might be useful during the processing of your Microjourney. Visualization of the essential parts of your business processes can include also non-technical stakeholders in the planning process.

For more information, see Creating a Microjourney for customer success *(on page)*.

Adopting feature-driven development

Develop capabilities in the context of a feature to maintain functional requirements and project status directly in your application. You associate features with elements of your application that you want to develop, for example, a language pack. For enhanced project management and effort tracking purposes, you can use features to estimate project costs and schedules before you start actual development.

For more information, see Adopting feature-driven development *(on page)*.

Reusing build-on capabilities

Reuse the functionality that you inherit from built-on applications, so that you can focus your development efforts on unique features and capabilities. For example, you can create an application for banking operations, and then reuse parts of it in applications for reviewing mortgage requests and loan requests. For those three applications, the basic process of collecting and validating information that customers provide might be the same. As a result, you save time but also deliver flexible applications that you can edit

independently from each other if your business requirements change, for example, when the process for reviewing loan requests needs additional elements.

For more information, see Adding built-on applications *(on page 24)*.

Engaging and communicating with stakeholders and team members

Communication is a crucial part of developing applications. Starting with application planning, App Studio offers tools to visualize business processes so that stakeholders from outside IT can be involved in designing a Microjourney – a path towards customer success. Apart from that, Pega Platform supports the uninterrupted exchange of information for development team members, as well as for users who process cases in your application. Enhance collaboration with shareable documents, spaces that gather professionally connected users, and messages that users can post in a Pulse gadget.

For more information, see Creating a Microjourney for customer success *(on page)* and Collaborating on cases *(on page)*.

Case management *(on page)*
Theme Cosmos *(on page)*

Introduction to application development

Start building your applications in a user-friendly and seamless way by implementing low-code digital solutions. Save time when you create your applications by using templates, develop various aspects of your application by engaging collaborators with different skills and roles, and increase work efficiency by configuring your application for reuse. To meet your unique needs, use the Pega Platform authoring portals.

Application development considerations

Before you start developing your application, consider the following factors:

Your business goal

In many situations, predicting an exact course of action for application development leads to fixed and inefficient solutions. With Pega Platform, you can avoid that and instead create agile applications that adjust to dynamically changing needs. center your application development around a goal that you want to achieve to ensure that work can reach a successful resolution under any circumstances.

People in your business process

Determine what types of users need to access your application, and then define the content that is appropriate for every user type. For example, a manager might need access more features than a regular worker. Additionally, consider what communication channels, such as email or chat bot, your users might need, and what devices they use to process work.

Data in your business process

Every business case requires data to reach a resolution. Think about the information that you need to collect from users, as well as the ways of gathering data. To save time, define how you can organize and reuse data in your application. For example, instead of storing separate data objects for every piece of personal information about a user, such as a name, surname, and phone number, you can gather all this information in a personal details data object. Then, you can update or reference one data object instead of multiple different instances.

Reusable elements

You can save considerable amounts of time and resources during application development by reusing various assets. For example, if you already have an application to review loan requests, you can reuse any relevant elements to develop an application for reviewing mortgage requests, or performing other banking operations.

Authoring portals

Pega Platform offers multiple tools that you can use based on your experience, role, and level of expertise. You can use the following authoring portals to develop your applications:

App Studio

Select this portal if you are a citizen developer, front-end developer, business analyst, or data engineer. App Studio holds a great variety of low-code, time-saving solutions for you to implement during your application development. With a real-time UI design, any person that is involved in a business process can contribute to application planning and development.

For more information, see App Studio overview *(on page 8)* and Creating a Microjourney for customer success *(on page)*.

Dev Studio

Select this portal if you are an experienced application developer, a security administrator, or an account administrator. Dev Studio offers tools for advanced application configuration, such as versioning, rules management, or branched development.

For more information, see Dev Studio overview *(on page 10)*.

Admin Studio

Select this portal if you are a system administrator. In Admin Studio, you can monitor and manage your system resources, such as nodes and background processes.

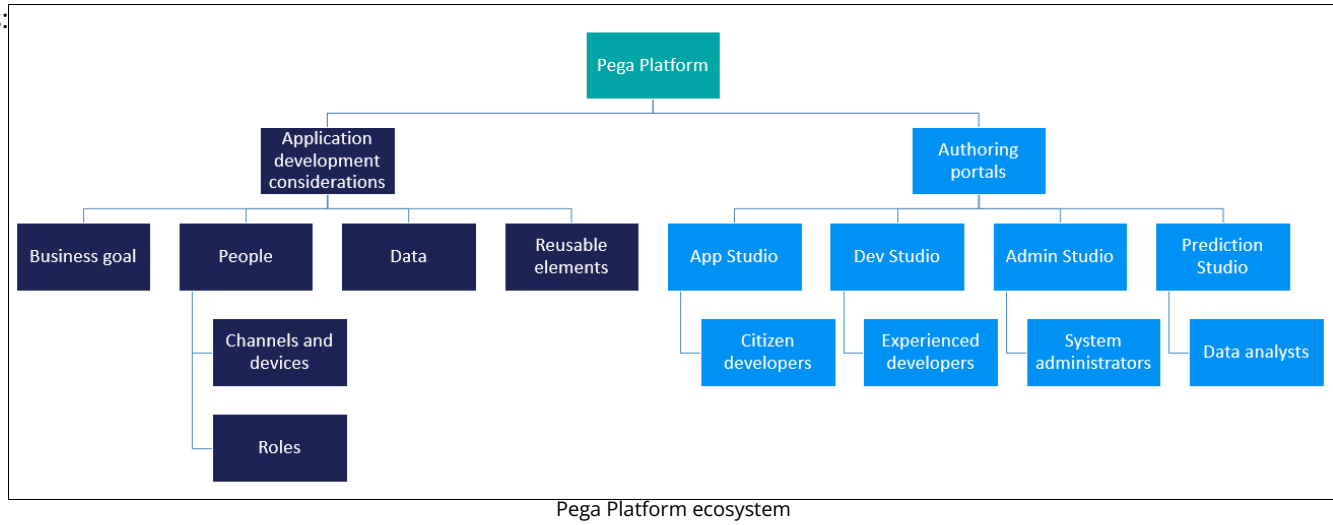
For more information, see Admin Studio overview *(on page 12)*.

Prediction Studio

Select this portal if you are a data scientist. In Prediction Studio, you can develop, monitor, and adjust models for analyzing customer interactions and communications to predict their future behavior.

For more information, see Prediction Studio overview *(on page 14)*.

The following figure summarizes factors and authoring portals to consider before you start creating your applications:



Pega Platform ecosystem

Related information

Case management *(on page)*
Theme Cosmos *(on page)*
Data modeling *(on page)*
Understanding project roles and personas *(on page 46)*
Creating a Microjourney for customer success *(on page)*
Designing applications for reuse and extension *(on page 74)*
Troubleshooting tools and techniques *(on page 340)*

App Studio overview

Turn your application development into a no-code, user-friendly experience by working in App Studio. In this authoring environment, you can configure the main elements of your applications that include templates for your business processes, personas that are involved in the processes, interaction channels, and data.

App Studio purpose

App Studio provides you with tools to develop your applications in a way that is convenient and understandable for a citizen developer with little or no knowledge of any programming language. In App Studio, you can prepare the following application elements:

- reusable templates for your business processes that Pega Platform refers to as case types, and that users complete at run time to reach a successful outcome.

  For more information, see Adding case types to organize work *(on page)*.

- personas that you create to visualize groups of users that interact with your application, such as customers, workers, or managers. You can define access settings for each group so that users view and process only relevant information.

  For more information, see Adding personas to organize users *(on page)*.

- data objects to ensure that the users have information required to resolve cases.

  For more information, see Adding data objects to organize data *(on page)*

The approach of building your business processes on case types, personas, and data objects is a part of the Pega Express delivery approach, the method of delivering projects in a no-code and user-oriented way. In the Pega Express delivery approach, you use App Studio to model your business processes by creating case types, personas, and data objects. A single business process that results in an expected outcome is a Microjourney.

App Studio roles

App Studio targets business analysts, citizen application developers, front-end developers, and data engineers. For improved communication between developers and stakeholders, App Studio supports real-time UI design, so that any person that is involved in a business process can contribute to application planning and development, the approach that helps achieve the best effects while providing IT solutions for business.

App Studio tools

In App Studio, you can find multiple tools for developing and maintaining applications. On the **Overview** page, you can find basic information about your application, for example, case types and personas that your application includes. By exploring work areas, you can quickly view and create the main parts of your application, such as case types, personas, channels, and UI elements. Each work area offers no-code solutions that you can use to model your business processes in the most efficient way. The following video shows what you can find in each work area:

https://players.brightcove.net/1519050010001/default_default/index.html?videoId=6282461521001

To see the results of your work at run time, preview an application display on a device, and launch portals associated with your application. While in a preview, you can turn on the design mode to conveniently make any necessary adjustments by interacting with UI elements. Apart from building applications, in App Studio you can work with tools to collaborate with other developers and track your development work, for example, by posting messages and toggling Agile Workbench. To ensure that you maintain a good health and quality of your application, explore options that you can use to monitor and maintain your application performance. The tools include Clipboard, Tracer, and Accessibility Inspector.

App Studio access

To access App Studio, you associate the *pxExpress* portal with your access group. For more information, see Granting portal access to an access group *(on page 68)*.

From App Studio you can switch to another workspace any time and change the tools and features that are available in your work environment. For more information, see Changing your workspace *(on page 15)*.

For relevant learning materials, see an App Studio module on Pega Academy.

What to do next: Start planning and implementing your Microjourney. See Creating a Microjourney for customer success *(on page)*

Related information

Creating a Microjourney for customer success *(on page)*
The Microjourney in the Pega Express delivery approach FAQ *(on page)*
Workspaces
Dev Studio overview *(on page 10)*
Admin Studio overview *(on page 12)*
Prediction Studio overview *(on page 14)*
Agile Workbench *(on page)*

Dev Studio overview

Configure advanced technical options for your applications in a low-code and goal-oriented way by exploring the tools available in Dev Studio. You can use Dev Studio to save time by managing the reusable resources in your application in a more detailed way, and to configure advanced settings for applications that focus on complex or exceptional business cases.

Dev Studio purpose and roles

Dev Studio contains tools that you can use to develop advanced aspects of your application in a convenient, low-code way. In many scenarios, Dev Studio expands the possibilities that App Studio offers, so that you can create applications for unique, complex, and less common business processes. Dev Studio combines access to all of the capabilities that you might need when you create an application, such as case management, security, reporting, mobile, and UI, so that you can provide end-to-end digital solutions for your projects.

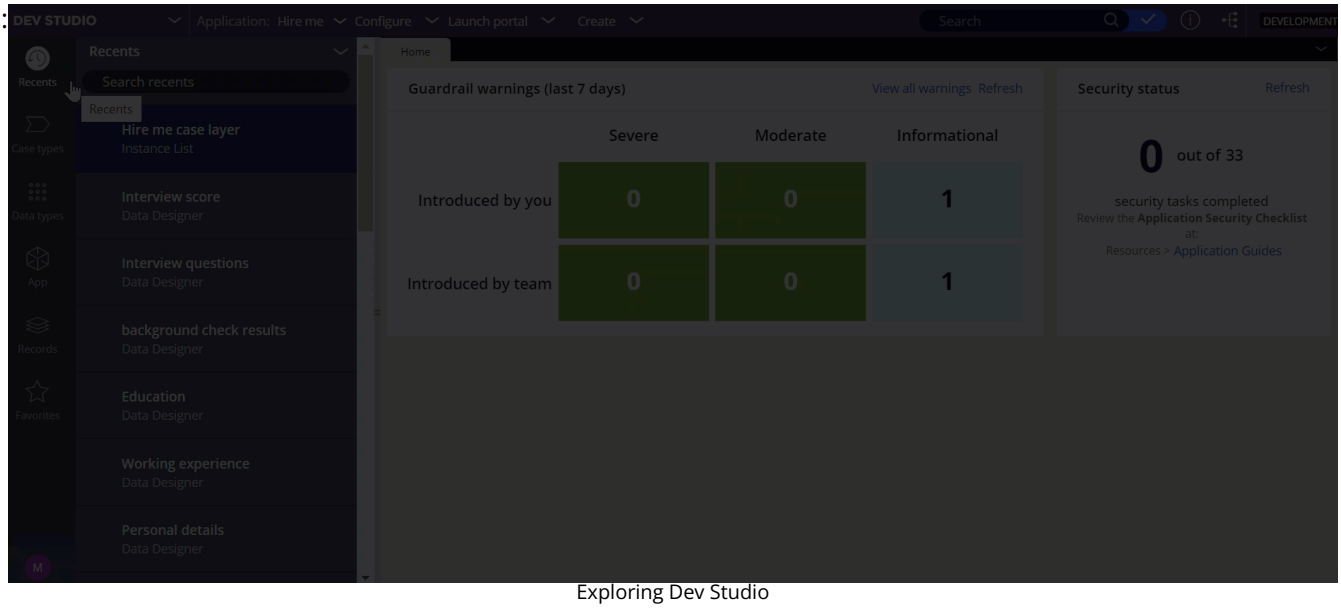
The focus in Dev Studio is on advanced functionality, such as system settings and rules, with access to rule forms and rules management. In Dev Studio, you can also manage security, versioning, and source control of your application. Target Dev Studio users include experienced application developers, security administrators, and account administrators.

Dev Studio tools

Dev Studio provides you with tools to first configure and than maintain your application. On the **Home** tab, you can quickly view recent guardrail warnings to check the compliance as well as the security status of your applications. By exploring the various work areas, you can access and manage the resources that your application includes. For example, you can view, edit, and create the reusable templates of your business processes that Pega Platform refers to as case types. The Data types work area helps you manage data for your application. By creating data types and supplementing them with data pages, you ensure that users of your application have the information they require to resolve business processes. In the App and Records areas, you can find the elements that are the technical building blocks of your application – the rules, classes, and branches. You can also quickly preview the results of your work at run time by launching a portal that you associate with your application. As a result, you can create applications that precisely meet your unique business needs.

Apart from building applications, Dev Studio provides you with tools for collaborating with other developers and tracking your development work, for example, by posting Pulse messages and opening Agile Workbench. To ensure that you maintain the good health and quality of your application, you can explore different options for monitoring and maintaining your application performance with tools such as the Clipboard, Tracer, and Accessibility Inspector.

The following figure explores some of the Dev Studio tools:



Exploring Dev Studio

Dev Studio access

To access Dev Studio, ensure that you associate the Developer portal with your access group. For more information, see Granting portal access to an access group *(on page 68)*.

From Dev Studio you can switch to another workspace at any time and change the tools and features that are available in your work environment. For more information, see Changing your workspace *(on page 15)*.

Admin Studio overview

Monitor and manage resources in your system more efficiently by using the tools that Admin Studio provides. You can use Admin Studio to access run-time information and configuration options for the resources that you create in Dev Studio.

Admin Studio purpose and roles

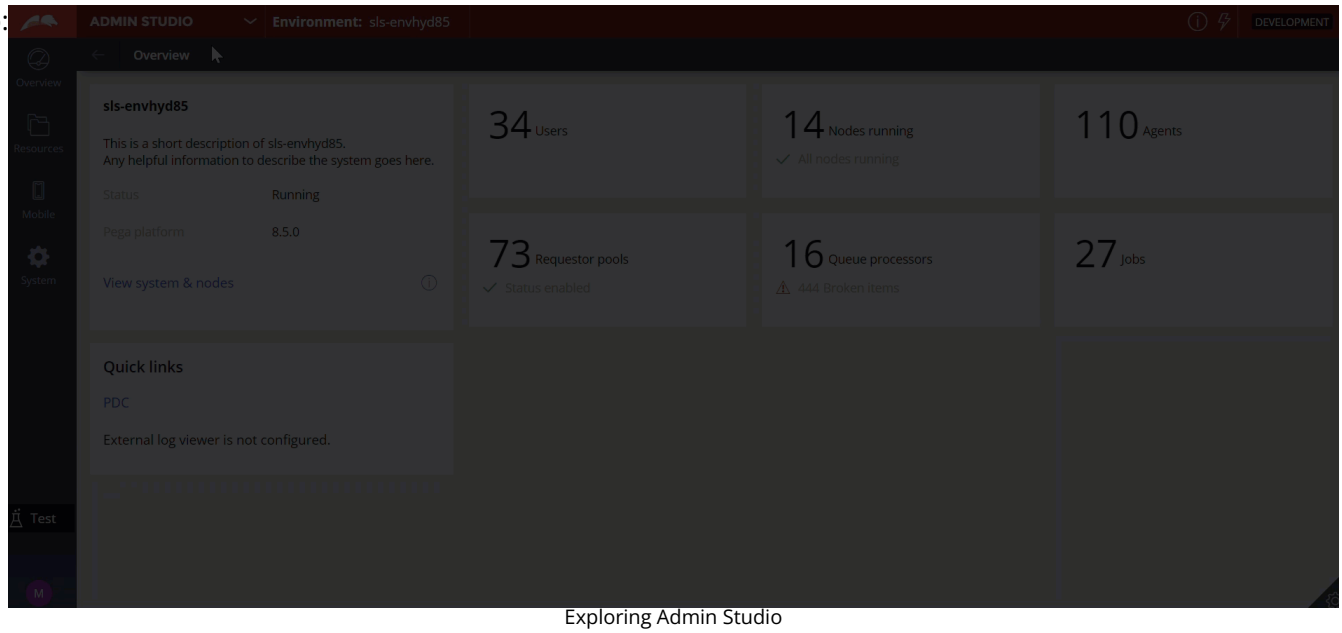
Admin Studio focuses on system administration and provides tools for monitoring and debugging resources in your system. Target users of Admin Studio are system administrators.

Admin Studio tools

By exploring the work areas that Admin Studio includes, you can maintain the top performance of your application. In the **Overview** work area, you can view data about your system and analyze information about all of the resources available in your environment. For example, you can check how many nodes your application includes in total, and how many of those nodes are currently running. The **Resources**

work area provides access to landing pages that focus on specific resources in your system. Depending on the resource, apart from viewing a list of resource instances available in your system, you can also investigate and fix issues, start and stop resources, or even remove resources from the system. In Admin Studio, you can also configure the settings for a mobile build server and manage the cache for users of your application's mobile channel, as well as preview system details, available nodes, other applications, and Java classes.

The following figure explores some of the tools in Admin Studio:



Exploring Admin Studio

Admin Studio access

To access Admin Studio, ensure that you associate the *pxAdminStudio* portal with your access group. For more information, see Granting portal access to an access group *(on page 68)*.

From Admin Studio you can switch to another workspace at any time and change the tools and features that are available in your work environment. For more information, see Changing your workspace *(on page 15)*.

For learning materials about Admin Studio, see an Admin Studio topic on Pega Academy.

Related information

Managing system resources *(on page)*

Prediction Studio overview

Prediction Studio is an authoring environment in which you can control the life cycle of AI and machine learning models (such as model building, monitoring, and update). From Prediction Studio, you can also manage additional resources, such as data sets, taxonomies, and sentiment lexicons.

To access Prediction Studio, you must specify `pxPredictionStudio` as one of the portals associated with your access group. From Prediction Studio, you can switch to another workspace any time and change the tools and features that are available in your work environment. For more information, see Changing your workspace *(on page 15)*.

The following components are available in Prediction Studio:

Prediction Studio Header

The header at the top of Prediction Studio enables you to view the name of the current application, perform a search across the rules in that workspace, view the online help, and toggle the Agile Workbench.

Page header

The page header at the top displays the name of the current work area, for example **Predictions** and enables you to perform a number of common actions such as viewing model reports, clearing deleted models, and so on. This toolbar also allows you to add models or additional resources.

Navigation panel

The navigation panel on the left provides quick access to the following work areas:

Predictions

In this work area, you can create predictions by answering several questions about what you want to predict. You can access, manage, and run existing predictions within your application. You can also perform Machine Learning Operations (MLOps) to replace predictive, adaptive, and scorecard models in predictions with external predictive models, and approve the model updates for deployment to your production environment.

For more information about predictions, see Creating and managing predictions *(on page)*.

Models

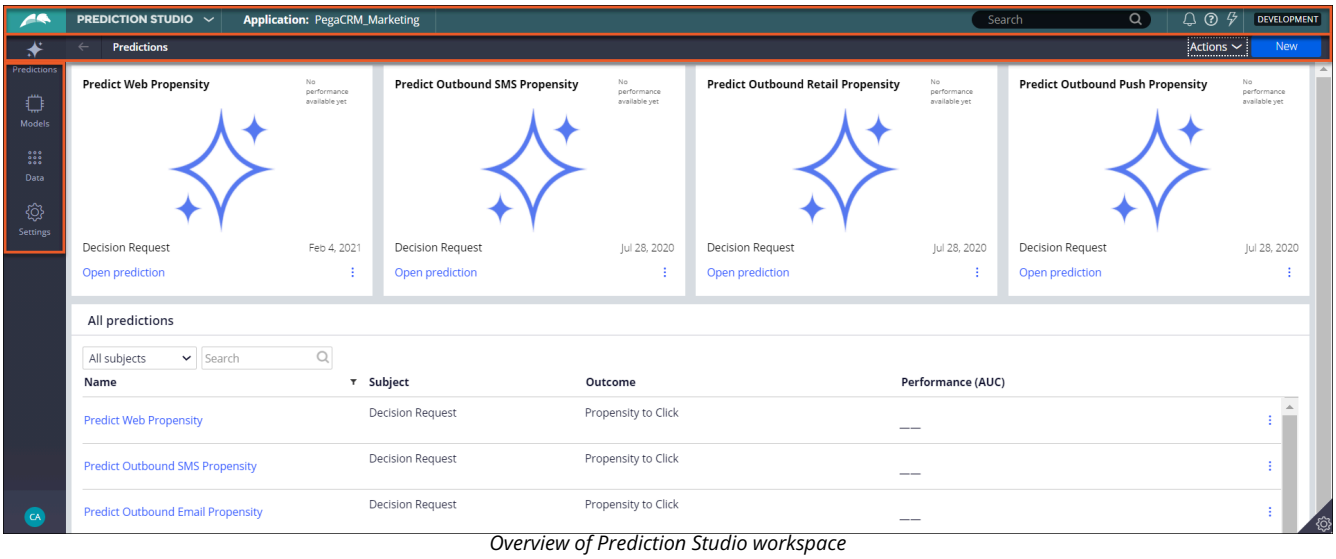
In this work area, you can access, sort, and manage predictive, adaptive, and text analytics models within your application. By default, the models are displayed as tiles. Each model tile contains an icon for quick identification of model type, the model name, and the indication of whether the model is completed or being built.

Data

In this work area, you can create and manage data sets or Interaction History summaries. In addition, you can access resources such as taxonomies or sentiment lexicons that provide features for building machine learning models.

Settings

This work area contains the global settings for model development, such as the internal database where model records and related resources are stored, their default application context, and model transparency policy.



Overview of Prediction Studio workspace

Prediction Studio transitions

You can quickly transition between Prediction Studio and business portals such as Pega Customer Decision Hub. When in Prediction Studio, you can click the **Back** widget to return to a business portal with no context change.

Developing and managing models *(on page)*

Changing your workspace

For the complete and multidimensional development of your application, switch from one workspace to another to change the tools and features that are available in your work environment. For example, you can create resources such as job schedulers in Dev Studio, and then manage and monitor those resources in Admin Studio.

1. In the header of your workspace, click the **Switch Studio** menu.
   The label for this menu is the name of your current workspace.
2. Select the workspace in which you want to work.

You have access only to the workspaces that are relevant to your role. For more information, see Granting portal access to an access group *(on page 68)*.
You have access only to the workspaces that are relevant to your role. For more information, contact your security administrator.
You have access only to the workspaces that are relevant to your role. For more information, review your access groups in Dev Studio.
Prediction Studio is a role-based workspace for data scientists. You have access only to the workspaces that are relevant to your role. For more information, contact your system administrator.

Adding personas to organize users *(on page)*

Exploring application-editing mode

Explore application-editing mode to learn about the tools and information that support application development.

Application-editing mode is enabled by default.

1. If you have turned off application-editing mode, turn it on.
    a. In the footer of App Studio, click the **Open runtime toolbar** icon.
    b. Click **Turn editing on**.
2. In the header of App Studio, click the **Resources** icon.
3. Click **Take a tour**.
4. Follow the prompts on the screen to discover the different tools that you can use to develop your application.

Building your first application *(on page 16)*

Building your first application

Create low-code applications in a user-friendly and intuitive way by working with Pega Platform. With various out-of-the-box tools, you can define customer journeys, develop applications, track your progress, and communicate with team members and stakeholders.

To ensure that your application is secure, use the Application Security Checklist. For more information, see Assessing your application using the Security Checklist *(on page)*.

Start building your application by performing the following tasks:

Automating work by creating case types *(on page)*
Theme Cosmos *(on page)*

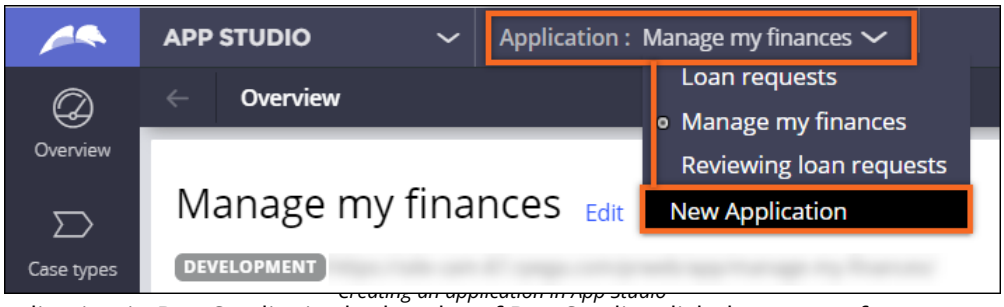
Security Checklist *(on page)*
Low-Code App Builder

Creating applications

Start developing your projects conveniently and intuitively by building your application on an application template. When you create an application on a template, you save time because you reuse key elements of an existing application, such as case types or data types.You can choose a default application template that Pega Platform provides or reuse one of your existing applications as a template. For example, you can create an application to review mortgage requests, and then supply the application with necessary case types, data types, and other elements. Then, you can build an application to review loan requests by using the application to review mortgage requests as a template. As a result, all assets from the application to review mortgage requests are automatically available in the new application to review loan requests.

For relevant training materials, see the Creating Pega Platform applications module on Pega Academy.

1. Navigate to the New Application wizard:
   - To build an application in App Studio, in the header of App Studio, click the name of your application, and then click **New Application**, as shown in the following figure:



Creating an application in App Studio

   - To build an application in Dev Studio, in the header of Dev Studio, click the name of your application, and then click **New Application**.
2. Choose a type of application to create:
   - To build an application that joins traditional UI authoring with a unique Theme Cosmos run-time experience, click **Build from scratch**, and then click **Theme Cosmos**.

     With a Theme Cosmos authoring experience you can save time by implementing a vast selection of auto-generated functionalities. You can also focus on low-code tools and solutions as Theme Cosmos is primarily designed for App Studio. For more information, see Theme Cosmos *(on page)*.
   - To build an application that uses Cosmos React UI authoring and run-time experience, and is also client-side rendered, click **Build from scratch**, and then select Cosmos React.

Cosmos React is an innovative way of designing applications that currently supports a limited range of case management functionalities. For more information, see Cosmos React *(on page)*.

- To build your application on a custom application template, hover over the application name, click **Continue**, and then click **Build with** *application template name*.

  For more information, see the implementation guide for your application.

- To search for an application template, click **Search all types**.

3. If you build your application on a custom template, and you want to reuse resources from the application, such as case types or data types, import the resources to your new application:
   a. In the **Select case types** section, select the check box next to each relevant case type.
   b. Click **Continue**.
   c. In the **Select data types** section, select the check box next to each relevant data type.
   d. Click **Continue**.
4. In the **Name your application** field, enter a unique name for your application.
5. **Optional:** To configure a custom class structure, organization, or select the base language for your application, click **Advanced configuration**.
   For more information about the settings that you can change, see Configuring advanced settings for new applications *(on page 19)*.
6. Click **Create application**.
7. **Optional:** To develop your application more quickly, in the **Optionally, add users** section, create a team:

   a. In the text field, press the Down arrow key, and then select a user name or an email address. If you enter an email address that is new in the system, the system creates a new user.

   b. Control which type of access the user has to your application by selecting a role in the list.

   c. Click **Add**.

   ⚠ **CAUTION:** If an email account error occurs when you add a user, take note of the user ID and generated password, because the user may not receive these credentials in an email.

8. Click **Go to app**.

Cosmos React *(on page)*
Configuring advanced application settings *(on page 30)*

Styling your application with design systems *(on page)*
Configuring applications for reuse *(on page 33)*
App Studio overview *(on page 8)*

Inviting collaborators to your application

Enhance your application and begin processing your business cases by inviting collaborators with different skills and roles.When you invite collaborators, you define the type of access that you grant to the users of your application. Your application illustrates different user and access types by personas. You can select from personas that reflect run-time users, and developer roles that correspond with design-time users. For example, you can invite members of a development team and associate them with developer roles, so that the team can start working on your application. You can also invite run-time users to process work in your application and, for example, associate them with a customer service representative (CSR) persona.

1. In the navigation pane of App Studio, click **Users**, and then click **User management**.
2. On the **People** tab, in the working area, click **Invite people to your application**.
3. In the **Add users to application** window, in the autocomplete field, press the Down arrow key, and then select a user name or email address.

   > If you enter an email address that does not exist as an operator in the system, the system creates a new user.

4. In the list of personas, define the type of access for the user of your application, by selecting a role:
   - To invite run-time users, select an option from the **Personas** section of the list.
   - To invite design-time users, select an option from the **Developer roles** section of the list.
5. Click **Add**.
6. Send an email with an invitation to the user by clicking **Send invitation**.
7. To invite more users, repeat steps 3 *(on page 19)* through 6 *(on page 19)*.
8. Close the **Add users to application** dialog box.

Creating a Microjourney for customer success *(on page)*
Adding personas to organize users *(on page)*
Creating personas *(on page)*
Creating a team *(on page)*
Creating applications *(on page 17)*
Managing work across your team *(on page)*

Configuring advanced settings for new applications

To fully benefit from possibilities Pega layer cake provides, configure advanced settings for your application and create organization layers. To save time, reuse common processes within your organization, and to

adjust the application to your departmental business needs, allow for greater customization of certain elements.

You can create division and unit layers in your application to customize different layers. For example, if your organization includes a Human Resources division and an Engineering division, configure your application to reuse common processes between the divisions, and change elements that are different for each division. For instance, reuse a process for calculating expenses, and specify different maximum amounts for each division.

1. In the **Application structure** section, select a structure for your application:
    - To create a generic application that you can reuse or extend later, select **Framework**.
    - To create a specialized application for a specific business audience, select **Implementation**.
2. In the **Application ID** field, enter an identifier for the application, for example, `HiringApp`.
   An identifier starts with a letter, and contains only letters, numbers, and hyphens. If you set an application name, the system generates an identifier based on the name.
3. In the **Version** field, enter a number that identifies the version of this application record, for example, `01.01.01`.
4. **Optional:** To create localization rules in the language in which you build the application, in the **Base language** field, select the default language for your application.

   > **Note:** This option is available only for applications that you build on Cosmos React.

5. In the **Organization settings** section, in the **Organization name** field, specify an organization that owns the application:
    - To reuse an existing organization, press the Down arrow key, and then select an organization.
    - To create an organization, enter a unique and descriptive name, for example, `HROrg`.

   The default value is an organization of the user that creates the application.
6. In the **Division name** field, specify a division that owns the application:
    - To reuse an existing division, press the Down arrow key, and then select a division.
    - To create a new division, enter a unique name.
7. In the **Unit** field, specify a unit in the division:
    - To reuse an existing unit, press the Down arrow key, and then select a unit.
    - To create a new unit, enter a descriptive name.
8. In the **Class layers** section, configure the class structure of your application:

a. **Optional:** To create a new division layer, select the **Generate division layer** check box, and then, in the **Division** field, specify a name of the division from the new layer.

b. **Optional:** To create a new class layer, select the **Generate class layer** check box, and then, in the **Class** field, specify the class for the new layer.

c. If you create a new organization, in the **Organization** field, enter the organization name. This option is read-only when you choose an organization that already exists.

9. In the **Application** field, enter a value that defines the application class layer for the application stack.
   If you leave this field blank, the application uses the value that you provide in the **Application ID** field.
10. In the **Class group name** field, enter a name that an application uses to create work queues for users.
    The default value is `Work`.
11. Click **Save**.

Creating rules *(on page)*
Organizing rules into rulesets *(on page)*
Organizing rules into classes *(on page)*
Rule resolution *(on page)*

Switching between applications

When you work on multiple applications, you can switch between them by using the **My Applications** gadget. By default, the **My Applications** gadget is available in App Studio.

1. In the header of App Studio, click **My Applications**.
2. From the list, select an application that you want to open.
3. **Optional:** If you have more than one access role in the application, select the role that you want to use, and then click **Go**.

   You can switch between applications without reentering login credentials if the applications share a common authentication service. However, if you are using two Pega Platform applications without a common authentication service or where the user is not authenticated with a common authentication service, and you try to switch applications, a dialog will launch with the message: `Switch application with different authentication service`. When you click the Login button, a new tab will open for you to reauthenticate using the default authentication service for that application.

   For more information, see Mapping authentication services in App Studio *(on page)*.

Changing your workspace *(on page 15)*

Adding the My Applications gadget

To conveniently navigate between your applications, add the *My Applications* gadget to your existing portal or application by embedding a header and a menu. By using the *My Applications* gadget, you can see an overview of your applications with short descriptions, customize the icons of your applications, and search the applications by name. By default, the *My Applications* gadget is available in App Studio.

1. In Dev Studio, open an application in which you want to add the **My Applications** gadget. For more information, see Switching between applications in Dev Studio *(on page 22)*.
2. In the navigation panel, click **Records > User Interface > Section**, and then open the **pyPortalHeader** section.
3. Right-click the embedded **pyAppLauncher** section, and then click **Copy**.
4. Open the section where you want to add the **My Applications** gadget.
5. Right-click the section, and then click **Paste**.
6. Open the **pyPortalHeader** section.
7. Repeat steps 3 *(on page 22)* through 5 *(on page 22)* for the **My Applications** icon.
8. Click **Save**.

Harness and section forms - Adding a section *(on page)*

Switching between applications in Dev Studio

When you work on multiple applications, you can switch between them by using the **Application** menu.

1. In the header of Dev Studio, click the name of the application, and then click **Switch Application > [***Application name***]**.
2. In the alphabetized list, select an application that you want to use as your current application.

> **Note:** The list of available applications is based on the access groups that are defined on your **Operator ID** form. If more than one access group points to the same application, the **Switch Application** menu displays the application label followed by the access group in parenthesis.

You can switch between applications without reentering login credentials if the applications share a common authentication service. However, if you are using two Pega Platform applications without a common authentication service or where the user is not authenticated with a common authentication service, and you try to switch applications, a dialog will launch with the message:

`Switch application with different authentication service`. When you click the Login button, a new tab will open for you to reauthenticate using the default authentication service for that application.

Exploring the application definition

Analyze and edit basic information about your application by exploring the application definition. In the application definition, you can edit what built-on applications your software uses to source elements, manage branched development, and add components to extend the functionality in your application.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. On the **Definition** tab, explore and edit the application definition to meet your business needs:
    - To reuse elements and assets from other applications, such as rules or features, explore the **Built on applications** section.

      For more information, see Adding built-on applications *(on page 24)*.
    - To expand the functionality of your application by adding components, explore the **Enabled components** section.

      Components represent reusable features, such as a tool to capture multimedia within an application. You can create your own component or you can use ready-made components that are available on Pega Marketplace.

      For more information, see Installing components *(on page 86)*.
    - To define the look and feel of your application by applying a skin, explore the **Presentation** section.

      For more information, see Specifying a skin for your application *(on page)*.
    - To define a URL to launch your application, explore the **Application URL alias** section.

      For more information, see Adding an application URL alias *(on page)*.
    - To define advanced configurations for your application, explore the **Advanced** section.

      For more information, see Configuring advanced application settings *(on page 30)*.
    - To provide separate sandboxes for application developers to avoid conflicts and overrides, explore the **Development branches** section.

      When you create branches, developers can build and modify rules without interfering with the work of another team that works on the same application.

      For more information, see Branches and branch rulesets *(on page 324)*.

- To enrich your application with a specific element, such as assets that are associated with a specific case type or access group, explore the **Application rulesets** section.

  For more information, see Organizing rules into rulesets *(on page)*.

Designing applications for reuse and extension *(on page 74)*
Basic requirements for deploying public-facing applications *(on page 40)*

Adding built-on applications

To save time and reduce development costs you can reuse elements between your applications, such as rules and features, by adding built-on applications. When you build your application stack, you ensure that you meet business requirements in an efficient way.For example, when you create an application to review mortgage requests, you can add a built-on application that workers use to review loan requests, and then reuse the elements that are relevant to the mortgage cases.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. In the **Built on applications** section, click **Add application**.
3. In the **Name** field, enter the name of the application that you want to add.
4. In the **Version** field, enter the version of the application that you want to add.
   As a best practice, add the latest version of your application.
5. To add more built-on applications, repeat steps 2 *(on page 24)* through 4 *(on page 24)*.
6. Click **Save**.

> When you work on your current application, you can reuse resources from your built-on applications.

Related information

Designing applications for reuse and extension *(on page 74)*
Organizing rules into rulesets *(on page)*
Organizing rules into classes *(on page)*
Rule resolution *(on page)*
Adopting feature-driven development *(on page)*

Best practices for using multiple built-on applications

To increase reuse between your Pega Platform™ applications and speed up the development process, enhance your applications by using multiple built-on applications. Built-on applications are structured as a hierarchical application tree.

By using multiple built-on applications, you break up common and stand-alone components into their own applications, and provide the ability to point an application to another application rule, helping to automatically update any required rulesets and versions across applications when applications undergo changes.

For example, the uPlus organization creates an application layer to contain the definition of common assets and integrations in corporate systems. Separating each integration into its own built-on application can simplify the revisioning of each integration by removing dependencies. As a result, applications use features independently and when developers want to update a feature, they can update only a specific built-on application, without affecting the others.

Best practices for using multiple built-on applications

Consider the following tips when developing an application that uses multiple built-on applications:

- Limit development to rulesets in the top-most application and keep built-on applications locked.
- To develop in the lower layers of the hierarchical application tree, switch to a built-on application layer.
- Avoid having the same ruleset in multiple applications. Instead, refactor the ruleset to its own application or a common application.
- Avoid using different versions of an application in the hierarchical application tree.
- Avoid branch IDs that span several applications.
- Avoid making any application structure changes without utilizing the Validation Tool to validate rules.
- Avoid using Ruleset Validation mode rulesets. Ruleset prerequisites can be challenging to maintain, especially when you need to refactor the application stack. Always default to using Application Validation mode rulesets, which create the ruleset stack depending on the application in which you define the rulesets.

Built-on applications and the Pega Process Fabric Hub

The Pega Process Fabric Hub is an application that gathers assignments from multiple distributed remote applications and provides a unified user interface to manage work that comes from multiple other sources. The Pega Process Fabric Hub and built-on applications are different solutions that fit different scenarios. When you need to compare scenarios that support multiple built-on applications or the Pega Process Fabric Hub, consider the following factors:

Built-on applications	The Pega Process Fabric Hub
You can deploy all built-on applications as a single stack.	Different stakeholders can manage different remote applications that publish work to the Pega

Built-on applications	The Pega Process Fabric Hub
	Process Fabric Hub, and you can connect applications that you host on different systems.
The life cycle of built-on applications is synchronized; for example, you deploy all applications simultaneously to a new environment.	Each remote application has a separate life cycle that you manage without interfering with other applications.
Maintenance and update of all built-on applications are simultaneous and smooth without contradicting the behavior of different applications.	Changes that you deploy in one remote application do not affect other applications that publish work to the Pega Process Fabric Hub.
Built-on applications provide the capacity for customizations of selected applications without negative interference to other applications.	The Pega Process Fabric Hub offers out-of-the-box functionalities without a need for any customizations, which helps you begin processing work faster.
All applications in the built-on applications stack are Pega Platform applications.	You can connect both Pega and non-Pega applications to the Pega Process Fabric Hub.
By implementing built-on applications, you can reuse different features between applications.	The Pega Process Fabric Hub focuses on processing assignments that different applications publish. Functionalities such as service-level agreements or notifications are specific to each remote application and are independent of the Pega Process Fabric Hub.

For more information on managing work from multiple applications in a unified interface, see Pega Process Fabric Hub *(on page)*.

Creating a unified view of work items in built-on applications and in the Pega Process Fabric Hub

You can use both built-on applications and the Pega Process Fabric Hub to create a unified view that summarizes assignments from multiple applications. The solution that you select depends on your scenario. Consider the following factors:

- Use built-on applications when you host your applications on one server without any need for external connections, and all your applications are based on the same version of Pega Platform.
- Use the Pega Process Fabric Hub when you gather work from applications that you host on different servers or when you use applications that are based on different versions of Pega Platform. You can also create connections between the Pega Process Fabric Hub and non-Pega applications.

Limits for applications in a built-on applications stack

The number of applications in a built-on applications stack depends on your scenario. If updating or modifying any of the applications in the stack might result in severe issues and extensive effort to maintain and adjust other applications, consider limiting your application stack. When you add an application as a built-on application, ensure that the benefits of reuse outweigh any risks connected with maintainability.

Related information

Application stack hierarchy for multiple built-on applications *(on page 27)*
Adding built-on applications *(on page 24)*
Validation tool *(on page 401)*
Configuring ruleset version settings *(on page)*

Application stack hierarchy for multiple built-on applications

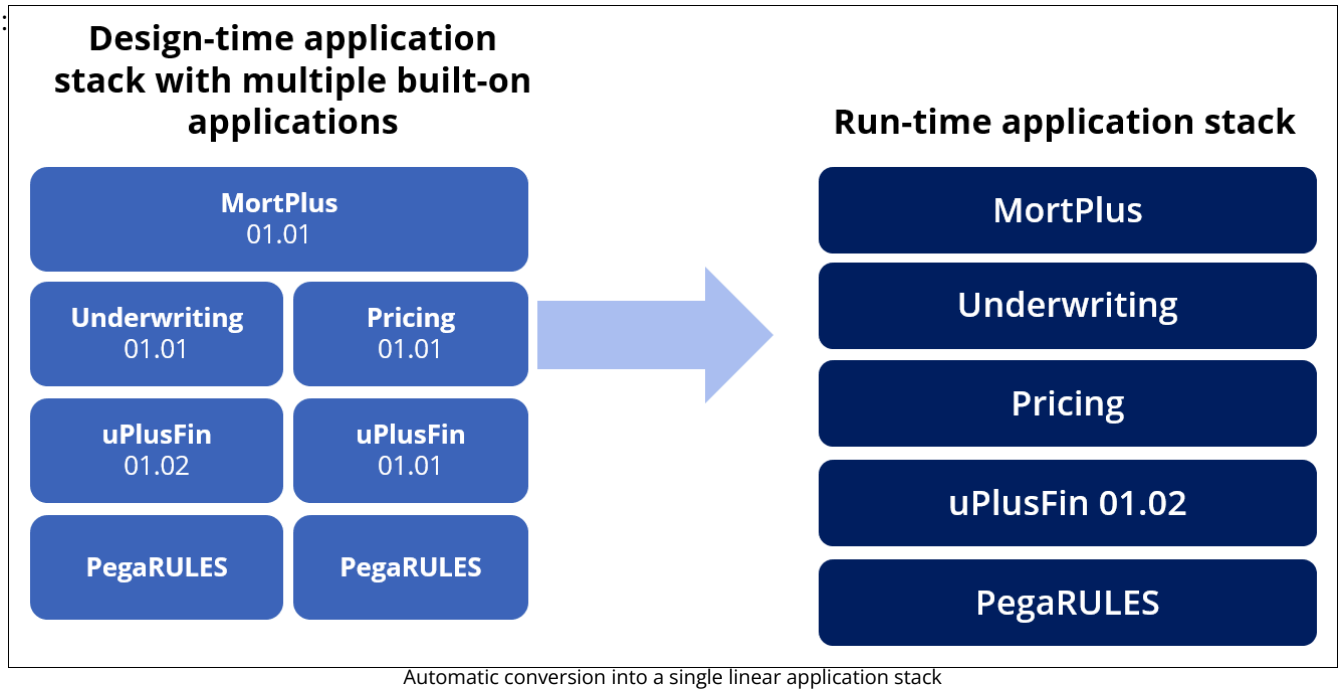
When you use multiple built-on applications, Pega Platform automatically converts the hierarchical application tree of multiple built-on applications into a single linear application stack. As a result, you can shorten development time and increase consistency in your application by reusing rulesets and frameworks.

Pega Platform creates the linear application stack using the following rules:

- The system always locates PegaRULES at the end of the linear application stack list.
- Pega Platform processes each built-on application by using a depth-first algorithm that searches as far as possible along an application tree branch before moving on to another branch.

  An application tree branch consists of a built-on application and its built-on applications.
- For duplicate applications with the same application version, Pega Platform retains only the application that has the lowest position in the tree in the application stack list, and ignores the other application.
- For duplicate applications with different application versions, as well as for PegaRULES, Pega Platform uses the numerically highest available application version.

The following image represents the automatic conversion of a hierarchical application tree into a single linear application stack at run time:



Automatic conversion into a single linear application stack

Ruleset behavior at design time

Each application in the hierarchical application tree consists of various rulesets that Pega Platform automatically assembles into a linear ruleset stack. This ruleset stack defines which rulesets a specific rule can use for context purposes at design time.

Ruleset stack assembly generally follows the same guidelines as for creating the linear application stack at run time. However, ruleset stack assembly for a specific rule is based on the specific application that the rule belongs to, which is also called the rule's owning application.

After Pega Platform determines a rule's owning application, Pega Platform automatically creates the linear ruleset stack for that specific rule, starting with the rule's owning application and working through the hierarchical application tree branch to PegaRULES.

Note: The behavior described above applies only when you save a rule in an Application Validation mode ruleset. When saving a rule in a Ruleset Validation mode ruleset, you must specify prerequisites in the **REQUIRED RULESETS AND VERSIONS** section on the **Versions** tab of the ruleset.

When you use Application Validation mode, at design time, the saved rule cannot reference certain rulesets that are available at run time. This restriction allows for the appropriate isolation of applications, and makes applications easier to reuse.

Example of the transition from hierarchical to linear ruleset stack assembly

In the following example, the hierarchical application tree contains only rulesets that use the Application Validation mode:

MortPlus 01.01 (*Ruleset1*)	
Underwriting 01.01 (*Ruleset2*)	Pricing 01.01 (*Ruleset4*)
uPlusFin 01.02 (*Ruleset3*)	uPlusFin 01.01 (*Ruleset5*)
PegaRULES	PegaRULES

If you save *RuleABC* in *Ruleset2*, the Underwriting application becomes the rule's owning application. Pega Platform then assembles the ruleset stack that *RuleABC* can use at design time according to the following pattern:

Ruleset2 **>** *Ruleset3* **>** *PegaRULES*

Pega Platform starts building the ruleset stack at the Underwriting application, and then processes the stack through PegaRULES in that specific application tree branch. Because the Pricing application is located on a separate application tree branch, at design time, ruleset stack processing cannot reference *Ruleset4* and *Ruleset5* from the other tree branch.

Ruleset behavior at run time

At run time, Pega Platform creates the linear application stack for the entire tree of built-on applications by following the preceding guidelines. This application stack is the basis for application functionality in a live environment.

Use case examples

The following use cases describe how Pega Platform processes the hierarchical application tree into a single linear application stack at run time.

> **Note:** These examples are for demonstration purposes only and might not represent the application structure of current application offerings.

Related information

Best practices for using multiple built-on applications *(on page 24)*

Adding built-on applications *(on page 24)*

Configuring advanced application settings

Incorporate advanced options into your application to ensure that the software that you develop meets your unique business requirements. For example, you can define how your application renders the UI, or enable your system to automatically manage background processes.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. On the **Definition** tab, expand the **Advanced** section.
3. Modify the advanced application settings to meet your needs:
    - To allow users to modify rules in a production environment, click **Add production ruleset**, and then in the **Production rulesets (customization)** field, enter the ruleset that includes the rules that you want to make available for users to edit.

      Ensure that you unlock at least one ruleset version in the production ruleset.

      For more information, see Production rulesets *(on page)*.
    - To configure your system to automatically manage the resolution of job scheduler and queue processor rules in an application without any manual configurations, select the **Include in background processing** check box.

      For more information, see Managing system resources *(on page)*.
    - To define how your application renders the UI at run time, in the **UI Runtime** section, define whether you want your application to use standard server-rendered UI or Cosmos React.

      For more information, see Cosmos React *(on page)*.
    - To maintain properties on both the *pxRequestor* and *pxThread* pages, clear the **Place properties on thread page only (5-4 or later)** check box.

      By default, your application maintains the properties on the *pxThread* page only. Change this setting only for backward compatibility with applications built on Pega Platform version 5.3 or earlier. In any other scenario, keep the default setting.
    - To define a screen that opens after a user logs out of the application, in the **Log off redirection** list, select the screen that you want to use.
4. Click **Save**.

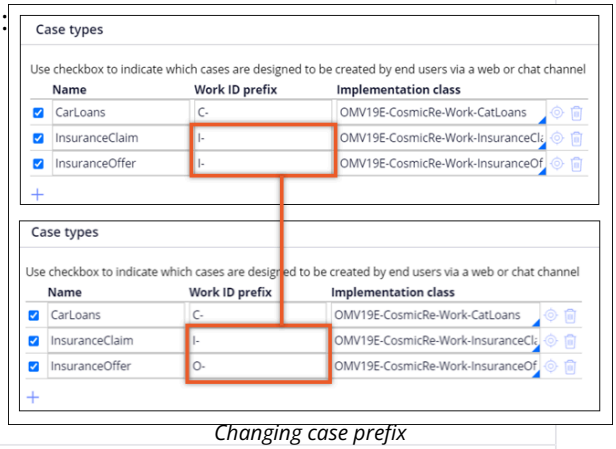
Basic requirements for deploying public-facing applications *(on page 40)*
Clipboard tool *(on page 373)*
Managing job schedulers *(on page)*
Managing queue processors *(on page)*
Organizing rules into rulesets *(on page)*
Theme Cosmos *(on page)*

Managing application cases and data

Explore the application definition to gain a holistic view of data in your application and cases that are available for creation at run time. As a result, you can globally manage resources in your application from one place.For example, you can select which cases users can create at run time and disable the cases that are not required for your business scenarios. Consequently, at run time, users have a more relevant choice of cases to create. To speed up application development, you can also provide a more relevant Data Explorer by adding necessary data types and removing data types that are not required.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. Click the **Cases & data** tab, and then manage any of the relevant settings for your application:

Choices	Actions
Enable creation of selected cases at run time	In the **Case types** section, select the check box in the row of the case type that you want to enable for run-time creation.
Change the prefix of run-time case instances of a case type	In the **Case types** section, in the **Work ID prefix** field of the selected case type, enter a new value. By default, the prefix is the first letter of the name of a case type. If you have multiple case types that start with the same letter, you can change prefixes so that users can conveniently distinguish cases at run time. The following figure shows an example of how changing the prefix can help

Choices	Actions
	you differentiate between cases at run time:  *Changing case prefix*
Change the scope of resources that are available for a case type	In the **Case types** section, in the **Implementation class** field, enter a new class to store the case type. For more information, see Understanding class layers *(on page)*.
Manage data types in your Data Explorer	a. In the **Data** section, in the text field, enter the class of the data type that you want to add to your Data Explorer, and then click **Add**. b. **Optional:** To remove a data type from the Data Explorer, click **Delete** in a row of the data type.
Change how your application stores information by associating new classes with main application elements	a. In the **Associated classes** section, in the **UI class** field, provide the default class for UI rules. b. In the **Associated classes** section, in the **Integration class** field, provide the default class for integration-related rules. c. In the **Associated classes** section, in the **Data class** field, provide the default class for data-related rules.

3. Click **Save**.

Configuring applications for reuse

To save time and immediately start application development, make your application available as a template. When you save your application as a template, other users can quickly build a new application from that sample. The newly created application automatically inherits template logic, data, and personas.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. On the **Application wizard** tab, select the **Show this application in the New Application wizard** check box.
3. In the **New Application wizard settings** section, in the **Short description** field, provide the name and purpose of your application:
    - To reuse existing information, press the Down arrow key, and then select a field value that stores the information that you need.
    - To use a new name, enter a unique and descriptive label, for example, `Hiring process`.
4. In the **Application description** field, provide a description so that users can decide whether to reuse your application:
    - To reuse existing information, press the Down arrow key, and then select a field value that stores the information that you need.
    - To use a new description, enter a unique and descriptive text, for example, `Use this application to process job applications`.
5. In the **New application access** list, select the access group that users receive when they reuse your application.
   You can only choose from development access groups, which are access groups that contain the developer or *pxExpress* portals.
6. **Optional:** To distinguish your application when users select reusable templates, replace the default logo and preview images:
    a. In the **Brand** section, click **Upload Image**.
    b. In the explorer dialog box, navigate to the file for your company or organization logo, and then click **Upload Image**.
    c. In the **Desktop snapshot** section, click **Upload Image**.
    d. Navigate to a screen capture that is at least 370 pixels by 233 pixels, and then click **Upload Image**.

e. In the **Mobile snapshot** section, click **Upload Image**.
f. Navigate to a screen capture that is at least 110 pixels by 194 pixels, and then click **Upload Image**.

7. Click **Save**.

> When users run the new application wizard to create an application, they can choose your application as a template and reuse its logic, data, and personas.

Creating applications *(on page 17)*
Creating a Microjourney for customer success *(on page)*
Adding built-on applications *(on page 24)*

Setting your application password

Improve your application security by setting an application password, which limits the number of users who can make changes to your application. For extra security, you can periodically update this password based on your security framework and security plan.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. Click the **Security** tab, and then in the **Application security** section, select the **Require password to update application** check box.
3. Click **Update password**.
4. In the **Update password** dialog box, enter the new password in the **Enter password** and **Confirm password** fields.

> **Note:** To successfully update your application password, you must enter identical values in both fields.

5. Click **Submit**.
6. Click **Save**.

> Users need to provide the password before they can save their changes to the application.

Basic requirements for deploying public-facing applications *(on page 40)*

Security tab of the Application Definition

Use this tab to define security settings in your application, map authentication services, and enable content, mashups, and digital messaging security.

Application Definition

To view the Application Definition, in the header ofDev Studio, click your application name, and then click **Definition**.

Security Tab

The Application Definition contains seven tabs, including the Security tab. To view the **Security** tab, in the header of Dev Studio, click your application name, and then click **Definition > Security**.

The **Security** tab contains the following sections: **Application security**, **Authentication**, **Content security**, **Mashup security** and **Digital Messaging security**.

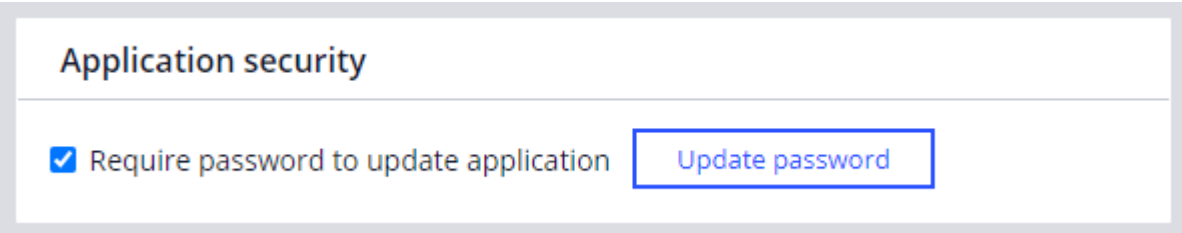
Application security

Application security has one setting: the **Require password to update application** check box, which should be selected when creating an application. The **Require password to update application** check box should remain selected.

Select the **Require password to update application** check box if you want to change or update the password that users must enter when updating the application, and then click **Update password** to set the password.

For more information, see Setting your application password *(on page 34)*.

The following figure shows the **Application security** section:



The application security section of the Security tab of the Application definition

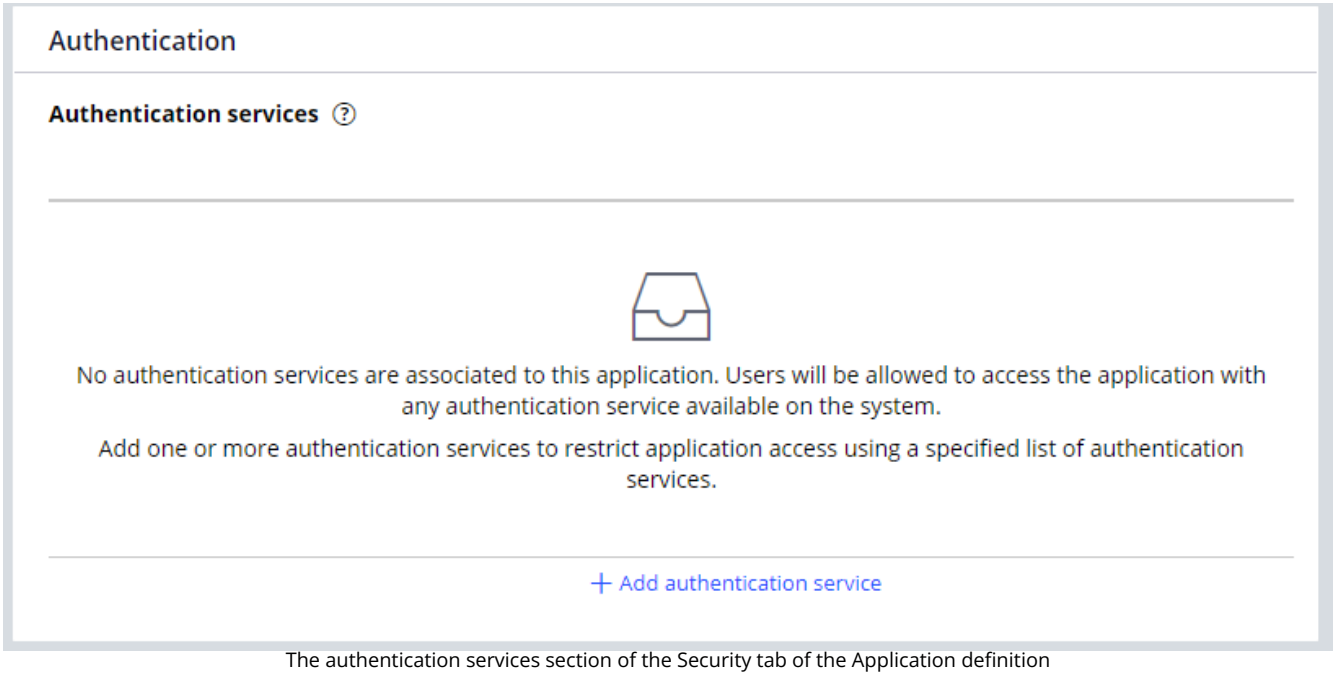
Authentication

Authentication services are new to Pega Platform. Users with the *pzAdvancedSecurityUser* privilege can map authentication services to an application. By default, the *PegaRULES:SecurityAdministrator* includes the *pzAdvancedSecurityUser* privilege.

By mapping an authentication service, a security administrator can define the authentication service mechanism that users use to log in to an application. Administrators can also create a default login mechanism and interactive screen the user selects an authentication service to login.

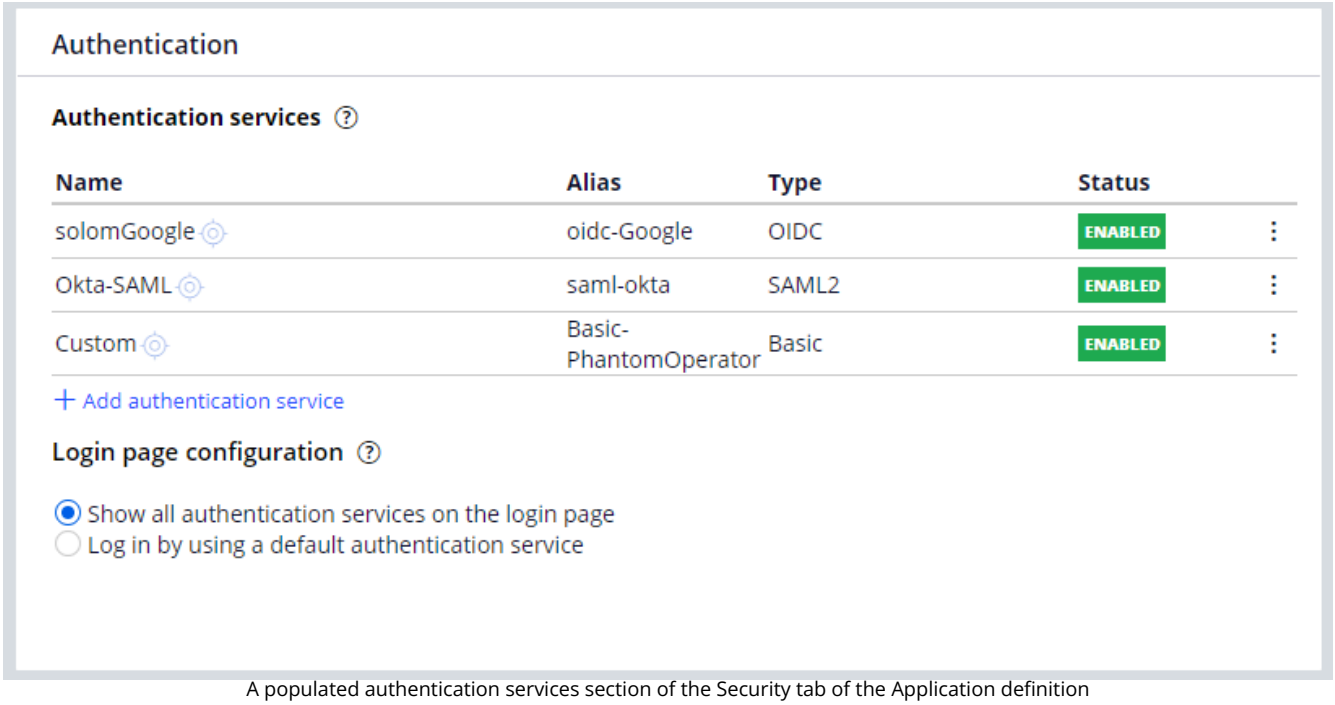
The Authentication section changes depending on whether you have already mapped authentication services.

The following figure shows the Authentication section for a system that does not have any mapped authentication services:



The authentication services section of the Security tab of the Application definition

The following figure shows the Authentication section for a system that already has mapped authentication services:



A populated authentication services section of the Security tab of the Application definition

For more information, see Mapping authentication services in Dev Studio *(on page)*.

You can also map authentication services in App Studio, without using the Application Definition. For more information, see Mapping authentication services in App Studio *(on page)*.

Content security

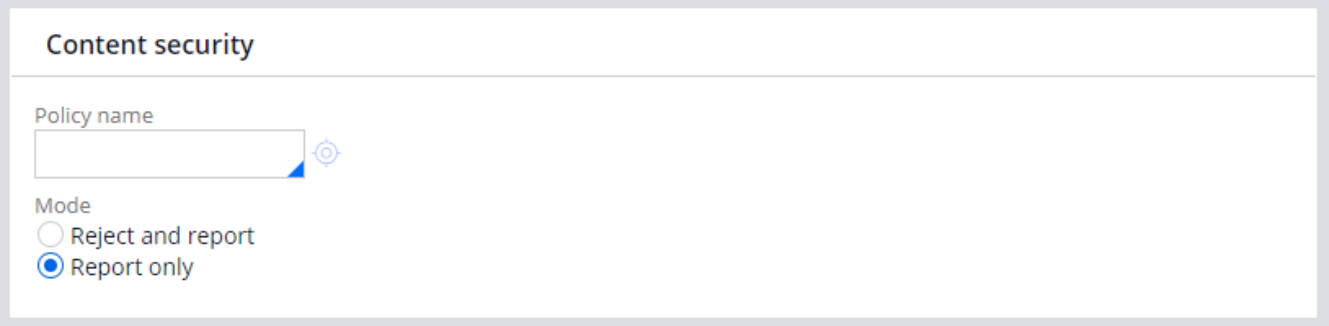
The content security policy (CSP) is a set of directives that inform the user's browser of locations from which an application is allowed to load resources, such as fonts, images, and style sheets.

Use the **Policy name** field to select the content security policy. You can configure this content security policy directly from this page by clicking the configure icon after you select the policy.

Use the **Mode** setting to specify what the browser does when a policy is violated:

Mode	System behavior
Reject and Report	Enforce the policy and report the violation. **Note:** The content source must be in the correct list.
Report Only	Report the violation, but do not enforce the policy.

The following figure shows the **Content security** section:



The content security section of the Security tab of the Application definition

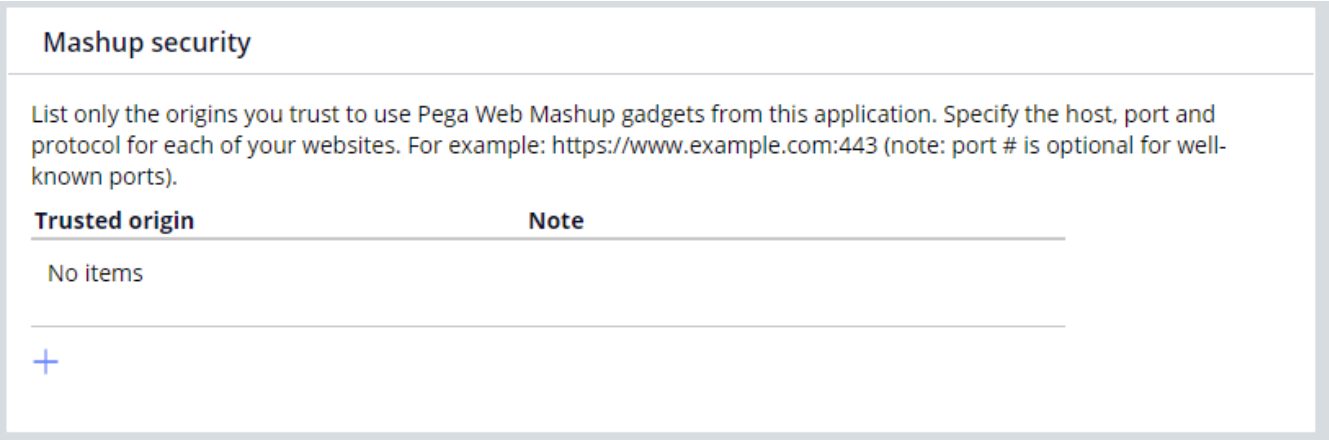
For more information, see:

- Content security policies *(on page)*
- Creating a content security policy *(on page)*

Mashup security

Mashup security is used to define the external URLs that are allowed to access Pega Platform so that the host page can communicate with the mashup gadget, if you use the mashup feature to embed Pega Platform content in an external application. This feature functions, in part, as an allow list.

The following figure shows the **Mashup security** section:



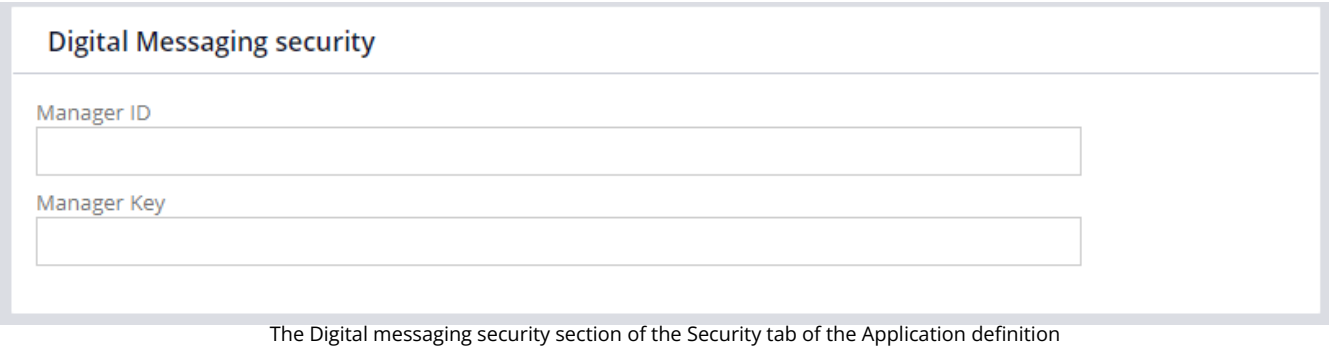
The mashup security section of the Security tab of the Application definition

For more information, see Securing your application for mashup communication *(on page)*.

Digital Messaging security

This security feature is used to set up messaging platforms for Pega Intelligent Virtual Assistant (IVA) for Digital Messaging. To start using Digital Messaging, you must configure and define the security settings for the channel and the system. Examples of messaging platforms that are useable by the IVA include Apple Business Chat, Facebook Messenger, MMS/SMS (Twilio), and WhatsApp Messenger.

The following figure shows the **Digital Messaging security** section:



The Digital messaging security section of the Security tab of the Application definition

For more information, see Configuring Digital Messaging channel security *(on page)*.

Specifying the content of application documentation

Provide guidance and methodology information to developers and other stakeholders by defining which information the application documentation includes. As a result, you can provide relevant details about your software and the methodology that your team implements.For example, you can define the business goals that you want your application to meet and provide additional relevant materials by adding attachments to your application overview.

Note: Defining the content of documentation in Dev Studio is suited to advanced developers or developers that use legacy document types. To implement low-code solutions, prepare application documentation in App Studio. For more information, see Creating project documents for stakeholders *(on page 326)*.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. Click the **Documentation** tab.
3. In the **Application guides** section, add guides that you want developers to follow during creating applications.
   For more information, see Creating a guide for application developers and administrators *(on page 330)* and Adding an application guide to a portal *(on page 336)*.
4. In the **Supporting specification types** section, describe the functionalities that you want to implement:
   a. In the **Name** field, enter a label for the specification.
   b. In the **Short description** field, provide brief information about the functionality.
   c. **Optional:** To add more specifications, click **Add specification**, and then repeat steps 4.a *(on page 39)* through 4.b *(on page 39)*.

Note: To take advantage of the latest Pega Platform tools, use features instead of specifications. For more information, see Adopting feature-driven development *(on page)* and Feature-driven development tracking *(on page)*.

5. In the **Business objectives** section, in the **Description** field, briefly describe a goal that you want to achieve in your application.

Tip: You can also provide business objectives in a low-code way in App Studio. For more information, see Editing application details in App Studio *(on page)*.

6. In the **Releases** section, click **Add item**, and then in the field, provide a new release version for your application.
7. **Optional:** To change the value of the release version that Pega Platform provides by default, click the field for the release, and then provide a value.
8. **Optional:** To provide a more detailed timeline of your application development, in the **Iterations** section, click **Add item**, and then provide an iteration for your application.
9. In the **Attachments** section, provide reference materials about your application development and planned design, by clicking **Add/Edit attachments**, and then attaching content:
    - To attach a file, in the **Attach files** section, drag a file or click **Select file(s)** and navigate to the file that you want to add, and then click **Submit**.
    - To attach a URL, in the **Enter a link** section, provide a URL and a short description of the URL content, and then click **Submit**.

    > The attached content is available in the **Overview** section of your application.

10. In the **Organization** section, specify the organizational owner for your application:
    a. Click **Update**.
    b. In the **Update Organizational Unit** dialog box, select the organization, division, and unit that own your application.
    c. Click **Submit**.

    The organizational structure impacts the routing and reporting in your application of, for example, notifications to managers.
11. In the **DCO settings** section, in the **Project Methodology** list, select a methodology that you want to implement during application development, for example, select **Scrum**.
12. Click **Save**.

Implementation methodologies *(on page)*
Estimating application development *(on page)*
Managing application inventory *(on page)*

Basic requirements for deploying public-facing applications

To ensure the security and branding of your public-facing applications, review and follow the minimum requirements for deployment through Pega Platform.

> **Note:** Do not deploy any public-facing applications that you develop in Pega Platform without following these basic requirements. If you cannot complete an item on this list, immediately notify the application owner of the security risks and of the omitted tasks.

The Security Checklist: Best practices for securely deploying your application

Security is critical in all applications, especially in applications that are used by external operators, such as customers and prospective customers, who are not your employees. Inadequate security might cause the deployment of your application to fail.

To secure your application, complete the Security Checklist to:

- Follow best practices for securely deploying applications.
- Ensure the confidentiality, integrity, and availability of your application in production.
- Identify when each task should be performed: at or near the beginning of development, on an ongoing basis, or just before deployment.
- Avoid expensive rework late in your development process.

Unless otherwise noted, these recommendations apply to all deployment environments, including Pega Cloud services.

The most important security requirement for all applications is to maintain guardrail-compliance because Pega Platform security features cannot always be successfully enforced in custom code. You can secure applications by configuring only the built-in features in Pega Platform, without relying on custom code that is built by developers who are not security experts.

For more information, see Security Checklist *(on page)*.

Pega Platform shows the overall completion of the Security Checklist on the Dev Studio home page, and provides tools to track the status of each task.

For more information, see Assessing your application using the Security Checklist *(on page)*.

Public users authentication

Unauthorized individuals should not have access to view or modify the application or its data. Always verify the identity of application users, and apply one of the following types of Authentication Services, based on the user type:

Anonymous user

A user is initially anonymous (identity unknown). The users may register themselves or have their identity verified partway through a session.

For example, in online shopping, unauthenticated users can browse and add items to a shopping cart, and then either create an account or enter their credentials to check out.

To configure authentication for applications that require anonymous users, you define the following parameters:

- An *Authentication Service* rule of the **Anonymous** type
- An access group for all anonymous users
- A limit for the access group to provide access to only data and applications functions that are not in any way confidential (such as the product catalog and shopping cart)

After a user becomes a known user, either by registering or entering credentials, you can use the Re-Authentication gadget to dynamically change their roles and privileges, so that the user can continue their session without logging in again.

For more information, see Configuring an application to use an anonymous authentication service *(on page)*.

Authenticated user

User credentials must be verified during login through an external data store or database, such as a customer master file.

User credentials are verified at the start of a session, but those credentials are stored outside of Pega Platform.

To configure authentication for applications with authenticated users, you define the following parameters:

- An*Authentication Service* rule of the **Basic Credentials** type
- The **Verify credentials using an external identity store** option
- A data page for accessing an external data store

Single sign-on

Single sign-on (SSO) is a property of identity and authentication that enables users to securely authenticate with a set of login credentials, such as a name and password. SSO is used and the user is authenticated through their credentials on a social media site, such as Google.

For more information, see:

- Creating an authentication service *(on page)*
- Creating a Google authentication service *(on page)*
- Authentication *(on page)*

Restrict access to data in *Data-Admin-OperatorID* class

In public-facing applications where end users do not need access to information about other operators, we recommend that you restrict all access to data in the *Data-Admin-OperatorID* class to only the end user's data through an access control policy. You can do this by enabling the out-of-the-box rules *pyDefault* and *pyRestrictToSelf* in the *Data-Admin-OperatorID* class.

If this rule does not exist in your Pega Platform release, you can create it in any release from 7.3 forward. For more information, see Restricting access to operator information in public-facing applications in Pega Platform 8.4 and earlier *(on page 45)*.

For more information, see:

- Attribute-based access control *(on page)*
- Creating an access control policy condition *(on page)*

Sensitive client data protection

Never deploy any application that creates or stores personally identifiable information (PII), such as social security numbers, account numbers, health data, date of birth, addresses, phone numbers, or any other sensitive data that could be used to identify an individual, without protecting that data from unauthorized access.

To encrypt sensitive property values, apply these guidelines:

- Set up a master key in an external Key Management System.
- Define a Keystore rule instance that references the master key.
- In Dev Studio, on the **Data Encryption** landing page, reference the Keystore rule that accesses the master key.
- Define PropertyEncrypt access control policies, and then list the properties to encrypt.
- Define PropertyRead access control policies to conditionally mask or obfuscate these property values for operators who should not be able to view them.

For more information, see:

- Encrypting system data by using a custom key management service *(on page)*
- Creating a keystore for application data encryption *(on page)*
- Creating an access control policy *(on page)*

Branding and exposed information during user logins

Set up a descriptive and memorable, or vanity, URL or custom domain name that does not unnecessarily expose server information to users.

For more information about configuring the domain name on Pega Cloud, see Requesting a custom domain name.

Restricting access to operator information in Pega 8.5

Restrict all access to data in the *Data-Admin-OperatorID* class to only the end user's data by using an access control policy. You can restrict access to personally identifiable information (PII) for security purposes, such as protection against unauthorized exposure of PII data. Restricting access to only the end users' data increases the security and peace of mind of users who must communicate with clients and customers through public-facing channels.

To enable the *pyDefault* and *pyRestrictToSelf* rules in the *Data-Admin-OperatorID* class, follow the steps below:

1. In the header of Dev Studio, click **Create > Security > Access Control Policy Condition**.
2. In the header of Dev Studio, click Records
3. Open the *pyRestrictToSelf* access control policy condition rule.
    a. In the **Policy Condition** field, select **pyRestrictToSelf**.
    b. In the **Applies To** field, select **Data-Admin-Operator-ID**.
4. Click **Save As**, then select the application ruleset for which you want to enforce this restriction, and click **Create and open**.
5. On the rule form, click **Availability**, and then select **Available** from the list.
6. In the header of Dev Studio, click **Records > Security > Access Policy Control**.
7. Open the *pyDefault* access control policy rule.
    a. In the **Policy name** field, select **pyDefault**.
    b. In the **Action** field, select **Read**.
    c. In the **Applies to** field, enter `Data-Admin-Operator-ID`.
8. Click **Save As**, then select the application ruleset for which you want to enforce this restriction, and click **Create and open**.
9. On the rule form, click **Availability**, and then select **Available** from the list.

Related information

Restricting access to operator information in public-facing applications in Pega Platform 8.4 and earlier *(on page 45)*

Configuring dynamic system settings *(on page)*

Access Control Policy rule *(on page)*

Restricting access to operator information in public-facing applications in Pega Platform 8.4 and earlier

Restrict all access to data in the *Data-Admin-OperatorID* class to only the end user's data by using an access control policy. Enable this access to personally identifiable information (PII) for security purposes, such as protection against unauthorized exposure of PII data. Restricting access to only end users' data increases the security and peace of mind of users who must communicate with clients and customers through public-facing channels.

If your installation of Pega Platform does not contain the *pyRestricttoSelf* rule, from Pega Platform 7.3 and later you can create it yourself.

1. In the header of Dev Studio, click **Create > Security > Access Control Policy Condition**.
2. Create an Access Control Policy Condition rule by clicking **Create > Security > Access Control Policy Condition**, and enter the following information:
    a. In the **Identifier** field, enter an identifying name.
    b. From the **Ruleset** list, select the application ruleset in which you want to enforce this restriction.
    c. In the **Apply To:** field, enter `Data-Admin-Operator-ID`.
3. On the **Pages & Classes** tab, enter the following information:
    a. In the **Page Name** field, enter `OperatorID`.
    b. In the **Class** field, enter `Data-Admin-Operator-ID`.
4. Click `Definition` and then enter the following conditions:
    a. In the **Conditional logic** section, enter a name for the condition.
    b. In the **Policy Conditions** section, in the **Condition** field, enter the same name that you provided in the **Conditional logic** field.
    c. In the **Column source** column, select **.pyUserIdentifer**.
    d. In the `Relationship` column, select **Is equal**.
    e. In the `Value` column, select **OperatorID.pyUserIdentifer**.
5. Click **Save**.
6. In the header of Dev Studio, click **Records > Security > Access Control Policy**.
7. Create an Access Control Policy rule with the following details:

a. In the **Identifier** field, enter a name for the rule.
b. In the **Action** field, select **Read**.
c. In the **Ruleset** field, enter any rulesets in the application for which you want to enforce this restriction.
d. In the **Applies To** field, enter `Data-Admin-Operator-ID`.

8. Click `Definition`, and then enter the name of the Access Control Policy condition rule that you create in Step 4 *(on page 45)* to the **Permit access if** field.
9. Save the rule form.

Related information

Restricting access to operator information in Pega 8.5 *(on page 44)*
Configuring dynamic system settings *(on page)*
Access Control Policy rule *(on page)*

Understanding project roles and personas

For the best user experience, define how users interact with your application by understanding what roles and personas your project requires. When you know who uses your application, and how, you can update the interactions to provide the most relevant content to users.

To enable customers to start using your application, define operators and access groups. By creating operators and access groups, you specify which elements of your application users can access. As a result, you ensure that every user can access only the features that are relevant to their role. You can increase efficiency exposing users only to data that they need, and improve workload management in your application.

For relevant training materials, see the Organization records and Creating and managing teams of users modules on Pega Academy.

Related information

Creating a Microjourney for customer success *(on page)*
Adding personas to organize users *(on page)*
Designing and tracking Microjourneys development *(on page)*

Operators

An operator defines a unique identifier, password, preferences, and personal information for a user. Create operators so that people and processes can access your application. For example, in an application

for reviewing loan requests, you can create operators for customer service representatives (CSRs), reporting managers, and an administrator to manage the application from the technical side.

By creating operators you provide access for users and processes to your application so that processing work can happen, and so that business processes can reach successful resolution. You can create operators not only for users, but also for automated, unattended processes. Unattended operators are robotic automation virtual machines (VMs). The system generates unattended operators for each registered VM in a robotic process automation (RPA) solution. Creating unattended operators increases the automation of your business processes, lowers costs, and speeds up work resolution.

Creating operators gives you the following benefits:

Increased security

Every operator can have a unique password with which to log in to your application. As a result, you protect your data and operations from inappropriate or random users.

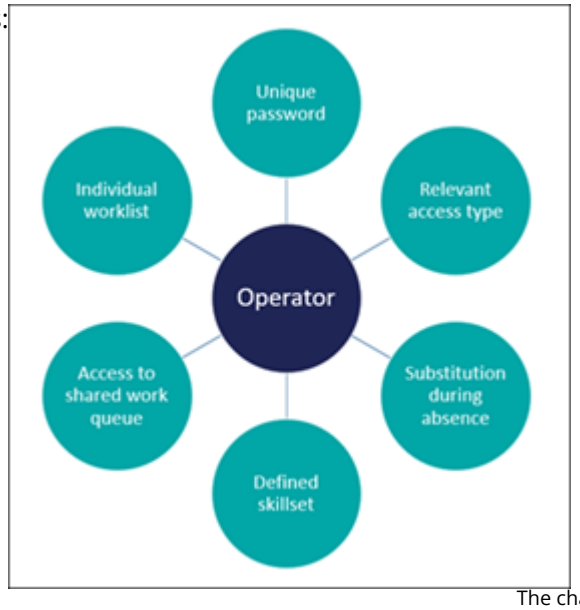
Improved routing of work

Every operator has an individual queue of assignments to complete that Pega Platform refers to as a worklist. Additionally, you can give an operator access to a work queue that groups assignments that are available for members of a specific team. For every operator, you can specify the level of relevant skills, for example, you can indicate that an operator is a highly advanced Java developer, or speaks French fluently. Grouping assignments and defining skillsets improves the routing of work and ensures that an application assigns tasks to the most appropriate users. For every operator, you can also define periods of unavailability and a substitute worker or work queue to ensure continuous case processing.

Relevant access type

You can define which data and operations an operator can access and perform in your application. For example, you can ensure that only a manager can approve or reject new job candidates, or that a CSR can access all information that a case requires to reach resolution. Consequently, the users of your application can perform only relevant actions and process their work faster.

The following figure summarizes the options that are available for operators in Pega Platform applications:



The characteristics of an operator

Clusters and operator IDs

After you save a new, or update an existing operator ID instance, the change might not be reflected on another node in a cluster until the Pega-RULES agent on that node performs the next system pulse — typically after no more than 60 seconds. Unlike instances of most other *Data-* classes, the system saves operator ID instances to the rule cache. As a result, until the next time that the rule cache is synchronized, one node might access a stale copy from its rules cache.

Bulk operator load

To save time, you can create operator ID instances by importing a comma-separated values (CSV) file, such as a Microsoft Excel spreadsheet.

Operator ID property

After a requestor logs in, the operator ID identifier is available on the *pxRequestor* page as the *pxUserIdentifier* property.

Operator IDs and external identity providers

If you implement authentication by using an external identity provider (IdP), the login process accesses IdP for authentication and ignores the password in this operator ID instance. However, the system still requires an operator ID data instance for each user.

Operator passwords

The system saves passwords for operator IDs as hashed values in the PegaRULES database, by first salting the clear text password value and then hashing it using the default bcrypt algorithm. The system uses two property types when changing the password: Password type for the **New Password** field, and Text type for the **Confirm Password** field. The *Data-Admin-Operator-ID.pyPwdCurrent* property stores the hashed password after the password passes validation.

Bcrypt hashing algorithm for Password property types *(on page)*
Managing Requestor Type data instances *(on page)*
Fields for operator contact information and application access *(on page)*
Defining security information for an operator *(on page 55)*

Creating an operator ID

Provide users with access to your application by creating operator IDs. When you create an operator ID, you can specify the background and general information about the user, define application access, and configure password credentials for the user. For example, in a banking application, create operator IDs for customer service representatives (CSRs) who review loan requests, and the manager to whom the CSRs report.

1. In the header of Dev Studio, click **Create > Organization > Operator ID**.
2. In the **Short description** field, enter the full name of the new operator.
   The value that you enter populates the **Full name** field in the operator contact information on the **Work** tab. The maximum length of the description is 64 characters.
3. In the **Operator ID** field, enter a unique identifier.
   When you enter the identifier, consider the following guidelines:
   - In addition to letters and digits, the identifier can include the following characters: period (.), single quote ('), tilde (~), underscore (_), exclamation point (!), ampersand (&), octothorpe (#) and no more than one at (@) character. Avoid using the forward slash (/) or backslash (\) characters in the identifier.
   - An operator ID can consist of a maximum of 128 characters.
   - Special processing applies to any user identifier that begins with the word `External`. Use such identifiers only when defining external operators that use Directed Web Access (DWA) for one-time processing of assignments.
   - Avoid using `system` as an operator ID name. This term is reserved and refers to agents.

4. Click **Create and open**.

> The operator rule form is displayed on the page and you can continue to add information about the user. For example, you can configure access settings or provide organizational information.

Adding an operator by using the organizational chart *(on page)*
Operators *(on page 46)*

Defining operator contact information and application access

Define contact information and application access for an operator so that all operators can communicate and access the required applications. The access groups that you specify affect which applications an operator can access and what their role is. For example, you can configure access settings so that a user can access a Loan requests application both as a manager and as a customer service representative (CSR).

1. In the header of Dev Studio, click **Configure > Org & Security > Organization > Operators**.
2. On the **Operators** tab, select the operator that you want to edit.

> **Tip:** To find your operator faster, in the **Search Text** field, enter the operator name, and then click **Search**.

3. **Optional:** To more quickly distinguish between operators, on the **Profile** tab, in the **Contact Information** section, provide an operator image:
    a. Click **Choose File**.
    b. In the files window, select a graphic file that you want to use, and then click **Open**.
    c. Click **Upload Image**.
    > The image is visible after you save the operator record.
4. Provide basic information about the operator:
    a. **Optional:** To inform users how to address the operator, in the **Title** field, enter a relevant personal title.
    b. **Optional:** To provide personal information about the operator, complete the **First Name** and **Last name** fields.

c. In the **Full name** field, enter a value that you want to use to represent the operator across your system.
By default, the system autopopulates the **Full name** field with the value that you provide in the **Short description** field when you create an operator ID. The operator's full name is visible, for example, in the list of operators.

d. **Optional:** To inform users about the job title of the operator, complete the **Position / job title** field.

e. **Optional:** To provide the phone contact information of the user, complete the **Phone** field by entering the phone number in the internationally standardized E.164 format. E.164 numbers can have a maximum of fifteen digits and use the following format: [+][*country code*][*subscriber number including area code*].
The system uses the phone number to send SMS correspondence to the user.

f. **Optional:** To provide the email contact information of the user, complete the **Email** field.
The system uses the email address to send email correspondence to the user.

5. In the **Application Access** section, list the access groups that the operator can access at run time:

a. In the **Access Group** field, enter an access group that you want to associate with the operator.

> **Tip:** To see the roles in an access group, click the **Expand row** icon.

b. **Optional:** To provide additional roles and access to additional applications, click **Add item**, and then in the **Access Group** field, enter another access group.

c. Indicate the default access group by selecting a relevant radio button.

> When a user logs in, the system directs the user to the application and role that the default access group defines. For example, if you define `LoanRequests:Managers` as a default access group, the user logs into the Loan requests application with a manager role.

6. In the **Localization** section, list the default locale and time zone of the operator.

Fields for operator contact information and application access *(on page)*
Learning about access groups *(on page 63)*
Operators *(on page 46)*

Defining work routing settings for an operator

For continuous and efficient workload management and routing, define skills, work groups, and work queues for operators. As a result, an operator can receive work based on skill set and team membership. For reporting purposes and advanced routing use cases, configure the reporting structure. For example, when you define the reporting manager of an operator, the manager might receive notifications about the operator's progress on a case.

Every operator has a unique worklist that accumulates tasks that the specific operator can retrieve. Additionally, an operator can access a work queue that collects tasks for a team to which the operator belongs. Any team member can pick up tasks from a work queue. Another name for a team is a work group.

For an automated selection of tasks to process, operators can use the Get Next Work logic. Get Next Work automatically prompts users with an assignment that currently has the highest urgency in work queues and work groups that the operator can access.

1. In the header of Dev Studio, click **Configure > Org & Security > Organization > Operators**.
2. On the **Operators** tab, select an operator that you want to edit.

> **Tip:** To find your operator faster, in the **Search Text** field, enter the operator name, and then click **Search**.

3. On the **Work** tab, in the **Organizational unit** section, define the affiliation of the operator within the organization by clicking **Update**, and then provide a relevant organization, division, and unit.
4. In the **Team** field, enter a work group to which the user belongs.
   A work group defines routing of tasks and a reporting manager of the user.
5. **Optional:** To assign a user to multiple work groups, click **Add a work group**, in the text box enter a work group, and then select the radio button next to the work group that you want to set as the default.
6. **Optional:** To enhance the reporting options of your application, in the **Reports to** field, enter the reporting manager for the user.
7. **Optional:** To enable routing of tasks based on a skillset, in the **Skill** field, enter the ability of the user, and then in the **Rating** field, enter the level of the ability.
   You can apply a rating from 1 to 10, where 10 implies the highest level of the skill.
8. **Optional:** To allow the user to process assignments from a work queue that corresponds with the user's work group, in the **Work queue** section, click **Add item**, and then provide a work queue and an urgency threshold.

During Get Next Work processing, your application ignores assignments with urgency lower than the urgency threshold. For more information about Get Next Work, see Customizing the Get Next Work logic *(on page)*.

9. **Optional:** To assign the user to multiple work queues, click **Add item**, and then repeat step 8 *(on page 52)*.
10. **Optional:** To trigger the Get Next Work algorithm to retrieve assignments from the work queues of the user first, select the **Get from work queues first** check box.
    Otherwise, Get Next Work picks up the top assignment on the user worklist, and accesses work queues only if this user's worklist is empty.
11. **Optional:** To trigger the Get Next Work algorithm to retrieve assignments from all work queues that correspond with the team that you provided in step 4 *(on page 52)*, select the **Use all work queue assignments in user's team** check box.
12. **Optional:** To trigger the Next Assignment item to consolidate assignments from all of the work queues of the user, sort by assignment urgency, and return the most urgent assignment in any work queue, select the **Merge work queues** check box.
    If you select the **Use all work queue assignments in user's team** check box in step 11 *(on page 53)*, the system selects the **Merge work queues** check box by default and you cannot clear this check box.
13. Click **Save**.

Fields for operator teams, work queues, and schedules *(on page)*

Operators *(on page 46)*

Defining operator availability

Ensure that the workflow in your organization is uninterrupted by defining operator availability and substituting operators that can receive work when the operator is unavailable. As a result, you ensure that the processing of your business cases continues even when some of the users are unavailable to perform work.

1. In the header of Dev Studio, click **Configure > Org & Security > Organization > Operators**.
2. On the **Operators** tab, select an operator that you want to edit.

> **Tip:** To find your operator faster, in the **Search Text** field, enter the operator name, and then click **Search**.

3. **Optional:** To allow applications to route additional assignments to the worklist of the user that the operator ID identifies, on the **Work** tab, in the **Availability** section, select the **Operator is available to receive work** check box.
   If you leave the check box cleared, the operator can log in, enter work items, or complete assignments that are already in the worklist. The operator can receive work from other operators and from a manager.
4. **Optional:** To define the working days and national holidays for the operator, in the **Business Calendar** field, select the relevant calendar.
5. If the operator might be unavailable in the future, in the **Scheduled absences** section, in the **Unavailable form** and **To** from, define a time period during which the operator is unavailable to perform work.
6. **Optional:** To define more periods of unavailability for an operator, click **Add item**, and then repeat step 5 *(on page 54)*.
7. **Optional:** To define how an application routes work when the operator is unavailable, in the **When unavailable** section, in the **Substitute operator type** list, select who or what work queue receives work:
   - To route work to another operator, select **Operator**.
   - To route work to a list of assignments for a team, select **Work queue**.
8. **Optional:** To determine a substitute for the unavailable operator at run time, in the **Decision tree to find substitute** field, enter a decision tree that evaluates conditions and finds another assignee for the assignment.
   Based on your configuration in step 7 *(on page 54)*, select a decision tree that returns either an operator ID or a work queue.
9. **Optional:** To identify the operator ID or work queue name of a substitute for new assignments that an application routes to the unavailable operator, enter a relevant value in the **Default to assignee** field:
   - To route work to a specific user, enter the operator ID of the user.
   - To route work to a team's list of assignments, enter the work queue.

   The system uses the value that you provide if you do not specify the decision tree to determine a substitute operator or the decision tree does not return a result.
10. Click **Save**.

Related information

Creating an operator ID *(on page 49)*
Creating a team *(on page)*
Adding work queues to a team *(on page)*
Creating work groups *(on page 59)*

Defining security information for an operator

Ensure that your organization complies with security policies by defining security information for the operators in your system. You can manage operator authentication, passwords, and license types, to allow rule check out, and enable or disable the operator. As a result, operators can access your application safely, and then perform only the actions that are relevant to their roles.

1. In the header of Dev Studio, click **Configure > Org & Security > Organization > Operators**.
2. On the **Operators** tab, select an operator that you want to edit.

> **Tip:** To find your operator faster, in the **Search Text** field, enter the operator name, and then click **Search**.

3. On the **Security** tab, in the **Access Settings** section, secure access to your application with a password:
   a. Click **Update password**.
   b. In the **Change Operator ID Password** dialog box, in the **New password** field, enter the new password.
   c. In the **Confirm new password** field, reenter the password.
   d. Click **Submit**.

> **Note:** If the operator is provided with Pega Platform, enter a password that is consistent with your security policies, and then send the new password to the enabled operator.

   The system converts the password to a hash value by using the salted bcrypt algorithm. The Storage Stream (BLOB) column of the pr_operators table contains the hashed value. By using the **View XML** action, you can discover only the hashed form of any operator password.

   For more information about security polices, see Security policies *(on page)*.

4. If the operator is an unattended operator, select the **This is an unattended operator (robot)** check box.
   Unattended operators are robotic automation virtual machines (VMs). The system generates unattended operators for each registered VM in a robotic process automation (RPA) solution.
5. **Optional:** To allow this operator to update rules in rulesets that use rule checkout, select the **Allow rule check out** check box.
6. **Optional:** To authenticate this operator only through external authentication facilities, select the **Use external authentication** check box.

7. **Optional:** To prompt the operator to change their password the next time the operator logs in, select the **Force password change on next login** check box.
8. **Optional:** To disable the operator, select the **Disable Operator** check box.
9. **Optional:** In the **Starting activity to execute** field, specify the first activity that the system runs after authentication for this user is complete.
   The default is `Data-Portal.ShowDesktop`.
10. In the **License type** list, indicate the operator type:
    - To indicate that the operator is a person who does business operations by using an application or customer-created interface, select **Named**.
    - To indicate that the operator is an abstract user to run agents, services, and other background processes, or an external user that interacts with the application through the Directed Web Access feature, select **Invocation**.

      For unattended operators, the system selects **Invocation** by default.
11. Click **Save**.

Operators *(on page 46)*
License compliance *(on page)*
Checking out a rule *(on page)*

Deleting operators

To ensure that only designated workers can access your application, you can disable access for users that are no longer active, for example, when they no longer work in your organization.

You cannot delete an operator when a data instance, such as an organization unit or work queue, still references the operator. To save time, instead of deleting all of the references individually, change the operator password and mark the user as unavailable.

1. Ensure that the user that you want to delete is not logged in.
2. Transfer or complete all assignments in the work queue for the user.
   For more information, see Transferring an assignment *(on page)*.
3. Update the password to a value that is unknown to the user, so that the person can no longer log in.
   For more information, see Defining security information for an operator *(on page 55)*.
4. Ensure that the user has no rules checked out. If any rules are checked out, sign on and delete or check in the checked-out rules.
   You cannot delete an operator ID if the operator has rules checked out. Ensure that the operator deletes or checks in all rules in their personal ruleset. For more information, see Checking in a rule *(on page)*.

5. Mark the user as unavailable to receive work, and then define the user's departure date and a substitute operator.
   For more information, see Defining operator availability *(on page 53)*.

Operators *(on page 46)*

Creating a work queue

To improve workload management and task routing in your application, create a work queue that holds assignments for operators and robotic queues. Because you associate a work queue with a group of users or a robotic queue, you logically and efficiently categorize work inside your organization.For example, create a work queue for customer service representatives (CSRs) who manage loan requests to ensure that an application routes loan request tasks to users with relevant skill set and qualifications.

> **Note:** When you create an application, the system creates a default work queue and work group. Because of mutual dependencies, you need to provide a work queue when you create a work group, and when you create a work group, you need to provide a work queue. Use the default values, and then provide the actual values after you create your work group and work queue.

1. In the header of Dev Studio, click **Create > Organization > Work queue**.
2. On the **Create Work queue** tab, in the **Short description** field, describe the purpose of the work queue.
3. In the **Work queue** field, enter a name that clearly conveys the purpose of the work queue and its contents to users of the application.

   > **Note:** For a better distinction between work queues and other organization elements, append the `WQ` characters to the work queue name.

4. Click **Create and open**.
5. On the **Work queue** tab, in the **General** section, specify a work queue type:

Choices	Actions
Create a work queue that distributes assignments to a virtual machine	a. In the **Type** list, select **Robotic**. b. In the **Maximum queue length** field, enter an integer that defines the maxi-

Choices	Actions
	mum number of tasks that a queue can contain. c. In the **Maximum automation execution time (seconds)**, enter the maximum amount of time in seconds that the virtual machine can take to run the assignment before the system cancels the assignment.
Create a work queue that distributes assignments to users	In the **Type** list, select **Standard**.

6. In the **Organization** section, specify an organization layer that contains your work queue:
    a. In the **Name** field, enter the name of an organization to which the work queue belongs.
    b. In the **Division** field, enter the work queue division.
    c. In the **Unit** field, enter the work queue unit.
    d. In the **Work group** field, enter a work group that you want to associate with a work queue. A work group consists of a work queue and a manager to which the work queue members report.
7. **Optional:** To notify operators about new assignments, in the **Contacts** section, in the text field, enter the ID of an operator who receives a notification when an application routes work to the work queue.
8. **Optional:** To add more operators who receive notifications, click **Add item**, and then enter the operator ID.
9. **Optional:** To specify which users can process work in this work queue, in the **Roles** section, in the text field, enter the required role for a user to resolve assignments from the work queue.

> ⚠ **CAUTION:** If you leave this field blank, any user who can access the work queue can process work.

10. **Optional:** To add more roles, click **Add item**, and then enter a role.
    To process work, a user needs to have at least one role from the set of roles that you specify.
11. Click **Save**.

Creating an operator ID *(on page 49)*
Creating a team *(on page)*
Configuring advanced settings for new applications *(on page 19)*
Copying a rule or data instance *(on page)*
Creating a rule specialized by circumstance *(on page)*

Creating work groups

Manage work inside your organization logically and efficiently by creating work groups. A work group connects a manager and a group of reporting users to enable relevant workflow management and improve communication.For example, at run time, when a user misses a deadline for a task, a reporting manager receives a notification. Your application determines the manager by using a work group setting.

> **Note:** When you create an application, the system creates a default work queue and work group. Because of mutual dependencies, you need to provide a work queue when you create a work group, and when you create a work group, you need to provide a work queue. Use the default values, and then provide the actual values after you create your work group and work queue.

1. In the header of Dev Studio, click **Create > Organization > Work Group**.
2. On the **Create Work Group** tab, in the **Short description** field, briefly describe the purpose of the group.
3. In the **Work Group Name**, enter a label for the work group.
   As a best practice to provide more information in the name, append to the work group name the at sign (@), followed by your organization name.
4. Click **Create and open**.
5. On the **Work group** tab, in the **Settings** section, provide basic information about the work group:
   a. In the **Manager** field, enter the ID of a user who is a reporting manager for the group.
   b. In the **Default work queue** field, enter a default work queue to receive work coming into this work group.
6. Click **Save**.

Related information

Creating a work queue *(on page 57)*
Creating a team *(on page)*
Adding work queues to a team *(on page)*

Updating the organizational structure by using the organizational chart

Reflect the growth of your company by creating new elements in your organizational structure. As your organization develops, you can add new divisions, units, and sub-units that correspond with new elements of your enterprise. Maintaining an organizational structure can accelerate your application development, as you can reuse different application elements across your organization.For a clear visualization of levels in your company, use an organizational chart to add new elements. The organizational chart displays each level with expandable sub-levels, such as a unit for a division. For example, when an Engineering division gains a new team, you can create a new unit that visualizes the new team.

In an organizational structure, an organization reflects your company. A division corresponds with a part of the company that gathers teams that operate in a similar business sector, such as engineering or human resources. A unit visualizes a single team. For further granularization, you can create sub-units. For example, if you need a unit to represent multiple teams that cooperate, a sub-unit might reflect individual teams. Consider a scenario in which a UPlusTelco organization needs to create an Engineering division. The division includes a Mobile unit with two sub-units, Prepaid and Postpaid.

You can reuse rules and other elements of your application across organizational layers. For example, reuse a process for calculating expenses for an entire organization and then specify different maximum amounts for each division.

1. In the header of Dev Studio, click **Configure > Org & Security > Organization > Organizational Chart**.
2. On the **Organizational Chart** tab, add a new organizational element:

Choices	Actions
Add an organization	a. Right-click an existing organization. b. In the list of available options, select **Add organization**. c. In the **Add Organization** dialog box, in the **Organization** field, enter the name of your organization. Use the organization's short name, ticket symbol, or Internet domain name. d. **Optional:** To provide more information about your organization, in the **Description** field, enter additional text. e. Click **OK**.

Choices	Actions
Add a division	a. Right-click the organization to which you want to add a division. b. In the list of available options, select **Add division to organization**. c. In the **Add Division** dialog box, in the **Division** field, enter the name of your division. d. **Optional:** To provide more information about your division, in the **Description** field, enter additional text. e. Click **OK**.
Add a unit	a. Right-click the division to which you want to add a unit. b. In the list of available options, select **Add unit to division**. c. In the **Add Unit** dialog box, in the **Unit name** field, enter the name of your unit. d. **Optional:** To provide more information about your unit, in the **Description** field, enter additional text. e. Click **OK**.
Add a sub-unit	a. Right-click the unit to which you want to add a sub-unit. b. In the list of available options, select **Add sub-unit to unit**. c. In the **Add Unit** dialog box, in the **Unit name** field, enter the name of your unit.

Choices	Actions
	d. **Optional:** To provide more information about your sub-unit, in the **Description** field, enter additional text. e. Click **OK**.

Creating an operator ID

When you create an operator ID in Pega Platform, you set up a unique account so that a user can access the system. By creating an operator ID you can set the identifying information of the user, define their access rights, and configure password settings. You can also add additional details, such as their skills, to ensure that work is routed to the user based on their capabilities.

1. In the header of Dev Studio, click **Create > Organization > Operator ID**.
2. In the **Short description** field, enter the full name of the new operator.
   The value you enter populates the **Full name** field on the following screen.
3. In the **Operator ID** field, enter a unique identifier.
   When you enter the identifier, consider the following guidelines:
   - In addition to letters and digits, the identifier can include the following characters: period (.), single quote ('), tilde (~), underscore (_), exclamation point (!), ampersand (&), octothorpe (#) and no more than one @ character. Avoid using forward slash (/) or backslash (\) characters in the identifier.
   - As a best practice, use the organization name as a suffix in operator IDs.
   - An operator ID can be up to 65 characters long.
   - Special processing applies to any user identifier that starts with the word `External`. Use such identifiers only to define external operators, those who use directed Web access for one-time processing of assignments.
   - Avoid using `system` as an operator ID name, which is reserved as it refers to agents.
4. Click **Create and open**.

   The operator rule form is displayed, where you can continue adding information about the user. For example, you can configure access settings, or provide organizational information.

Adding an operator by using the organizational chart *(on page)*
Operators *(on page 46)*

Learning about access groups

An access group is a group of permissions within an application. Pega Platform uses these permissions for operators, external system access, and background processes. You define an access group for operators who have similar responsibilities. For example, most applications allow case managers to do actions that are different from the actions of regular operators, so case managers and regular operators belong to different access groups.

Access group names have the format *application name:access group name*. For example, for the MyApp application, you can define the MyApp:Administrators access group for administrators and the MyApp:Users access group for regular operators.

Operators can belong to multiple access groups. You select one of the access groups as the default, which is used when the operator initially logs in. If an operator belongs to multiple access groups, the operator can switch between groups. Only one access group is in effect at any given time during a session.

When you create an access group, you define permissions and settings that are used for operators who belong to that access group and who use the application defined for that access group. These permissions and settings include the following:

- Access roles and privileges
- The portal layout
- The work pools that are available
- The types of work items that operators can work on
- The rulesets that are displayed at the top of the ruleset list
- Details of rule caching for performance
- For developers, the initially displayed ruleset and version for rules that they create

Access groups and ruleset lists

When an operator logs in, Pega Platform looks for an access group in the following order until an access group is found, and uses that access group to assemble the operator's ruleset list:

1. The default access group defined on the **Profile** tab of the **Operator ID** form
2. The default access group for the Org Division that is identified on the **Work** tab of the **Operator ID** form
3. The default access group for the Org that is identified on the **Work** tab of the **Operator ID** form
4. The default access group for the appropriate requestor type

Access groups and external systems

An access group determines the ruleset list that is available to an external system that requests services. The following data instances and rules reference access groups directly, or indirectly by specifying an operator:

- Listener data instances
- Service package data instances
- Agent rules
- Agent schedule data instances

When effective

When you save an access group, active requestor sessions on the current node that are associated with that access group are immediately updated. Requestors at other nodes in a cluster are updated when the next system pulse occurs on their nodes.

Facilities provided to unauthenticated (guest) requestors

Guest users, or unauthenticated requestors, typically have access to only the rules in the rulesets in the *PRPC:Unauthenticated* access group, as referenced in the requestor type instance named `pega Browser`.

CAUTION: You should review **Security** tab of each activity in the rulesets, to verify that the **Require authentication to run** check box is not selected if you update the:

⚠

- `pega Browser` requestor type to reference a different access group, or
- *PRPC:Unauthenticated* access group to make additional rulesets available to unauthenticated users.

The clipboard for a guest requestor does not include pages for the operator ID, organization, division, or organization unit.

Managing access roles *(on page)*
Fields for operator contact information and application access *(on page)*
Managing Requestor Type data instances *(on page)*

Viewing access groups and operators

You can view all the access groups or view only the groups that reference an application. You can see all the access groups across the Pega Platform applications and the operators who have access to those applications.

1. To view all the access groups in Pega Platform, complete the following steps.
   You can also view the operators in a group.

   a. In the header of Dev Studio, click **Configure > Org & Security > Groups & Roles > Access Groups**.

   b. The fields on this page allow you to learn more about access groups and operators for the current application.
      - **Search text** – Filter the list to display only those access groups with a specific text value in the name or description.
      - **View in Excel** – Click to export access group information and operator counts to Microsoft Excel file.
      - **Name** – Click the access group name to open the access group instance.
      - **Description** – A short description of the access group.
      - **Operators** – The number of operators associated with this access group as defined in the operator ID instance. Click the number to display the list of operators.

2. To view only the groups that reference the current application, click **Configure > Application > Structure > Access Groups and Users**. The names of the operators in each group are also displayed.
3. To view the operators who belong to a selected access group, complete the following steps.

   a. Open an existing access group from the navigation panel by clicking **Records > Security > Access Group** and selecting an instance.

   b. On the **Operators** tab, review the operator ID instances that reference this access group. You can click a row to open an operator ID record.
   The following information is displayed for each operator:
      - Full name
      - Operator ID
      - Job position
      - Date and time of last sign-on

4. To view the operators who are assigned to a particular organization, division, or unit, complete the following steps.

a. Display the organizational chart by doing one of the following steps.
   - Click **Configure > Org & Security > Organization > Organizational Chart**.
   - Open the rule form for the organization, division, or unit, and go to the **Chart** tab.

b. Select any node within the organization.

c. Right-click to open the pop-up panel and click **View Operators**.

5. To view all operators, do one of the following.
   - Click **Configure > Org & Security > Organization > Operators**.
   - In the navigation panel, click **Records > Organization > Operator ID**.

Operators *(on page 46)*
Fields for operator contact information and application access *(on page)*

Creating an access group

An access group is a group of application permissions that are used by an operator, external system, or background process. Create an access group to define the actions that are allowed when such an entity uses an application.

1. In the Dev Studio header, click **Create > Security > Access Group**.
2. Enter a short description for this access group data instance.
3. In the **Access Group Name** field, enter a name. As a best practice, use the format of an application name followed by a single colon and a description of the job or user function of the users or other requestors who belong to this access group.
   Do not use an asterisk character(*), parentheses (), or spaces in an access group name. Begin the name with a letter and use only letters, digits, dash, ampersand, period, and colon characters.
4. Click **Create and open**.

   > The access group is created in a ruleset for the application that you are currently logged in to. You can change the ruleset by clicking **Edit** next to the ruleset name.

5. In the **Application** section, in the **Name** field, press the Down Arrow key and select the name of the application that this access group can access.
6. In the **Application** section, in the **Version** field, press the Down Arrow key and select the version of the application that this access group can access.
   When a user logs in, the ruleset names and ruleset versions identified in the application rule are added to the user's ruleset list.
7. Grant portal access to the access group *(on page 68)*.
8. Add roles to the access group *(on page)*.

Learning about access groups *(on page 63)*

Assigning work pools to an access group

Work pools are the case types in which users in an access group are allowed to create cases. You specify the work pools that are available to an access group.

1. Create an access group, or open an existing instance from the navigation panel by clicking **Records > Security > Access Group** and selecting an instance.
2. On the **Advanced** tab, in the **Name** field of the **Work Pools** section, list the work pools that are accessible to this access group.
    a. In the **Name** field, press the Down Arrow key and select a work pool.
    b. To add more work pools to the access group, click **Add item** and select a work pool.
    c. To select the user's default work pool, click the button next to the work pool name.
    As a best practice, select the work pool that contains the case types in which users that belong to this access group most often create cases. This selection determines the default work pool that appears on the top of the Application Explorer tree when you open an application.

Adding personas to organize users *(on page)*

Work pools and access groups

Work pools are the case types in which users in an access group are allowed to create cases. The work pools that you add to an access group affect the user experience at run time.

- Work pools are classes that are marked as **is a class group** on the **General** tab of the class rule form. A work pool is a named collection of case types in which users that belong to the access group associated with the work pool are permitted to create cases.
- When you define the access group, in the access group's work pool list, you can enter any class group for which the container class (the *Rule-Obj-Class* instance with the same name as the class group) belongs to a ruleset listed in the **Production Rulesets** list for the access group or that is available to users through application rules. This restriction ensures that the rules of the application are available to users associated with the access group.
- At run time, the work pools that you add to an access group appear in the Application menu **Switch Work Pool** list of the users who belong to the access group. Work pool names appear in the order in which you list the work pools for the access group.

> **Tip:** If this list contains many work pools, consider reordering them to display the work pool names alphabetically or in another order that is meaningful to users.

- Leave a work pool list empty if users who create new cases and belong to the access group associated with the work pool use a composite portal that includes the standard section *@baseclass.NewWork*. The cases that such users can create appear on the **Cases and Data** tab of the Application rule, not in the access group.
- At run time, if you leave a work pool list empty and the Application rule is not used for work pools, the system behaves in the following manner:
  - When users that belong to the access group associated with the work pool open an application, the Application Explorer class autocomplete field is empty, and no classes appear in the tree.
  - Users that belong to the access group associated with the work pool cannot create cases. The work pool list does not affect which cases the user can update. The list determines only the types of cases that the user can create. The assignments that the user can open and update are determined by access roles and ruleset lists, not by work pools.

Assigning work pools to an access group *(on page 67)*
Learning about access groups *(on page 63)*
Working with class groups *(on page)*

Granting portal access to an access group

Associate a portal with an access group to control which workspaces or web channels are available to users while they work in your application.

1. Open an access group by searching for it or by using the Records Explorer.

> **Tip:** To open your current access group, select the **Access group** option from the **Profile** menu.

2. Click the **Definition** tab.
3. In the **Available portals** section, click **+ Add portal**.
4. In the **Name** field, press the Down Arrow key, and then select a portal.

For example, select *pxPredictionStudio* to give the data scientists on your team access to the Prediction Studio workspace.

5. **Optional:** To make this portal the default for all operators in the access group, select **Default** next to the portal name.
6. Click **Save**.
7. Log off and then log in to Dev Studio.

> Users who belong to the access group can switch from their current workspace or web channel to the portal that you provide.

Configuring portals *(on page)*
Changing your workspace *(on page 15)*
Learning about access groups *(on page 63)*

Configuring tools access

The Access Manager **Tools** tab provides information about the authorizations that users have to the tools you can secure in your application. Use Access Manager to configure the actions that access groups can do with tools.

You can view all access groups in the application and see a summary of authorizations for each access group.

You can also customize the tools that appear on the **Tools** tab. For more information, see Adding custom tools to Access Manager *(on page)*.

When you expand a tool category, icons indicate whether full access, no access, or conditional access is granted to members of the access group.

Summarized views can alert you to areas in your business processes that need tighter restrictions. You can expand the view to edit the authorizations of a role.

Access Manager *(on page)*
Application tools *(on page)*

Viewing tools authorizations in all access groups

You cannot modify authorizations when viewing all access groups. The view shows you only whether:

- All operators in an access group are authorized to use all the tools listed in a tool category.
- No operators in an access group are authorized to use any tools listed in a tool category.
- A combination of conditional access and possibly no access to this tool among operators in the access group.

1. In the header of Dev Studio, click **Configure > Org & Security > Access Manager > Tools**.
2. Click **Application**.
3. Select or deselect the applications you want to modify.
4. Click **Apply**.
5. In the **Access Group** field, select **All Access Groups**.
6. The page displays the access groups for the application and their access status for each tool. Expand a tool category to see the authorization for each tool in the category.

Access Manager *(on page)*
Application tools *(on page)*

Viewing tools authorization for a single access group

Access Manager allows you to view settings for access groups other than your own. You can see the following information:

- Operators in this access group have full access to the tool.
- Operators in this access group have no access to the tool. The menu item or UI element is disabled.
- A combination of conditional access and possibly no access to this tool among operators in the access group.

1. In the header of Dev Studio, click **Configure > Org & Security > Access Manager > Tools**.
2. Click **Application**.
3. Select or deselect the applications you want to process.
4. Click **Apply**.
5. Select an access group from the **Access Group** list.

   > The page displays the tools and their access status for the access group.

6. View or edit the tool category.
   a. Click an individual access group to edit.
   b. Expand a tool category to see the authorization for each tool in the category.

Access Manager *(on page)*
Application tools *(on page)*

Editing tools authorization for a single access group

You can change the access status of a role for all the operators in the same access group.

1. In the header of Dev Studio, click **Configure > Org & Security > Access Manager > Tools**.
2. Click **Application**.
3. Select or deselect applications that you want to process.
4. Click **Apply**.
5. Select an access group from the **Access Group** list.
6. Expand a tool category to see the authorization for each tool in the category.
7. In the column for the access role to be authorized, click the icon.
8. Select the access type.
    - **Full Access** – Grant operators with this role full access to the item.
    - **No Access** – Deny operators with this role access to the item. At run time, the system can hide or deactivate the item.
    - **Conditional** – Grant access based on an Access When condition. Select an Access When condition under which an operator in the access group can access the item.
9. Click **OK**.

Access Manager *(on page)*
Application tools *(on page)*

Learning about access groups

An access group is a group of permissions within an application. Pega Platform uses these permissions for operators, external system access, and background processes. You define an access group for operators who have similar responsibilities. For example, most applications allow case managers to do actions that are different from the actions of regular operators, so case managers and regular operators belong to different access groups.

Access group names have the format *application name:access group name*. For example, for the MyApp application, you can define the MyApp:Administrators access group for administrators and the MyApp:Users access group for regular operators.

Operators can belong to multiple access groups. You select one of the access groups as the default, which is used when the operator initially logs in. If an operator belongs to multiple access groups, the operator can switch between groups. Only one access group is in effect at any given time during a session.

When you create an access group, you define permissions and settings that are used for operators who belong to that access group and who use the application defined for that access group. These permissions and settings include the following:

- Access roles and privileges
- The portal layout
- The work pools that are available
- The types of work items that operators can work on
- The rulesets that are displayed at the top of the ruleset list
- Details of rule caching for performance
- For developers, the initially displayed ruleset and version for rules that they create

Access groups and ruleset lists

When an operator logs in, Pega Platform looks for an access group in the following order until an access group is found, and uses that access group to assemble the operator's ruleset list:

1. The default access group defined on the **Profile** tab of the **Operator ID** form
2. The default access group for the Org Division that is identified on the **Work** tab of the **Operator ID** form
3. The default access group for the Org that is identified on the **Work** tab of the **Operator ID** form
4. The default access group for the appropriate requestor type

Access groups and external systems

An access group determines the ruleset list that is available to an external system that requests services. The following data instances and rules reference access groups directly, or indirectly by specifying an operator:

- Listener data instances
- Service package data instances
- Agent rules
- Agent schedule data instances

When effective

When you save an access group, active requestor sessions on the current node that are associated with that access group are immediately updated. Requestors at other nodes in a cluster are updated when the next system pulse occurs on their nodes.

Facilities provided to unauthenticated (guest) requestors

Guest users, or unauthenticated requestors, typically have access to only the rules in the rulesets in the `PRPC:Unauthenticated` access group, as referenced in the requestor type instance named `pega.BROWSER`.

⚠ **CAUTION:** If you update the `pega.BROWSER` requestor type to reference a different access group, or update the `PRPC:Unauthenticated` access group to make additional rulesets available to unauthenticated users, review the **Require authentication to run** check box on the **Security** tab of each activity in the rulesets. Select this check box for only the activities that guests need to run.

The clipboard for a guest requestor does not include pages for the operator ID, organization, division, or organization unit.

Managing access roles *(on page)*
Fields for operator contact information and application access *(on page)*
Managing Requestor Type data instances *(on page)*

Creating an access group

An access group is a group of application permissions that are used by an operator, external system, or background process. Create an access group to define the actions that are allowed when such an entity uses an application.

1. In the Dev Studio header, click **Create > Security > Access Group**.
2. Enter a short description for this access group data instance.
3. In the **Access Group Name** field, enter a name. As a best practice, use the format of an application name followed by a single colon and a description of the job or user function of the users or other requestors who belong to this access group.
   Do not use an asterisk character(*), parentheses (), or spaces in an access group name. Begin the name with a letter and use only letters, digits, dash, ampersand, period, and colon characters.
4. Click **Create and open**.

   > The access group is created in a ruleset for the application that you are currently logged in to. You can change the ruleset by clicking **Edit** next to the ruleset name.

5. In the **Application** section, in the **Name** field, press the Down Arrow key and select the name of the application that this access group can access.

6. In the **Application** section, in the **Version** field, press the Down Arrow key and select the version of the application that this access group can access.
   When a user logs in, the ruleset names and ruleset versions identified in the application rule are added to the user's ruleset list.
7. Grant portal access to the access group *(on page 68)*.
8. Add roles to the access group *(on page)*.

Learning about access groups *(on page 63)*

Designing applications for reuse and extension

Save time and speed up your application development process by designing your applications for reuse and extension. When you create reusable elements, such as rules and classes, you can implement them again in future projects, to make your work more efficient. For example, if you create a rule that defines a service-level agreement, you can include this rule in any future cases that you design. At the same time, you improve the flexibility of your projects because you can select from specific elements that your current business project requires.

Related information

Troubleshooting tools and techniques *(on page 340)*

Relevant records for rule reuse

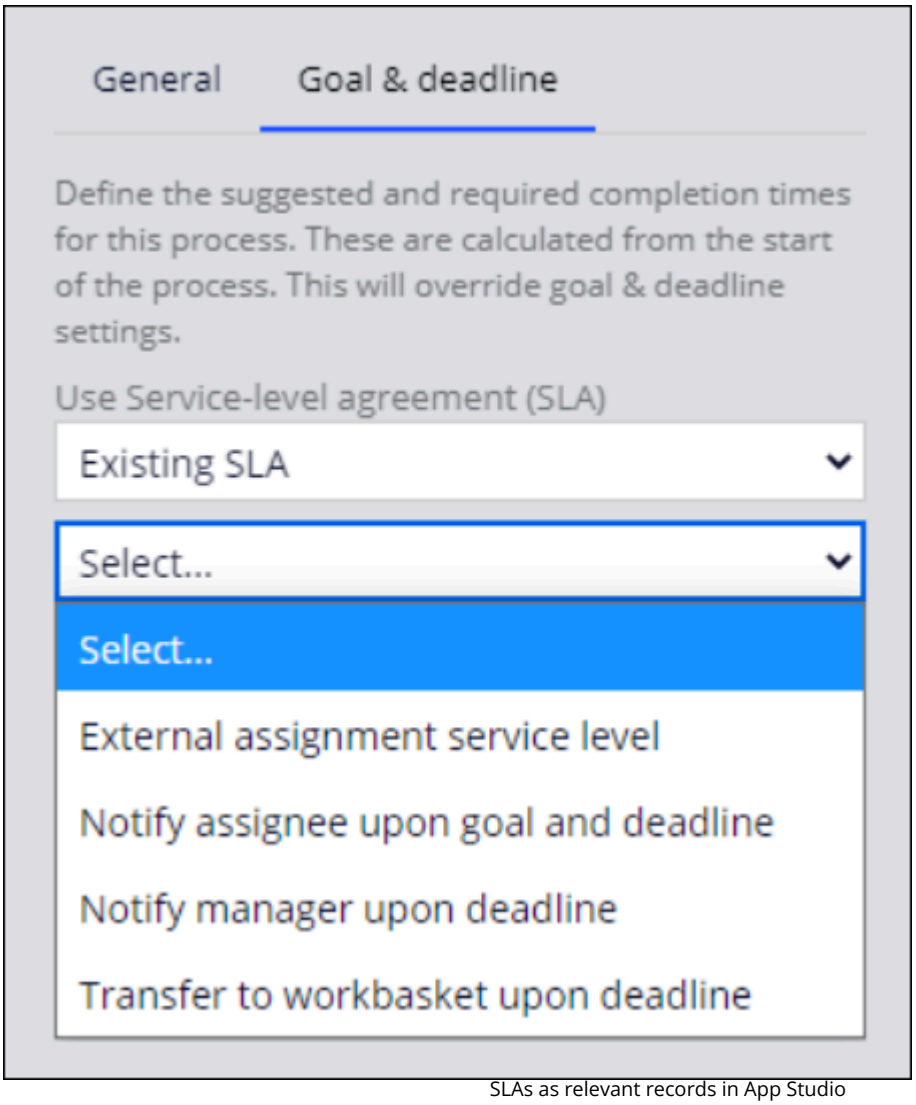
Relevant records are rules that Pega Platform automatically marks for reuse in App Studio, or that application developers manually designate for reuse in Dev Studio. By using relevant records, you configure your application with developer-approved rules, improve its quality, and reduce development time.

Relevant records creation

Typically, Pega Platform automatically manages relevant records for you. In App Studio, when you create records, such as fields, views, processes, and user actions in the context of a case type or data type, Pega Platform automatically marks these records as relevant.

You can also configure a rule in Dev Studio, then mark the rule as a relevant record so that the rule is accessible to other developers from prompts in App Studio.

For example, when you configure an assignment and select an existing service-level agreement (SLA) option in App Studio, the SLA is available in the drop-down list only if another developer first creates and marks the SLA as a relevant record in Dev Studio.



SLAs as relevant records in App Studio

Rule types that you can manually mark as relevant records in Dev Studio include:

- Properties, also known as fields
- Sections, also known as views
- Harnesses
- Paragraphs
- Correspondences
- Service-level agreements
- Strategies
- Processes, also known as flows
- User actions, also known as flow actions
- Data Transforms
- Decision Tables
- Notifications
- Pulse Feed rules

- Validate rules
- When rules

Note: You can mark records that are outside classes that Pega Platform provides by default, such as `@baseclass` or `Work-` classes.

You can manually designate records as relevant in the following ways:

Mark a rule as a relevant record in the rule form

You can mark a selected rule as relevant directly in the rule form in Dev Studio.

For more information, see Marking a record as relevant *(on page 83)*.

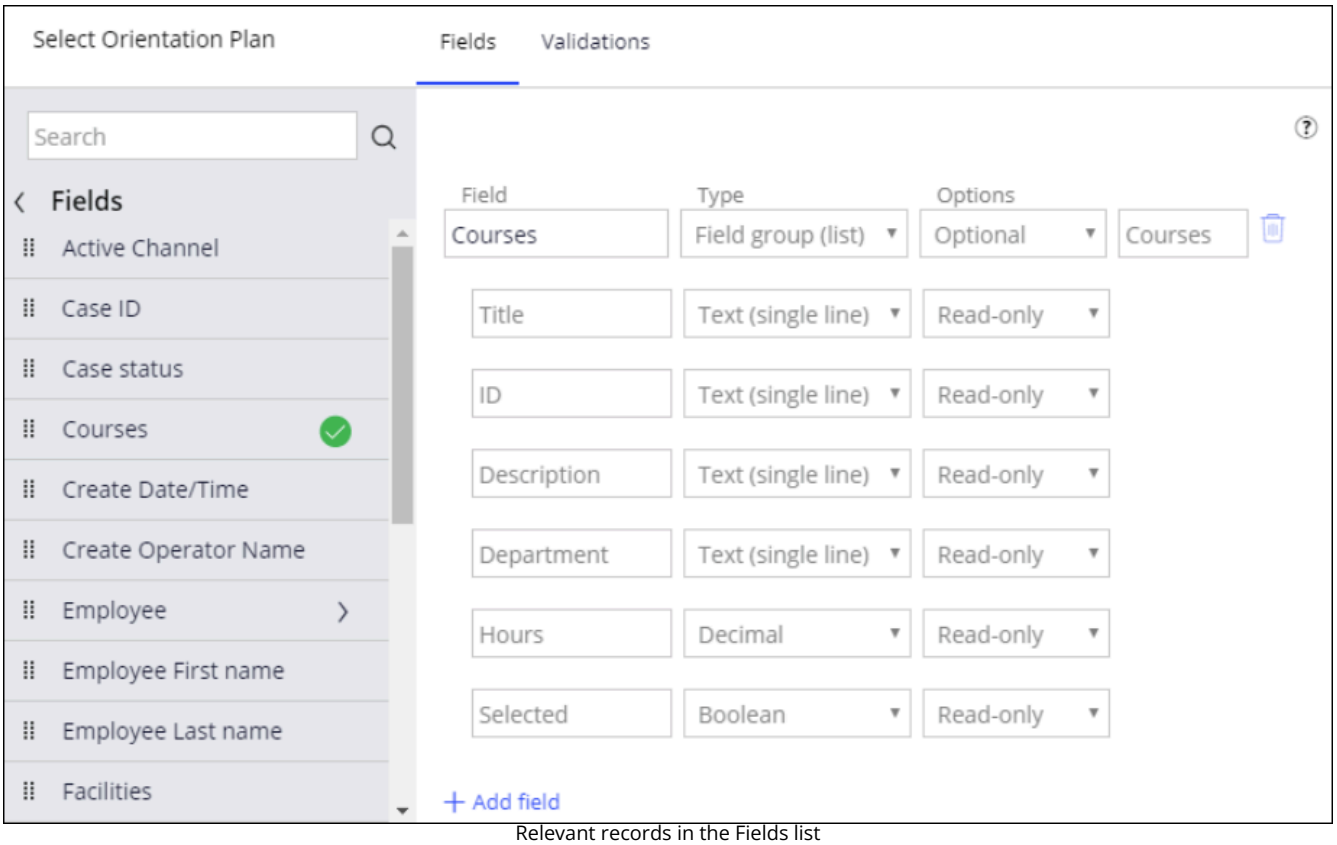
Add a rule as a relevant record on the Relevant records tab

You can designate a selected rule as a relevant record on the **Relevant records** tab in Dev Studio.

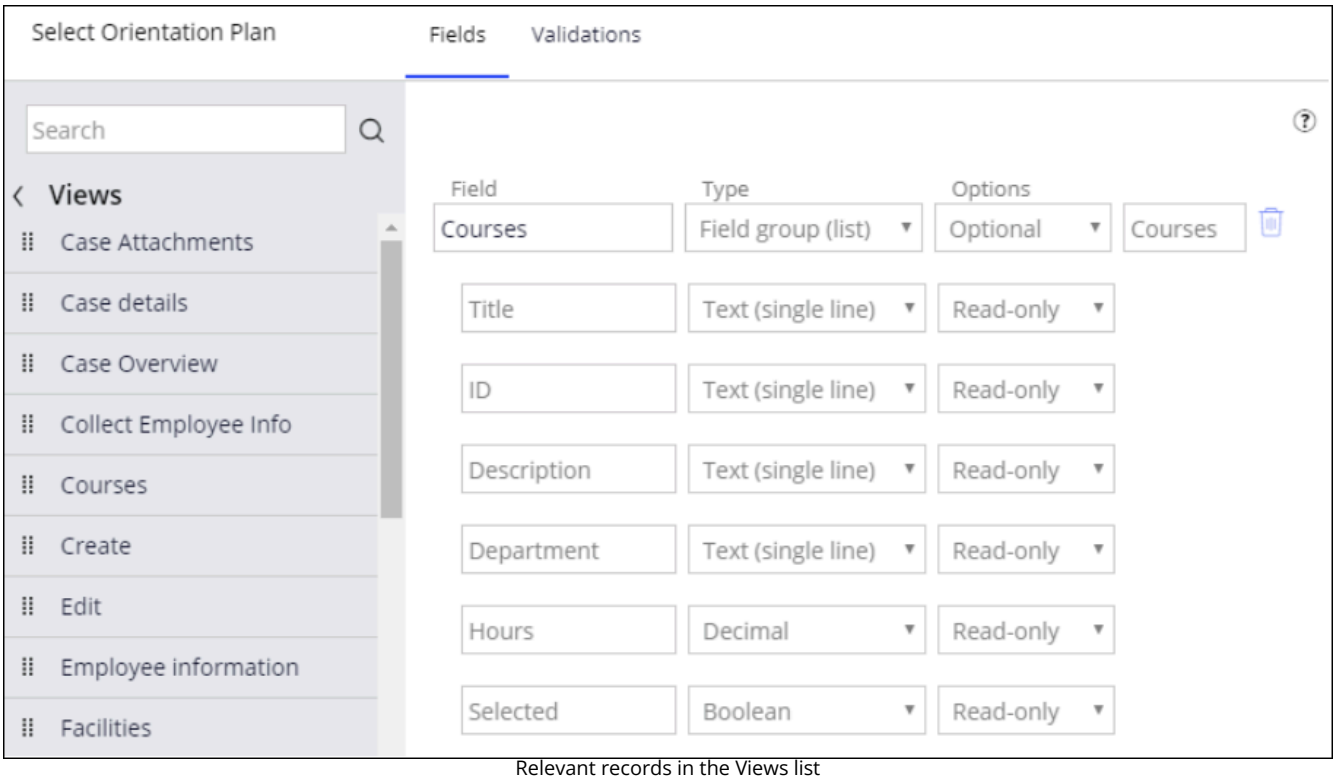
For more information, see Adding a relevant record to a specified class in your application *(on page 84)*.

You can access relevant records in App Studio, in the Data Designer or the Case Designer, when you add a step to a process, add fields to a user view, or apply a service-level agreement.

For example, when you configure a user view for a step in a case life cycle, Pega Platform populates the Fields and Views lists with relevant records, as shown in the figure:



Relevant records in the Fields list



Relevant records in the Views list

The functions of relevant records

Relevant records control design-time prompting and filtering in the following areas:

In case types

Relevant records for a case type can include references to properties, sections, processes, and user actions that are explicitly important to your case.

Properties marked as relevant define the data model of the case. Processes and user actions marked as relevant appear in prompts for case type settings to encourage reuse. Sections marked as relevant appear as reusable sections.

In data types

Relevant records designate the most important inherited fields for a data type. Relevant records can include fields that are defined for the class of the data type and fields inherited from parent classes.

In condition builders

When you build conditions in a condition builder in App Studio, or on the **Condition** tab of a when rule in Dev Studio, you see a list of fields and when conditions that you can use in your custom condition. The list populates with relevant records. If a record is not in the list, you can add it to relevant records in your application.

For more information, see Adding a relevant record to a specified class in your application *(on page 84)* and Defining conditions in the condition builder *(on page 271)*.

In proposition filters

When you use properties, strategies, and when rules as proposition filter conditions, you designate these elements as relevant records for your primary context class, which by default is the *Customer* class.

For more information, see About Proposition Filter rules *(on page)*.

Creating new properties on the **Properties** tab of a strategy or a proposition automatically marks the properties as relevant records. You must manually add strategies and when rules to the list of relevant records for a specific class.

For more information, see Managing relevant records *(on page 82)*.

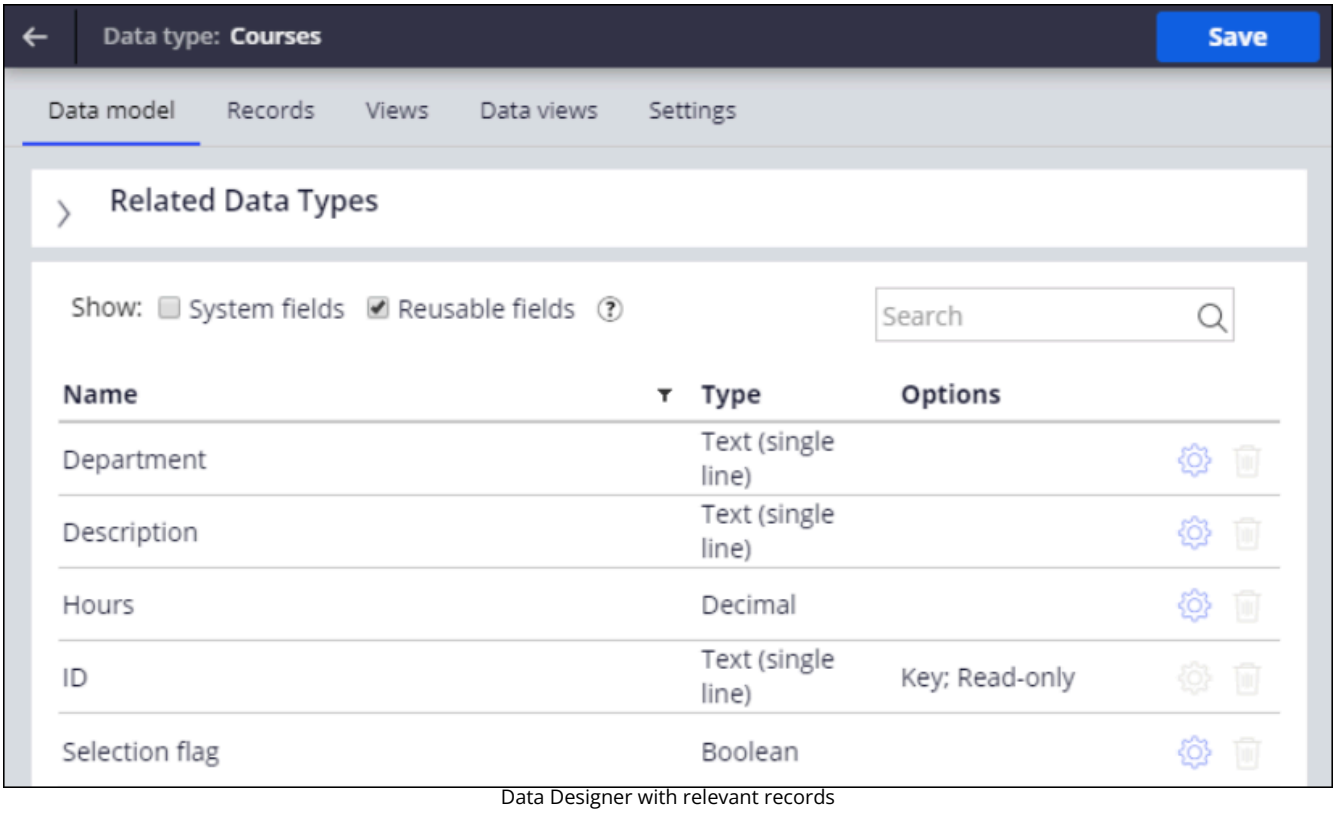
Relevant records in Data Designer

The Data Designer displays properties for the selected data type that the system marks as relevant records. In Dev Studio and App Studio, use the filtering options to show reusable fields, which are relevant records defined elsewhere in the selected data type's inheritance path, and to show internal system fields.

The Dev Studio Data Designer provides the **Show inherited** and **Show relevant records** filtering options.

The Data Designer in App Studio displays only relevant records (properties) for the selected data type, with an option to display or hide relevant fields defined in inherited classes.

New fields that you add to the data types are automatically marked as relevant records as shown in the figure.



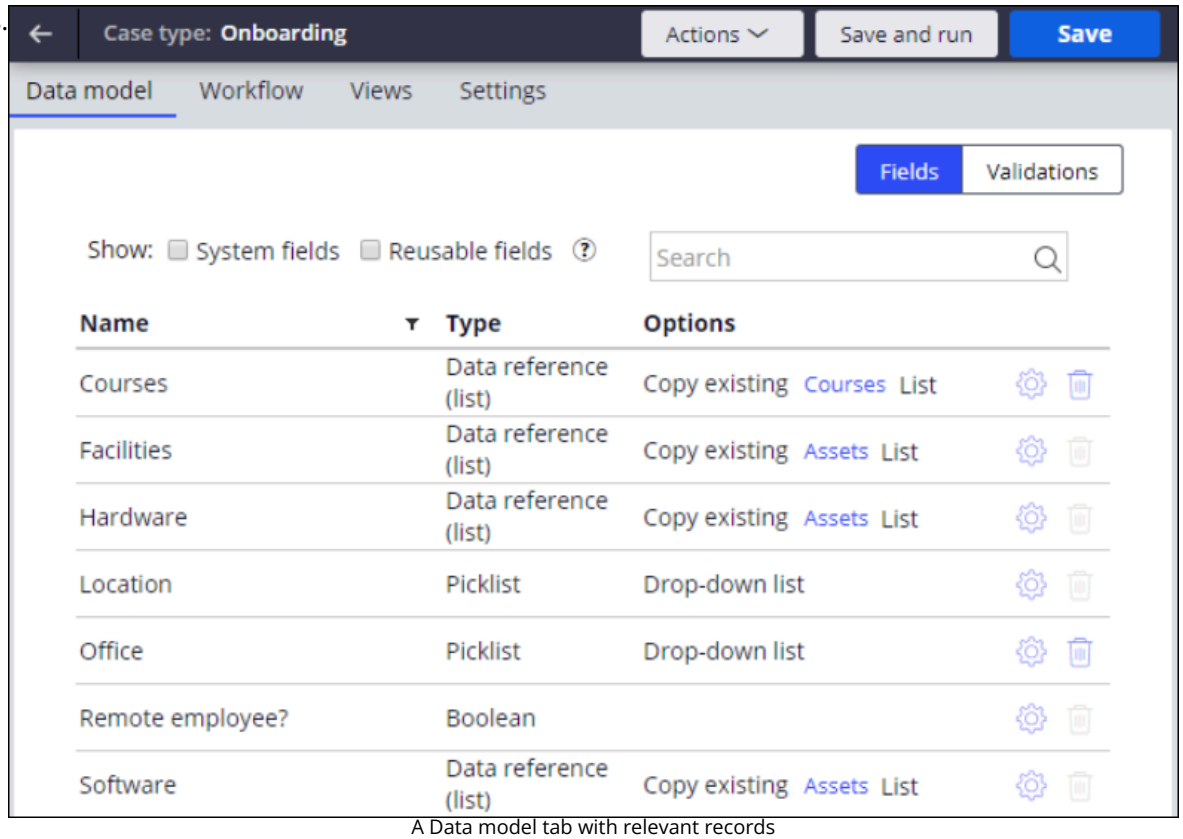
Data Designer with relevant records

Relevant records in Case Designer

You can access relevant records from several locations in Case Designer:

From the Data model tab

The **Data model** tab displays properties for the selected case type that the system marks as relevant. Use the filtering options to display reusable fields or system fields.

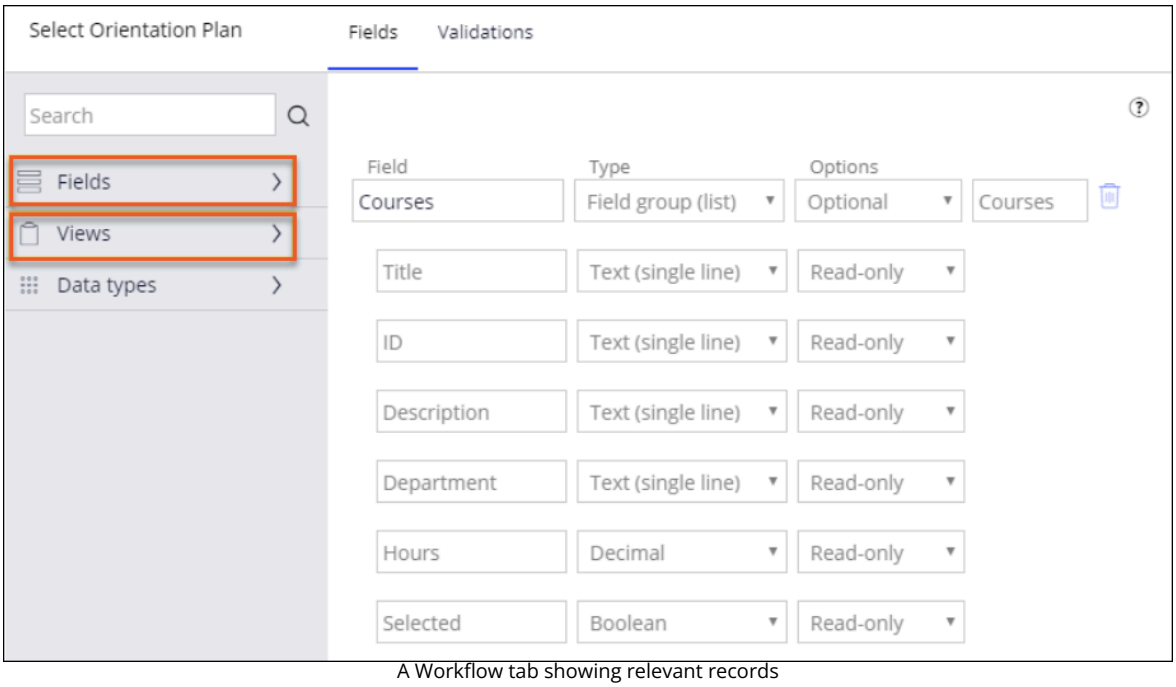


A Data model tab with relevant records

From the Workflow tab

On the **Workflow** tab, the view configuration window for a selected assignment displays records for the selected case type that the system marks as relevant. The **Fields** list displays fields (properties) that are configured on the current case type and marked as relevant records. The **Views** list displays views that are configured on the current case type and marked as relevant records.

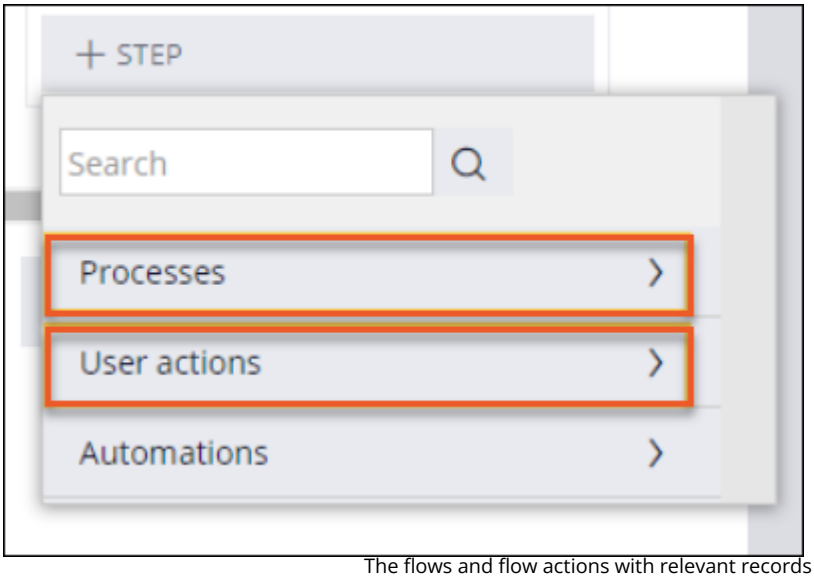
The system automatically marks new fields that you add to a form as relevant for that case type.



A Workflow tab showing relevant records

On the **Workflow** tab, the **More** section of the step menu displays flows and flow actions that the system marks as relevant records from the Process and User actions lists, respectively.

The system automatically adds new user actions that you add to the case to the relevant records for that case type as shown in the figure.

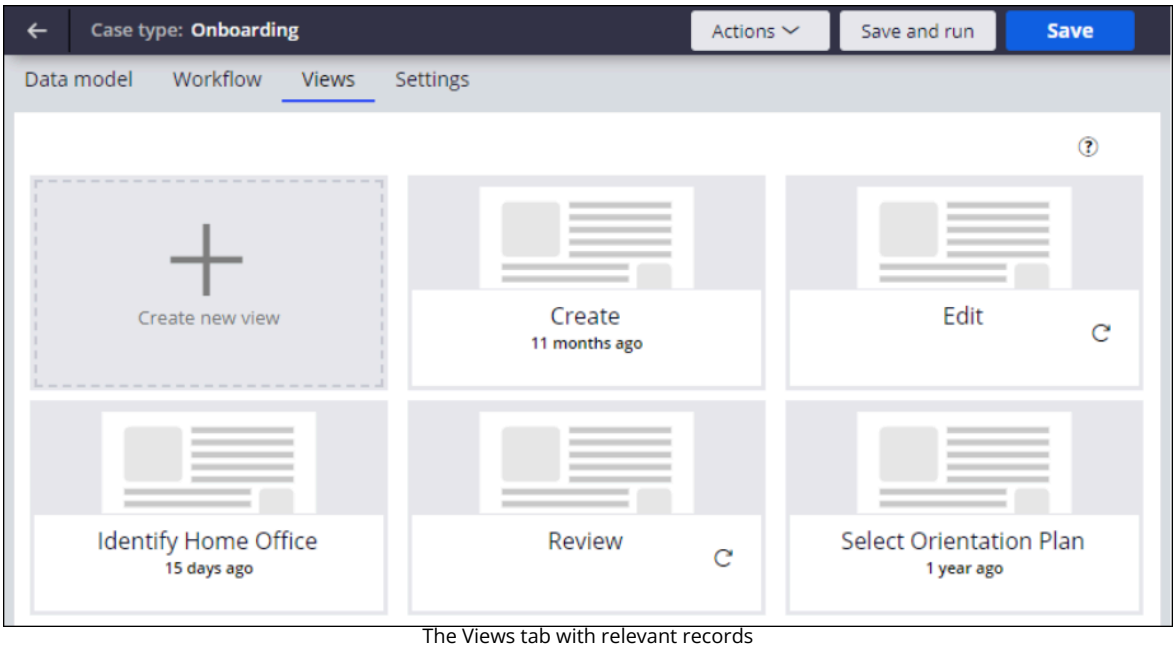


The flows and flow actions with relevant records

From the Views tab

The **Views** tab displays the views (sections) that the system marks as relevant records for the current case type.

The system automatically marks any additional views that you create as relevant to the current case type, as shown in the figure.



The Views tab with relevant records

Related information

Configuring Relevant records - a demo

Managing relevant records

Speed up your application development by curating records that you need to include in your case types and data types. Relevant records control design-time prompting and filtering in several areas of App Studio, and as a result, reduce time-consuming searching through unrelated records in your cases.For example, in an insurance application, you can mark a value field of an insured object as a relevant record so that it is available to all child classes, such as a class that stores records for jewelry insurance or a vehicle insurance case type.

1. In the header of Dev Studio, click **Configure > Application > Inventory > Relevant Records**.
2. In the **Class Name** field, enter the class of a case type or data type for which you want to display relevant records.
   By default, the system displays only active relevant records for the selected class.

   > The system displays a list of active relevant records for the specified class.

3. Do any of the following actions:
   - To display inactive records for the specified class, select the **Show inactive relevant records for the class** check box.
   - To display records that the current class inherits in the class hierarchy, select the **Show inherited relevant records for the class** check box.

> **Note:** The **Mark relevant at** column shows at which class level the particular record is relevant.

- To mark a record as active or inactive for the current class, click the **Actions** icon, and then select an option that meets your need. For example, select **Mark active for current class**.

Related information

App Studio overview *(on page 8)*
Rules management *(on page)*
Configuring a data model for a case *(on page)*
Views for cases *(on page)*

Marking a record as relevant

To save resources and speed up application development, promote reuse in data types and case types by marking a rule as a relevant record. Relevant records control design-time filtering options in case types and data types in App Studio. As a result, when you create an application, you receive a set of relevant options, instead of an extensive or incomplete selection of choices.For example, in a banking application, you can mark a property that holds an interest rate as a relevant record for a loan case type, so that it is easily available in App Studio when you build your loan application form.

You can mark only certain rule types as relevant records. For more information, see Relevant records for rule reuse *(on page 74)*.

You can mark records that are outside classes that Pega Platform provides by default, such as `@baseclass` or `Work-` classes.

Pega Platform automatically marks any records that you create in App Studio, such as properties, as records relevant to an appropriate context.

1. In the navigation pane of Dev Studio, click **Records**.
2. Open the rule form of a record that you want to mark as relevant:
   - To open a property, click **Data Model > Property**.
   - To open a data transform, click **Data Model > Data Transform**.
   - To open a strategy, click **Decision > Strategy**.
   - To open a decision table, click **Decision > Decision Table**.
   - To open a when rule, click **Decision > When**.
   - To open a correspondence rule, click **Process > Correspondence**.

- To open a flow, click **Process > Flow**.
- To open a flow action, click **Process > Flow Action**.
- To open a notification rule, click **Process > Notification**.
- To open a Pulse feed rule, click **Process > Pulse Feed**.
- To open a validate rule, click **Process > Validate**.
- To open a SLA, click **Process > Service Level Agreement**.
- To open a harness, click **User Interface > Harness**.
- To open a paragraph, click **User Interface > Paragraph**.
- To open a section, click **User Interface > Section**.

3. In the list of instances, open the rule that you want to mark as a relevant record.
4. On the header of the rule form, click **Actions > Mark as relevant record**.
5. Click **Submit**.

   > Your application marks the record as relevant for the `applies to` class of the record. The record is now available for reuse and configuration in your application.

6. **Optional:** To verify that the record is now relevant for the class, on the confirmation message, click **View**, and then review the relevant records landing page.

Related information

Configuring Relevant records - a demo
App Studio overview *(on page 8)*
Installing components *(on page 86)*
Creating rules *(on page)*
Configuring a data model for a case *(on page)*
Views for cases *(on page)*

Adding a relevant record to a specified class in your application

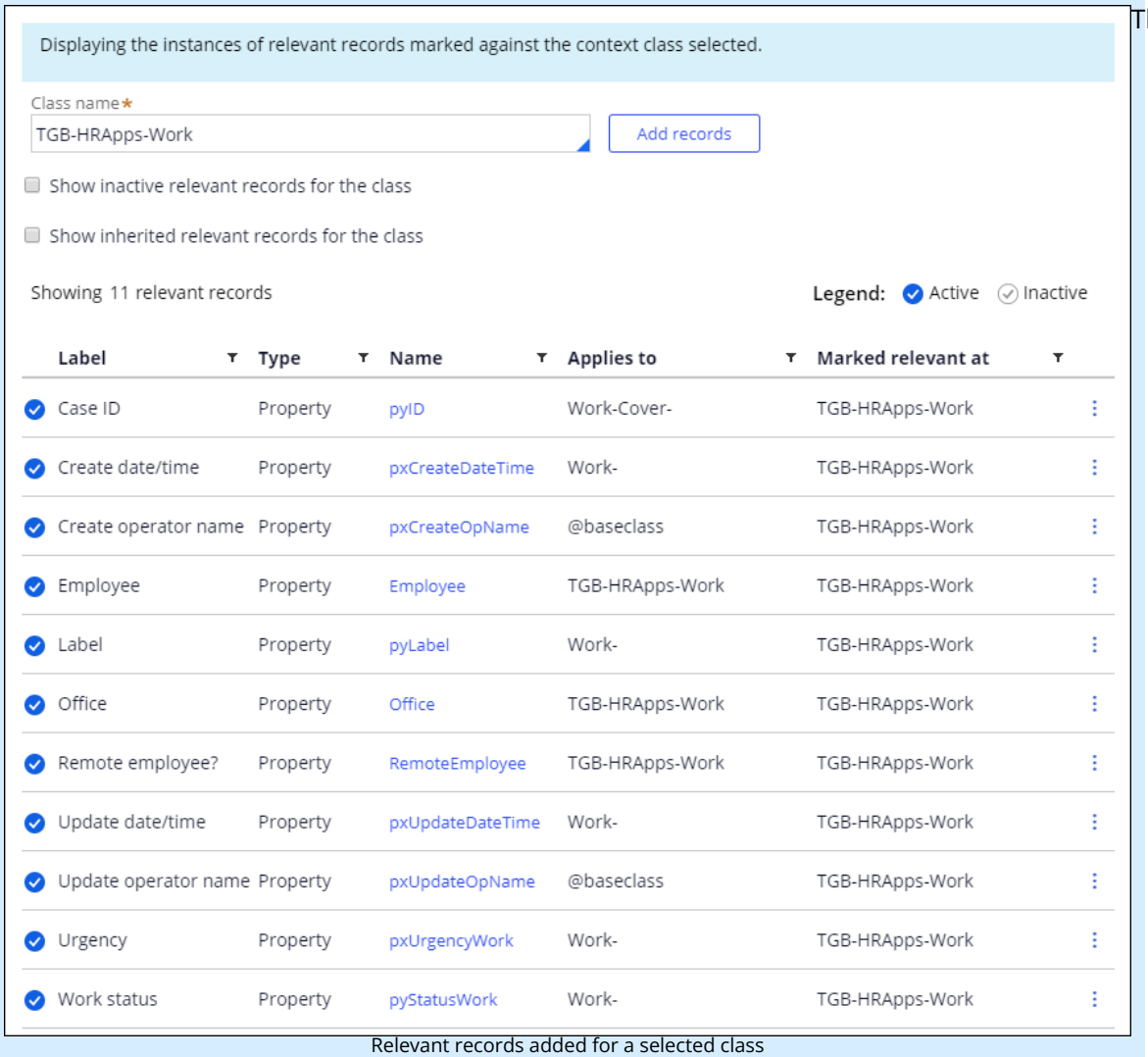
Expand the library of relevant records by adding records and classes that are most likely to be reused for a case or data type.For efficient management of resources, promote a rule from a built-on application so that you can reuse the rule in another class. By supplementing your inherited rules with relevant rules of your choice, you can create data types and case types in a quicker and more efficient way.

1. In the header of Dev Studio, click **Configure > Application > Inventory > Relevant Records**.
2. On the **Relevant Records** tab, in the **Class Name** field, enter the class of a case type or data type in which you plan to use the rule.
3. Click **Add Record**.
4. In the **Add Records** dialog box, select the type and name of the rule to reuse.

Note: For data-type classes, you can select only properties.

5. Click **Submit**.

The



Relevant records added for a selected class

relevant rule is ready for reuse in the list of relevant records that is available in the application.

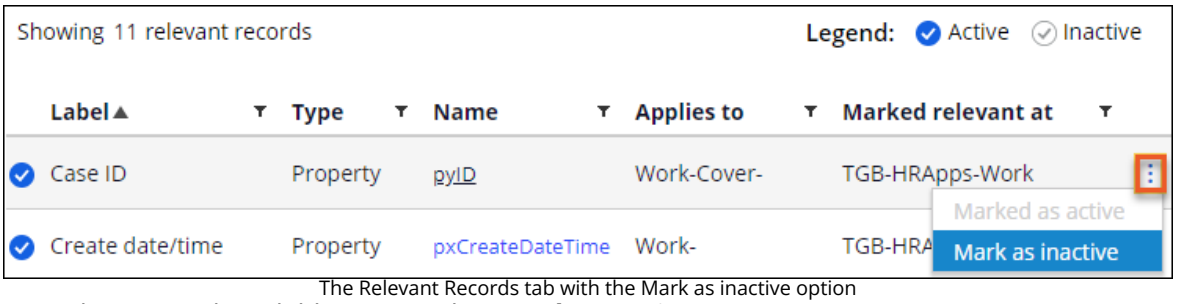
Related information

How to add a relevant record-a demo

Marking relevant records as active or inactive

Make a record available or unavailable in App Studio by marking the record as active or inactive. By marking certain records as inactive you can narrow down the list of relevant records to suit the current implementation context best, without the need to delete any records from your application.

1. In the header of Dev Studio, click **Configure > Application > Inventory > Relevant Records**.
2. On the **Relevant Records** tab, select the record of interest.
3. Click the **Actions** menu, and then define the availability of the record:
   - To make a record unavailable, select **Mark as inactive**.



The Relevant Records tab with the Mark as inactive option

   - To make a record available again, select **Mark as active**.
4. Click **Submit**.

Related information

How to mark a relevant record as inactive - a demo

Inactive Record Removal Tool

Installing components

Install a component to make a reusable feature available to applications on your system.

1. Open your Application form.
2. Click **Manage components**.
3. Select **Install new** and select the zip file that includes the component.
4. **Optional:** Browse for components on the Marketplace site:
   a. Click **Browse Apps and Components** to open the Marketplace site to search for other components.
   b. Browse the available components. When you find a component to add to your application, follow the instructions on the component page to download the component zip file to your system.
   c. Return to your application, and then select the component zip file.
5. Click **Open**.

6. **Optional:** Select **Enabled** to enable the new component for this application.
7. Click **OK**.

Adding a relevant record to a specified class in your application *(on page 84)*

Components

A component is a collection of rulesets that create a small feature that can be added to any application created in the Pega Platform.

Creating a component creates a new ruleset. As a best practice, do all component-related development in this ruleset to avoid naming conflicts. After you create a component, you can package the component and import it into another system. Then, you can enable the component in other applications on the target system.

Access the Components landing page by clicking **Configure > Application > Components**.

Creating a component *(on page 88)*
Documenting a component *(on page 88)*
Packaging a component *(on page 89)*
Installing components *(on page 86)*
Enabling components *(on page 87)*
Disabling components *(on page 90)*

Enabling components

Enable components to add the component feature to your application.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. On the **Definition** tab, in the **Enabled components** section, click the **Manage components** button.
3. Select the **Enabled** check box for each component that you want to use in your application.

> **Note:** The locked icon indicates that the component is ready to add to an application and is password-protected. Unlocked components may still be in development. Use caution when you add unlocked components to your application.

4. Click **OK**.
5. Click **Save**.

Components *(on page 87)*
Disabling components *(on page 90)*
Installing components *(on page 86)*

Creating a component

Create a component to support feature reuse in another system or application. You can control which rulesets define your component and whether your component has dependencies on other rulesets or applications.

1. In the header of Dev Studio, click **Configure > Application > Components**.
2. Click **New component**.
3. In the **Label** field, enter a name for your component ruleset.
4. In the **Version** field, enter a number that distinguishes this version of the component from previous versions.
5. Click **Create and open**.
6. Update the Component rulesets. Change the default ruleset or add rulesets to the component.
7. List required Prerequisites for this component.
8. Click **Save**.

Components *(on page 87)*
Adding a relevant record to a specified class in your application *(on page 84)*

Documenting a component

Document the components that you create to help users decide whether they can reuse your component in other applications.

1. In the header of Dev Studio, click **Configure > Application > Components**.
2. Select the component that you want to document.
3. Click the Documentation tab.
4. **Optional:** Upload an icon to associate with this component.

> **Note:** The icon file must be a .png, .jpg, or a .bmp file. The icon file size must be 3 MB or smaller. The icon must be exactly 120 pixels high and 120 pixels wide.

   a. In the Overview section, click **Upload icon**.
   b. Click **Choose File**.

c. Select the icon file, and click **Open**.
d. Click **Submit**.

5. Enter a **Description** of the component.
6. In the Key features section, enter a key feature of the component to help users understand what it does. Click **Add new** to add additional features.
7. Upload a PDF file with a maximum size of 25 MB that provides information about this component.
    a. **Optional:** Click **download template** to download a documentation template. Click **download example** to see a sample installation guide.
    b. Click **Upload**.
    c. Click **Choose File**.
    d. Select the file you want to add, and click **Open**.
    e. Click **Submit**.
8. Click **Save**.

Components *(on page 87)*
Packaging a component *(on page 89)*
Creating a component *(on page 88)*

Packaging a component

You can use a component in applications on other systems. To import a component into another system, package the component.

To package a component, follow these steps:

1. Click **Configure > Application > Components**.
2. Select the component you want to package.
3. On the Definition tab, select Locked.
4. Enter and confirm the password. You will need this password to unlock the component.
5. Click **Submit**.
6. Click **Package Component**. The system creates a zip file.
7. Click **Download** to download the zip file.
8. Save the component .zip file. Do not change the default name of the file.

Shared ruleset

A shared ruleset contains a small number of rules that each operate on a common, top-level page of a single class (or of subclasses of that class).

Shared rulesets have the following restrictions:

- A shared ruleset cannot contain a class rule.
- Rules must apply to classes that extend existing classes such as the `Work-` base class and the `Data-` base class.

Typically, shared rulesets contain only a few rules. The intent is to provide a small group of rules that provide common functions that can be shared among (unrelated) applications.

To facilitate reuse of the ruleset in multiple situations, use the keyword `Top` in property references in the rules of a shared ruleset to refer to the top-level class on which the rules operate. Do not identify this class explicitly.

When you log in, a ruleset list is developed from the information in your access group and rules that it references. Shared ruleset versions appear near the bottom of the list, just above the standard `Pega-` rulesets.

To mark a ruleset as a shared ruleset , set the **RuleSet Type** on the **Category** tab of the ruleset form to `Shared`.

To include a component ruleset in your own ruleset list, identify the ruleset in the **Enabled components** array on the **Definition** tab of an application rule referenced by your access group.

Disabling components

Disable components to remove access to the component feature from your application.

1. In Dev Studio, open the application in which you plan to use components.
2. Click **Manage components**.
3. Clear **Enabled** for each component you want to remove from your application.
4. Click **OK**.
5. Click **Save**.

Components *(on page 87)*

Creating a configuration set

Allow users of your application to change application settings at run time by creating configuration sets. As a result, users can adjust an application to their business needs in a no-code way. You also provide software that is easier to maintain and update, as configuration sets minimize the need for rule overrides and custom rule changes.

Configuration sets gather together settings called configurations. In the background, by adding a configuration, you create a property that the application references. At run time, users can change the value of the property in a no-code way to provide flexible application behavior.

1. In the navigation pane of App Studio, click **Settings > Configurations**.
2. In the header of the **Configurations** section, click **Manage configuration sets**.
3. In the **Configuration sets** dialog box, click **Add**.
4. In the **Configuration set name** field, enter a descriptive name for your configuration set, and then click **OK**.
5. **Optional:** To add more configuration sets, repeat steps 3 *(on page 91)* through 4 *(on page 91)*.
6. Close the dialog box by clicking **Close modal**.

Referencing properties *(on page 96)*
Designing applications for reuse and extension *(on page 74)*
Building logic and calculating values in your application *(on page 100)*

Creating configurations

Provide no-code tools for run-time administrators of your application by creating configurations. Configurations determine run-time behavior of your application that users can flexibly adjust to meet their unique business needs.For example, in a financial application, you can create a configuration that displays currency rates for every case that case workers process. When the business objective changes, an application administrator can disable the configuration to hide the currency rates. For more organized management of your resources, you can combine configurations into configuration sets.

1. In the navigation pane of App Studio, click **Settings > Configurations**.
2. In the header of the **Configurations** section, click **Add configuration**.
3. In the **Create configuration** dialog box, in the **Name** field, enter a name for your configuration.
4. In the **Description** field, briefly describe the purpose of the configuration.
5. In the **Configuration set** list, select the configuration set to store the configuration.
6. In the **Configuration scope** list, define elements to which the configuration applies:
    - To apply the configuration to an entire application, select **Application**.

      The configuration can apply to assets in an entire application, such as data model, UI elements etc.
    - To apply the configuration to a specific case type, select **Case type**, and then select the case type that you want to use.

The configuration can apply to assets related to a selected case type, such as a case type data model or views in the case type.

> **Tip:** You can also select all case types.

7. In the **Type** list, select a type for the configuration.
8. If you create a picklist, in the **Add choices** section, click **Add choice**, and then, in the text field, enter a value that users can select at run time.
9. **Optional:** To add more choices for the picklist, repeat step 8 *(on page 92)*.
10. In the **Default value** list, select a default value for the configuration:

Choices	Actions
Select a specific value	a. Select **Constant**. b. In the value field, select or enter a default value for the configuration.
Select a specific value for each production level (only in the Application Configuration scope)	a. Select **Constant per production level**. b. In the value field, enter a default value for the configuration. c. **Optional:** To use a different value for any of the production levels, in the value field for the selected production level, enter a new value.
Calculate a value at run time	a. Select **Calculated value from decision table**. b. In the list of decision tables, select a decision table that you want to use, or create a new decision table. For more information about decision tables, see Authoring decision tables in App Studio *(on page)*.

Choices	Actions
	When you build a decision table, the system sources the columns for the table from the data model of the scope that you select for the configuration.

11. **Optional:** To provide a unique ID for the configuration, expand the **Advanced** section, and then, in the **ID** field, enter a new value.
12. Click **Submit**.

Referencing properties *(on page 96)*
Designing applications for reuse and extension *(on page 74)*
Building logic and calculating values in your application *(on page 100)*

Adding Administration landing page to an application

To ensure that application administrators can edit values in configurations at run time, add the Administration landing page to an end user portal of your application. As a result, you provide no-code tools that users can apply at run time to adjust the application to dynamically changing business circumstances.For example, at run time, users can hide or display various UI elements in the application, such as currency rates or financial forecasts, without logging in to any design-time portal.

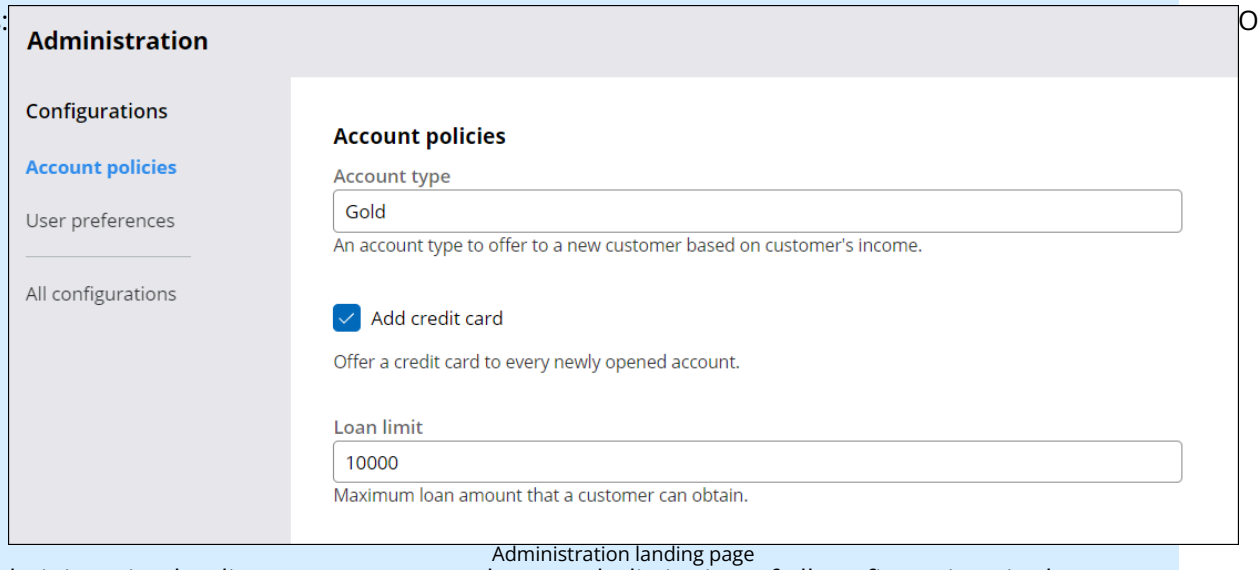
1. In Dev Studio, mark the *pxAdministration* harness as a relevant record.
   For more information, see Marking a record as relevant *(on page 83)*.
2. Add the Administration landing page to your application:
   - If you build a Cosmos React application, add the landing page to your portal.

     For more information, see Organizing the main navigation for a portal *(on page)*.
   - If you build a standard Pega Platform application, add the *pxAdministration* harness to your application.

     For more information, see Harnesses *(on page)*.

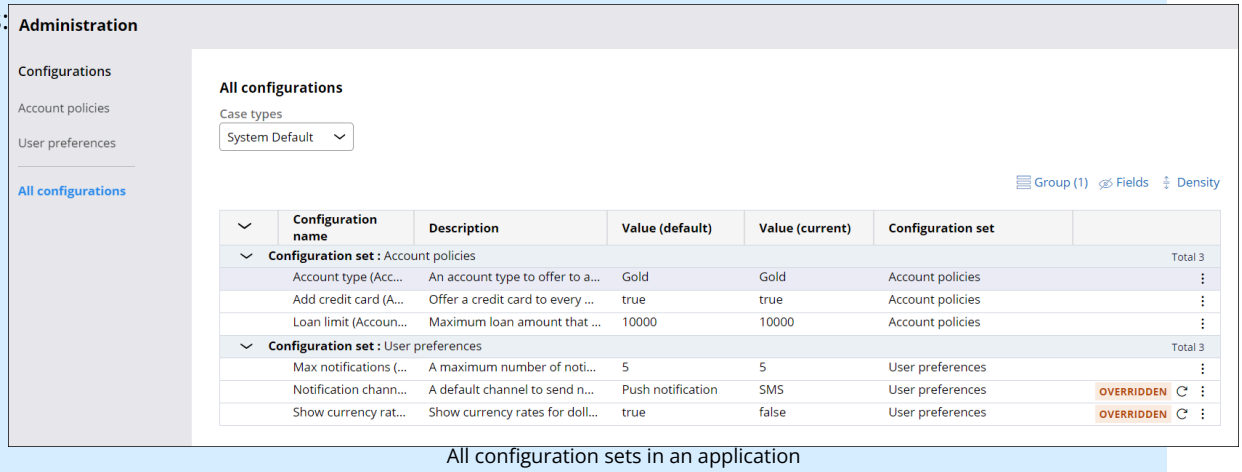
> Your application includes the Administration landing page that users can edit at run time to change the application behavior. The following image shows a configuration set with editable

fields:



Administration landing page

On the Administration landing page, users can also get a holistic view of all configurations in the application, check default and current values for configurations, and override configuration values:



All configuration sets in an application

Referencing properties *(on page 96)*
Designing applications for reuse and extension *(on page 74)*
Building logic and calculating values in your application *(on page 100)*

Technical considerations for configuration sets

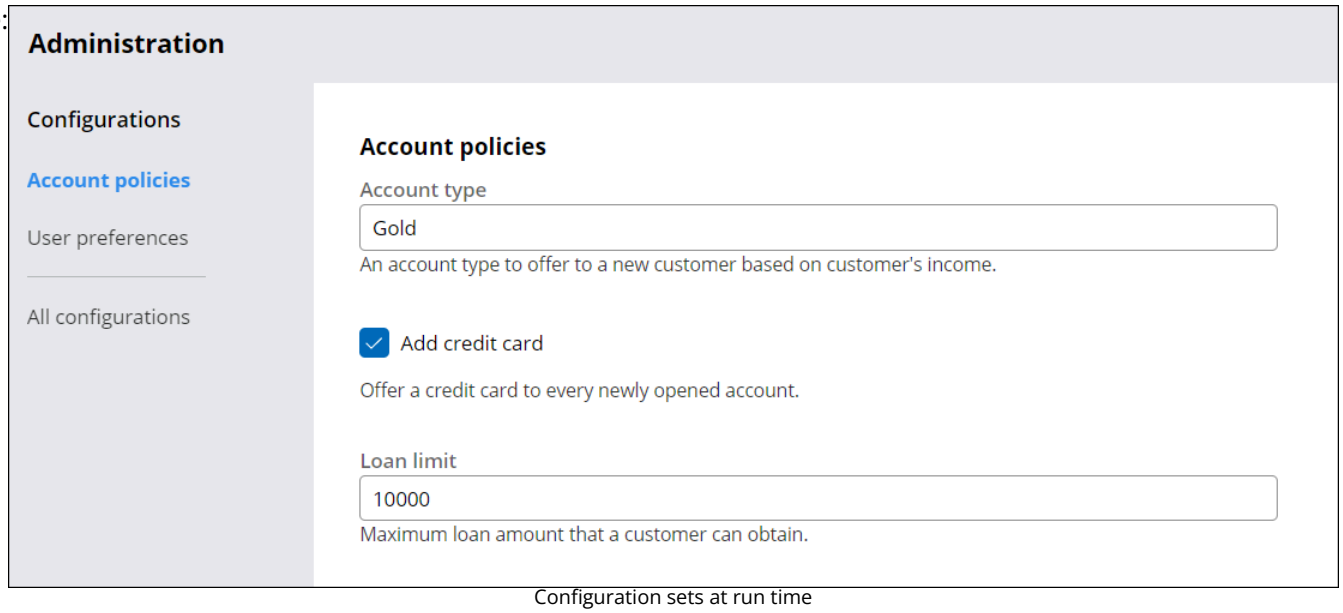
Configuration sets provide tools for run-time users to flexibly adjust application behavior in a no-code way. When you create configuration sets, considering crucial technical aspects can help you deliver an application that precisely targets business objectives. Additionally, configuration sets minimize custom rule overrides, so that maintaining and upgrading your application is more convenient.

Rule status

When you create a configuration, in the background the system creates a property whose value users can adjust at run time. You can also create a decision table to calculate the value automatically. When you move your application to the production environment, ensure that the rules that include configuration sets and decision tables are final. As a result, you ensure that users cannot edit the rules, and avoid future update issues caused by custom rule overrides. For more information, see Setting rule status and availability *(on page)*.

One configuration set per feature

For efficient and transparent organization of resources, ensure that one configuration set includes configurations that define one feature. For example, in a financial application, you can create a configuration set that gathers configurations that define the application UI, and another set for configurations related to account policies and limits. At run time, an application displays each configuration set on a separate tab. By collecting related configurations in one set, you help users quickly find the configuration that they need to edit, as in the following example:



Configuration sets at run time

Impact of access groups on the run-time experience

An application user's access group defines the actions that the user can perform on configurations at run time in the following ways:

- Users with the *ApplicationName:Administrators* access group can view, edit, and delete values for configurations.
- Users with the *ApplicationName:Managers* access group can view and edit values for configurations.
- Users with the *ApplicationName:Users* access group can view values for configurations.

To ensure that users can perform additional actions without modifying an access group, you can grant access to configuration sets to personas in App Studio. For more information, see Configuring access options for a persona *(on page)*.

The user access group also impacts the editing of decision tables at run time. Only users with the *ApplicationName:Administrators* access group can view and edit decision tables in the configuration sets. For users with other access roles, you must first edit their access groups by adding the *PegaRULES:ViewConfigTable* privilege to view decision tables, and the *PegaRULES:EditConfigTable* privilege to edit decision tables. For more information, see Adding a role to an access group *(on page)*.

Related information

Managing access roles *(on page)*

Learning about access groups *(on page 63)*

Authoring decision tables in App Studio *(on page)*

Referencing properties

Provide the data necessary to process your cases by referencing information in the form of properties. When you refer to a property, your application calls a specific piece of information, such as a customer phone number or an address.

You can reference value properties that are strings of data such as text, number, or dates. You can also reference page properties that contain multiple value properties. For example, a page property `.Customer` might include value properties `.Name`, `.Address`, and `.Phone`.

- To refer to a property, prefix the property name with a period.
- To refer to a Single Value property, enter the property name.
- To refer to an entry in a Value Group property, enter the Value Group name, and then the property name in parentheses.
- To refer to an entry in a Value List property, enter the Value List name, and then the index number of the entry in the list.

Apply the same guidelines when you reference Page properties.

- To refer to a Page property, enter the Page property name.
- To refer to an entry in a Page Group property, enter the Page Group name, and then the entry in parentheses.
- To refer to a page in a Page List property, enter the Page List name, and then the page number in parentheses.
- To refer to a specific property on a page, enter the page name as a prefix for the property name.

Data transforms *(on page 140)*

Naming conventions for properties

Creating descriptive and logical names for your properties can help you convey essential information just by looking at the name. As a result, you speed up development of your application, promote reuse across your application, and avoid duplicating properties that already exist in your system.For example, you can create *CustomerName* and *PhoneNumber* properties that clearly describe the type of data that the properties reference.

For the names of a property, choose nouns or noun phrases that clearly describe the property. Use names that are descriptive and meaningful, so that the contents of the property are easier to understand. For example, *LoanNumber* is a good choice for a numeric account number, while *LoanID* is a better choice for an alphanumeric account number. Enter the name in a camel case convention with the first letter in uppercase and then capitalize the first letter of each additional connected word.

For clarity, follow the following guidelines:

- End names for lists, that are Page List properties, with `list`.
- End names for groups, that are Page Group properties, with `group`.
- Because property names are case-sensitive, use only letters and digits.
- Do not start property names with an underscore (_) or dollar sign ($).

Standard property names

Standard property names that Pega Platform includes start with the `px`, `py`, and `pz` characters. Do not start names of your properties with these prefixes. The following table describes the purpose of each of the prefixes:

Prefix	Description
px	Computed properties that users see on a form, but cannot directly enter or change values, for example a *pxCreateDateTime* property.
py	Properties that users can explicitly enter or change, for example a *pyDescription* property.
pz	Properties that are reserved for internal use, for example a *pzInsKey* property. Users cannot see, enter, or change properties with the `pz` prefix.

Related information

Naming conventions for records *(on page)*
Naming conventions for classes *(on page)*
Rules management *(on page)*
Building logic and calculating values in your application *(on page 100)*

Configuring page, page group, and page list properties

To provide complete data for your cases, define properties that store related information in a form of a page, page group, or page list. By providing properties of a page type, you organize data in your application in a logical way, and as a result, improve application development and speed up case resolution.

When you create single page and page list properties, you also define how your application sources data for the properties. To save time, you can configure your application to refer to a data page or to copy data from a data page while providing values for properties. For greater flexibility, you can also allow users to provide values manually.

1. In the header of Dev Studio, click **Create > Data Model > Property**.
2. In the **Label** field, enter a short description for the property.
3. **Optional:** To manually set the name key part of your record to a value that is different from the default, in the **Identifier** field, click Edit, and then update the name.
   The default value of this field is `To be determined`. The system automatically populates the field with a read-only value based on the sentence that you enter in the **Label** field. The system ignores spaces and special characters.
4. In the **Context** section, select the application layer in which you want to store the record.
5. In the **Apply to** field, select the class to which this record applies.
6. In the **Add to ruleset** field, select the name and version number of a ruleset to contain the record.
7. Click **Create and open**.
8. On the **General** tab of the property form, in the **Property type** section, click **Change**, and then define a property type:
   - To create a page, click **Single Page**.

     Single pages contain embedded pages as values. For example, you can create a single page that stores information about a user that creates a case, such as a user ID and an email address.
   - To create a page group, click **Page Group**.

     Page group properties contain unordered groups of embedded pages. For example, you can create a page group that stores information about work parties, and each embedded page stores information about one work party, such as a customer or a worker.

- To create a page list, click **Page List**.

  Page list properties contain ordered lists of embedded pages. For example, you can create a page list that store multiple addresses of a customer.

9. In the **Page definition** field, enter a page class.
   The page definition stores information about what is the class of each page in this list. When you provide the page definition, the system determines what fields each page has or what rules the system can use on pages in this list.
10. If you configure a single page or page list property, in the **Data access** section, define how property sources values:

Choices	Actions
Users provide data manually at run time	Select **Manual**.
Property refers to a data page	a. Select **Refer to a data page**. b. **Optional:** If you configure a PageList, to retrieve each embedded page in the PageList separately, select the **Load each page in this page list individually** check box. c. In the **Data Page** field, enter a data page that stores values to refer.
Property copies data from a data page	a. Select **Copy data from a data page**. b. **Optional:** If you configure a PageList, to retrieve each embedded page in the PageList separately, select the **Load each page in this page list individually** check box. c. In the **Data Page** field, enter a data page that stores values to copy. d. **Optional:** To copy only selected information from the data page, in the **Optional Data Mapping** field, enter a data transform that determines what information the system copies.

11. Click **Save**.

Related information

Configuring a data transform *(on page 144)*

Building logic and calculating values in your application

Provide automations for calculating values and implementing logic so that your application can flexibly respond to unique conditions at run time. For example, you can create a process of transforming data, and, as a result, present users with relevant information when they perform work.

Creating an activity

Automate a system task for which a more appropriate rule type is not available by creating an activity. With activities, you define a sequential set of instructions, or steps, that the activity completes automatically. Each step calls a method or supported rule type to perform the required processing.Consider a scenario in which an insurance company must submit insurance claims to the Registry of Motor Vehicles. To minimize the impact on users, you can configure an activity to automate claim uploads so that your application submits insurance claims outside of peak hours, without user intervention.

1. In the header of Dev Studio, click **Create > Technical > Activity**.
2. In the **Activity Record Configuration** section, select **Activity (legacy)**.
3. In the **Label** field, enter a name that describes the purpose of the activity and helps identify the activity.
   Start activity names with a verb that indicates the purpose of the activity. Follow the verb with a noun or noun phrase that indicates an element on which the activity operates. Capitalize the first letter of each word in the name of the activity.
4. **Optional:** To change the identifier that other rules use to reference this activity, in the **Identifier** section, click **Edit**, and then enter a name that is unique within the **Apply to** class.
   Choose a name that starts with a letter and contains only letters, numbers, and hyphens. The name must be a valid Java identifier. The length of the class name plus the length of the identifier cannot exceed 128 characters.
5. **Optional:** To automatically set the activity type and, in some cases, add prepopulated steps to the form, click **View additional configuration options**, and then select one of the available activity templates:
   - To create a route activity that you can use in an assignment shape to route an assignment to a worklist or work queue by using custom routing criteria, select **Template for Route type activity for worklist**. For more information, see Assignment shapes in processes *(on page)*.

- To create a trigger activity that you can use in a Declare Trigger rule to set the values of parameters, select **Template for Trigger type activity**. For more information, see Creating Declare Trigger rules *(on page 238)*.
- To create a utility activity that you can use to automate the processing of a case, select **Template for Utility type activity**. For more information, see Calling an activity or automation from a process *(on page)*.

6. In the **Context** section, define the context in which to execute the rule:
   a. In the list of built-on applications, select an application layer for the activity.
   b. In the **Apply to** field, enter a class that you want to associate with the activity.
      At run time, the activity runs in the context of a page. The class that you specify in the **Apply to** field must be either the class of that page or in the class hierarchy of that page's class. For more information, see Understanding class hierarchy and inheritance *(on page)*.

      The list of available class names depends on the application context that you select.
   c. In the **Add to ruleset** list, select a ruleset and a ruleset version in which you want to store the activity.
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 *(on page)*.
8. Click **Create and open**.

Unit testing an activity *(on page 127)*
Creating rules *(on page)*
Branches and branch rulesets *(on page 324)*
Creating an automation *(on page)*
Methods *(on page)*

Configuring steps in an activity

After you create an activity, define a sequential set of instructions, or *"steps,"* that the activity completes automatically. Each step calls a method or supported rule type to perform the required processing.Consider a scenario in which a customer service department wants to make available the content of their support requests through a REST service. To fulfill this requirement, you can configure an activity to validate the service parameters, gather the appropriate data, and construct a JSON response.

Note: Limit each activity to 25 or fewer steps. To promote modularity and maintainability, use 15 or fewer steps and set the activity to complete one task.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Technical** category, and then click **Activity**.
3. Open the activity that you want to configure.
4. **Optional:** To temporarily exclude a step from running by commenting that step out, on the **Steps** tab, in the **Label** field, enter `//`.

   Note: An excluded loop step excludes all child steps from running.

5. **Optional:** To allow other steps to reference this step in their When or Jump criteria, on the **Steps** tab, in the **Label** field, enter a short string.
6. **Optional:** To repeat a step or a contiguous sequence of steps a number of times, click **Loop**, and then define a loop.
   The most common way to loop is to iterate over a Page List property, such as *pxResults*, by using the **For each embedded page** option. For more information, see Repeating steps in an activity *(on page 110)*.
7. **Optional:** To complete a step only if specific parameters fulfill conditions that you define, click **When**, and then define the preconditions for the step.
   For more information, see Activity form - Completing the Steps tab - Entering preconditions *(on page 105)*.
8. Click the **Method** field, press the Down arrow key, and then select a method or instruction that you want to complete.
   For more information, see Methods *(on page)*.
9. **Optional:** To set a context for the step that is different than the primary page of the activity, specify the name of the page or property that you want to use as the context.
   By default, an activity runs within the context that calls that activity. For example, an activity that a Utility shape calls during case processing runs within the context of *pyWorkPage*, which is the page that is assigned to the case type. Complete the following steps only if you want to change the default context:

   a. On the **Pages & Classes** tab, add the page or property that you want the system to use at run time.
      For more information, see Defining the pages and classes of a rule *(on page)*.

b. In the **Step Page** field, enter the name of the property that you want the step to use as context.
Depending on the scenario, you can use additional syntax. For example, if you select the **For each embedded page** loop in step 6 *(on page 102)*, enter a full property reference to a Page List property, such as *pagename.pxResults*.

10. Click **Expand to see method parameters**.
11. In the **Method Parameters** section, configure how your application applies the method or instruction that you selected in step 8 *(on page 102)*:
    - For methods, enter parameters specific to a particular method.

      For example, if you selected the **Obj-Browse** method, in the **PageName** field, enter the name of the page on which you want to store the results, and then in the **ObjClass** field, enter the name of the class that you want the method to browse.
    - For instructions, such as **Call**, **Branch**, **Collect**, or **Queue**, enter parameters specific to the rule that is associated with the instruction.
    - If you selected **Java**, in the **Java Source** field, enter Java source code that you want the activity to run.

    For more information, see Activity form - Completing the Steps tab - Entering Method parameters *(on page 104)*

    > At run time, the system passes the values that you specify to the method or instruction. The system holds the parameter names for a step on a special clipboard page that is called the parameter page, which is visible during tracing, but not with the Clipboard tool.

12. **Optional:** To specify conditions to evaluate after the method in a step runs but before the activity moves to the next step, click the **Jump** link, and then define a transition.
    For more information, see Defining step transitions in an activity *(on page 107)*.
13. Click **Save**.
14. If your activity contains Java steps, ensure that the activity causes no security issues by running the Rule Security Analyzer tool before locking a ruleset version.
    For more information, see Implementing security guidelines for custom HTML *(on page)*.
15. **Optional:** To configure another step in the activity, click **Add a step**, and then repeat steps 4 *(on page 102)* through 14 *(on page 103)*.

Creating an activity *(on page 100)*
Unit testing an activity *(on page 127)*
Creating rules *(on page)*
Branches and branch rulesets *(on page 324)*
Methods *(on page)*

Activity form - Completing the Steps tab - Entering Method parameters

The Method Parameters section allows you to pass information from an activity to a method or instruction specified in the **Method** field of a step.

Completing the Method Parameters section in a step

1. Enter a method name or instruction in the **Method** field.
2. Click the **Expand** icon to expand the step.
3. Review the fields in the Method Parameters section. These fields dynamically update based on the value you specified in the **Method** field:
    - For method names, parameters specific to the method appear.
    - For instructions, parameters specific to the rule name associated with the instruction appear. The keywords `Call`, `Branch`, `Collect`, `Queue`, `Flow-New`, or `Rule`, followed by a rule name are examples of instructions.
    - For a `Java` instruction, a text area appears where you can enter inline, Java source code.
    - The message "No parameters" appears when the specified method or instruction does not accept any parameters.
4. **Note:** Enter values in each field, using the SmartPrompt to help you select an option from a list where available. Icons appear next to a field when appropriate:

    - **Star** – Indicates the field is required.
    - **Magnifying glass** – Opens the rule referenced in the field.
    - **Build an expression** – Launches the Expression Builder.

At run time, the system passes the values specified in the Method Parameters section to the method or instruction. If applicable, a parameter page is created for the step that is separate from the parameter page of the activity. This separate parameter page is destroyed when the step completes.

In some cases, you can use the parameter page of the activity to pass information instead of explicitly setting values in the Method Parameters section. This technique is referred to as passing parameters by reference.

Passing parameters by reference

1. Ensure that the parameter values required by the method or instruction are present on the parameter page of the activity before the step executes.

   This is typically done by a previous `Property-Set` step in the activity or a rule higher up in the call stack.
2. Enter a method name or instruction in the **Method** field.
3. Click the **Expand** icon to expand the step.
4. Select the **Pass current parameter page** check box.

At run time, the parameter page of the current activity is passed to the method or instruction. This page returns to the activity when the step completes.

Passing parameters by reference allows the called method or instruction to modify the parameter page.

Creating an activity *(on page 100)*
Methods *(on page)*

Activity form - Completing the Steps tab - Entering preconditions

Use the **When** button to define a when precondition that conditionally skips execution of a step. The system evaluates whether specified conditions are met before it executes the method or instruction referenced in the step. If false, the rest of the step is skipped.

Click the button to access the details of the precondition.

Use the add row and delete row icons to define the condition. Order is significant. You can rearrange the order by selecting the row number and dragging it up or down. Click **OK** to save your edits and close the dialog.

Field	Description
Enable conditions before this action	Select to turn on the precondition test. If this box is not selected, all the rows are hidden and are ignored at runtime. The gray **When** button appears.
When	Identify a when condition rule that specifies this precondition. Or enter an expression starting with "=" that returns true or false, such as `=(.pyDoorisLocked == false).`

Field	Description
	Click the **Open** icon to review or update the when condition rule. Click ⚙ to start the Expression Builder *(on page)*. `You can use the keyword <CURRENT>` within the precondition and transition elements of a step that also includes a repeat element to evaluate the current element of the repeat.
If True	Select to indicate what is to happen if the when condition evaluates to True. Complete both an If True and an If False option for each precondition row. • `Continue Whens` — Advance to the next row of the precondition array, before executing the step. • `Skip Whens` — Skip all the later rows and execute the step. • `Jump To` — Jump to a specific higher-numbered activity step that contains a label. Entering the step's Label name in the True Param field. Processing resumes at the labeled step. You cannot jump to a label for a lower-numbered step. • `Skip step` — Skip the current step. For a repeating step, this ends the iteration. • `Exit This Iteration` — End the current iteration, and advance to the next iteration. • `End Activity` — End execution of this activity. The rest of the step is not executed.
True Param	If you selected `Jump To`, enter the Label name of the step to jump to. If you entered Exit Activity, enter an integer: `0` for a status of `Good`, any negative integer for a status of `Fail`, and any positive integer for a status of `Warn`.
If False	Choose what is to happen if the condition evaluates to false. The choices are the same as those for If True. As a good practice, complete both an If True and If False option for each row.
False Param	If you selected `JumpTo`, enter the label of the step to jump to.

Activities *(on page)*

Creating an activity *(on page 100)*

Defining step transitions in an activity

Use the blue **Jump** button to open a pop-up dialog, and identify transitions, optional fields that can end iteration, terminate the activity, or cause control to jump to a later (higher-numbered) labeled step. Use transitions to specify conditions that are evaluated after the method in the step is performed, but before the execution continues with other steps.

Often a transition condition tests whether the method in the current step was successful. See How to test method results using a transition *(on page 109)* for more on this use of transitions.

Use the add row and delete row icons to define the condition. Order is significant. You can rearrange the order by selecting the row number and dragging it up or down. Click **OK** to save your edits and close the dialog.

Field	Description
Enable conditions after this action	Select to activate the transition. If this box is not selected, all rows of the transition are hidden and are ignored at run time. The gray **Jump** button is available.
On exception, Jump to a later step label:	Optional. If you anticipate that this method might cause a Java exception, enter the label of a higher-numbered step to execute next. This can prevent users from seeing any evidence of the exception, such as the red X. This setting does not apply if an exception occurs in a precondition to the step.
When	Identify a when condition rule that the transition is to evaluate. Alternatively, enter an expression that returns true or false, starting with =. Click the **Open** icon to review or update the when condition rule. Click to start the Expression Builder *(on page)*. `You can use the keyword <CURRENT>` within the precondition and transition elements of a step that also includes a repeat element to evaluate the current element of the repeat.
If True	Select which action the system is to take when the when condition or conditions evaluate to true. Complete both an If True and If False option for each row. • `Continue Whens` — Advance to the next row of the transition. • `Skip Whens` — Skip all the other, later rows of the transition. • `Jump to Later Step` — Jump to a higher numbered step that contains the specified label. Indicate which step to jump to by entering the step's Label name in

Field	Description
	the True Param field. After the activity jumps to the specified step, it processes that step and later steps in the activity. You cannot use `Jump to Later Step` to transfer control to a label of a lower-numbered step. • `Skip Step` — End processing of this step, continuing at the next step. If the current step involves an iteration, this choice operates similarly to a break statement in many programming languages. • `End Activity` — End execution of this activity. This is similar to the Exit-Activity method, not the Activity-End method.
True Param	If you selected `Jump to Later Step` for the If True field, enter the label of a higher-numbered step to jump to. If the If True value is `Jump to Later Step` and the transition is true, but this field is empty, a Java exception occurs.
If False	Select what action the system is to take if the condition evaluates to false. The choices are the same as those for If True . Complete both an If True and If False option for each row.
False Param	If you selected `Jump to Later Step` for the If False field, enter the label of a higher-numbered step to jump to. If the If False value is `Jump to Later Step` and the transition is false, but this field is empty, a Java exception occurs.

Note:

You cannot use an iteration to jump from outside an iteration sequence to any child step within the iteration. You can jump to the parent step — first step (NNN.0) of an iteration sequence. From a child step in an iteration, you can jump to a later step within the iteration or to a later step outside that iteration.

When a method in an activity step affects the value of a property referenced in a Declare Expression rule, the declarative rule execution occurs before the system evaluates the transition. For example, if the method updates the value of a Diameter property, and a Declare Expression rule computes a CircleArea property based upon the Diameter value, only the updated CircleArea value is available to the transition.

When a step contains an enabled when-based transition, the Tracer shows a step status of "Good" regardless of the step status that existed before the transition evaluation. This status is consistent with the processing status that will be perceived by the next activity step; it reflects that any error condition that existed is "noted" by the activity.

Activities *(on page)*
Creating an activity *(on page 100)*

Testing method results using a transition

As an activity executes, the method referenced in each step (implemented in Java) is called and executes. Most methods update the `pxMethodStatus` property on the `pxThread` page with a status that starts with one of the values `Good:`, `Warn` :, or `Fail:`. These three values are prefixes to a message key that is looked up as a field value rule.

Optionally, your activities can place additional information about an error or result in the property named `pxMethodStatusInfo`.

As a best practice, test this status prefix against `Good` in those activity steps that may fail. In the When dialog, reference a when condition rule, and branch or jump to handle the failure.

In transitions, you can use three standard when conditions: `@baseclass.StepStatusGood`, `@baseclass.StepStatusGood`, and `@baseclass.StepStatusFail`. Note the following information:

- The `StepStatusFail` condition returns `True` when the status is `Fail`.
- The `StepStatusWarn` condition returns `True` when the status contains `Warn`.
- The `StepStatusGood` condition returns `True` when the status is `Good`.
- The Obj-Validate method does not usually update the `pxMethodStatus` property. Reference the standard when rule `@baseclass.hasMessages` to test whether a page contains one or more page messages.

To enter a transition:

1. Select the **Jump** button on the step row to open the condition pop-up dialog.
2. Enter the second key part of a when condition rule.
3. In the **If True** field, indicate what processing will occur when the condition is true.
4. In the **If False** field, indicate the processing to occur when the condition is false.
5. Make sure the **Enable condition after this action** check box is selected, and click **OK** to save your edits.

Note:

- The value in the `pxMethodStatus` property changes often. The value displayed when you review the clipboard using the Clipboard tool may be stale or deleted. Use the Tracer to see the current value.
- An activity step can explicitly reset the value of `pxMethodStatus` to a less severe status by the Activity-Clear-Status method.
- Using a transition in a step alters the Tracer display for the step. Normally, a red Fail row in Tracer results indicates an unhandled exception condition. If a method returns a Fail status but the step contains a transition, the Tracer row displays the status as `Good` and has a gray background. The transition mechanism is intended to allow you to catch previously unhandled Java `RuntimeExceptions`, not checked exceptions.

Repeating steps in an activity *(on page 110)*

Repeating steps in an activity

Use the **Loop** button to identify iterations, optional fields that can define an enumeration of or loop condition for this step.

Loops repeat a step or a contiguous sequence of steps a number of times, or until a condition is met. You can iterate over the elements in an aggregate property — a `Value List`, `Value Group`, `Page List`, or `Page Group`.

This feature provides a limited form of enumeration or looping, similar to the "repeat until" or "do while" or "repeat from X to Y" structures of programming languages.

Note: In a child step within a multistep loop, right-click to access a context menu. You can copy, cut, or paste child steps within the same iteration.

Repeating a single step using the loop field

Click the **Loop** button to access the details of the iteration. Use the pop-up dialog to indicate whether and how to repeat the step. You can cause the step to repeat a specific number of times, or until a specific result is reached. Click **OK** to save your edits and close the dialog.

Use the Repeat selection box to apply the step to more than one page or object. Select:

- `For each top level page`
- `For each embedded page`
- `For each element in value list`
- `For each element in value group`
- `For loop`

To exit the iteration before processing all elements in the iteration, complete a transition that is true after the last desired element is processed.

When processing the `For loop` or iterating over a `Value List` or `Page List` property, the parameter `param.pyForEachCount` holds the current iteration number. You can examine (but not alter) this value in the step. For example, to iterate only over the first ten pages or values, you can exit the iteration when `pyForEachCount` is greater than 10.

> **Note:** If the body of the iteration requires more than one step to code, create an activity and call the activity in the iterated step. Select the **Pass current parameter page?** check box to share the parameter page of the current activity with the called activity.

For each top level page

Select the `For each top level page` option to sequence through all pages of a specified class or classes and perform a method or instruction for each top level page. Leave the Step Page field blank. Each time the step repeats, the step's page changes.

For the optional Only Loop for Certain Classes parameter, enter a class or classes. (Click the Add Row icon to add more than one class.)

When Only Loop for Certain Classes is not blank, iteration processing skips over any page with a class that does not appear on the list, or is not derived from one of these classes.

Use the add row icon to add a class. You can rearrange the order by selecting the row number and moving it up or down.

For each embedded page

Use `For each embedded page` option to apply a method or instruction to each embedded page in a `Page List` or `Page Group` property. Use the iteration choice to specify processing that is to occur for each embedded page. For example, you can:

- Copy each embedded page
- Delete each embedded page using Page-Remove
- Delete a property on each embedded page
- Add a property to each embedded page

and so on.

Identify the property containing the embedded pages in the Target field. The SmartPrompt list for the Target field shows only `Page List` and `Page Group` properties.

For the optional Only Loop for Certain Classes parameter, enter a class or classes. (Click the Add Row icon o add more than one class.)

- When Valid Classes is blank, all pages of the `Page List` or `Page Group` are included in the iteration.
- When Only Loop for Certain Classes is not blank, iteration processing skips over any page with a class that does not appear on the list, or is not derived from one of these classes.

For each element in a value list

Select the `For each element in value list` option to repeat the step for each element in a `Value List` property.

Each time the step repeats, the system applies a method or instruction of your choice (such as Property-Set) to the current property. Use the `<CURRENT>` index value to refer to the current property element.

When you select this iteration form, a Value List Property field appears. Identify the `Value List` property in the field.

For each element in a value group

Select the `For each element in value group` option to repeat the step for each element in a `Value Group` property.

When you select this iteration form, a Value Group Property field appears. Identify the `Value Group` property in the field.

Each time the step repeats, the system applies a method or instruction of your choice to the current property. Use the `<CURRENT>` index value to refer to the current property.

For loop

Select the `For loop` option to repeat the step a number of times determined by the values of integer constants or integer properties.

Enter integer constant values or integer property references for the Minimum Value, Maximum Value, and Increment fields. The Increment must be a positive integer.

The system accesses these three values once before starting the iteration. The iteration may occur zero, one, or many times, repeating until the incremented value is greater than or equal to the stop value (unless a transition causes processing to end earlier).

Understanding `<CURRENT>`

The symbolic reference `<CURRENT>` can be used only in the activity step that defines the iteration. If that step calls another activity, the symbolic reference `<CURRENT>` cannot be used in the called activity; rather, aspects of the current iteration are available on the parameter page of the called activity:

- For a repeat type of `For each element in value list` or `For each element in value group`, the symbol `<CURRENT>` refers to the value of ClipboardProperty corresponding to the iteration. For the called activity, this corresponds to the `pyPropertyValue` parameter described below.
- For a repeat type of `For each embedded list`, `<CURRENT>` refers to the `ClipboardPage` of the current iteration: for example, when used in an RUF call that takes a `ClipboardPage` parameter or on the right side of a property set when the target is a page (when you add to a page list using `.list(<append>)`).
- For other repeat types, `<CURRENT>` refers to the index number of the current iteration, and corresponds to the `pyForEachCount` parameter.
- `<CURRENT>` can be used within the precondition and transition elements of a step that also includes a repeat element to evaluate the current element of the repeat.
- Within a repeat block, `<CURRENT>` can appear in the Expression Builder for the **Target** field of a Property-Set method; the keyword will refer to the current element's property so that it can receive the new value.

Multistep loops

The scope of a loop can be more than one step. The loop type and conditions are set in the first step of a multistep loop.

To create a multistep loop:

1. Identify the step that is to be first in the loop. Click **Loop** to access the pop-up dialog, and complete the loop details.
2. Complete information for the first step. Identify the step page. Optionally, you can leave the method blank.
3. Select the stage, open the right-click context menu, and select *Add Child*. The current step is renumbered as NNN.0, where NNN was the previous step number. The form changes to include a

new later step NNN.1. The scope of the iteration is now both NNN.0 and NNN.1. This zero-numbered step is known as the parent step.

4. Complete NNN.1, a child step. Leave the Step Page field blank and the iteration criteria blank in most cases, to have the child steps operate on the step page identified in the parent step. Optionally, you can override this step page of a child step by entering one of the following:

- a top-level page (such as the primary page).
- a reference to a page relative to the step page of the parent.
- a reference to a `Page List` or `Page Group` property on the step page of the parent. (In this case, complete the `For each embedded page` iteration details for the child step.)

1. To create an iteration of more than two steps, click within the parent step NNN.0 again to make it active, and select the *Add Child* menu option gain.

Numbered child steps are executed in sequence NNN.0, NNN.1, NNN.2 ... then NNN.0 again, until the iteration ends or a transition causes a jump to a labeled step.

You can add addition child steps at the same level by placing the mouse pointer in the Description field and pressing the Enter key.

To convert an iteration sequence to comments, enter two slash characters // in the Label field of the parent step. The parent and all child steps become comments.

> **Note:** You cannot jump from outside a multistep iteration to any step within the iteration (except to the starting — parent — step). You can jump from any step in an iteration (parent or child) to a higher-numbered step (that is not within any iteration).

For an example of multistep and nested iterations, see the Pega Community article *How to use multistep loops and nested loops in activities*.

Nested iterations

By using the *Add Child* menu option on a child step numbered NNN.Z, you can create a nested iteration.

For example, clicking a child step numbered 14.8 converts this to 14.8.0, and inserts a second-level child step numbered 14.8.1. Complete the iteration conditions for step 14.8.0.

Use care in entering the Step Page for the parent step of a nested iteration (14.8.0 in this example). If you refer to an aggregate property (Value Group, Value List, Page Group or Page List), select a property that reside on the `<CURRENT>` step page of the parent's parent. (That is, the structure of the iteration must follow

the structure of the page). Within a nested step that does not have a repeat element, `<CURRENT>` inherits the semantics of its parent step that does have a repeat element.

> **Note:** From within a nested iteration, you can jump (using a transition) to a labeled step that is within an outer iteration but higher numbered. You can jump from 14.8.9 to 14.10 or 17, but not from 14.8.9 to 14.3.

Restrictions

If a step is part of an iteration sequence, the method cannot be any of the following:

Connect-Java	Page-Change-Class
Obj-Open	Page-Merge-Into
Obj-Open-By-Handle	Page-New
Obj-Sort	Page-Validate
	RDB-Open

Parameters available to called activity

The Pega Platform automatically adds the following parameters to the parameter page of a called activity with a step that contains an iteration.

- `pyForEachCount` — Index of the current iteration, normally starting at 1 and incrementing by one. (If the iteration type is `For loop`, the starting value may be a value other than 1 and the increment may be a value other than 1.) This parameter is available throughout the iteration as well as in called activities.
- `pyIterationType` — Indicates the iteration type, internally represented by one of the constants `page`, `embedded`, `propertylist`, `propertygroup`, or `repeat`. (This parameter values are not available to other methods in an iteration step.)
- `pyIterationTarget` — Identifies the name or reference of the iteration. For page and repeat types, it is blank. For embedded, it is the step page reference. For `propertylist` or `propertygroup`, it is the property specified in the iteration. (This parameter values are not available to other methods in an iteration step.) (This parameter values are not available to other methods in an iteration step.)

- `pyPropertyValue` — Available only for `propertylist` and `propertygroup` iterations. The text value of the current property. (This parameter values are not available to other methods in an iteration step.)
- `pyPropertyReference` — Obsolete. Set to "???" for backward compatibility.
- `pyPropertyType` — Obsolete. Set to "???" for backward compatibility.

Activities *(on page)*
Creating an activity *(on page 100)*

Calling an automation from an activity

Call an automation from an activity without using a Java step.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Technical** category, and then click **Activity**.
3. Click in the row of the activity from which you want to call an automation.
4. On the **Steps** tab, in the **Method** field of a step in the activity, select **Call-Automation**.
5. Click the arrow to the left of the **Method** field to expand the step and view the method parameters.
6. In the **Resource** field, select the type of automation that you want to call.
7. In the **Automation** field, select the automation that you want to call.
8. Complete the **Map from** and **Map to** fields for the inputs and outputs for the automation you are using.
   This establishes the relationship between the resource inputs settings, and the resource outputs generated automation errors. You can view a description of these fields by viewing the automation rule. For more information, see Viewing automations *(on page)*.

   You do not have to use all of the outputs. To use an output, you must provide a property reference that can hold that type of output, for example, a string property for a string output. Like inputs, outputs are also validated. Errors that occur when the automation runs are returned in the `Automation Errors` output. If an error occurs that is not in the automation rule, it is converted to a run-time error before being returned.

   > **Note:** Page type outputs expect a property type string. A page reference is stored in that property so that you can return to that page later

9. If you specify a default value in the automation, in the **Inputs** section, a **Use default value** option is displayed next to the corresponding **Map from** field. You can use the default value or specify a new default value:

- To use the default value specified in the automation, click **Use default value**. The value in the **Map from** field becomes read-only, and the **Use default value** option changes to **Specify value**.
- To specify a new default value, in the field for which you want to change the value, press the Down arrow key to select a value.

> **Note:** If a list of values is specified for a text input in the automation rule, only those values are available in the **Map from** field.

10. Click **Save**.

Call-Automation method *(on page)*
Automations *(on page)*

Best practices for writing activities for background jobs

Ensure that work items are processed correctly and that data is not corrupted during a node shutdown by using best practices for writing activities for background jobs.

Queue processors, job schedulers, and agents are background jobs that use an activity to implement processing logic. These activities are expected to finish processing in less than a few seconds. If an activity takes longer to complete, use the following best practices to ensure that work items are not affected during the shutdown process.

Create small work items

If possible, break down large work items into smaller work items, queue them to a queue processor, and process them individually. Use a queue processor instead of an agent because queue processors provide better performance, resiliency, fault tolerance, and flexibility than agents. Scale queue processors horizontally by increasing the number of nodes configured with the node type. Scale queue processors vertically by increasing the number of threads. For more information, see Managing queue processors in Admin Studio help.

Use checkpoints

To ensure that activity logic can gracefully handle restarts without corrupting data or introducing inconsistencies into your application, use checkpointing by saving the state of jobs in persistent storage or as messages in a queue that is appropriate to the use case.

For example, consider a scenario where a work object can have the following statuses: NEW, IN-PROGRESS, and PROCESSED. You can read all of the work object records to be processed, loop through them one by one, mark the status of the current work object as IN-PROGRESS, continue with the business logic, and when processing is finished, change the status to PROCESSED.

Divide and scale tasks

If you have background tasks that can be broken down into multiple tasks that have different performance capabilities, consider dividing them and scaling each independently. Queue processors can be scaled horizontally by increasing the number of relevant nodes and can be scaled vertically by increasing the number of threads. For more information, see Managing queue processors in Admin Studio help.

Create sub-activities

Split activities into sub-activities when possible so that the state is saved frequently, ensuring that if there is an abrupt node shutdown, the sub-activities can proceed based on the recovered state.

For more information about agents, job schedulers, and queue processors, see the Admin Studio help.

Activities *(on page)*

Security tab on the Activity form

Use the Security tab to specify the activity type and optionally to restrict which users (or other requestors) can execute the activity. This optional security supplants restrictions based on ruleset and version.

You can specify zero, one, or more than one privilege to restrict access. Order is not significant. At run time, any match between a privilege listed and those a user possesses allows users to execute this rule.

Field	Description
Restrict access	
Allow invocation from browser	Select to allow users to start this activity directly through user input processing, for example through a Submit button or a `pyActivity=` element in an URL. Clear this if this activity is to be started only from another activity, through a Call, Branch, or other means. For example, select the box for a service activity, or if this activity is called by an AJAX event from a form.

Field	Description
	At run time, if the box is not selected and a user attempts to start this activity by user input, the activity does not run and returns a method status of `Fail:Security`. For most activities, leave this box cleared to promote security of your application. Unless needed by your design, allowing activities to be started from a URL or other user input — whether the requestor is authenticated or a guest — may let users bypass important checking, security, or setup.
Require authentication to run	Select to require that only authenticated requestors can start this activity. Clear to allow guest users to run this activity, if they meet other security and access criteria. Guest users — unauthenticated requestors — typically have access to rules in the RuleSets provided in the `PRPC:Unauthenticated` access group, as referenced in the Requestor type instance named `pega.BROWSER`. ⚠ **CAUTION:** If you update the `BROWSER` requestor type to reference a different access group, or update the `PegaRULES:Unauthenticated` access group to make additional rulesets available to unauthenticated users, review carefully this check box for each activity in the rulesets. Select this check box for all but those specific activities that guests need to run. In most cases, clear this check box if the activity is for an agent. Agents are not true authenticated users and by default cannot run activities that are restricted to authenticated users. However, this check box is ignored by agents for which the Bypass activity authentication check box (on the Security tab is checked; they can run activities regardless of the Authenticate? value. Identify privileges in this array to restrict which users and other requestors can execute this activity. At run time, if the user does not possess an access role that — through an Access of Role to Object rule — provides access to one of the identified privileges, the execution of the activity fails.
Privilege Class	Optional. Identify the Applies To key part of a class to use at run time to locate a privilege rule. Normally this is the same as the Applies To key part of this activity.

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>Privilege Name</td><td>Optional. Identify the name for a privilege in that class (or an ancestor class). The class you enter and the name must together identify a privilege (using rule resolution including class inheritance.)</td></tr>
<tr><td>Usage</td><td></td></tr>
<tr><td>Activity Type</td><td>Determine whether and how this activity can be referenced in other rules. For an activity that is not to be referenced in a flow, choose one of these values:
<table>
<tr><td>Activity</td><td>Select when no more specific value is applicable. Activities with this value cannot be referenced directly in flows. (Select <code>Activity</code> for this field on Parse Structured forms.)</td></tr>
<tr><td>Asynchronous</td><td>Select for an activity that runs in a background thread asynchronously. The activity can call only these step methods: Load-DataPage, Connect-Wait, and Call-Async-Activity (step instruction to allow user to load an asynchronous activity)</td></tr>
<tr><td>Load-Declarative-Page</td><td>Select for an activity that adds values to data pages. Reference this activity on the Definition tab of a data page rule (<code>Rule-Declare-Pages</code> rule type).</td></tr>
<tr><td>Locate</td><td>The Locate activity type is not supported. Existing activities that use locatable pages display this option. New activities do not.</td></tr>
<tr><td>On-Change</td><td>Select for an activity to be executed automatically by a Declare OnChange rule, Such activities must not use any methods that directly change the properties or the database.

Declare Expression rules do not evaluate during the execution of an On-Change.activity. OnChange activities must not perform any forward chaining themselves.</td></tr>
<tr><td>Trigger</td><td>Select for an activity to be executed automatically by a Declare Trigger rule. Because triggered activities run during database commits, they cannot themselves commit database transactions.</td></tr>
</table>
Select <code>Notify</code>, <code>Rule Connect</code>, <code>Assign</code>, <code>Route</code>, or <code>Utility</code> if the class of the activity inherits from the Work- base class and you designed and implemented the activity to be referenced in a flow shape. See How to create activities for use in flows (on page 122) for more details on these choices:</td></tr>
</table>

Field	Description
	Choose `Notify` for a notification activity. For example, a notify activity can send an email message to someone conveying news of the assignment.
	Choose `Assign` for an assignment activity, one that creates an assignment.
	Choose `Route` for a router activity, one that determines which user worklist or work queue is to receive an assignment.
	Choose `Utility` for a utility activity, one that automates processing without user interaction.
	Choose `Rule Connect` for an activity that operates without user interaction and calls a connector rule to interface with an external system.
	Do not choose `Validate` or `Assembler` for this field.

Activities *(on page)*
Declare OnChange rules *(on page 225)*
Declare Trigger rules *(on page 236)*
Activity form - how to create activities for flows *(on page 122)*

Defining the local variables for an activity

Create scalar variables that you can use to pass information between the steps of an activity. Local variables require less memory and run faster than regular parameters.

For example, you can simplify a highly nested expression by using local variables to break up that expression into component parts. As a result, the expression is easier to read and troubleshoot.

Your application stores local variables together as fields in the Java class that Pega Platform generates to implement the activity. Local variables do not appear on the parameter page.

You can reference a local variable in a Property-Set method by using the `Local` keyword, as well as in Java steps.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Technical** category, and then click **Activity**.
3. Open the activity that you want to configure.
4. Click the **Parameters** tab.
5. In the **Local variables** section, in the **Name** field, enter a unique identifier for the local variable.

You can choose any valid Java identifier. Do not choose a name that starts with pz or any of the reserved names.

6. In the **Description** field, enter text that describes how your rule logic processes the local variable, so that other developers can understand that rule more quickly.
7. In the **Data type** list, select the type of data that the value for the variable has:
    - For a `Java.lang.string` object, select **String**.
    - For a Java `StringBuffer` object, select **StringBuffer**.
    - For a Java integer, select **Integer**.
    - For a Java double, select **Double**.
    - For a Java Boolean, select **Boolean**.
    - For a Java char, select **Char**.
    - For the `BigDecimal` class, select **BigDecimal**.
    - For any Java object, select **Object**.

> **Note:** An application can assemble a local variable of the StringBuffer type in multiple steps, because a Property-Set method on a string buffer class appends the value to the string buffer.

> **Note:** To avoid Java null pointer exceptions, the system initializes `String` local variables to the empty string, and the `StringBuffer` local variable as: `public StringBuffer aBuffer = new StringBuffer();`

8. **Optional:** To define another local variable, click **Add item**, and then repeat steps 5 *(on page 121)* through 7 *(on page 122)*.
9. **Optional:** To use a flow action at run time that produces a prompt form in which the user can enter values for the flow parameters that are specified in this tab, in the **Local action for parameter display** section, specify the flow action.
   For more information, see Prompting users for parameter values *(on page)*.

Activity form - how to create activities for flows

Use these instructions to create activities that can be called directly in flows.

Flow processing automatically controls locking and transaction boundaries for work items. Do not use the Commit method (or the Obj-Save method with the **WriteNow** parameter selected) in any custom activity called by a flow.

Activity Type

The value of the **Activity Type** field on the **Security** tab determines which flow shapes accept this activity. To create an activity for a flow, set the **Activity Type** field to `Utility`, `Rule Connect`, `Assign`, `Notify`, or `Router` as appropriate.

First, explore the standard activities of the type you need, as examples and as starting points for your activity.

In all but rare situations, choose a class derived from the `Work-` base class for the **Applies To** key part of your activity. This makes your activity available to flows in that class plus all subclasses of the class (subject to security restrictions, rule resolution restrictions and so on).

Guidelines for Utility activities

Utility 	Follow these guidelines to create a Utility activity, one that can be referenced in a utility shape: • Select `Utility` as the **Activity Type**. • In the activity, do not use the Show-HTML method or interact with a user. • The activity can save a new or updated object with the Obj-Save method. • Optionally, this activity can use the TaskStatus-Set method to return a text value in the `pxTaskStatus` property to the flow as a basis for decisions. Connectors leading from the activity can be based on this status value. Using this method can simplify flow diagrams by eliminating fork or decision shapes that follow a utility shape.

Guidelines for Assign activities

Assign 	Follow these guidelines and constraints to create an activity of type `Assign`, one that you can reference in an assignment shape: • In the activity, do not use the Show-HTML method or other means to interact with human users. • Create and save an instance of a concrete class derived from the `Assign-` base class. Typically, a new assignment is an instance of the `Assign-Workbasket` or `Assign-Worklist` standard class.

Guidelines for Integrator activities

Integrator 	Use an **Activity Type** value of `Rule Connect` to make the activity available to an Integrator shape. Generally, use this for activities in a `Work-` class that call a rule to start a connector interface to an external system. This activity type (and the corresponding Integrator shape) is a reminder that this shape depends on an external system. Response time, availability, and performance may be affected by outside factors. Like the `Utility` activity type, a `Connect` activity type cannot include HTML displays.

Guidelines for Notify activities

Notify	Select an **Activity Type**of `Notify` to identify activities in a `Work-` class that generate correspondence. Correspondence generation can occur without any user interaction, or can capture user input in a simple HTML form, or may start Microsoft Word on the user desktop.

Guidelines for Route activities

Route	Follow these guidelines to create an activity with **Activity Type** of `Route`, one that can be referenced in a router shape: • The results of the activity are a work queue name, an Operator ID (for the worklist) or an agent name. • The activity does not display HTML forms or interact with a user. • The activity returns its results in an output parameter named `AssignTo`. This parameter is an output only, and does not need to be declared on the Parameters tab. Many routing activities accept these Boolean input parameters:

Parameter	Description
`CheckForAvailability`	If this parameter is true, the activity is to check Operator ID instances for availability, and follow the rules to obtain a substitute if the operator is not available. (Standard routing activities support this parameter.)
`SwitchToWorkbasket`	Allows an assignment to be routed to either a work queue or worklist. Use this Boolean output parameter in a router activity that is referenced in assignment

Parameter	Description
	shapes where the **Rule** field on the **Assignment Parameters** panel is set to `Worklist`. When `SwitchToWorkbasket` is set to false, the `AssignTo` parameter identifies an Operator ID for a worklist; when set to true, the `AssignTo` parameter is to identify a work queue name.
`Passthrough-Assignment`	Allows creation of an assignment instance to be bypassed, with the flow execution continuing with a default flow action — the most likely flow action — that requires no user input. Set this Boolean output parameter to false in the normal case, or true to skip the assignment and continue flow execution as if the user chose the highest-likelihood flow action.
`UsesSkills`	Influences the **Properties** panel to allow the entry of one or more skill names (literal values or property references) and skill levels (integers). A check box for each skill can mark the skill as required to perform an assignment, or (if not selected) desirable to perform the assignment. The activity can call standard functions to test whether a specific operator possesses the required skills, or possesses the desired skills, at the indicated level of proficiency.

The standard `Routing` library (in the Pega-ProCom RuleSet) contains several functions useful in router activities.

Available parameters

When a flow calls an `Integrator`, `Utility`, `Notify`, or `Assign` activity, or another flow, it provides twelve parameters to the activity beyond those declared in the **Parameters** tab. Click the **Show System Parameters** button on the **Parameters** tab to review the names and purpose of the six parameters most often used.

Parameter	Description
`flowName`	The subscript of this flow in the work page `pxFlow` property, a `Page Group`.

Parameter	Description
`TaskName`	The shape name of the shape that this activity or flow was called from, such as SendResolutionEmail or SplitForEach999.
`pyDraftMode`	True if the flow is in draft mode.
`ReferenceInsKey`	`pzInsKey` of the work item page.
`ReferenceClass`	Class of the work item page (the work type).
`ReferenceInsName`	`pxInsName` of the work item page.
`ReferencePageName`	Clipboard page name of the work item page.
`TimeFlowStarted`	Date and time that the flow execution began.
`InterestPage`	Property reference to the embedded page of the work item that is the primary page of the flow. (This is null if the flow's primary page contains the work item.) For example, a flow executing on the `Customer` work party has an interest page of `.pyWorkParty(Customer)`.
`InterestPageClass`	Class of the interest page, for example `Data-Party`. This is an empty string if the flow's primary page is the work item itself.
`FlowHasEnded`	A Boolean used by internal flow processing to control whether one flow calling another is to wait or to continue.
`flowType`	Second key part of the flow.

Activities *(on page)*

Intelligent routing

Intelligent routing is the process of comparing the characteristics of a new assignment with the characteristics of the workforce to route the assignment to the most appropriate operator. Like a supervisor who thoughtfully distributes work to her team, intelligent routing in your application can significantly affect the productivity and throughput of a team.

An intelligent routing algorithm can examine many factors, such as backlogs, the presence or absence of operators, operator skills, the urgency or priority of an assignment, and a customer's location or time zone. Your system includes the following three standard activities as examples:

- `Work-.ToSkilledGroup` — Implements skill-based routing. This activity sends an assignment to a randomly selected operator within a specific team. This operator is available between the time the assignment starts and is due, and has the minimum skill proficiencies to complete the assignment. Using indexes maintained in the `Index-OperatorSkills` class, this activity operates efficiently.
- `Work-.ToLeveledGroup` — Sends an assignment to an operator within a specific team who has the least urgent total worklist, based on a computed score.
- `Work-.SampleToSkilledGroupList` — Demonstrates use of the `getAvailableSkilledOperatorsList()` function. Part of the `Routing` library, returns a list of operators who are available and meet the required skill requirements. Your routing algorithm can apply additional filtering and criteria to select which of the operators on the list receives the assignment. (If none of the operators in the group meet the standard function qualifications, the list contains only the team manager's Operator ID.)

Skill-based routing is an important type of intelligent routing. Skill-based routing compares the skill proficiency of operators in a team with the required and desired skills to perform an assignment accurately and quickly. For example, an assignment might require an operator who passed the NASD Series 7 exam, and might prefer that the operator have a speaking proficiency in Spanish.

Skill-based routing affects only which work queue or which operator worklist receives an assignment. Skill-based routing does not prevent an operator who does not meet the skill profile used in routing from accessing an assignment in a work queue and performing the assignment.

Unit testing an activity

You can test an activity individually before testing it in the context of the application that you are developing. Additionally, you can convert the test run to a Pega unit test case.

You can use unit test the activity if one of the following criteria is met:

- The **Require Authentication to run** check box is selected on the **Security** tab of the **Activity** form.
- Your access role allows you to update rules that have the `Applies To` class of this activity.

In the navigation pane of Dev Studio, click **Records > Technical > Activity**, and then select the activity that you want to open.

Activities *(on page)*
Unit testing individual rules *(on page 345)*
Using the Clipboard tool *(on page 374)*
Opening a unit test case *(on page)*
Application debugging by using the Tracer tool *(on page 347)*

Clipboard contents created from activity unit tests

When you test activities with the Run Rule feature, the system might create one or more clipboard pages, including copies of the named pages that you specified in the **Run** dialog box. The system retains these pages when you close the dialog box; properties on these pages remain available for you to use when you perform subsequent testing of this or other rules with the Run Rule feature.

If the activity returns results to a parameter marked as `Out` on the Parameter tab of the activity, you cannot see the results in the Clipboard. Out parameters reside on the parameter page of the activity, which you can view only with the Tracer tool. To simplify testing, you can modify the activity temporarily to store an Out parameter value in a property that is visible on a clipboard page. Delete the temporary step when you finish testing.

Testing an activity in context

If the activity requires extensive setup and is more appropriately tested in context rather than through unit testing, you can trigger the Tracer to start only when the activity is reached.

To test an activity in the context of other activities and rules that produce the needed initial conditions, complete the following actions:

1. Open the activity.
2. Select **Actions > Trace**. The Tracer tool starts.
3. To avoid using more memory than needed while tracing, click **Settings** in the Tracer window and ensure that only the ruleset containing the activity is selected in the **Rulesets To Trace** section.
4. Begin the other processing that eventually calls the activity in your own requestor session. Tracer output begins when the activity starts.

Application debugging by using the Tracer tool *(on page 347)*

Creating an activity

Automate a system task for which a more appropriate rule type is not available by creating an activity. With activities, you define a sequential set of instructions, or steps, that the activity completes automatically. Each step calls a method or supported rule type to perform the required processing.Consider a scenario in which an insurance company must submit insurance claims to the Registry of Motor Vehicles. To minimize the impact on users, you can configure an activity to automate claim uploads so that your application submits insurance claims outside of peak hours, without user intervention.

1. In the header of Dev Studio, click **Create > Technical > Activity**.
2. In the **Activity Record Configuration** section, in the **Label** field, enter a name that describes the purpose of the activity and helps identify the activity.
3. **Optional:** To change the identifier that other rules use to reference this activity, in the **Identifier** section, click **Edit**, and then enter a name that is unique within the **Apply to** class.
   Choose a name that starts with a letter and contains only letters, numbers, and hyphens. The name must be a valid Java identifier. The length of the class name plus the length of the identifier cannot exceed 128 characters.
4. **Optional:** To automatically set the activity type and, in some cases, add prepopulated steps to the form, click **View additional configuration options**, and then select one of the available activity templates:
   - To create a route activity that you can use in an assignment shape to route an assignment to a worklist or work queue by using custom routing criteria, select **Template for Route type activity for worklist**. For more information, see Assignment shapes in processes *(on page)*.
   - To create a trigger activity that you can use in a Declare Trigger rule to set the values of parameters, select **Template for Trigger type activity**. For more information, see Creating Declare Trigger rules *(on page 238)*.
   - To create a utility activity that you can use to automate the processing of a case, select **Template for Utility type activity**. For more information, see Calling an activity or automation from a process *(on page)*.
5. In the **Context** section, define the context in which to execute the rule:

   a. In the list of built-on applications, select an application layer for the activity.

   b. In the **Apply to** field, enter a class that you want to associate with the activity.
   At run time, the activity runs in the context of a page. The class that you specify in the **Apply to** field must be either the class of that page or in the class hierarchy of that page's class. For more information, see Understanding class hierarchy and inheritance *(on page)*.

The list of available class names depends on the application context that you select.

c. In the **Add to ruleset** list, select a ruleset and a ruleset version in which you want to store the activity.

6. **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 *(on page)*.
7. Click **Create and open**.
8. Before populating any steps, on the **Security** tab, select an appropriate activity type to determine whether and how other rules can reference this activity.
   For more information, see Security tab on the Activity form *(on page 118)*.
9. **Optional:** If you want to skip the step or use a when condition that references this step, on the **Steps** tab, complete the **Label** field.
   - To skip the step, enter only `//`.

     An excluded loop step excludes all child steps from execution.
   - To allow other steps to reference this step in their When or Jump criteria, enter a short string.
10. **Optional:** To repeat a step or a contiguous sequence of steps a number of times, click **Loop**, and then define a loop.
    The most common way to loop is to iterate over a Page List property, such as *pxResults*, by using the **For each embedded page** option. For more information, see Repeating steps in an activity *(on page 110)*.
11. **Optional:** To complete a step only if a set of parameters fulfill conditions that you define, click **When**, and then define the preconditions for the step.
    For more information, see Activity form - Completing the Steps tab - Entering preconditions *(on page 105)*.
12. In the **Method** field, press the Down arrow key, and then select a method or instruction that you want to complete.
13. **Optional:** To set a context for the step that is different than the primary page of the activity, specify the name of the page or property that you want to use as the context.
    By default, an activity executes within the context that calls it. For example, an activity that a utility shape calls during case processing executes against *pyWorkPage*, which is the page that is assigned to the case type. Complete the following steps only if you want to change the default context:

    a. On the **Pages & Classes** tab, add the page or property that you want the system to use at run time.
    For more information, see Defining the pages and classes of a rule *(on page)*.

b. In the **Step Page** field, enter the name of the property that you want the step to use as context.
Depending on the scenario, you can use additional syntax. For example, if you selected the **For each embedded page** loop in step 10 *(on page 130)*, you must enter a full property reference to a Page List property, such as *pagename.pxResults*.

14. Click **Expand to see method parameters**.
15. In the **Method Parameters** section, configure how your application applies the method or instruction that you selected in step 12 *(on page 130)*:
    - If you selected a method, enter parameters specific to that method.

      For example, if you selected the **Obj-Browse** method, in the **PageName** field, enter the name of the page on which you want to store the results, and then in the **ObjClass** field, enter the name of the class that you want the method to browse.
    - If you selected an instruction, such as **Call**, **Branch**, **Collect**, or **Queue**, enter parameters specific to the rule that is associated with the instruction.
    - If you selected **Java**, in the **Java Source** field, enter Java source code that you want the activity to execute.

    > At run time, the system passes the values that you specify to the method or instruction. The system holds the parameter names for a step on a special clipboard page called the parameter page, which is visible while tracing, but not with the Clipboard tool.

16. **Optional:** To specify conditions to evaluate after the method in a step runs, but before the activity moves to the next step, click the **Jump** link, and then define a transition.
    For more information, see Defining step transitions in an activity *(on page 107)*.
17. Click **Save**.
18. If your activity contains Java steps, ensure that the activity causes no security issues by running the Rule Security Analyzer tool before locking a ruleset version.
    For more information, see Implementing security guidelines for custom HTML *(on page)*.
19. To configure another step in the activity, click **Add a step**, and then repeat steps 9 *(on page 130)* through 18 *(on page 131)*.

Activities *(on page)*
Unit testing an activity *(on page 127)*
Creating rules *(on page)*
Branches and branch rulesets *(on page 324)*

Constraints rules

Create constraints rules to define and enforce comparison relationships among property values. Constraints rules can provide an automatic form of property validation every time the property's value is "touched", in addition to the validation provided by the property or other means.

The following tabs are available on this form:

- Constraints *(on page 134)*

The system evaluates constraints automatically each time a property identified in a constraints rule is changed. This technique is known as forward chaining.

Where referenced

No other rules explicitly reference constraints rules. When you save a constraints rule, it is enforced immediately and thereafter. The system automatically adds a message to any property that is present on the clipboard and fails a constraint. This message marks the page containing the property as invalid and ordinarily prevents the page from being saved.

Use the Application Explorer to access the constraints rules that apply to work types in your application. Use the Records Explorer to list all the constraints rules available to you.

After you complete initial development and testing, you can delegate selected rules to line managers or other non-developers. Consider which business changes might require rule updates and if delegation to a user or group of users is appropriate. For more details, see Delegating a rule or data type *(on page)*.

Category

Constraints rules are instances of the `Rule-Declare-Constraints` class. They are part of the Decision category.

Note the s in the rule type. The `Rule-Declare-Constraints` rule type replaces the `Rule-Declare-Constraint` rule type, which is deprecated.

Viewing rule history *(on page)*

Creating constraints rules

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing rules by creating a copy in a specific ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a constraints rule by selecting `Constraints` from the `Decision` category.

> **Note:** Because constraints extend the definition of a property value, use nouns in the declarative expression names for better distinction from property values.

Key parts:

A constraints rule has two key parts:

Field	Description
Apply to	Select a class for this rule. At runtime, a clipboard page of this class must be a top-level page. The properties to be constrained may be in this class (or in the class of an embedded page). You cannot use a Rule-Declare-* class or any ancestor of the `Rule-Declare` - class (including `@baseclass`) here. You cannot use a class derived from the `Code-` or `Embed-` class here.
Identifier	Enter a name for this constraint, unique within the **Apply to** class. Begin the name with a letter and use only letters, numbers, the ampersand character, and hyphens. No other rules explicitly reference this **Identifier** value. However, because of normal class inheritance, a constraints rule named MaximumLimit at one level in the class structure may override (and so prevent execution of) a constraints rule named MaximumLimit at a higher level in the class structure.

Rule resolution

When searching for instances of this rule type, the system uses full rule resolution which:

- Filters candidate rules based on a requestor's ruleset list of rulesets and versions
- Searches through ancestor classes in the class hierarchy for candidates when no matching rule is found in the starting class
- Finds circumstance-qualified rules that override base rules
- Finds time-qualified rules that override base rules

Constraints rules *(on page 132)*
Viewing rule history *(on page)*
More about Constraints rules *(on page 138)*

Constraints form - Completing the Constraints tab

Record the computations that constrain the property values. Each row defines a separate constraint. Enter at least one row. Order is not significant.

When the system detects that a constraint fails, it adds a message to the page of the property identified in the left portion of the **Require That** field (if any). The presence of this message ordinarily prevents the system from saving the page containing this property value to the database.

You can cause a specific additional message (`Rule-Message` rule type) to become associated with that property or another property when the constraint fails, in the **Else** row.

Detection of this message and any later processing to correct the constraint — such as displaying a message to a user, recalculation of the property value, and so on — is an application responsibility.

Each property involved in the constraint expression must be on the primary page (the page that has the class corresponding to the **Applies To** key part), or on an embedded page identified in **Page Context** field on the **Pages & Classes** tab, or on a linked property page.

Using the controls

Click any constraint to enable controls.

Control	Action
open icon 	Click to review a property (for a field that contains a property reference). If the property does not exist, click the **Open** icon to create a new property.
expression selector	Click the drop-down button to display a drop-down list of all boolean expressions.
row buttons	Select any constraint to enable. Click the Add a row icon to add a new constraint, or click the Delete this row icon to delete the selected constraint. (Equivalently, use the **Insert**, **Shift +Insert**, and **Delete** keys.)
Display Label	When reviewing this tab, you can see a presentation using either property names or the Short Description (`pyLabel`) of properties. Click to toggle the display. (Note that two or more properties may have the same Short Description text.)

Modifying constraints with drag and drop operations

To move a constraint to a different location, click and drag the small circle located to the left of the entry.

To copy a constraint, drag the entry while holding down the `CTRL` key.

Completing fields

Field	Description
When	Click the drop-down button to select a boolean expression from the drop-down list. The form changes to reflect the expression selected. Most expressions require properties. Use SmartPrompt to select a property. You can enter a linked property *(on page 134)* reference here.
Require that	This area defines the constraint. Click the drop-down button to select an expression from the drop-down list. The form changes to reflect the expression selected. Most expressions require properties. Use SmartPrompt to select a property, or enter a linked property reference. You can't specify single elements of a `Value List`, `Value Group`, `Page List` or `Page Group` in property references here. Use only empty parentheses containing no index or subscript, for example: `pyWorkPage.pyWorkParty().pxCity = "Chicago"`
Else add message	Optional. Type the text of a message within quotes or use the SmartPrompt to select the name of a `Rule-Message`. The list of options is limited to `Error` and blank category messages. Commonly used messages include: • `ValueRequired` — Value cannot be null (blank) • `ValueOutOfRange` — Value is out of range Some messages require parameters. To provide fill-in-the-blank text values for each parameter, follow the message name with a backslash character, the single letter t, and text for each message parameter.
to	Optional. Identify the property to be marked with the message. The message remains until this property has a new value (or until an activity performs a Page-Clear-Messages method). Ordinarily, choose the property that failed the constraint. Some constraints involve two or more properties, on the same or different pages. Select the one most likely to have a new value that will restore the constraint condition.

Field	Description

	For example, if the constraint is of the form
	`BoxWidth IS LESS THAN 40`
	then the message can be associated with the BoxWidth property. If the constraint has the form
	`BoxWidth IS LESS THAN BoxLength`
	then the message can be associated with either property.
	In unusual cases, you can leave this blank. When you leave this blank, the message is associated with the entire page rather than a property on that page. However, this is temporary, as page messages are removed upon the next user HTTP Submit operation.
	You can't use symbolic page names (such as `primary` or `steppage`) here, as they are not meaningful except within an activity execution context.

About Constraints rules *(on page 132)*

Constraints rules *(on page 132)*
Creating constraints rules *(on page 132)*
Viewing rule history *(on page)*
More about Constraints rules *(on page 138)*

Constraints form - Completing the Pages & Classes tab

Complete this tab to list the pages referenced by name in the **Constraints** tab, and to identify the context of the properties constrained.

1. About *(on page 132)*
2. New *(on page 132)*
3. Constraints *(on page 134)*
4. Pages & Classes *(on page 136)*
5. History *(on page)*
6. More... *(on page 138)*

For basic instructions, see How to Complete a Pages & Classes tab *(on page)*.

Field	Description
Page Name	Optional. Identify a page that is the source of properties referenced on the **Constraints** tab. Optionally, add a row with the keyword `Top` as the page name, to identify a top-level page. The Top keyword allows you to use the syntax Top.propertyref to identify properties, on other tabs of this rule form.
Class	Optional. Select the class of that page.
Locate Page Expression	The Locate Page Expression option is not supported. Use a data page instead. Rules created in previous versions that use the Locate Page Expression display this field. New rules do not display this field. Changes to a rule that uses this option cannot be saved unless Locate Page Expression is replaced by supported functionality.
Locate Page Activity	The Locate Page Activity option is not supported. Use a data page instead. Rules created in previous versions that use the Locate Page Activity display this field. New rules do not display this field. Changes to a rule that uses this option cannot be saved unless Locate Page Activity is replaced by supported functionality.
Page Context Data	

Page Context	Optional. Leave this blank if the constrained property or properties (on the Constraints tab) all have mode `Single Value`.
	
	Otherwise, identify a `Page List` or `Page Group` property on the clipboard. You can supply literal index (subscript) values or property-reference index values. Precede the property reference with a period.
	
	To indicate that all the values of the index are acceptable, omit any index between parentheses. For example, these are acceptable:
	
	`Invoices.py Orders(2).pyItems("Manuals").pyItems`
	
	`Invoices.py Orders().pyItems().pyItemNames`

Field	Description
Page Class	Optional. If the Page Context field is not blank, identify the class of the page or pages that are identified in the Page Context field. (You cannot use the `$ANY`, `$NONE` or `$CLASS` keywords here.)

About Constraints rules *(on page 132)*

More about Constraints rules

Conflicting constraints

Multiple constraints that are impossible to meet are neither detected or rejected, such as A > B and B > A. Both properties are marked as not valid each time either is changed.

Primary page

During execution of a constraints rule, the page on which the rule operates temporarily becomes the primary page. The page keyword `PRIMARY` and the results of the `tools.getPrimaryPage()` PublicAPI method reflect this change. When the constraints rule completes, the primary page of the calling activity resumes as primary.

Alias Function rules

The contents of the selection lists on the Constraints tab depend on property alias rules and alias function rules.

Change tracking exceptions

The constraints are tested when a property value on the primary page (the page that matches the Applies To key part of the rule) changes. The following do not cause the constraints to be evaluated:

- Changes to properties on other pages referenced in the Pages & Classes tab
- Saving a rule form that contains a property on the page
- Retrieving a property value with the Obj-List or Obj-Browse method
- Retrieving a property with the RDB-List method, unless the ApplyDeclaratives parameter of that method is selected

Workstation display of constraint failures

When appropriate, your application can display constraint rule violations immediately as a user changes an input value on a user form or flow action form, rather than later when the form is submitted. The user can see the constraint message immediately.

For example, the constraint may require that the Due Date field be at least a week after the Start Date field. As a user enters and revises work item data, the constraint is checked immediately as user focus leaves either input field.

This feature can improve user productivity and accuracy, while also reducing the number of server interactions and HTTP traffic required to complete a valid input form. Consider whether and where such interactive checking may simplify the user's task of completing complex input forms in your application.

To implement this capability:

1. For best results, use this feature on flow actions or harnesses that use the SmartFrames layout and JSP tags rather than declaratives.
2. Include the properties involved in the constraints on the same flow action form, or on the same user form (in the same or different sections). (Don't place the input in the flow action form and the target in the harness form or vice versa.)
3. Select the Enable Expression Calculation? box on the HTML tab of the Flow Action form or Harness form.
4. Test.

This feature is based on AJAX technology.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Testing and debugging constraints rules

Using the Tracer, you can watch the evaluation of a constraints rule. Start the Tracer and select a requestor session. Click **Settings** and check the Declare Constraint box in the Event Types to Trace section. Also check the RuleSet that contains the rule to be traced.

The statistic Tracked Property Changes on the full details page of the Performance tool shows how many property changes have occurred (for the current requestor since log-in) that are tracked for declarative rules computations. />.

OnChange activities

Constraints rules, like Declare Expression rules, do not evaluate during the execution of an OnChange activity.

Property aliases *(on page)*
Function Alias rules *(on page)*
Application debugging by using the Tracer tool *(on page 347)*
Constraints rules *(on page 132)*
Creating constraints rules *(on page 132)*
Viewing rule history *(on page)*

Data transforms

A data transform defines how to convert data that is in one format and class (the source) into data of another format and class (the target). The supported formats are clipboard and JSON. Using a data transform instead of an activity to set property values speeds up development and makes application maintenance easier.

The two main types of data transforms are the standard data transform and the JSON data transform. When you want to convert data formats, use the standard data transform to manage your application data within internal Pega Platform applications. When you need to integrate disparate data, for example, web-based data with information provided by internal sources, use the JSON data transform.

For more information about configuring a data transform, see the following topics:

- Configuring a data transform *(on page 144)*
- Configuring the JSON data transforms settings column *(on page 167)*
- Unit testing a data transform *(on page 173)*

Data transforms are useful in the following situations:

- When using data pages *(on page)* to manage application data (see the Where referenced section below).
- In activities (see the Access section below).

You can create a data transform for either a clipboard or JSON model format. You select the type of data transform in the **Additional configuration options** section on the Create Data Transform form. The options are different depending on which model format you choose.

For the JSON model format, you can choose whether to automatically map all data in the JSON string, or you can map individual properties. Mapping individual properties is useful when you have a large JSON string and you only care about a couple of properties, when you want to change the JSON structure, and when the fields in the JSON have different names from the clipboard properties to which you are mapping. In addition, you can select classes to exclude.

Where referenced

The following data transforms that are used for data management use data pages:

- Optional data mapping on the property *(on page)* form – On the Edit Property form, the **Optional Data Mapping** field appears when you select **Copy data from a data page**. Use this data transform to copy a subset of the data from the data page to the property. If you do not specify a data transform, the system copies all of the data from the data page to the property.
- Request data transform – When a data source for a data page is a connector (an integration with an external data service), the request data transform lets you map Pega Platform data to the fields that the connector needs to communicate with the data service. Select the data transform to use in the **Request Data Transform** field on the Data Page rule form on the **Definition** tab, in the **Data sources** section.
- Response data transform – Response data transforms normalize data provided by the data sources into the common application data model. A response data transform is required when the data source class is incompatible with the data page class (the recommended pattern to achieve true data virtualization). Select the data transform to use in the **Response Data Transform field** on the Data Page rule form on the **Definition** tab, in the **Data sources** section.

Other parts of Pega Platform that reference data transforms include:

- Several activity methods that operate on pages can use a data transform, such as the Apply-DataTransform, Apply-Model (deprecated), Page-New, Page-Copy, and Page-Change-Class methods.
- Starter flows, flows that create work items, can specify a data transform (on the flow's **Process** tab) for a Work- class to set initial properties for the work item.
- The **Action** tab for a flow action can specify that the system apply a data transform before or after displaying the flow action user interface to the user.
- For flows that have been edited by using the Flow form, data transforms can be specified on the **Set Properties** tab of the **Connector Properties** panel.
- In various user interface elements, which include the Client Event Editor's:
  - `Refresh This Section` action
  - `Refresh When` conditions
  - On-click control rule actions `Refresh Section`, `Show Harness`, and `Open URL In Window`

Data transform rules can be referenced by other data transforms.

Rule types that create clipboard pages (such as Connector rules, Service rules, and other rule types) can use data transforms.

Actions

For more information about a specific data transform action, see the following topics:

- Append and Map To *(on page 151)*
- Append To *(on page 153)*
- Apply Data Transform *(on page 154)*
- Auto-map *(on page 156)*
- Exit For Each Page *(on page 157)*
- For Each Page In *(on page 157)*
- Otherwise *(on page 166)*
- Otherwise When *(on page 166)*
- Remove *(on page 158)*
- Set *(on page 159)*
- Sort *(on page 161)*
- Update Page *(on page 163)*
- When *(on page 166)*

Action restrictions

The following property types have limited support in data transforms and have no support in JSON data format transforms:

- Page Groups
- Value Lists
- Value Groups

Access

Use the Application Explorer to access data transforms that apply to the work types in your application. Use the Records Explorer to list all data transforms available to you.

- Normalize data for use with a data page.
- Define, copy, or map data with activities:
  - Copy a clipboard page to make a new page
  - Map properties (and their values) on one page to another, existing page
  - Map properties (and their values) on one page to a new page

- On a specific clipboard page, define one or more initial properties on that page and set their values. A data transform can set many property values on a page in one processing step.
- Append pages from one Page List property to another.

Data transforms can be created either for clipboard or JSON formats, and are a structured sequence of actions. Each action is invoked in the sequence defined by the data transform.

Category

Data transform rules are part of the Data Model category. A data transform is an instance of the `Rule-Obj-Model` class.

Creating unit test cases for rules *(on page)*

Creating a data transform

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing rules by creating a copy in a specific ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

- Creating rules *(on page)*
- Creating a rule specialized by circumstance *(on page)*
- Creating a rule or data instance *(on page)*
- Creating a specialized or circumstance rule *(on page)*

This information identifies the key parts and options that apply to the record type that you are creating.

Create a data transform rule by selecting **Data Transform** from the **Data Model** category.

A data transform rule has two key parts:

Field	Description
Apply to	Select a class that this data transform applies to.

Field	Description
	The list of available class names depends on the ruleset that you select. Each class can restrict applying rules to an explicit set of rulesets as specified on the **Advanced** tab of the class form.
Name	Enter a name that is a Java identifier. Begin the name with a letter, and use only letters, numbers, and hyphens. See How to enter a Java identifier. A conventional name for the initial, and typically the simplest, data transform in a class is `pyDefault`. This name is not reserved and has no special significance.

Rule resolution

When searching for instances of this rule type, the system uses full rule resolution which:

- Filters candidate rules based on a requestor's ruleset list of rulesets and versions
- Searches through ancestor classes in the class hierarchy for candidates when no matching rule is found in the starting class
- Finds circumstance-qualified rules that override base rules
- Finds time-qualified rules that override base rules

Data transforms *(on page 140)*
Unit testing a data transform *(on page 173)*
More about data transforms *(on page 172)*

Configuring a data transform

To make application development maintenance more convenient and efficient, set property values by using data transforms. When you create a data transform, you convert data from one format and class to another format and class. For example, if a customer enters a shipping address on an order form, you can configure a data transform to populate the billing address with the same details.

1. In the header of Dev Studio, click **Create > Data Model > Data Transform**.
2. In the **Label** field, enter a short description for your data transform.
   For data transforms that initialize cases, use the `pyDefault` name.
3. **Optional:** To manually set the identifier of your data transform, in the **Identifier** section, click **Edit**. By default, the system automatically populates this field with a read-only value that is based on the sentence that you enter in the **Identifier** section. The system ignores spaces and special characters.
4. In the **Additional configuration options** section, select a data model format for your data transform:

- If you want to use the Clipboard tool as your data model, select **Clipboard**.
- If you want to use JSON as your data model, select **JSON**.

5. In the **Context** section, specify the Pega Platform ruleset context for your data transform:
   a. Select an application layer in which you want to store the record.
   b. In the **Apply to** field, select the class to which this data transform applies.
   c. In the **Add to ruleset** field, select the name of a ruleset to which you want to add the record, and then, in the list, select the version number.
6. **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 default work items, see Setting your current work item *(on page)*.
7. Click **Create and open**.
8. On the **Definition** tab, configure details of the data transform, based on the selected data model:

Choices	Actions
Configure the data transform by using the Clipboard tool	a. In the **Action** column, select an action that you want to perform on the data. For more information about actions available for the Clipboard tool, see Data transform actions for Clipboard *(on page 148)*. b. In the **Target** column, specify the target of the selected action. c. If available, in the **Relation** column, specify the relationship between the target and the source. The **Relation** column will show available selections based on the action selected in step a. d. In the **Source** column, specify the source of the selected action. e. **Optional:** To add more actions in your data transform, click **Add a row**, and then repeat steps 8.a *(on page 145)* through 8.d *(on page 145)*. f. **Optional:** To chain together this data transform and data transforms with the

Choices	Actions
	same name in any of its parent classes, select the **Call superclass data transform** checkbox. At run time, the system first performs the actions in the highest-level data transform.
Auto-map the data transform by using JSON	a. Select the **Auto-map all data** check box. > At run time, the system maps the JSON string to the clipboard. b. In the **Top element structure** section, select whether the structure for your data transform is an object or an array. c. If your element structure is an array, in the **Pagelist Property** field, select the property to which to map the array, and then select the JSON elements.
Configure the data transform by using JSON	a. In the **Top element structure** section, select whether the structure for your data transform is an object or an array. b. If your element structure is an array, in the **Pagelist Property** field, select the property to which to map the array, and then select the JSON elements. c. In the **Action** column, select an action that you want to perform on the data. For more information about actions available for JSON, see Data transform actions for JSON *(on page 150)*. d. In the **Clipboard** column, select the clipboard value to use for JSON serialization or deserialization.

Choices	Actions
	e. If available, in the **Relation** column, specify the relationship between the target and the source. The **Relation** column will show available selections based on the action selected in step a. f. In the **JSON** column, enter the array, object, or simple field name from the JSON data. g. In the **Empty behavior** column, control the mapping results of a JSON data transform by selecting an empty behavior. The configuration choices include the following options: **Default value** The JSON data transform serializer maintains the preexisting value. This configuration supports backward compatibility with Pega Platform that are earlier than 8.5. **Null** Represents a null value in the JSON data transform when the associated *ClipboardPage / ClipboardProperty* exists with an empty value, **Skip** Skips mapping in JSON when the associated *ClipboardPage / ClipboardProp-*

Choices	Actions
	erty exists with an empty value. h. **Optional:** To add more actions to your data transform, click **Add element**, and then repeat steps 8.c *(on page 146)* through 8.g *(on page 147)*.

9. **Optional:** To hide all child rows, click **Collapse All**.
10. **Optional:** To show all child rows, click **Expand All**.
11. Click **Save**.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data transform actions for Clipboard

To save time while processing your cases, populate data in your application by using data transforms. When you use data transforms, you can convert data from one format and class to another format and class.

When you configure a data transform using the Clipboard data model, you select an action that you want to perform on the data. You can select the following actions:

Set

Sets the target to equal the source. You can set single-value properties, top-level pages, embedded pages, page lists, and page groups. You typically use this action to initialize properties. If the target pages that you specify do not exist on the clipboard, the Set action creates the pages.

Remove

Deletes the target and any associated values from the clipboard.

Update Page

Sets the context for setting properties on the target page. Use to set properties on a page that is different from the primary page. If the target page does not already exist on the clipboard, the system creates the page. If the source page is different from the primary page, from the Relation list, select with values from, and then specify the source page.

Apply Data Transform

Applies another data transform to a clipboard page that the previous action specifies in the current context.

Sort

Creates custom ordered lists. You can sort a page list according to single-value properties in that page list. You can specify a sorting order for each property as either ascending or descending order. In the Target column, click the View Sorting Properties icon to specify the properties on which to sort and the sorting order for each property.

Comment

Specifies descriptive text. Use this action to provide comments within the sequence. For example, you can explain the goal of a branch of steps.

When

Specifies a condition to evaluate to determine whether to apply the subsequent actions. If the condition evaluates to true, the system applies the action within the child rows. If the condition evaluates to false, the system continues with the action specified within the next row. If the action in the next row is Otherwise or Otherwise When, the system applies the action if the condition evaluates to true.

Otherwise When

Specifies another condition to meet before the system applies an alternate condition. If the condition evaluates to true, the system applies actions within the child rows. If the condition evaluates to false, the system continues with the action specified within the next row. You can use this action after you use a When or Otherwise When action.

Otherwise

Specifies the alternate actions to apply if the preceding When or Otherwise When action evaluates to false. You can only use this action after a When or Otherwise When action.

> **Note:** If you add When, Otherwise When, and Otherwise actions to your data transform, the system performs only one of these actions, depending on the first of these actions that evaluate to true.

Append to

Copies a page to the target. You can copy a page if the source and the target are of the same class, or if one of the classes is higher in the class hierarchy in the inheritance path.

Append and Map to

Adds a page to the target and maps properties and their source values to the target properties. The source and target can be of different classes.

For Each Page In

Iteratively applies actions specified in the subsequent rows to all the pages in the specified target. Ensure that the target is a Page List or a Page Group.

Exit For Each

Exits the current iteration defined by the preceding For Each Page action. You can use this action only as a child row of a For Each Page In action.

Exit Data Transform

Stops the data transform at the row that contains the Exit Data Transform action.

Standard Data Transforms *(on page 170)*
Data transform actions for JSON *(on page 150)*

Data transform actions for JSON

Save time while processing your cases by providing data for your application through data transforms. When you apply a data transform, you populate target values with values form a source that you specify, so that you avoid providing the same data twice.

When you configure a data transform by using JSON, you select an action that you want to perform on the data. You can select the following actions:

Set

Sets the target to equal the source. You can set single-value properties, top-level pages, embedded pages, page lists, and page groups. You typically use this action to initialize properties. If the target pages that you specify do not exist on the clipboard, the Set action creates the pages.

Auto-map

Automatically maps the JSON data to the clipboard.

Update page

Sets the context for setting properties on the target page. Use to set properties on a page that is different from the primary page. If the target page does not already exist on the clipboard, the system creates the page. If the source page is different from the primary page, from the Relation list, select with values from, and then specify the source page.

Append and Map to

Adds a page to the target and maps properties and their source values to the target properties. The source and target can be of different classes.

Apply data transform

Applies another data transform to a clipboard page that the previous action specifies in the current context.

Comment

Specifies descriptive text. Use this action to provide comments within the sequence. For example, you can explain the goal of a branch of steps.

Standard Data Transforms *(on page 170)*
Data transform actions for Clipboard *(on page 148)*

Data Transform form - Append and Map to action

Use the **Append and Map** to action to append a page to the target Page List mode property and set the context to that page for subsequent child actions to map properties on that page. The target and source can be of different **Applies To** classes.

Usage

The **Append and Map to** action is equivalent to doing an **Update Page** action using the `<APPEND>` keyword. When the system invokes the **Append and Map to** action, it creates a new page in the target Page List property, without creating any embedded properties in that new page. Subsequent child actions create the embedded properties on the page.

When you select the **Append and Map to** action, the system automatically displays a child **Set** action because no properties exist yet on the newly created page.

When you use a source that is a Page List or a Page Group property (using the `each page in` choice for the Relation), the **When** icon appears next to the `each page in` choice. Click this button to specify a when condition rule for the **Append and Map to** action, so that a page from the source is appended to the target only if the condition is true. For example, when your target is a Page List property of PreferredCustomers, and you want to map data from an AllCustomers source Page List property only for those customers that are "preferred" customers, click the **When** icon and specify a when condition rule named `IsPreferred`.

JSON data model format

When the data model format for a data transform is JSON, there are the following differences in behavior from the clipboard data model format:

- The order of the steps is not guaranteed.
- The data transform is bidirectional and can be called in either direction using the *executionMode* parameter. This parameter indicates whether to use JSON as the source and the clipboard as the target or use the clipboard as the source and JSON as the target.
- There are two parameters for the JSON transform. JSON transforms cannot have custom parameters.
    - *jsonData* – Specifies the incoming or outgoing JSON data
    - *executionMode* – Specifies the direction of the transform, SERIALIZE or DESERIALIZE.
- The **Pages & Classes** tab is only used to specify the class of classless properties, such as pxResults.
- JSON data model format transforms cannot reference other top-level pages. You can only reference fields on the primary page.

In addition, the **JSON** field requires a JSON array name.

Example

Append and Map to `pyWorkPage`

Relationship settings

The following relationship settings are available when the data model format is clipboard:

- **a new page** – The default. Use to create a new page in the target Page List, and subsequently create properties on that page.
- **an existing page** – Use to create a new page in the target Page List, and subsequently create properties on that page that are to be mapped from a source page.
- **each page in** – Use to create a new page in the target Page List, and subsequently create properties on that page that are to be mapped from a source Page List or Page Group.

The following relationship settings are available when the data model format is JSON. Select the setting that matches the structure of the JSON array.

- **An array of objects** – This is the only option that translates directly to the clipboard. There are no restrictions on the number of child steps or siblings.
- **An array of scalars** – In JSON, an array of scalars has the format [1,2,3,...] or ["a","b","c",...]. When you choose this option, there is exactly one child step for the **Append and Map to** step. You cannot add siblings or disable the step. The step is a set step and the JSON side states, "each scalar element." You can specify the clipboard field to map to.
- **An array of arrays** – In JSON, an array of arrays has the format [[...],[...],...], where each element is a nameless array. When you choose this option, there is exactly one child step for the **Append and Map to** step. You cannot add siblings or disable the step. The child step can have either an **Auto-**

map or **Append and Map to** action. The JSON side states, "each array element." You can pick a Page List for the clipboard side. If you add an **Append and Map to** step, the same options are available again (array of objects, array or scalars, and array of arrays), so that you can configure embedded child steps.

- **A group of pages** – Select this option to configure mapping to a property that contains an unordered group of embedded pages. You can map a group of pages in either direction, from the source to the target or the reverse. When you choose this option, you can set child steps to map the data for the **Append and Map to** step. You can add siblings, disable the steps, and perform the full set of actions available in the **Action** column.

Supported features

You can use the following items in the **Append and Map** to action:

- In the **Target** column, Page List mode properties
- In the **Source** column, Page, Page List, and Page Group mode properties
- Conditionalizing the **Append and Map to** action, when the source is a Page List or a Page Group (using the **When** icon)

Restrictions

You cannot specify a Page Group mode property as the target for the **Append and Map to** action. That is, you cannot append to a Page Group mode property by using this action.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - Append to action

Usage

Use the Append to action to copy a page from the source to the target, or to copy all of the elements in a source Page List property to a target Page List property. The target must be of a class that is compatible with the source page. (To append a page to a target having a class that is not compatible with the source's class, or to conditionally map source data, use the Append and Map to action *(on page 151)*.)

Example

In this example, `Shipments` and `ShipmentGroup` are properties having the same Page Class. `Shipments` is a Page List property, and `ShipmentGroup` is a Page Group property. The Append to action specifies to copy each page from `ShipmentGroup` to `Shipments`.

Append to `.Shipments` each page in `.ShipmentGroup`

Relationship settings

- `a new page` — The default. Use when you want to create a new page in the target Page List property.
- `an existing page` — Use when you want to copy a source page as the new page in the target Page List property.
- `each page in` — Use when you have a Page List or Page Group for the source, and want to copy pages in the Page List or Page Group as new pages in the target Page List.

Supported features

You can use the following items in the Append to action:

- In the Target, Page List mode properties
- In the Source, Page, Page List, and Page Group mode properties

Restrictions

You cannot specify a Page Group mode property as the target for the Append to action. That is, you cannot append to a Page Group mode property.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - Apply Data Transform action

Use the **Apply Data Transform** action to apply actions defined in another data transform (the applied data transform).

Usage

To apply the data transform to a specific page, specify a row by using an **Update Page** action before the **Apply Data Transform** action row. To apply the data transform to a specific page, precede the **Apply Data Transform** action with an **Update Page** action to identify the page.

JSON data model format

When the data model format is JSON, there are the following differences in behavior:

- The order of the steps is not guaranteed.
- The data transform is bidirectional.

Relation column

The **Apply Data Transform** action has no setting for the **Relation** field.

Parameters in the to-be-applied data transform

The applied data transform can have parameters defined on its **Parameters** tab. In the **Apply Data Transform** action row, click the **View Data Transform Parameters** icon to provide values for the applied data transform's input parameters (if any), or specify whether to pass the current parameter page.

When the **Pass current parameter page?** check box is selected, the two data transforms (the one that has the **Apply Data Transform** row and the applied data transform specified in that row) share one parameter page. If the check box is not selected, the parameters on the parameter page for the applied data transform are not available to the other data transform, and vice versa.

If the current parameter page is shared, and the applied data transform has a parameter declared as `Out`, if the applied data transform updates the value of that parameter, the resulting value is available to the other data transform.

> **Note:** If the data transform that you are calling changes any values on the parameter page, those values will be lost if you do not select the **Pass current parameter page?** check box. For example, when calling a JSON data transform using the SERIALIZE value for the *executionMode* parameter, the resulting JSON data is written to the parameter page as a value of *jsonData.*

Restrictions

You cannot call a clipboard data transform from a JSON data transform.

You cannot use the `Top` or `Parent` keywords in the entry field when specifying the applied data transform.

You cannot use expressions when specifying the applied data transform.

When `$ANY` is used in the entry field (to make the class of the applied data transform assignable at run time), design-time validation does not check whether the specified data transform exists when you save this data transform form.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - Auto-map action

Use the **Auto-map** action to contextually and automatically map data from JSON to the clipboard or from the clipboard to JSON.

Usage

Use the **Auto-map** action with a Page or Page List property that has a data model of the property's page class that exactly matches the JSON data model. The name and type of every child field is the same in both the JSON data and the clipboard.

> **Note:** To ensure that the **Auto-map** action runs properly, as a best practice, define the desired Pega Platform data model in your application hierarchy.

For more information about mapping and JSON serialization and deserialization, see the following articles on Pega Community:

- *Using the mapping API for high-performance JSON serialization*
- *Using the mapping API for high-performance JSON deserialization*

Relation column

The **Auto-map** action has a fixed **Relation** field setting of `To`. That is, the **Auto-map** action always sets the target to the source and establishes no other sort of relationship between the two.

> **Note:** When serializing JSON, the source is clipboard, and when deserializing JSON, clipboard is the target.

Restrictions

The only property type values supported are:

- Page
- Page List

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - For Each Page In and Exit For Each actions

Usage

Use the For Each Page In action to have the system apply the subsequent actions to every page in a target Page List or Page Group mode property. After the For Each Page In action row, the system goes down the subsequent children rows and iteratively applies the actions in those rows to each page in the target.

Use the Exit For Each action to stop the iterations of the For Each Page In action. Actions at the same level as the Exit For Each row (its siblings) are not performed by the system.

The Exit For Each action is typically used as part of an overall conditional structure. For example, in the following shopping cart example, `Items` is a Page List property. if the user adds a product item (.PRODUCTNAME) and quantity to his shopping cart, the When action checks to see if that product is already in the cart, for each page of `Items` and if so, updates the quantity in the shopping cart with the additional quantity for that item. After the item is found in the cart, the Exit For Each action stops the system from iterating over the rest of the pages in `Items` and omits the actions for the Otherwise case (which are not needed once the item is found to exist in the cart).

```
For Each Page In pyWorkPage.Items

  When .Name == Primary.PRODUCTNAME

    Set pyWorkPage.ItemToAdd.LocationFound equal to pyWorkPage.Items(<CURRENT>).pxListSubscript

    Set pyWorkPage.ItemToAdd.Found equal to "true"

    Set pyWorkPage.ItemToAdd.Quantity equal to pyWorkpage.Items(<CURRENT>).Quantity

    Exit For Each

  Otherwise

    Set pyWorkPage.ItemToAdd.Found> equal to "false"
```

Example

```
For Each Page In .Customers
  When IsPreferred
    Set Discount equal to 15
  Otherwise When IsExisting
    Set Discount equal to 5
```

Relationship settings

The For Each Page In and Exit For Each actions have no settings for the Relation field.

Supported features

You can use the following items in the For Each Page In action:

- Page Lists
- Page Groups
- Parameters specified as `Page Name`

Restrictions

You cannot use Page List properties that have Lightweight specified in their rule forms as sources.

You cannot iterate over a source. You can only iterate over target pages.

You cannot use the `<APPEND>` keyword in the For Each Page In action. Instead, follow the For Each Page action with an Append to action or Append and Map to action.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - Remove action

Usage

The Remove action deletes the specified target and any associated values from the clipboard.

Example

Remove `pyWorkPage.ItemToAdd.Quantity`

Relationship settings

The Remove action has no setting for the Relation field.

Supported features

You can use the following items in the Remove action:

- Properties
- Pages (top-level or embedded clipboard pages)
- Page Lists
- Page Groups
- Value Lists
- Value Groups
- Data pages (pages created by data page rules).
- `Primary` keyword and a property (for example, `Primary.Firstname`)

Restrictions

To remove a top-level page, the page must be specified on the Pages & Classes tab *(on page 168)*.

You cannot use `Top`, `Parent`, or `param` in the Remove action.

You cannot remove the primary page (the page of the Applies To class of the data transform).

You cannot remove a parameter (specified on the Parameters tab).

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - Set action

Use the **Set** action to set the target equal to the source. When you specify a target page that does not yet exist on the clipboard, the **Set** action creates the page.

Usage

To select a value for the target property from the set of values defined in case instances for the property, click **Select values**.

JSON data model format

When the data model format is JSON, there are the following differences in behavior:

- The order of the steps is not guaranteed.
- The data transform is bidirectional.

Examples

```
Set .pyWorkIDPrefix equal to "W-"

Set .HardwareItems(1).Price equal to ""
```

Relation column

The **Set** action has a fixed **Relation** setting of `equal to`. That is, the **Set** action always sets the target equal to the source, and establishes no other sort of relationship between the target and source.

> **Note:** When serializing JSON, the source is clipboard, and when deserializing JSON, clipboard is the target.

Supported features clipboard data model format

When the data model format is clipboard, the **Set** action supports the target and source combinations shown in the following table. These features are not supported when the data model format is JSON.

The class of the target does not have to match the class of the source.

Target	Relation	Source	Example
Single Value Property	equal to	Property Reference	.Department
Single Value Property	equal to	Literal	"W-"
Single Value Property	equal to	Expression	.FIRSTNAME + " " + .LASTNAME
Single Value Property	equal to	Linked Property Reference	.pxCreateOperator.pyUserName

Target	Relation	Source	Example
Page	equal to	Page	OrderItems(1) equal to HardwareItems(1)

The source and target support the following items:

- Properties
- Literals
- Expressions (in the source column only)
- Parameters specified on the **Parameters** tab (for example, `param.Name`)
- Page Lists
- Page Groups
- Value Lists
- Value Groups
- Keywords: `Primary`, `<APPEND>`
- Function calls (in the source column only)

When the **Set** action is preceded by an **Update Page**, the context of the source is reset and all property references are relative to the new page.

Restrictions

The use of `Top` and `Parent` keywords is not supported.

You cannot set properties on data pages. Such properties cannot be specified as targets.

References to linked properties are read-only. Such properties can be specified as sources, but not as targets.

You cannot set reference properties (properties that have the **Reference Property** check box selected on the **Advanced** tab of their Property forms). They cannot be specified as targets.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - Sort action

Usage

Use the Sort action to create custom ordered lists. A typical scenario is you are displaying a multi-column list and you want the columns automatically sorted according to values in one or more of the columns. With the Sort action, you can sort a Page List property according to single-value properties in that Page List. For each single-value embedded property, you can specify sorting in either ascending or descending order of that property. Click the **View sorting properties** icon to specify the properties on which to sort and the sorting order for each property.

If you specify more than one embedded property to sort on, the first property specified is given precedence at runtime. For example, you want a section to display a list of product items in an order, and the page list is `Items`, with embedded properties:

- `Quantity` (Integer type)
- `TotalPrice` (Decimal type)

And you want to show the user the list of items sorted first by quantity from smallest to largest, and then by total price from highest to lowest within a given quantity. In the Sort By Property window, specify the `Quantity` property in ascending order and the `TotalPrice` property in descending order.

At runtime, the displayed rows run from the lowest quantities to the highest, and rows that have the same quantity are displayed with the highest total price first, then the next, and so on.

Relationship settings

The Sort action has no setting for the Relation field.

Supported features

You can use the following items in the Sort action:

- In the Target, Page List mode properties
- As the embedded properties on which to sort by, single-value properties. For each sorted-by property, you can sort in ascending or descending order.

Restrictions

You cannot specify a Page Group mode property as the target for the Sort action. That is, you cannot sort by properties in a Page Group mode property.

You cannot sort by properties in the Page List that are not single-value properties. For example, you cannot sort by a Page property in a Page List property.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - Update Page action

Use the **Update Page** action to set values on a target page that is different from the data transform's primary page.

Usage

The **Update Page** action sets the page context for the subsequent (children) steps. Using the **Update Page** allows you to:

- Avoid entering the target page name each time in the children **Set** actions
- Use source values from a page that is different from the data transform's primary page

For example, the following data transforms are equivalent:

`Set pyWorkPage.CustomerType equal to "Existing"`

Set `pyWorkPage.CustomerLevel` equal to "Bronze"

and

```
Update Page pyWorkPage
  Set .CustomerType equal to "Existing"
  Set .CustomerLevel equal to "Bronze"
```

In the second example, the **Update Page** action sets the page context to the `pyWorkPage`, so that the page does not have to be specified for the **Target** fields in the following child steps.

You can set the page context for the target only, or for both the target and source. To set the page context for the source, select `with values from` in the **Relation** field.

JSON data model format

When the data model format is JSON, there are the following differences in behavior:

- The order of the steps is not guaranteed.
- The data transform is bidirectional.

In addition, the **JSON** field requires a JSON object name.

Examples

```
Update Page pyWorkPage

  Set .ZipCode equal to "02139"

  Set .TelPrefix equal to "617"

Update Page pyWorkPage with values from Employees

  Set .ZipCode equal to .PostalCode

  Set .TelPrefix equal to .Prefix
```

Relation column

The following relationship settings are available when the data model format is clipboard:

- blank – The default. Use when setting values on a target page using source values that are on this data transform's primary page (the clipboard page with the same class as the data transform's **Applies To** class).
- **With values from** – Use when the source values are on a page other than the data transform's primary page.

The following relationship settings are available when the data model format is JSON:

- **For both sides** – Set the page context for both the JSON and clipboard sides.
- **For JSON only** – Set the page context for the JSON side only.
- **For clipboard only** – Set the page context for the clipboard side only.

JSON relationship settings example

This example illustrates how to use the relationship settings. The JSON for a customer is as follows:

```
{
          "id": 123456,
       "name": "John Doe",
       "email": "john.doe@example.com",
       "address": {
           "addressLines": {
               "line1": "1 example st",
               "line2": "example building"
           },
           "city": "Example",
           "zip": "02142"
```

```
      }
   }
```

The clipboard structure is as follows:

```
Customer
       ID
      Name
      Email
       Address
          AddressLine1
          AddressLine2
          City
           Zip
```

The tree structures match except for the address line fields. `line1` on the JSON side is under the `address > addressLines` hierarchy, but its corresponding `AddressLine1` property on the clipboard is directly under `Address`. In this case, use the **Update Page** action to update context only on one side. To map the entire structure, update the context for address on both sides, then update the context only on the JSON side, and then map individual properties, as shown in the following sample:

Update Page `Address` **For both sides** `address`

Update Page `No context change` **For JSON only** `addressLines`

Set `AddressLine1` **equal to** `line1`

Supported features

You can use the following items in the **Update Page** action:

- Page (top-level or embedded clipboard page)
- Page in a Page List
- Page in a Page Group
- Keywords: `<APPEND>`, `<LAST>`, `Primary`
- Expressions in the index (subscript) of an aggregate property

Restrictions

- You cannot use the `Parent` keyword.
- Do not use the `Top` keyword in data transforms. In some situations, using `Top` does not work if a top-level page is not clearly defined (for example, if a row has both source and target pages).
- You cannot use data pages (pages that are instances of data page rules).

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - When, Otherwise, and Otherwise When actions

Usage

Use the When, Otherwise, and Otherwise When actions to conditionalize data transform actions, so that the system invokes the actions only when the specified conditions are true.

For example, given an ordering process with a form where customers can select a check box to indicate whether their billing address is the same as their shipping address, you can use a When action to tell the system to perform a set of actions (children actions of the When) when that check box is selected (`BillAddressSameAsShipping` property is true). By using an Otherwise action after the When action, you can specify another set of data transform actions to take if the check box is not selected.

In the list of rows on the Definition tab, an Otherwise When action row must be preceded by a When action row at some point in the list. An Otherwise action row must be preceded by either a When action row or an Otherwise When action row.

Example

In the following example, if the user selects **Billing address same as shipping address** check box in the order form, the property `BillingAddressSameAsShipping` is set to "true".
In that case, the data transform actions following the When action specify to update the `pyWorkPage.pyWorkParty(Customer).BillingInformation` page using the Address value from the `pyWorkPage.pyWorkParty(Customer).ShippingInformation` page (that is, use the shipping address for the billing address). If the check box is not selected, the children actions of the Otherwise action specify to set the billing information page values to null strings, so that the user has to enter the data.

```
When pyWorkPage.pyWorkParty(Customer).BillingAddressSameAsShipping == "true"

  Update Page pyWorkPage.pyWorkParty(Customer).BillingInformation with values from

 pyWorkPage.pyWorkParty(Customer).ShippingInformation
```

```
    Set .Address equal to .Address
Otherwise
  Update Page pyWorkPage.pyWorkParty(Customer).BillingInformation
    Set .Address equal to ""
```

Relationship settings

The When, Otherwise, and Otherwise When actions have no settings for the Relation field.

Supported features

You can use the following items in the rows for the When and Otherwise When actions:

- When condition rules
- Expressions
- `Primary` keyword

Restrictions

Do not use `<` or `>` in an expression. Use `<=` or `>=` instead.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Configuring the JSON data transforms settings column on the Data Transform form

Pega Platform displays the **Settings** tab on the data transform rule form for the JSON data model format. Use the **Settings** tab to enable multidimensional array output and to specify fields to skip during auto-mapping. You might want to skip fields that do not have meaning outside of Pega Platform, such as *pyObjClass*.

1. In the data transform that you create, click the **Settings** tab.
2. To enable multidimensional array output, select the **Enable multidimensional array output** check box.
   Select this option when auto-mapping is enabled on the **Definition** tab and you have a *Page List* property with only one non-skipped property that has the same name as its parent. If you enable multidimensional array output, the *Page List* property will result in a multidimensional array in the JSON output. If you do not select the **Enable multidimensional array output** check box, the *Page List* property is displayed as an array of objects.

3. In the **Date format for serialization** field, select the output format to use when serializing a data page:
    - When you select **Pega API**, the system formats the output as:

      Date: `1996-07-26`

      Date Time: `1996-07-26T09:30:00.000Z`

      Time of Day: `09:30:00.000Z`
    - When you select **Pega Internal**, the system formats the output as:

      Date: `19960726`

      Date Time: `19960726T093000.000 GMT`

      Time of Day: `093000`
4. To specify which fields to skip processing during auto-mapping, in the **Fields to skip when auto-mapping** section, click **Add field** to add a field to the list.
5. To specify the default auto-map empty behavior, select an empty behavior in the **Default Automap Empty Behavior** field.

> **Note:** Select this option when the **Auto-map all data** check box is selected on the **Definition** tab.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Data Transform form - Completing the Pages & Classes tab

To use page names in fields on the Definition tab, you must specify those pages on this Pages & Classes tab.

At runtime, the system uses information from this tab to locate properties on the clipboard.

Specifying the pages on the Pages & Classes tab also assists when working with the Definition tab at design time. For example, when a page is specified here, it is listed in the SmartPrompts in the fields on the Definition tab.

See How to Complete a Pages & Classes tab *(on page)* for more details.

Note: For JSON data model format transforms, this page is only used to specify the class of classless properties, such as pxResults.

Click the **Add item** icon to add new rows to the array. Click the **Delete** icon to delete a row.

Field	Description
Page Name	Enter the name of a clipboard page referenced on the Definition tab.
Class	Specify the class of that page. Click the **Open** icon to open the class rule, or to create the class if it does not already exist.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

Adding data transform to a process

To save time, define a place in your case where a data transform occurs by adding the data transform to a process. You typically add a data transform between two assignments. For example, while processing a purchase order, you can populate a shipping address with data that a user provides as a billing address.

Note: You can also call a data transform automatically from a case life cycle in a low-code way in App Studio. For more information, see Calling a data transform from a case *(on page)*.

1. In the navigation pane of Dev Studio, click **Case types**, and then click the case type that you want to open.
2. On the **Workflow** tab, hover over a process to which you want to apply a data transform, and then click **Configure process**.
3. On the toolbar, click **Open process**.
4. Double-click a connector between two assignments where you want to add a data transform.
5. In the **Connector properties** dialog box, in the **Set properties** section, select **Apply data transform**.
6. In the **Data transform** field, press the Down arrow key, and then select the data transform.
7. Click **Submit**.
8. Click **Save**.

Referencing properties *(on page 96)*
Calling a data transform from a case *(on page)*
Data transforms *(on page 140)*

Standard Data Transforms

Naming convention

By convention, the Pega Platform includes a standard data transform named `pyDefault` for most standard concrete classes and several standard abstract classes. For example, the `Data-Admin-Operator-ID.pyDefault` data transform provides initial values for about 25 properties in the Operator ID data instance.

This is not a reserved name and has no special characteristics. You may create data transforms with this name or other names in any class or RuleSet.

Data transforms that apply to classes derived from Work-

Whenever a new work item is created, the Pega Platform uses the `pyFlowName` property in the object to identify a flow rule to start.

If you create a data transform named `pyDefault` with the Applies To key part class derived from the `Work-` base class, include a value for the `pyFlowName` property. In most cases, select the Call SuperClass data transform check box to apply the ancestor `pyDefault` data transforms also.

Set the value of this property to the Flow Name key part of a flow rule. When a user starts a new flow in the context of that Work- class, the system uses this value, with rule resolution to find the flow rule to run.

Applies To	Data transform	Purpose
`Work-`	`pyCopy-Default`	Determines which properties are copied when a user copies a work item. See Harnesses — How to copy a work item *(on page)*.
`Work-`	`pyDe-fault`	Provides default initial values for properties in new work items.
`Work-Cover-`	`pyDe-fault`	Provides default initial values for properties in new work items that are covers.

Applies To	Data transform	Purpose
Work-Folder-	pyDefault	Provides default initial values for properties in new work items that are folders.
Work-.Rule-Checkin	pyDefault	Provides default initial values for properties in new work items for the rule check-in flow.
PegaSample-SimpleTask	pyDefault	Demonstrates entry of a Simple Task work item, not part of a cover.
PegaSample-CustomerRequest	pyDefault	Demonstrates entry of a cover work item.
PegaSample-Task	pyDefault	Demonstrates entry of a covered work item.

Data transforms that apply to System-User-MyRules class

By default, the order and link labels for reports in the Monitor Activity work area are controlled by data transforms with System-User-MyRules as the Applies To key part. Override these standard data transforms and then modify your rule to reference and label reports as desired.

Applies To	Data transform	Purpose
System-User-MyRules	Assignment-Monitoring-Reports	Determines the order and labels of reports (that are available to all users) in the Monitor Assignments area of the Monitor Activity work area.
System-User-MyRules	DataReports	Not used in PRPC Version 6.
System-User-MyRules	Performance-AnalysisReports	Determines the order and labels of reports (that are available to all users) in the Analyze Performance area of the Monitor Activity work area.
System-User-MyRules	RuleReports	Not used in PRPC Version 6.

`System-User-MyRules`	`WorkAnalysisReports`	Determines the order and labels of reports (that are available to all users) in the Analyze Process Quality area of the Monitor Activity work area.
`System-User-MyRules`	`WorkMonitoringReports`	Determines the order and labels of reports (that are available to all users) in the Monitor Processes area of the Monitor Activity work area.

Other standard data transforms

Applies To	Data transform	Purpose
(various)	`pyTrackSecurityChanges`	Affects data retained in history for data objects or rules.

Data transforms *(on page 140)*
Creating a data transform *(on page 143)*
More about data transforms *(on page 172)*

More about data transforms

Using a data transform, you can specify initial properties for objects of a specific class and also map properties (and their values) of an instance of a class to an instance of a different class.

For example, you can ensure that a special set of risk-related properties is present on all instances of class Data-Loan-Aircraft-SingleEngine. When a page of this class is created, you can instruct the system to apply a data transform, named Aircraft-High-Risk. The system places into the page the risk-related properties defined in the data transform. When the page is later saved into the database as a persistent instance, the properties are present with meaningful values.

Typically, you use the actions within a data transform to update and apply data to clipboard pages. Additionally, there are four activity methods that can optionally use a data transform as a parameter:

- Apply-DataTransform — Updates existing clipboard pages with values computed from a data transform.
- Page-New — Creates a new named page of a specified class. If a data transform is specified in this method, the data transform's actions are applied to the page.
- Page-Copy — Copies the contents of a clipboard page to a new named page or replaces the contents of an existing page. If a data transform is specified in this method, the data transform's actions are applied to the destination page before the source's contents are copied.
- Page-Change-Class — Creates a new page of the specified class, and then copies properties from another page into it, according to the method's Keep parameter and the specified data transform (if any).

The Apply-Model method is deprecated in version 6.2, and replaced by the Apply-DataTransform method. To use the Apply-DataTransform method to achieve the same result of using the Apply-Model method with the OverwriteProperties parameter set to `'false'`, use When actions in the data transform to prevent values from being applied when properties are already present on the target page. For example:

```
When !@PropertyHasValue(.FirstName)

Set .FirstName = "Bob"
```

In the preceding example, when the `FirstName` property has a value, the condition is not satisfied and the Set action is skipped.

Service rules (`Rule-Service-` classes) and a few other rule types can reference a data transform. The data transform is processed and applied at runtime to a newly created page.

Viewing the Java code of a rule

Click **Actions > View Java** to view the generated Java of a rule. You can use the Java code to debug your application or to examine how rules are implemented.

Changing the value of a TimeOfDay property

When you use an expression on TimeofDay property type values, the expression converts the value into a BigDecimal where the whole number is the number of days and the fraction is the hour of the day expressed as a fraction of a day, for example, 12 hours is 0.5 (half of a day). The format is useful to know when you want to change the value of a property that is a Date, DateTime, or TimeofDayproperty type.

For example, assume that you have a property of type TimeofDaythat is formatted as a standard time with hour, minute, and second without punctuation. To add 12 hours to the property:

1. Use a Set action to assign the property value to a second property that is of type TimeofDay.
2. On the right side of the expression, enter the property reference plus the fraction. In this example, enter .5 to add 12 hours.

Constants in expressions *(on page 284)*
Data transforms *(on page 140)*
Creating a data transform *(on page 143)*

Unit testing a data transform

You can use the **Run Rule** feature to test a data transform individually before testing it in the context of the application that you are developing. Additionally, you can convert the test run into a Pega unit test case.

1. In the navigation pane of Dev Studio, click **Records > Data Model > Data Transform** , and then select the data transform that you want to open.
2. Click **Clipboard** to open the Clipboard and examine the pages that are generated by the unit test. For more information, see Clipboard pages created by the Run Rule feature *(on page 383)*.

Data transforms *(on page 140)*
Viewing test case results *(on page)*
Exporting a list of test cases *(on page)*

Decision tables

Decision tables derive values that have one of a few possible outcomes, where each outcome can result from a test of a condition that includes multiple variables. A decision table lists two or more rows, and each row contains test conditions, optional actions, and a result.Decision tables are appropriate for evaluations that include more elements compared to simple true/false evaluations. For example, you can use a decision table to estimate a car insurance payment based on the age of the driver, age of the car, and the make of the car.

You build decision tables by adding columns and rows. Each column represents a condition, while rows store corresponding values that the system tests at run time. For example, you can build a decision table to determine shipment costs based on the following conditions:

- The value of the order
- The weight of the parcel
- Whether the shipment is to a foreign country

Each condition is a separate column in the decision table. Additionally, every decision table includes a column that stores results that the decision table returns if all values in a row evaluate to true.

Each row includes cells that store values that the system evaluates at run time. You can insert more than one value in a cell by adding an OR condition, and then the system continues processing a row if any of the values in the cell evaluate to true.

At run time, the system evaluates the rows starting at the topmost row:

- If any condition in a row evaluates to false, processing continues with the next row. The system ignores the **Return** column for that row.

> **Note:** You can leave empty cells in a row. Empty cells evaluate to true except when a cell is empty and includes an OR condition. Then, the parser ignores that cell and parses only the cell that contains a value.

- If all the conditions in a row evaluate to true, the system processes the **Actions** and **Return** columns of that row. The next action depends on how you configure your decision table:
  - To return only one result, you can configure processing to end. The system then returns the value in the **Return** column as the value of the entire rule.

    In scenarios that require only one result, such as estimating delivery costs, when you end processing after receiving the first result, you improve performance and avoid unnecessary computations.
  - To return multiple results, you can configure processing to continue through all remaining rows, performing the Actions and Return calculations for any rows for which the conditions are all true.

    For example, when the decision table returns the email address of a person in a process to approve an expense, the expense might require the approval of more than one person, based on the conditions. Such scenarios require more than one result.
- If all rows in the table evaluate to false, the system returns a default result.

Applying decision tables

You can apply decision tables in the following elements of your application:

- In a flow rule, you can reference a decision table in a decision shape  .

  For more information, see Adding decisions to processes *(on page)*.
- In an activity, you can evaluate a decision tree by using the *Property-Map-DecisionTable* method.

  For more information, see Creating an activity *(on page 100)*.
- A Declare Expression rule can call a decision table.

  For more information, see Creating Declare Expression rules *(on page 210)*.
- A collection rule can call a decision table.

Category

Decision table rules are instances of the *Rule-Declare-DecisionTable* class. Decision tables are part of the Decision category.

Viewing rule history *(on page)*

Creating decision tables

Better adjust to the varied factors in your business processes by creating a decision table. Decision tables test a series of property values to match conditions, so that your application performs a specific action under conditions that you define.For example, you can define a decision table in your application to calculate a loan interest rate for a customer. The decision table evaluates the customer's credit score, type of account, and loan term, and then returns a result that matches values applicable to the customer.

1. Create a rule to store the decision table:
    a. In the header of Dev Studio, click **Create > Decision > Decision Table**.
    b. In the **Label** field, enter a name that describes the purpose of the table.
    c. In the **Apply to** field, select the class in which you want to create the decision table.
    d. Click **Create and open**.
2. **Optional:** To prepopulate the decision table with values, upload an .xls file that stores starting information:
    a. In the toolbar, click **Import**.
    b. In the **Upload file** dialog box, click **Choose file**.
    c. In the **Open** window, navigate to the file on your local machine, and then click **Open**.
    d. Click **Upload file**.
3. On the **Table** tab, in the **Conditions** column, click the header cell.
4. In the **Decision table property chooser** window, in the **Property** field, enter or select a property that you want to use as a condition.
5. In the **Label** field, enter the name of the property that you want to display in the header of the decision table.
6. Select a comparison method:
    - To use a simple comparison, in the **Use operator** list, select the operator.

      For example, select >= to indicate that the value in the decision table must be equal to or greater than the value that the decision table evaluates at run time.
    - To specify a range for the condition property, select the **Use range** check box, and then define the start and end of a range.

      For example, you can configure a property value to be greater than and lower than certain amounts.

7. Click **Save**.
8. **Optional:** To consider additional factors in a decision, add more condition properties:
    a. In the **Conditions** column, click a cell.
    b. In the toolbar, click the **Insert column after** icon.
    c. Define the condition property by repeating steps 3 *(on page 176)* through 7 *(on page 177)*.
9. In the **if** row, click the cell under a property, and then enter a value that the decision table evaluates at run time.
    If you configure two or more conditions, enter a value for at least one of the conditions. Your application ignores cells without values.
10. In the **Return** column, enter a return result.
11. **Optional:** To insert an additional value in a cell, add an OR condition:
    - To insert the OR condition before the current value, on the toolbar, click the **Insert OR before** icon.
    - To insert the OR condition after the current value, on the toolbar, click the **Insert OR after** icon.

    When an application evaluates the cell at run time, the application starts with the value in the top part of the cell.
12. **Optional:** To provide more outcomes, populate more rows with values:
    a. Click a cell in the **if** row.
    b. In the toolbar, click the **Insert row after** icon.
    c. Define the condition by repeating steps 3 *(on page 176)* through 10 *(on page 177)*.
13. In the **otherwise** row, in the **Return** column, select or enter a property that defines an application behavior when no condition in the table returns a true value.
14. **Optional:** To ensure that your application can process the table, check the table for conflicts by clicking **Show conflicts** on the toolbar.

    A warning icon appears in rows that are unreachable or empty.

15. **Optional:** To increase the possibility of reaching a return value, improve the completeness of the table by clicking **Show completeness** on the toolbar.

    The system automatically adds suggested rows to the decision table, that cover additional cases.

16. Click **Save**.

⚠ **Important:** If you receive an error message that says that the code of a method exceeds the 65535 bytes size limit, you must create the *DecisionTable/legacy/splitSizeLimit* dynamic system setting (DSS) in the *Pega-RULES* ruleset, set its value to `true`, and then save the decision

> ⚠ table again. This fix applies to decision tables for which you configure legacy Java generation instead of the standard Java generation.

17. **Optional:** To import the decision table to an .xls file, for example to share with stakeholders offline, in the toolbar, click **Export**.

> At run time, your application processes rows in the table starting from the top row and returns a result for the row that is first to evaluate to true.

Calculating values with decision tables *(on page)*
Authoring decision tables in App Studio *(on page)*
Decision tables *(on page 174)*
Viewing rule history *(on page)*
More about decision tables *(on page 181)*

Specifying pages and classes of a decision table rule

To ensure that a decision table accesses or updates information on clipboard pages, specify the page name and class of the pages. At run time, these pages contain the properties that a decision table references on other tabs of its rule form.

Define the pages and classes of a rule to:

- Set up rule prompting during rule configuration.
- Configure rule validation.
- Apply default class for list elements.

For more information about pages and classes, see Defining the pages and classes of a rule *(on page)*.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **Decision Table**.
3. In the list of decision table instances, click the decision table that you want to edit.
4. On the rule form, click the **Pages & Classes** tab.
5. In the **Page name** field, enter the name of the page that contains the property that this rule references.
6. In the **Class** field, select the class of the page.
7. **Optional:** To add more pages and classes, click **Add item**, and then repeat steps 5 *(on page 178)* and 6 *(on page 178)*.
8. Click **Save**.

Creating decision tables *(on page 176)*
Configuring additional options for a decision table *(on page 179)*
Organizing rules into classes *(on page)*

Configuring additional options for a decision table

Adjust how your decision table works by configuring additional options. For example, you can configure how a decision table behaves after one row evaluates to true, or define which elements of a decision table are available for edits.Consider a scenario in which a user with a managerial role needs to update the results that the decision table returns. To avoid unintentional changes in the decision table, you can prevent users from manipulating columns and rows, and allow for editing only the results.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **Decision Table**.
3. In the list of decision table instances, open the decision table that you want to edit.
4. On the **Results** tab, in the **Delegation options** section, configure the initial presentation and available options for the decision table:
    - To continue processing after one row in the decision table evaluates to true, select the **Evaluate all rows** check box.

      At run time, when a decision table evaluates all rows, the application performs the results from all rows that evaluate to true. If you leave this check box clear, the application performs the result from the first row that evaluates to true.
    - To disable row manipulation in the decision table, for example by adding or removing rows, clear the **Allowed to update row layout** check box.

      > **Note:** Users with rule-editing privileges can update the cell values within an individual row, even if this check box is clear.

    - To disable column manipulation in the decision table, for example by adding or removing columns, clear the **Allowed to update column layout**.

      > **Note:** Users with rule-editing privileges can update the cell values within an individual column even if this check box is clear.

    - To prevent changes to the cell values, clear the **Allowed to change property sets** check box.

- To prevent access to the Expression Builder from any cell of the decision table, select the **Allowed to build expressions**.

> **Note:** Users with rule-editing privileges can add constants or property references in a row or column cell even if this check box is clear.

- To hide the **Result** column of the decision table to indicate that the decision table does not return an explicit value that represents the overall processing result, clear the **Allowed to return values** check box.

  If you select this check box, you can also restrict the list of possible return values. This check box is available only when you leave the **Evaluate all rows** check box clear.

5. **Optional:** To specify allowed results for the decision table, in the **Results** section, in the **Results defined by property** field, enter a property that lists the allowed results.
6. **Optional:** To match the allowed results that you provided in step 5 *(on page 180)* with specific values, map the results to properties:
   a. Expand the **Additional Allowed Results. When selected, each target property will be assigned the value specified** section.
   b. In the **Result** field, enter a value that the decision table can return at run time.
   c. In the **Target property** field, enter a property that the system updates after the decision table returns the result that you specified in step 6.b *(on page 180)*.
   d. In the **Value** field, enter a value to update the property that you specified in step 6.c *(on page 180)*.
   e. **Optional:** To update more properties when the table returns a specific result, under the **Target property** field, click **Add item**, and then repeat steps 6.c *(on page 180)* and 6.d *(on page 180)*.
   f. **Optional:** To match more results with properties, under the **Result** field, click **Add item**, and then repeat steps 6.b *(on page 180)* through 6.d *(on page 180)*.
7. **Optional:** To set properties before the system starts processing the decision table, define properties to precalculate:
   a. Expand the **Preset Properties** section.
   b. In the **Property** field, enter a property that you want to set before processing of the decision table.
   c. In the text field, provide a value for the property by entering a constant, property name, expression, or input parameter.
   d. To set more properties, click **Add a row**, and then repeat steps 7.b *(on page 180)* and 7.c *(on page 180)*.
8. Click **Save**.

More about decision tables

When you learn more about decision tables, you can better understand how to use decision tables in a way that most efficiently meets your business scenarios. For example, you can understand how decision tables are related to selected methods and activities, and how these connections affect processing at run time.

Evaluating a decision table

In an activity, to evaluate a decision table and derive a value, your application can use the following objects:

- The *Property-Map-DecisionTable* method
- The standard *DecisionTable.ObtainValue()* function
- The *pxEvaluateDecisionTable* utilities library
- The standard *@baseclass.DecisionTableLookup* activity

For more information about calling objects from an activity, see Configuring steps in an activity *(on page 101)*.

Property-Map-DecisionTable method

To evaluate a decision table and save the result as the value of a property, in an activity, call the *Property-Map-DecisionTable* method. As parameters, enter the target property name and the name of the decision table. For more information, see Property-Map-DecisionTable method *(on page)*.

DecisionTable.ObtainValue standard function

In an activity, you can call the standard *DecisionTable.ObtainValue* function to evaluate a decision table. Use the following syntax:

```
Lib(Pega-RULES:DecisionTable).ObtainValue(tools, myStepPage, DecisionTableName)
```

For more information about calling objects from an activity, see Configuring steps in an activity *(on page 101)*.

pxEvaluateDecisionTable utility library

In an activity, you can call the *pxEvaluateDecisionTable* utility library to evaluate a decision table. Use the following syntax:

```
Lib(Pega-RulesEngine:Utilities).pxEvaluateDecisionTable(PrimaryPageName, DecisionTableName)
```

For more information about calling objects from an activity, see Configuring steps in an activity *(on page 101)*.

Performance

Pega Platform does not limit the number of rows in a decision table. However, as a best practice to avoid slow performance when updating the form and also avoid the Java 64 KB code maximum, limit your decision tables to no more than 300 to 500 rows.

> ⚠ **Important:** When you save a decision table, if you receive an error message that says that the code of a method exceeds the 65535 bytes size limit, you must create the *DecisionTable/legacy/splitSizeLimit* dynamic system setting (DSS) in the *Pega-RULES* ruleset, set its value to `true`, and then save the decision table again. This fix applies only to decision tables for which you configure legacy Java generation instead of the standard Java generation.

Special processing with Declare Expression calls

When you create a Declare Expression rule, you can incorporate the result of a decision table by providing the result property of the decision table in the **Set Property To** field. Special processing occurs at run time when a property from the decision table is not present on the clipboard. Such decision rules fail with an error message, and the system returns the Otherwise value from the decision table instead.

Not declarative

Despite the class name, the *Rule-Declare-DecisionTable* rule type does not produce forward or backward chaining. Technically, decision tables are not a declarative rule type.

Decision tables *(on page 174)*
Creating decision tables *(on page 176)*
Viewing rule history *(on page)*

More about Decision Tables

Use a decision table to derive a value that has one of a few possible outcomes, where each outcome can be detected by a test condition. A decision table lists two or more rows, each containing test conditions, optional actions, and a result.

Uploading an Excel spreadsheet to start

If you have in advance a Microsoft Excel spreadsheet in XLS file format that contains useful starting information for a decision table, you can incorporate (or "harvest") the XLS file and the information it contains directly into the decision table.

This feature lets people with no access to the Pega Platform record their decision rules using a familiar software program.

Evaluating a decision table

In an activity, to evaluate a decision table and derive a value, your application can:

- Use the Property-Map-DecisionTable method
- Call the standard function `DecisionTable.ObtainValue()`
- Call the standard activity `@baseclass.DecisionTableLookup`

Method

In an activity, call the method Property-Map-DecisionTable method. As parameters, enter the target property name and the name of the decision table.

Standard function

In an activity, call the standard function named `DecisionTable.ObtainValue` to evaluate a decision table. Use the syntax:

```
Lib(Pega-RULES:DecisionTable).ObtainValue(this, myStepPage, "decisiontablename")
```

Performance

The Pega Platform does not limit the number of rows in a decision table. However, as a best practice to avoid slow performance when updating the form and also avoid the Java 64KB code maximum, limit your decision tables to no more than 300 to 500 rows.

> ⚠ **Important:** When you save a decision table, if you receive an error message that says that the code of a method exceeds the 65535 bytes size limit, you must create the *DecisionTable/legacy/splitSizeLimit* dynamic system setting (DSS) in the *Pega-RULES* ruleset, set its value to `true`, and then save the decision table again. This fix applies only to decision tables for which you configure legacy Java generation instead of the standard Java generation.

Standard activity

The standard activity named `@baseclass.DecisionTableLookup` also evaluates a decision table. (This approach is deprecated.)

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Special processing with Declare Expression calls

When a Declare Expression rule has `Result of decision table` for the Set Property To field, special processing occurs at runtime when a property referenced in the decision table is not present on the clipboard. Ordinarily such decision rules fail with an error message; in this case the Otherwise value is returned instead. For details, see the Pega Community article *Troubleshooting: Declarative Expression does not execute when a decision rule provides no return value.*

Not declarative

Despite the class name, the `Rule-Declare-DecisionTable` rule type does not produce forward or backward chaining. Technically, it is not a declarative rule type.

Decision tables *(on page 174)*
Creating decision tables *(on page 176)*
Viewing rule history *(on page)*

Unit testing a decision table

You can test a decision table individually, before testing it in the context of the application that you are developing. Additionally, you can convert the test run to a Pega unit test case. Unit testing a decision table involves specifying a test page for the rule to use, providing sample data as the input, running the rule, and examining the results.

1. In the navigation pane of Dev Studio, click **Records > Decision > Decision Table**, and then select the decision table you want to test.
2. Click **Actions > Run**.
3. In the **Data Context** list, click the thread in which you want to run the rule.
4. Select a method for creating the test page.
    - Select **Copy existing page** to copy values from a thread of an existing clipboard page to the main test page.
    - Select **Create or reset test page** to create a new test page or reset the values of an existing test page.
        - To apply a data transform to the values on the test page, click the link in the **Apply** field and then select a data transform.
        - To clear the Result pane, click **Reset Page**.
        - To switch the current context, select a thread in the **Data Context** list and then click **Switch Context**.
5. To display the test results, enter data in the Result panel and then click **Run again**.

    > The value that you enter and the result that is returned are the values that are used for the default decision result assertion that is generated when you convert this test to a test case.

6. **Optional:** To view the pages that are generated by the unit test, click **Show Clipboard**. For more information, see Clipboard pages created by the Run Rule feature *(on page 383)*.
7. To convert the test into a Pega unit test case, click **Convert to Test**. For more information, see Creating unit test cases for rules *(on page)*.
8. **Optional:** To view the row on the **Table** tab that produced the test result, click the **Result Decision Paths** link. If the **Evaluate All Rows** option on the **Results** tab is selected, all the rows that are true are highlighted.

Debugging decision trees with the Tracer *(on page 208)*
Unit testing individual rules *(on page 345)*
Decision tables *(on page 174)*
Running a unit test case *(on page)*
Viewing test case results *(on page)*
Exporting a list of test cases *(on page)*
Creating decision tables *(on page 176)*
Viewing rule history *(on page)*
More about decision tables *(on page 181)*

Debugging decision tables with the Tracer

If your decision table rule does not give you the results that you expect, and you cannot determine the problem by running the rule and examining the clipboard pages, run the Tracer tool. With the Tracer, you can watch each step in the evaluation of a decision table as it occurs.

1. On the developer toolbar, click **Tracer**.
2. Select the ruleset that contains the rule to be traced.
    a. In the Tracer window, click **Settings**.
    b. In the **Event Types to Trace** section, select the **Decision Table** check box.
    c. Select the ruleset that contains the rule to be traced.
    d. Clear the other rulesets to avoid memory usage from tracing events occurring in other rulesets.
    e. Click **OK**.
3. Return to the main portal and run the decision table.
4. Watch the Tracer output as the rule runs.

Unit testing a decision table *(on page 184)*

Decision trees

Use a decision tree to record `if .. then` logic that calculates a value from a set of test conditions organized as a tree structure on the Decision tab, with the 'base' of the tree at the left.

The following tabs are available on this form:

- Decision *(on page 190)*
- Input *(on page 206)*
- Results *(on page 200)*
- Test Cases *(on page 207)*

Where referenced

Rules of four other types can reference decision trees:

- In a flow rule, you can reference a decision tree in a decision shape, identified by the Decision shape  .
- In an activity, you can evaluate a decision tree using the Property-Map-DecisionTree method.
- A Declare Expression rule can call a decision tree.
- A collection *(on page)* rule can call a decision table.

Access

Use the Application Explorer to access decision trees that apply to work types in your current work pool. Use the Records Explorer to list all decision trees available to you.

Development

The Decision tab offers various formats and choices, depending on settings on the Results tab:

- For an advanced decision tree, complete the Input tab before the Decision tab.
- For a basic decision tree, complete the Results tab first. To restrict the results to one of a few constant values, complete the Results tab before the Decision tab.

After you complete initial development and testing, you can delegate selected rules to line managers or other non-developers. Consider which business changes might require rule updates and if delegation to a user or group of users is appropriate. For more details, see Delegating a rule or data type *(on page)*.

Category

Decision tree rules are instances of the `Rule-Declare-DecisionTree` class. They are part of the Decision category.

Viewing rule history *(on page)*

Creating decision trees

Calculate a value from a set of properties or conditions where true comparisons can lead to additional comparisons, organized and displayed as a tree structure, by creating a decision tree. For example, you can create a condition that checks whether the location of a job candidate is equal to a specific city. If the condition is true, your application evaluates additional conditions, such as work experience and education.

1. In the header of Dev Studio, click **Create > Decision > Decision Tree**.
2. In the **Label** field, enter a name that describes the purpose of the decision tree.
3. In the **Apply to** field, select the class in which you want to create the decision tree.
4. Click **Create and open**.
5. Select the branch to display the columns.
6. Define a condition and a result:

Choices	Actions
Define a single condition	a. In the first field, enter a property or a value. b. In the drop-down list, select a comparator. c. In the next field, enter a property or value that your application compares against the first property or value. d. In the **then** list, select **return**. e. In the last field, enter a property or value result that you want your application to return.
Define nested conditions	a. In the first field, enter a property or a value. b. In the drop-down list, select a comparator. c. In the next field, enter a property or value that your application compares against the first property or value. d. In the **then** list, select **continue**. e. Select the next branch to display the columns. f. Define a nested condition by providing a property or value, a comparator, and a result.

7. **Optional:** To create complex conditions, click **Add row**, and then repeat step 6 *(on page 187)*.
8. In the **otherwise** section, define the behavior of your application if all of the conditions evaluate as false:

Choices	Actions
Return a value	a. From the list, select **Return**. b. In the **Default return value** field, enter a value that you want to use. c. **Optional:** To configure your application to perform an action, click **Take actions**, click **Add a row**, and then define the action.
Perform an action **Note: In order to select this option, on the Configuration tab, select the Allow selection of additional return actions check box.**	a. From the list, select **Take action**. b. Click **Actions**. c. Click **Add a row**. d. Define an action by setting a value for the action property.

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

   A warning icon appears in rows that are unreachable or empty.

10. **Optional:** To increase the possibility of reaching a return value, test for completeness of the tree by clicking **Show completeness** on the toolbar.

    The system automatically adds suggested rows of the decision tree that cover additional cases.

11. Click **Save**.

Decision trees *(on page 186)*
Rules management *(on page)*
Viewing rule history *(on page)*
More about Decision Trees *(on page 203)*

Completing the Decision tab (Advanced format)

Record the `if.. then..` logic of the decision tree in the three-column array. These unlabeled columns are known as the comparison, action, and next value columns.

This help topic describes the advanced format of the **Decision** tab. If you encounter a **Decision** tab that does not contain an Evaluate Parameter or Evaluate property name see Completing the Decision tab (Basic format) *(on page 196)*.

At run time, the system evaluates the if portion of the array, starting at the top row, and continues until it reaches a `Return` statement. If the system processes the entire tree but does not reach a `Return` statement, it returns the `Otherwise` value.

The **Evaluate** field at the topic identifies the Property value, if any, from the **Configuration** tab. When this field is blank, the value is taken from a parameter of the Property-Map-DecisionTree method. If this decision tree was created in basic mode or if the **Allowed to Evaluate Properties?** box on the **Configuration** tab is not selected, the **Evaluate** field does not appear

If the **Redirect this Rule?** check box on the **Configuration** tab is selected, this circumstance-qualified rule is redirected and tab is blank.

Understanding the branch structure

Each indent level supports comparisons against a single value, which are determined by context:

- The top (leftmost) level tests values against the value of the property that is identified on the **Configuration** tab, or a parameter value specified in the method that calls the decision tree. Comparisons are implicit: the property on the **Configuration** tab (or the parameter in the method) is not displayed on this tab.
- An indented level tests values against a property identified in the **Evaluate** field of the statement above the indented level.
- Each text box can contain a value, a comparison operator followed by a value, or a Boolean expression. The context is not relevant when a Boolean expression is evaluated.

Using the controls

To display controls for that row or field, click it.

Control	Action
Collapse All	Click to hide subtree structures, or to hide specific subtree structures, click the minus sign.

Control	Action
Expand All	Click to show all the subtree structures, or to display specific subtrees, click the plus sign.
Open expression builder icon	Click to start the Expression Builder. This tool provides prompting and guidance when you create complex expressions involving functions. See Building expressions with the Expression Builder *(on page)*.
Open icon	Click to review a property for a field that contains a property reference.
Add row and **Delete row** buttons	Click to select a row. Then click the appropriate buttons to insert, append, or delete a row, respectively.
Show Conflicts	Click to analyze the consistency of the tree. This button displays a **Warning** icon next to any parts of the tree that are unreachable. For example, a branch that extends below the two mutually contradictory tests (if Width > 100) and (if Width < 100) is unreachable. To highlight the parts of the tree that cause that branch to be unreachable, click the **Warning** icon. A decision tree that contains no unreachable parts is called consistent. The presence of unreachable portions of the tree does not prevent you from saving the rule. Comparisons involving two properties such as Width > Length are ignored in this analysis. Conflicts are also checked when you save the form, and when you use the Guardrails landing page for the application. Conflicts do not prevent the rule from being validated or executed, but could indicate that a rule does not work as intended.
Show Completeness	Click to automatically add suggested portions of the decision tree that cover additional cases and reduce or eliminate the situations that fall through to the Otherwise Return expression. Suggested additions are displayed with a light green highlight and can refer to values that you must modify such as Result or DecisionTreeInputParam. These additions are only suggestions; you can alter or delete them.

Modifying branches with drag and drop operations

To move a subtree structure, drag the small circle at its left.

To copy a subtree structure, drag while holding down the CTRL key, and drop it on the destination node.

Completing fields in a branch

Field	Description
if / if value is	Enter a value for the current context, or a comparison starting with one of the six comparison operators: <, >, =, !=, >= or <=. The value can be an expression, such as a literal value between quotations or a `Single Value` property reference. (For more information, see Building expressions with the Expression Builder *(on page)*.) To select a pattern that helps you enter Boolean expressions, click the drop-down button. The form changes to reflect your pattern decision. If the **Action** field is set to `Otherwise`, this field is not visible .
(action)	Select an action from the selection list. The action that you choose determines which branch of this decision tree the system follows at run time when the condition to its left is reached and evaluates to true. Select a keyword: **Return** Causes this branch of the decision tree to end processing when reached. If the system finds a Return row to be true, the value in the right column of this row becomes the result of the decision tree evaluation. **Continue** Causes the next row of the decision tree to be nested within this branch. The system reflects the nesting by indenting the next row on the form display and changing the → arrow to ↴ so that it points down to the indented row. The context for the `Continue` statement is the same as for the current statement. **Evaluate** Causes the system to use a new property, identified in the right column, as the context for nested comparisons below the current row. Enter a `Single Value` property reference in the **Value** field to the right of the **Action** field. This choice is not available for decision trees that are created in basic mode, or when the **Allowed to Evaluate Properties** check box on the **Configuration** tab is not selected.

Field	Description
	Call Decision Tree Causes the system to evaluate another decision tree, which is identified in the field to the right of this value. The result of the second decision tree becomes the result of this decision tree, and evaluation ends. At run time, if this decision table evaluates in a backward-chaining context (the AllowMissingProperties parameter to the method is `true`), the system evaluates the called decision tree in the same way. This choice is not available for decision trees that are created in basic mode, or when the **Allowed to Call Decision** check box on the **Configuration** tab is not selected. **Call Map Value** Causes the system to evaluate a map value, identified in the next field. The result of the map value becomes the result of this decision tree, and evaluation ends. At run time, if this decision table evaluates in a backward-chaining context (the AllowMissingProperties parameter to the method is true), the system evaluates the called map value in the same way. This choice is not available for decision trees that are created in basic mode, or when the **Allowed to Call Decision** check box on the **Configuration** tab is not selected. **Call Decision Table** Causes the system to evaluate a decision table, identified in the next field. The result of the decision table becomes the result of this decision tree, and evaluation ends. At run time, if this decision table evaluates in a backward-chaining context (the AllowMissingProperties parameter to the method is true), the system evaluates the called map value in the same way. This choice is not available for decision trees that are created in basic mode, or when the **Allowed to Call Decision** check box on the **Configuration** tab is not selected. **Otherwise**

Field	Description
	Select `Otherwise` only as the final choice in a set of alternatives. The value in the right column of this bottom row becomes the result of this decision tree evaluation.
→ **(next value)**	Identify a target based on the action value. If you selected `Return` as the action and the **Configuration** tab is not blank, select one of the values listed on the **Configuration** tab. Otherwise, enter a value or expression here that allows evaluation of the decision tree to continue. You can reference a property on any page, but be sure to enter any page you reference on the **Pages & Classes** tab. Enter a value that depends on the one of the following action value keywords: **Return or Otherwise** Enter an expression for the result of this decision tree when this row is the final one evaluated. **Evaluate** Identify a property reference that the system uses at run time to evaluate the nested comparisons beneath the row that contains the `Evaluate` action. This option is not available for decision trees that are created in basic mode, or when the **Allowed to Evaluate Properties** check box on the **Configuration** tab is not selected. **Call Decision Tree** Select another decision tree. The result of that rule becomes the result of this rule. **Call Map Value** Select a map value. The result of that rule becomes the result of this rule. **Call Decision Table** Select a decision table. The result of that rule becomes the result of this rule. **Call Base Decision Tree** Available only for decision trees that are circumstance-qualified. When selected, the base (or non-qualified) decision tree of the same name, ruleset, and version is executed. **Take Action**

Field	Description
	Set one or more properties to values as the only outcome of the decision tree. This ends evaluation of the rule, returning the null string as its result. This capability is not available for decision trees that are created in basic mode, or when the **Allowed to Take Action** check box on the **Configuration** tab is not selected. This input field is not displayed when the action value is `Continue`. To open a referenced decision tree, map value, or decision table, Click the **Open icon**. (The `Call Decision Tree`, `Call Map Value`, and `Call Decision Table` choices are not available for decision trees that are created in basic mode, or when the **Allowed to Call Decisions?** field on the **Configuration** tab is not selected.)
	Click to access an optional array of properties and values. To hide this array, click  . When the system evaluates the decision tree at run time and this row is the source of the results, the system also recomputes the value of the target properties that are identified in this array. Order is significant. This capability is not available for decision trees that are created in basic mode, or for decision trees when the **Allowed to Set Take Action?** check box on the **Configuration** tab is not selected.
Property	Optional. Identify a property reference corresponding to a value to be set.
Value	Enter a value for that property — a constant, property reference, or expression.

Completing the Otherwise branch

Field	Description
Otherwise	
Return	To define the outcome of this decision tree when no result is determined by the tree structure above, choose `Return` or `Take Action`. If this field is blank and no other return value is computed, the system returns the null value. **Return**

Field	Description
	Choose `Return` to specify a value to return if an earlier branch does not return a value. If the Allowed Results list on the **Configuration** tab is not blank, this field is required and is limited to one of the constant values that are listed on that tab. For guided assistance in entering an expression start the Expression Builder by clicking the **Open expression builder** icon. **Note:** During backward chaining computations for Declare Expression rules, if the Otherwise Return value can be computed, but properties that are needed for other parts of the form are not defined, the Otherwise Return value is returned as the value of the decision table. **Take Action** Choose `Take Action` (when this choice is visible) to return the empty string as the value of the decision tree, but to also evaluate a function that is identified by an alias in the **Allowed Action Functions** field of the **Configuration** tab. Most commonly, the `Take Action` choice allows one or more property values to be set as the outcome of a decision tree. Select a property in the **Set** field. Enter a value for the property in the **Equal to** field.

Decision trees *(on page 186)*
Creating decision trees *(on page 187)*
Viewing rule history *(on page)*
More about Decision Trees *(on page 203)*

Completing the Decision tab (Basic format)

Record the `if.. then..` logic of the decision tree in this array, which has three columns. The unlabeled columns are known as the comparison, action, and next value columns.

This help topic describes the basic format of the **Decision** tab. If you encounter a **Decision** tab that contains an Evaluate Parameter or Evaluate property name, see Completing the Decision tab (Advanced format) *(on page 196)*.

At run time, the system evaluates the if portion of the array, starting at the top row, and continues as described here until it reaches a `Return` statement. If the system processes all rows but does not reach a `Return` statement, it returns the `Otherwise` value.

If the **Redirect this Rule?** check box on the **Results** tab is selected, this circumstance-qualified rule is redirected and this tab is blank.

Understanding the rows

Each text box can contain a value, a comparison operation for two values, followed by an outcome. The comparison can be between two properties or between a property and a constant value.

To make controls for a row or field visible, click that field :

- To hide subtree structures, click **Collapse All**. To hide specific subtree structures, click the minus sign.
- To show all subtree structures, click **Expand All**. To display specific subtrees, click the plus sign
- To review a property for a field that contains a property reference, click the **Open icon**.
- To append or delete a row, select a row and then click the **Add** or **Delete** icon, respectively.

Field	Description
if / if value is	Enter a comparison by using one of the six comparison operators: <, >, =, !=, >= or <=. The value can be a constant or a `Single Value` property reference. If the **Action** field is set to `Otherwise`, this field is not visible.
(action)	Select an action from the selection list. The action that you choose determines which branch of this decision tree the system follows at run time when the condition to its left is reached and evaluates to true. Select a keyword: **Return** Causes this branch of the decision tree to end processing when reached. If the system finds a `Return` row to be true, the value in the right column of this row becomes the result of the decision tree evaluation. **Continue** Causes the next row of the decision tree to be nested within this branch. The system reflects the nesting by indenting the next row on the form display and changing the → arrow to ↴ to point down to that indented row. The context for the continue statement is the same as for the current statement. **Call Decision Tree**

Field	Description
	Causes the system to evaluate another decision tree, identified in the next field. This choice might not be present in all cases, depending on settings on the **Results** tab. At run time, if this decision table evaluates in a backward-chaining context (the AllowMissingProperties parameter to the method is true), the system evaluates the called decision tree in the same way. **Call Map Value** Causes the system to evaluate a map value, identified in the next field. This choice might not be present in all cases, depending on settings on the **Results** tab. At run time, if this decision table evaluates in a backward-chaining context (the AllowMissingProperties parameter to the method is true), the system evaluates the called map value in the same way. **Call Decision Table** Causes the system to evaluate a decision table, identified in the next field. This choice might not be present in all cases, depending on settings on the **Results** tab. At run time, if this decision table evaluates in a backward-chaining context (the AllowMissingProperties parameter to the method is true), the system evaluates the called decision table in the same way. **Otherwise** Select `Otherwise` only as the bottom, final choice in a set of alternatives, marking the final choice. The value in the right column of this row becomes the result of this decision tree evaluation.
(next value)	Identify a target based on the action value. If you selected `Return` as the action and the **Results** tab is not blank, select one of the values listed on the **Results** tab. Otherwise, enter a value or expression here that allows the evaluation of the decision tree to continue. You can reference a property on any page, but be sure to enter any page you reference on the **Pages & Classes** tab. Enter a value that depends on the action value keyword:

Field	Description
	• `Return` or `Otherwise` — Enter an expression for the result of this decision tree when this row is the final one evaluated. • `Call Decision Tree` — Select another decision tree. The result of that rule becomes the result of this rule. This choice might not be present in all cases, depending on settings on the **Results** tab. • `Call Map Value` — Select a map value. The result of that rule becomes the result of this rule. This choice might not be present in all cases, depending on settings on the **Results** tab. • `Call Decision Table` — Select a decision table. The result of that rule becomes the result of this rule. This choice might not be present in all cases, depending on settings on the **Results** tab. This input field is not displayed when the action value is `Continue`. To open a referenced decision tree, map value, or decision table, click the **Open icon**.
Expand icon	Click to access an optional array of properties and values. To hide this array, click the **Collapse** icon . This choice might not be present in all cases, depending on settings on the **Results** tab. When the decision tree evaluates and this row is the source of the results, the system also recomputes the value of the target properties that are identified in this array. Order is significant.
Property	Optional. Identify a property reference to be set.
Value	Enter a value for that property.
Otherwise	
Return	Optional. Enter an expression defining a value to return when the decision tree evaluation does not return another value. When the Allowed Results list on the **Results** tab is not blank, this field is required and limited to one of the constant values listed on that tab. If this field is blank and no other return value is computed, the system returns the null value.

Completing the Configuration tab *(on page 200)*
Decision trees *(on page 186)*
Creating decision trees *(on page 187)*
Viewing rule history *(on page)*
More about Decision Trees *(on page 203)*

Completing the Pages & Classes tab

The system uses this tab to locate properties on the clipboard.

The system completes a row from the Applies To key part of this decision tree. If your decision tree does not reference any properties other than those in the Applies To class, you do not need to add other rows this array.

See How to Complete a Pages & Classes tab *(on page)* for basic instructions.

Field	Description
Page Name	Optional. Enter the name of the clipboard page on which the property or properties are to be found at runtime. Optionally, add a row with the keyword `Top` as the page name, to identify a top-level page. The Top keyword allows you to use the syntax Top.propertyref to identify properties, on other tabs of this rule form. Decision tree rules can apply to embedded pages that appear within top-level pages with various names. In such cases, you can use the keywords `Top` or `Parent` in the page name here.
Class	Optional. Select the class of that page.

About Decision Trees *(on page 186)*

Decision trees *(on page 186)*
Creating decision trees *(on page 187)*
Viewing rule history *(on page)*
More about Decision Trees *(on page 203)*

Completing the Configuration tab

Complete the fields on this tab to restrict the possible values returned by this decision tree. Additional options allow you to control the actions that other users can take on the Decision tab.

Options

The fields in this section impact the initial presentation and available options on the Decision tab of the decision tree.

For example, you can prevent users from calling specific function aliases or adding new nodes to the tree structure. This helps you customize the development experience for delegated users, such as line managers, who may not require access to the full set of decision tree options.

All users, including delegated users, can remove these restrictions if they hold a rule editing privilege.

Field	Description
Allow changes to function lists	Select this check box to allows users to change the function aliases called by each tree node on the Decision tab. Clear this check box to hide the function alias picker on the Decision tab. Users with rule editing privileges can still update the constant values in each tree node. With this option selected you can: • Populate the Functions Allowed list to restrict the function aliases a user can select. Function aliases commonly used by managers include: `CompareTwoValues`, `allEntriesSatisfyCondition`, and `anyEntrySatisfiesCondition`. • Leave the Functions Allowed list empty to let users select any available function alias. • Open any function alias in the Functions Allowed list.
Allow adding of nodes to the decision tree	Select this check box to allow users to append and insert top-level tree nodes on the Decision tab. Clear this check box to hide the **add icon** on the Decision tab.
Allow selection of 'evaluate property' option	Select this check box to allow users to evaluate the value of a property *(on page 206)* from a tree node on the Decisiontab. Clear this check box to hide the **evaluate** option from **then** drop-down list on the Decision tab.

Field	Description
	You must have the **Allow adding of nodes to the decision tree** option selected before you can change the state of this check box.
Allow selection of 'call decision' option	Select this check box to allow users to call a map value, decision tree, or decision table from a tree node on the Decision tab. Clear this check box to hide decision rules from the list of available options in the **then** statement of the Decision tab. You must select the **Allow adding of nodes to the decision tree** option before you can change the state of this check box.
Allow selection of additional return actions	Select this check box to make the **Take Action** option visible on the Decision tab. Users can take action within each tree node or as part of the **otherwise** statement on the Decision tab. With this option selected, you can: • Populate the **Allowed Action Functions** list to restrict the function aliases a user can call from an action. The `setPropertyValue` function alias is commonly used by managers. • Leave the **Allowed Action Functions** list empty to let users call any available function alias. • Open any function alias in the **Allowed Action Functions** list.

Results

Use the options in this section of the tab to define the possible values this decision tree can return. You can also specify a list of preset properties that are calculated before the decision tree runs.

To define allowed results:

1. Enter a property or linked property name in the **Results defined by property** field.

   This property must use table validation *(on page)* because the table values are used to populate the **Result** field.

2. Select a value from the **Result** list.

Alternatively, you can enter a string value without quotes to supplement the existing values from the property that uses table validation.

3. Define a list of **Target Property** and **Value** pairs that are set when the decision tree returns the corresponding **Result**.

   You can enter a constant, property name, or expression in the **Value** fields.

4. Repeat steps 2 and 3 as necessary.

At run time, the system sets target properties using the order you specify.

To define preset properties:

1. Enter a property name in the **Property** field.
2. Enter a constant, property name, expression, or input parameter in the **Value** field.
3. Click the **add icon** and repeat this process for as many properties as are required.

These properties are set before the Decision tab is processed.

Decision trees *(on page 186)*
Creating decision trees *(on page 187)*
Viewing rule history *(on page)*
More about Decision Trees *(on page 203)*

More about Decision Trees

Overriding the property

Decision tree evaluation may be based on a known property identified on the Configuration tab, or on a parameter supplied in the Property-Map-DecisionTree method, or both.

If you leave the Property field blank, evaluation is always based on the parameter. If a parameter is supplied, the parameter value is used even when the Property field is not blank.

Standard functions

As an alternative to the Property-Map-DecisionTree method, you can use these standard functions to evaluate a decision tree:

```
@(Pega-RULES:DecisionTree).ObtainValue(tools, myStepPage, decisiontree, inputproperty)
```

```
@(Pega-RULES:DecisionTree).ObtainValue(tools, myStepPage, decisiontree, inputproperty, bAllowMissingProperties)
```

Decision tree rules can also be evaluated as part of a collection *(on page)* rule (`Rule-Declare-Collection` rule type).

Pega Community note

See the Pega Community article *How to evaluate a decision tree and handle errors* for an example.

Uploading a text file to start

You can create a decision tree by importing (or "harvesting) a specially formatted text file. This capability lets others not familiar with the Pega Platform create decision trees.

Performance

The number of nodes in a decision tree is not limited. However, as a best practice to avoid slow performance when updating the form and also avoid the Java 64 KB code maximum, limit your decision trees to no more than 300 to 500 rows.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Special processing with Declare Expression calls

When a Declare Expression rule has `Result of decision tree` for the Set Property To field, special processing occurs at runtime when a property referenced in the decision table is not present on the clipboard. Ordinarily such decision rules fail with an error message; in this case the Otherwise value is returned instead. For details, see the Pega Community article *Troubleshooting: Declarative Expression does not execute when a decision rule provides no return value.*

Not declarative

Despite the class name, the `Rule-Declare-DecisionTable` rule type does not produce forward or backward chaining. Technically, it is not a declarative rule type.

Custom function aliases

You can use **Configuration** tab to enable custom function aliases instead of the default comparison. For some of those aliases, you can click **Select values** and select from a list of values that are available for the selected property.

Application debugging by using the Tracer tool *(on page 347)*
Decision trees *(on page 186)*
Creating decision trees *(on page 187)*
Viewing rule history *(on page)*

More about Decision Trees

Overriding the property

Decision tree evaluation may be based on a known property identified on the Configuration tab, or on a parameter supplied in the Property-Map-DecisionTree method, or both.

If you leave the Property field blank, evaluation is always based on the parameter. If a parameter is supplied, the parameter value is used even when the Property field is not blank.

Standard functions

As an alternative to the Property-Map-DecisionTree method, you can use these standard functions to evaluate a decision tree:

```
@(Pega-RULES:DecisionTree).ObtainValue(tools, myStepPage, decisiontree, inputproperty)
```

```
@(Pega-RULES:DecisionTree).ObtainValue(tools, myStepPage, decisiontree, inputproperty, bAllowMissingProperties)
```

Decision tree rules can also be evaluated as part of a collection *(on page)* rule (`Rule-Declare-Collection` rule type).

Pega Community note

See the Pega Community article *How to evaluate a decision tree and handle errors* for an example.

Uploading a text file to start

You can create a decision tree by importing (or "harvesting) a specially formatted text file. This capability lets others not familiar with the Pega Platform create decision trees.

Performance

The number of nodes in a decision tree is not limited. However, as a best practice to avoid slow performance when updating the form and also avoid the Java 64 KB code maximum, limit your decision trees to no more than 300 to 500 rows.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Special processing with Declare Expression calls

When a Declare Expression rule has `Result of decision tree` for the Set Property To field, special processing occurs at runtime when a property referenced in the decision table is not present on the clipboard. Ordinarily such decision rules fail with an error message; in this case the Otherwise value is returned instead. For details, see the Pega Community article *Troubleshooting: Declarative Expression does not execute when a decision rule provides no return value.*

Not declarative

Despite the class name, the `Rule-Declare-DecisionTable` rule type does not produce forward or backward chaining. Technically, it is not a declarative rule type.

Custom function aliases

You can use **Configuration** tab to enable custom function aliases instead of the default comparison. For some of those aliases, you can click **Select values** and select from a list of values that are available for the selected property.

Function Alias rules *(on page)*
Application debugging by using the Tracer tool *(on page 347)*
Decision trees *(on page 186)*
Creating decision trees *(on page 187)*
Viewing rule history *(on page)*

Configuring property evaluation in a decision tree

The run-time result of a decision tree can depend on the value of a property or the optional, third parameter of the Property-Map-DecisionTree method.

The following fields are visible when the **Allow selection of 'evaluate property' option** check box on the Configuration tab is selected. Use them to configure the property used by **evaluate** nodes in the decision tree.

Field	Description
Data Type	Choose `String`, `Number`, or `Boolean` to specify how the system evaluates the comparisons defined on the Decision tab when an optional parameter value is supplied in the Property-Map-DecisionTree method.

Field	Description
	The Data Type value you select affects comparisons on the Decision tab when the system obtains the input value as a method parameter. For example, if the method parameter is "007" and the Data Type is `String`, then a comparison of "007" < "7" is true. If the method parameter is "007" and the Data Type is `Number`, then the comparison of "007" < "7" is false. The Data Type is ignored when the property identified in the next field is used at runtime. In that case, comparisons depend on the type of that property. This Data Type is independent of — and need not match — the type of the property to contain the decision tree result (the first parameter to the Property-Map-DecisionTree method). For example, you can evaluate comparisons of inputs based on numbers and return a result property of type `Text`.
Property	Optional. Enter a `Single Value` property reference, or a literal value between double quotes. (If your property reference doesn't identify a class, the system uses the Applies To portion of the key to this decision tree as the class of the property). At runtime, if the value of the third parameter to the Property-Map-DecisionTree method is blank, the system uses the value of this property for comparisons.
Label	Optional. Enter a text label for the input property. This label appears on the Decision tab. Choose a meaningful label, as certain users may see and update only the Decision tab.

About Decision Trees *(on page 186)*

Decision trees *(on page 186)*
Creating decision trees *(on page 187)*
Viewing rule history *(on page)*
More about Decision Trees *(on page 203)*

Unit testing a decision tree

You can use the **Run Rule** feature to test a decision tree individually before testing it in the context of the application that you are developing.

You specify a test page for the rule to use, provide sample data as the input, run the rule, and examine the results. Additionally, you can convert the test run to a Pega unit test case.

1. In the navigation pane of Dev Studio, click **Records > Decision > Decision Tree**, and then select the decision tree you want to test.
2. Click **Actions > Run**.
3. In the **Data Context** list, click the thread in which you want to run the rule.
4. Select a method for creating the test page.
    - Select **Copy existing page** to copy values from a thread of an existing clipboard page to the main test page.
    - Select **Create or reset test page** to create a new test page or reset the values of an existing test page.
        - To apply a data transform to the values on the test page, click the link in the **Apply** field and then select a data transform.
        - To clear the Result pane, click **Reset Page**.
        - To switch the current context, select a thread in the **Data Context** list and then click **Switch Context**.
5. To display the test results, enter data in the Result panel and then click **Run again**.

    > The value that you enter and the result that is returned are the values that are used for the default decision result assertion that is generated when you convert this test to a test case.

6. **Optional:** To view the pages that are generated by the unit test, click **Show Clipboard**. For more information, see Clipboard pages created by the Run Rule feature *(on page 383)*.
7. To convert the test into a Pega unit test case, click **Convert to Test**. For more information, see Creating unit test cases for rules *(on page)*.
8. **Optional:** To view the row on the **Decision** tab that produced the test result, click the **Result Decision Paths** link.

Unit testing individual rules *(on page 345)*
Decision trees *(on page 186)*
Running a unit test case *(on page)*
Viewing test case results *(on page)*
Exporting a list of test cases *(on page)*
Creating decision trees *(on page 187)*
Viewing rule history *(on page)*
More about Decision Trees *(on page 203)*

Debugging decision trees with the Tracer

If your decision tree does not give you the results you expect and you cannot determine the problem by running the rule and examining the clipboard pages, run the Tracer tool. With the Tracer you can watch each step in the evaluation of a decision tree as it occurs.

1. On the developer toolbar, click **Tracer**.
2. Select the ruleset that contains the rule to be traced.
    a. In the Tracer window, click **Settings**.
    b. In the **Event Types to Trace** section, select the **Decision Tree** check box.
    c. Select the ruleset that contains the rule to be traced.
    d. Clear the other rulesets to avoid memory usage from tracing events occurring in other rulesets.
    e. Click **OK**.
3. Return to the main portal and run the decision tree.
4. Watch the Tracer output as the rule runs.

Unit testing a decision tree *(on page 207)*

Declare Expression rules

Declare Expression rules automatically calculate values based on other values that are available in your application. When you include Declare Expression rules in calculations, you increase automation and efficiently reuse resources.For example, you can use a Declare Expression rule to automatically calculate a tax amount for an order in an online shop, based on the prices of items that the order includes.

Declare Expression rules use two methods of calculating values: forward chaining and backward chaining. Starting with Pega Platform version 8.3, Pega Platform automatically applies the most suitable calculation method to each business scenario.

Forward chaining

Forward chaining occurs when the target property changes because the input values in the calculation change. Consider a scenario in which a customer creates an order in an online shop by selecting items that sum up to $60. A shopping application calculates the tax value that is 10% of the order amount and updates the target property with the value of $6. When the customer adds another item to the shopping cart, the application adds the price of the new item to the calculation. Consequently, input values change. The application performs forward chaining and updates the target value, which is the tax amount, based on the updated input values.

Backward chaining

Backward chaining occurs when an application seeks values of input parameters to calculate a target property. For example, in an online shopping application, when a customer provides a code to get a discount on specific items in a shopping order, the application seeks properties that hold prices of the selected items, and then updates the total order amount. In backward chaining, target values remain

unchanged if input values change, and the application performs calculation only after receiving a call for a target property.

Backward chaining directly corresponds to the *Property-Seek-Value* method. The *Property-Seek-Value* method can access the computational relationships among properties in a Declare Expression rule to identify source values that are needed to compute a missing property value.

Backward chaining is also applicable in the following specific scenarios:

- Some of the fields in the expression are from data pages that are outside of the current page structure. For example, consider a scenario in which you want to sum up values of multiple sales opportunities, but each opportunity is a separate and independent page.
- A declare expression invokes a function that has no parameter, such as *calculateEmployeeCount()*. In such cases with no parameter, change tracking does not happen and the system applies backward chaining.

Users are no longer required to explicitly specify forward or backward chaining. Users can switch to legacy expressions if they want to explicitly specify a change tracking method.

Category

Declare Expression rules are instances of the *Rule-Declare-Expressions* rule types that are part of the Decision category.

Note: Do not confuse Declare Expression rules with simple expressions. Expressions — which contain syntax that includes constants, function calls, operators, and property references — are often applicable in addition to Declare Expression rules.

Expressions *(on page 281)*
Viewing rule history *(on page)*
Property-Seek-Value method *(on page)*

Creating Declare Expression rules

Create Declare Expression rules to automatically calculate values in your application. Declare Expression rules take advantage of values that exist in your system to calculate values of target properties. As a result, you increase automation of calculations in your application.For example, in an online shopping order, an application can automatically calculate the tax value based on prices of the items and the information

that tax equals a specific percentage of the item price. By implementing Declare Expression rules, the application recalculates the tax value every time the customer updates the shopping order.

1. In the header of Dev Studio, click **Create > Decision > Declare Expression**.
2. In the **Target Property** field, enter a property that you want to calculate, preceding the property name with a period.
3. **Optional:** To restrict search for the result of the declare expression to a specific page, in the **Page Context** field, enter a data page that stores available results.
   For more information, see Data pages overview *(on page)*.
4. In the **Context** section, select the application layer in which you want to store the rule.
5. In the **Apply to** field, enter the class that stores the target property.

> **Note:** When you create a declare expression, you can first provide the class, and then provide the target property in the **Target Property** field. If you select a class first, the system limits the available target properties to the properties that the selected class stores.

6. In the **Add to ruleset** field, enter a ruleset to store the rule.
7. Click **Create and open**.

Declare Expression rules *(on page 209)*
Viewing rule history *(on page)*
More about Declare Expression rules *(on page 214)*

Defining expressions in Declare Expression rules

Define how your application calculates target properties by building expressions. When you build expressions, you create relations between properties so that an application can automatically update property values at run time. As a result, you increase the level of automation and flexibly respond to changing circumstances.For example, you can create a Declare Expression rule that automatically calculates shipping costs based on the total weight of all items that a customer orders in an online shop. When the customer adds items to the order and the total weight changes, a Declare Expression rule automatically updates the shipping costs.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **Declare Expression**.
3. In the list of instances of declare expressions, open the instance that you want to edit.
4. **Optional:** To provide additional information about the rule, on the **Expressions** tab, in the **Overview** section, enter a brief description:

a. In the **Description** field, briefly explain the purpose of the rule.

b. In the **Usage** field, briefly explain the scenarios and elements of your application where the declare expression applies.

> **Tip:** Providing additional information about a declare expression is very useful to quickly share details of the rule with other developers.

5. In the **Build Expressions** section, click **Add condition**.
6. In the **IF** field, enter or select a property to evaluate at run time.
   The list includes properties from the Apply To class that you selected during rule creation. For more information, see Creating Declare Expression rules *(on page 210)*.
7. In the list of comparators, select a comparator to apply during property evaluation.
8. In the text field, provide a property to evaluate against the property from the **IF** field.
9. **Optional:** To define more conditions, click **Add a row**, add a condition before or after the current condition, and then repeat steps 6 *(on page 212)* through 8 *(on page 212)*.

   > By default, the system groups conditions by using the AND comparator. At run time, your application applies the result after all the conditions with the AND comparator evaluate to true.

10. **Optional:** If you have multiple rows, to apply the result of the expression when any of the rows evaluates to true, click **AND**, and then click **OR**.
11. In the **Then set** row for your target property, define how the declare expression determines the target property:
    - To determine the target property as a result of a string, in the list, select **Value of**, and then, in the text field, enter the string that calculates the property.
    - To determine the target property as a result of a decision tree, in the list, select **Result of Decision Tree**, and then, in the text field, enter a decision tree to compute at run time.
    - To determine the target property as a result of a decision table, in the list, select **Result of Decision Table**, and then, in the text field, enter a decision table to compute at run time.
    - To determine the target property as a result of a map value, in the list, select **Result of Map Value**, and then, in the text field, enter a map value to evaluate at run time.
12. **Optional:** To add more conditions to the expression, click **Add condition**, and then repeat steps 6 *(on page 212)* through 11 *(on page 212)*.

> **Note:** If you create multiple conditions, at run time, processing stops after your application determines the first expression that evaluates to true.

13. In the **Set** row for the target property, provide that you want the system to apply if all the conditions evaluate to false:
    - To determine the target property as a result of a string, in the list, select **Value of**, and then, in the text field, enter the string that calculates the property.
    - To determine the target property as a result of a decision tree, in the list, select **Result of Decision Tree**, and then, in the text field, enter a decision tree to compute at run time.
    - To determine the target property as a result of a decision table, in the list, select **Result of Decision Table**, and then, in the text field, enter a decision table to compute at run time.
    - To determine the target property as a result of a map value, in the list, select **Result of Map Value**, and then, in the text field, enter a map value to evaluate at run time.
14. Click **Save**.

Declare Expression rules *(on page 209)*
Creating Declare Expression rules *(on page 210)*
Viewing rule history *(on page)*
More about Declare Expression rules *(on page 214)*

Specifying pages and classes for a declare expression

Define pages and classes that store properties to ensure that declare expressions can access these properties to correctly calculate values for your business scenario. As a result, you increase the level of automation in your application because automatic calculations can occur without user intervention.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **Declare Expression**.
3. In the list of instances of declare expressions, click the expression that you want to edit.
4. On the rule form, click the **Pages & Classes** tab.
5. In the **Page name** field, enter the name of the page that contains the property that this rule references.

> **Note:**

To identify a top-level page, you can add a row with the keyword `Top` as the page name. With the `Top` keyword, you can use the syntax `Top.`*propertyref* to identify properties that you reference on other tabs of this rule form.

Declare Expression rules can apply to embedded pages that appear within top-level pages with various names. In such cases, you can use the keywords `Top` or `Parent` in the page name. If the **Page Context** key part of this rule is not blank, this tab automatically includes a definition of the `Top` and `Parent` keywords.

6. In the **Class** field, select the class of the page.
7. **Optional:** To add more pages and classes, click **Add item**, and then repeat steps 5 *(on page 213)* and 6 *(on page 214)*.
8. Click **Save**.

Defining the pages and classes of a rule *(on page)*
Declare Expression rules *(on page 209)*
Creating Declare Expression rules *(on page 210)*
Viewing rule history *(on page)*
More about Declare Expression rules *(on page 214)*

More about Declare Expression rules

This topic provides additional information about declare expression rules.

Avoid direct updates to target properties

CAUTION: Do not change the value of a property computed by a Declare Expression rule through the Property-Set method, application of a data transform, or user input into a form. As you develop your application, the Pega Platform attempts to detect and prohibit such situations.

For example, if someone creates a Declare Expression rule that computes the property AverageWait, a developer tomorrow cannot save an activity that uses the Property-Set method to change the value of AverageWait.

However, if yesterday — before the Declare Expression rule was defined — someone created and saved an activity that set the value of AverageWait, then after today both the activity and the Declare Expression rule can execute to save the value. This practice is not recommended.

To reduce the risk of similar rule conflicts, you can adopt a naming convention for properties designed to be computed in Declare Expression rules, and create Declare Expression rules early, using stub or dummy expressions.

⚠ **CAUTION:** At run time, the system neither detects nor prevents the value of a target property from changing through user input, through the Property-Set method, or by data transforms. If the value changes through these or other direct means, the target property value can temporarily not equal the last computed expression value. This situation is corrected the next time any input values for the expression change.

How this rule runs with forward chaining

For more information about forward chaining, see forward chaining

If you select `Whenever inputs change` in the Compute Values field, then each time the value of any property referenced in any Declare Expression rule — or properties in other rules (such as decision trees, decision tables, or map values) referenced in the Declare Expression rule — changes, the system computes the values of the target property.

Ordinarily, code your activity to place all properties of interest on the clipboard before the activity accesses the value of a property referenced in the key of a Declare Expression rule. Your activity can create placeholder values (with Property-Set or with a data transform), or create the properties by opening instances with the Obj-Open method.

If the Declare Expression rule contains a non-blank Page Context field, the expression is evaluated at run time only when the clipboard contains a page matching that full context.

When more than one Declare Expression rule is to run, you cannot control or predetermine the order in which the multiple Declare Expression rules run.

Forward chaining does not create new embedded pages. Declare expressions that have a non-blank **Page Context** field and that use `top` or `parent` keywords do not run unless at least one embedded page exists at each level of the page context.

When a property is the target of two Declare Expression rules, one context-free and the second with a context, the rule with the context runs, and the system ignores the context-free rule. If two context-free Declare Expression rules reference the same property by using distinct property reference forms, for

example *workpage.targetproperty* and *.targetproperty*, the rule with the longer reference runs, and the system igonres the rule with the shorter reference.

Sample rules for backward chaining (goal seeking)

The Property-Seek-Value method uses Declare Expression rules to compute a property value on request. For an example, the standard flow action named `Work-.VerifyProperty` calls the standard activity `Work-.VerifyProperty`. If you use this flow action in a flow rule, a user can select it to cause the system to use goal-seeking to compute the value of the `pyResolutionCost` property.

If the backward chaining process fails, it can indicate a property with no current value that if set could aid in the computation. Your flow can then prompt a user for help or for a value that can allow the computation to complete.

Primary Page

During execution of a Declare Expression rule, the page on which the rule operates temporarily becomes the primary page. The page keyword `PRIMARY` and the results of the `tools.getPrimaryPage()` PublicAPI method reflect this change. When the rule execution completes, the primary page of the calling activity resumes as primary.

Embedded properties

Declare expressions do not support displaying values of target properties in user interface if the target property is an embedded property and if the calculation engages forward chaining. Declare expressions always display property values if the calculation uses backward chaining. During forward chaining, the system might render the UI before populating the clipboard, and the properties are not visible in the UI. To show updated values, define refresh conditions in the UI to get new values from the server when the values change. For more information, see Defining refresh conditions for UI areas *(on page)*.

Declarative Network Analysis gadget

In Dev Studio, click **Configure > Case Management > Business Rules > Declarative Network** to view and interact with the declarative expression rules in your application.

Alias Function Rules

The contents of the selection lists on the Expressions tab depend on property alias rules and alias function rules.

Testing and debugging Declare Expression rules

Using the Tracer tool, you can watch the evaluation of a Declare Expression rule. Start the Tracer tool and select a requestor session. Click **Settings** and check the Declare Expression box in the Event Types to Trace section. Also check the RuleSet that contains the rule you want to trace.

The statistic Tracked Property Changes on the full details page of the Performance tool shows how many property changes have occurred (for the current requestor since log-in) that are tracked for declarative rules computations.

Like other rules, Declare Expression rules won't evaluate as expected if the RuleSets needed for correct execution are not available to the requestor at run time. For an example, see the Pega Community article *Declarative rules require access to correct input property rules*.

Workstation display of calculated values using AJAX

When appropriate, your application can recompute the value of target properties (presented as read-only fields) immediately as a user changes an input value on a user form or flow action form, rather than later when the form is submitted. Users can see the new value immediately.

For example, the target property can represent an order total amount for a sales order. As a user enters and revises sales details, the total changes immediately as user focus leaves an input field.

> **Note:** This feature can improve user productivity and accuracy, while also reducing the number of server interactions and HTTP traffic required to complete a valid input form. Consider whether and where such interactive operation can simplify the user's task of completing complex input forms in your application.

To implement this capability:

1. Use this feature on flow actions or harnesses that use the SmartFrames layout and JSP tags.
2. Include at least one input to the expression as another field or flow action. You can't place the input in the flow action form and the target in the harness form or vice versa.
3. Select the Enable Expression Calculation? box on the HTML tab of the Flow Action form or Harness form.
4. Test.

This feature uses AJAX between the browser client and the server.

Special processing for map value, decision table, and decision tree calls

When a Declare Expression rule has `Result of decision table`, `Result of decision tree` or `Result of map value` for the Set Property To field, special processing occurs at run time when a property referenced in that decision rule is not present on the clipboard. Ordinarily such decision rules fail with an error message; in this case the system returns the Otherwise value instead. For details, see the Pega Community article *Troubleshooting: Declarative Expression does not execute when a decision rule provides no return value*.

Advanced debugging

Use the Tracer tool to detect that a Declare Expression rule executes when expected. For more detailed debugging help, use logging categories.

For more information, see Creating custom log categories *(on page)*.

OnChange rules

Declare Expression rules do not evaluate during the execution of an activity of type `OnChange`. Such executions are typically brief.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Application debugging by using the Tracer tool *(on page 347)*
Declare Expression rules *(on page 209)*
Creating Declare Expression rules *(on page 210)*
Viewing rule history *(on page)*

Declare Expression form - Completing the Change Tracking tab

Complete this tab to determine the conditions that cause automatic recomputation of the expression. These conditions affect how often the computation on the **Expressions** tab is performed.

> **Note:** Declare expressions that are created as of Pega Platform 8.3 do not include the **Change tracking** tab. To use the legacy declare expression **Change tracking** tab, click **Actions > Use legacy expression**.

Note: Change tracking applies to properties on the primary page — the page that matches the Applies To key part of the rule. Changes to properties on other pages referenced in the expression or on the Pages & Classes tab are not tracked and do not trigger recomputation.

Additionally, the following operations that can change a property value do not cause change tracking to occur:

- Saving a rule form that contains a property
- Retrieving a property with the Obj-List or Obj-Browse method
- Retrieving a property with the RDB-List method, unless the ApplyDeclaratives parameter of that method is selected

Note: Forward chaining operates only when the source properties are not marked as invalid. If a property has an associated message, forward chaining halts.

Note: Changes made to a property value within a Java step of an activity do cause backward changing computations to start, but do not cause forward chaining computation.

Field	Description
Target Property Data	Select to determine how often Pega Platform recomputes the value of the Target Property (identified in the second key part).
Calculate Value	Select an option to determine which events cause this rule to run: **Whenever inputs change** (Default). Forward-chaining recomputation. The target property is not recomputed when the property is used as a source, only when one of the expression inputs change. If the target property is not present on the page when needed or is present but has no value, the Declare Expression rule does not run. **When used if no value present** Compute when the value is null, blank, zero, or does not yet appear on the page using backward chaining. Assuming the computation results are not null,

Field	Description
	later requests for the target property do not cause the Declare Expression rule to run. **When used if property is missing** Recomputation using backward chaining when the target property is not present on the clipboard. **Whenever used** Just-in-time recomputation even when the property already has a value, using backward chaining. ⚠ **CAUTION:** This choice ensures that the target property value matches the computed value at the start of every activity step, by evaluating the rule upon every read access of the target property. Because this choice can be costly in terms of performance, a warning icon appears when you save the rule form. ⓘ **Note:** During backward chaining computations, if the Otherwise section of a decision tree or decision table, or the Default row in a map value referenced in a Declare Expression rule can be computed, but properties needed for other parts of the form are not defined, the Otherwise value is returned as the value of the rule. **When applied by a Rule Collection** The target property is computed only when the defined Declare Expression rule is invoked by a collection rule *(on page)*. The chained expressions also are evaluated if they too are set to "When applied by a Rule Collection." For example, if the defined rule is TotalCostAfterTax=TotalCost*(1+TaxRate) where TotalCost=ItemCost*Quantity , then this expression is run only when a collection references this rule. In this case, TotalCost is also calculated if the properties on it were set to "When applied by a Rule Collection." This option supports self-referential expressions, for example,

Field	Description
	MyProperty=.MyProperty+1 **Note:** This option is compatible with versions earlier than PRPC 7.1.6. **When invoked procedurally** The target property is computed only when an independent collection rule *(on page)* directly invokes the target property. If the defined Declare Expression rule is chained into another expression that is set to "When invoked procedurally," the chained expression will not run. For example, if the defined rule is TotalCostAfterTax=TotalCost*(1+TaxRate) where TotalCost=ItemCost*Quantity , then this expression is run only when a collection references this rule. In this case, TotalCost is not calculated if the properties on it were set to "When invoked procedurally." This option supports self-referential expressions, for example, **Note:** As a best practice, select `Whenever used` when the expression involves values from properties on multiple pages.
Additional Dependencies	Optional. Enter property references to identify additional properties (in the same page context) to be tracked. Order is not significant. If not blank, changes to any of the properties cause this rule to execute. This array appears only when you select `Whenever inputs change` for the Calculate Value field.
Context Execution Behavior	Select one of three settings to further define which situations cause this expression to be evaluated: **Only when the top-level page is of the Applies to class** To execute this rule only if the property is on a top-level page.

Field	Description
	Note: Do not select this option if the Applies To key part of this rule identifies a class derived from the `Embed-` base class; by definition, pages of such classes are never at the top-level. **When the top-level page is of the Applies To class, or one of the following** To restrict execution to contexts in which the top-level page matches the Applies To class, one of a specified list, or any descendent classes in the class hierarchy. **Regardless of any pages it is embedded in** To execute the rule regardless of the page context, making this rule completely "context-free". **Note:** If you select this option, you cannot use the `Top` or `Parent` keywords in the computations on the Expressions tab, either explicitly or in decision rules referenced on that tab. **CAUTION:** The third selection can produce high execution frequencies. Select the most restrictive value that meets your application requirements. **Note:** If this Pega Platform system was updated from Version 5.2 or earlier, execution of context-free expressions is disabled by default. Enable them through changes to the `prconfig.xml` file or Dynamic System Settings. See More about Declare Expression rules *(on page 214)*.
Execute this Expression	

Field	Description
Allowed top-level classes	This array appears only if you select the second choice (`When the top-level page..`.) for the Execute this Expression field. Enter a list of classes. Execution occurs if the top-level page has a class that exactly matches any class on this list, or is a descendent of one of the classes on this list. Do not list the Applies To key part here; that class is included automatically.

Declare Expression rules *(on page 209)*
Creating Declare Expression rules *(on page 210)*
Viewing rule history *(on page)*
More about Declare Expression rules *(on page 214)*

Supported and unsupported configurations in simplified declare expressions

As of Pega Platform 8.3, the **Change tracking** tab has been removed from the declare expression rule form to simplify expression configuration. In these simplified declare expressions, some **Target Property Data** and **Context Execution Behavior** configurations are not supported.

Note: To access the **Change tracking** tab, click **Actions > Use legacy expression**.

Supported configurations

In simplified declare expressions the target property's value is only calculated whenever used. Additionally, expressions are always executed regardless of the context of the pages they are contained in.

Unsupported configurations

The following **Target Property Data** field options are not supported for simplified declare expressions. For detailed information about these options, see Declare Expression form - Completing the Change Tracking tab *(on page 218)*.

- **Whenever inputs change**
- **When used, if no value present**
- **When used, if property is missing**
- **When applied by a Rule Collection**
- **When invoked procedurally**

The following **Context Execution Behavior** field options are not supported for simplified declare expressions. For detailed information about these options, see Declare Expression form - Completing the Change Tracking tab *(on page 218)*.

- **Only when the top-level page is of the Applies To class**
- **When the top-level page is of the Applies To class, or specified Allowed Top-Level Classes**

Page context changes

In simplified declare expressions, context-bound rules are not supported.

Non-atomic targets

Non-atomic targets are not supported in simplified declare expressions, for example, .page.targetProp.

Related information

Declare Expression rules *(on page 209)*

Declare Expression form - Completing the Change Tracking tab *(on page 218)*

Viewing the declarative network

Quickly gain insight into the relationships between declarative properties, which your system calculates automatically based on the value of other properties. By accessing the declarative network you can view and test declarative rules, which reduces processing errors and development effort.

1. In the header of Dev Studio, click **Configure > Case Management > Business Rules > Declarative Network**.
2. On the **Declarative Network Analysis** tab, click **application**, and then select the application whose declarative network you want to review.

   > The system displays the declarative rules that your application uses, divided by class.

3. Review the relationship between the declarative rules by performing any of the following actions:
   - To display the rule and related properties in a pop-up tree diagram, click the **Display this top-level declarative network** icon.
   - To open the declare expression for the rule in a new tab, click the **Display declare expression** icon.
   - To open the target property of the declarative rule in a new tab, click the property name.
   - To open the class to which a rule belongs in a new tab, click the class name.

Testing rules in the Declarative Network Viewer *(on page 225)*

Testing rules in the Declarative Network Viewer

Reduce the number of processing errors by testing calculated properties in the Declarative Network Viewer. By determining the value of a property automatically (declaratively), you reduce the number of manual calculations required and improve system efficiency. The viewer is a tool that displays the relationship between declarative properties based on changes to other property values. You can also use the viewer to test whether the properties calculate correctly, and set up tests for future use.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **Declare Expression**.
3. In the list, select the declare expression that you want to test.
4. In the declare expression, click **Actions > Run**.
5. In the **Test Page** section, select the context to use for the test:
    a. In the **Data Context** list, click the thread in which you want to run the rule.
    b. **Optional:** To discard all previous test results and start from a blank test page, click **Reset Page**.
6. In the section with the declare expression that you want to test, define the value of the properties that the system uses to calculate the declarative property:
    a. Click a property that you want to define, and then enter its value in the **Property** field.
    b. Click **Update**.
    c. Repeat steps 6.a *(on page 225)* through 6.b *(on page 225)* for all properties in the expression.

    > The value of the declare expression changes to reflect the updated properties. You can now see whether the declarative property calculates correctly.

7. **Optional:** To save the settings as a test scenario, click **Convert to test**, and then complete the **New** tab.
    For more information, see Creating unit test cases for rules *(on page)*.

Viewing the declarative network *(on page 224)*

Declare OnChange rules

Create Declare OnChange rules to run an activity automatically at activity step boundaries when the value of a specified property changes. This capability provides a form of automatic forward chaining.

The following tabs are available on this form:

- OnChange Properties *(on page 227)*

For example, a Declare OnChange rule can update a year-to-date counter property (an integer) to track how many times a work item status changed in value. Another Declare OnChange rule can compute the average dollar amount of work items entered by a team, in real time.

Declare OnChange rules can force all processing on a work item to be suspended pending an independent review of the situation. The independent review is supported by one or more flows. This feature can support compliance, fraud detection, and quality control staffs.

Where referenced

No other rules explicitly reference Declare OnChange rules. After you save a Declare OnChange rule, it is run immediately and as needed.

Access

Use the Application Explorer to access OnChange rules that apply to work types in your application. Use the Records Explorer to list all OnChange rules available to you.

Category

Declare OnChange rules are instances of the `Rule-Declare-OnChange` class. They are part of the Decision category.

Policy overrides and suspended work items *(on page)*
Viewing rule history *(on page)*

Creating Declare OnChange rules

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing rules by creating a copy in a specific ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a Declare OnChange rule by selecting `Declare OnChange` from the `Decision` category.

Key parts:

A Declare OnChange rule has two key parts:

Field	Description
Apply to	Select a class for this rule. At runtime, a clipboard page of this class must be a top-level page. The properties to be watched may be in this class (or in the class of an embedded page). You cannot use a Rule-Declare-* class or any ancestor of the Rule-Declare- class (including `@baseclass`) here. You cannot use a class derived from the `Code-` or `Embed-` class here.
Identifier	Enter a unique name for this OnChange rule within the class. Begin the name with a letter and use only letters, numbers, the ampersand character, and hyphens. No other rules explicitly reference this **Identifier** value. However, because of normal class inheritance, a Declare OnChange rule named OutofStock at one level in the class structure may override (and so prevent execution of) a Declare OnChange rule named OutofStock at a higher level in the class structure.

Rule resolution

Declare OnChange rules *(on page 225)*
Viewing rule history *(on page)*
More about Declare OnChange rules *(on page 231)*

Declare OnChange form - Completing the OnChange Properties tab

Define the conditions that will cause this rule to be executed. Specify in an array the properties to be watched for change.

Properties, Conditions, and Action

Field	Description
Properties to Watch	This array identifies some of the properties that the rule tracks. Order is not significant. See More about Declare OnChange rules *(on page 231)* for information about the properties tracked by the rule.
(1, 2, 3...)	Enter the desired property references. Each property entered is monitored for changes. Identify a property on the top-level page (with a dot followed by the page name) or on a page identified by the **Page Context** field on the Pages & Classes tab.

Field	Description
	• If you list more than one property, they must be on the same page, either the top-level page or a common embedded page. • If you list more than one property, when two or more of the properties change value (for example, within a single step of an activity) this `OnChange` activity runs only once. • Start each property reference with a dot. You can't reference a property on a page other than the page corresponding to the Applies To key part of the rule or on a page identified by the **Page Context** field on the Pages & Classes tab.
Conditions	

When	Optional. Identify a when condition rule to be evaluated at the time a property value changes.
	
	The system uses the Page Context value on the Pages & Classes tab (if not blank) as the Applies To key part of the when condition rule. If the Page Context value is blank, the system uses the Applies To key part of this OnChange rule.
	
	Alternatively, you can enter a simple Boolean expression here, for example in one of these formats:
	
	`property = "constant value"`
	
	`property1 > property2`
	
	You can reference linked properties in the expression. Don't use a complex Java expression or an expression that calls a function here.
	
	When this OnChange rule used to suspend work item processing, this when condition is known as the business exception.

Choose Action...	
Choose Action	Select: • `Suspend Work Object` — To cause all executing flows for a work item to be suspended. See Understanding policy overrides and suspended work items *(on page)*. • `Call Activity` — For all other OnChange processing.

When True Run and When False Run

Complete these fields when you select `Call Activity` in the Choose Action field.

Field	Description
When True Run	
Activity	Enter the Activity Name key part of an activity to execute when any of the specified properties change (and the test in the When field evaluates to true). Choose an activity with an Activity Type of `OnChange` (recorded on the Security tab). If the activity has input parameters, click **Params** to enter literal constant values for them. The system uses the Page Class value on the Pages & Classes tab (if not blank) as the Applies To key part of the activity. If the Page Context value is blank, the system uses the Applies To key part of this OnChange rule. See More about Declare OnChange rules *(on page 231)* for more about the activity.
When False Run	
When False Run	Optional. Enter the Activity Name key part of an activity to execute when any of the specified properties change and the test in the When field evaluates to false. Choose an activity with an Activity Type of `OnChange` (recorded on the Security tab). If the activity has input parameters, click **Params** to enter literal constant values for them. See More about Declare OnChange rules *(on page 231)* for more about the activity.

Suspend Section

Complete these fields when you select `Suspend Work Object` in the Choose Action field. See Understanding policy overrides and suspended work items *(on page)*.

Field	Description
Suspend Section	

Field	Description
Policy Override Flow to run	Enter the Flow Name key part of a flow to execute to support review of the suspended flow. Ensure that the flow meets the requirements listed in How to implement business exception processing with policy overrides *(on page 233)*.
Error message	Enter the third key part of a field value rule with first key part `@baseclass` and second key part `pyMessageLabel`, or enter literal text between double quotes to identify this source of policy override reviews.

Declare OnChange rules *(on page 225)*
Creating Declare OnChange rules *(on page 226)*
Viewing rule history *(on page)*
More about Declare OnChange rules *(on page 231)*

Declare OnChange form - Completing the Pages & Classes tab

Use this tab to list the pages referenced by name in the OnChange tab. Specify the pages in an array containing these fields. See How to Complete a Pages & Classes tab *(on page)* for basic instructions.

1. About *(on page 225)*
2. New *(on page 226)*
3. OnChange Properties *(on page 227)*;
4. Pages & Classes *(on page 230)*
5. History *(on page)*
6. More... *(on page 231)*

If the watched properties are an aggregate or are on an embedded page, complete the Page Context Data fields.

Field	Description
Pages and	

Field	Description
Classes	
Page Name	Optional. Name of a page referenced on the OnChange Properties tab. You can identify named pages here to support evaluation of a when condition rule referenced on the OnChange Properties tab. However, all tracked properties must be on the page corresponding to the Applies To key part, or a context within that page.
Class	Optional. Select the class of that page.
Page Context Data	
Page Context	Optional. Leave this blank if the monitored properties are `Single Value` properties that appear on a top-level page. Otherwise, identify a `Page List` or `Page Group` property reference on the page. You can supply literal index (subscript) values or property-reference index values. Omit any index to indicate that all values of the index are acceptable.
Page Class	Optional. Identify the class of the page or pages that are identified in the Page Context field. You cannot use the keywords `$ANY`, `$NONE` or `$CLASS` here.

About Declare OnChange rules *(on page 225)*

More about Declare OnChange rules

Pega Platform tracks properties listed in Declare OnChange rules for changed values. After each activity step, and after forward chaining for Declare Expression rules and constraints rules occurs, tracked changes start the activity specified on the OnChange Properties tab.

Properties tracked by the OnChange rule

This rule tracks the following properties:

- Properties identified in the Properties to Watch field
- Properties used in the when condition rule identified in the When field
- Properties used in the decision rules referred by the when condition rule

Restrictions for OnChange activities

The activity identified on the OnChange Properties tab must have an Activity Type of `OnChange`. It can call or branch to other activities, but only if they too have an Activity Type of `OnChange`.

Your activity can examine the `Value List` property `pyChangedProperties` to discover the property that changed value. This property is on the page named `pyDeclarativeContext`, of class `Code-Pega-DeclarativeContext`. This page exists only while the activity runs.

The primary page passed to OnChange activities is the top-level page.

OnChange processing may start as the result of a user HTTP Post operation. If users complete an HTTP form that changes the value of a property tracked by a Declare OnChange processing, this change is detected.

Use care not to start an infinite processing loop within declarative rules. For example, in an OnChange activity, do not update any of the properties that caused the activity to start. You can to update other properties in the OnChange activity.

- Declare Expression rules do not evaluate during the execution of an `OnChange` activity. However, do not attempt in an OnChange activity to overtly set the value of a property that is the target of a Declare Expression rule.
- Similarly, constraints rules do not evaluate during the execution of an OnChange activity.

Primary page

During execution of a Declare OnChange rule, the page on which the rule operates becomes the primary page. The page keyword `PRIMARY` and the results of the `tools.getPrimaryPage()` PublicAPI method reflect this change. When the Declare OnChange rule execution completes, the primary page of the calling activity resumes as primary.

Performance

Design the OnChange activity to execute quickly, as some properties may change values often. Remember that properties may change values (and thus cause the rule to run) during development and testing tasks, such as the preview of a harness or flow action form.

Declare OnChange rules may be expensive (computing several values) compared with Declare Trigger rules that test the same property. Consider which rule type better meets your needs. For example, if a value changes every few seconds but is part of an object that is saved only hourly on average, a Declare Trigger rule computes a new value only hourly, not in real time.

Testing and debugging Declare OnChange rules

Using the Tracer, you can watch the evaluation of a Declare OnChange rule. Start the Tracer and select a requestor session. Click **Settings** and check the Declare OnChange box in the Event Types to Trace section. Also check the RuleSet that contains the rule to be traced.

The statistic Tracked Property Changes on the full details page of the Performance tool shows how many property changes have occurred (for the current requestor since log-in) that are tracked for declarative rules computations.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Application debugging by using the Tracer tool *(on page 347)*
Declare OnChange rules *(on page 225)*
Creating Declare OnChange rules *(on page 226)*
Viewing rule history *(on page)*

How to implement business exception processing with policy overrides

Through OnChange rules and special flows, your application can create application-wide temporary pauses in flow processing — for example, to detect and review exceptions, suspect data or special cases — without altering the flows. This provides an alternative to use of tickets.

For a background on the benefits and purposes of this capability, see Understanding policy overrides and suspended work items *(on page)*.

Configuration overview

This facility requires three components:

- A property or properties present in a work item that can signify a business exception.
- An OnChange rule that responds to a business exception by promptly suspending all assignments on all currently executing flows for the work item.

- A flow that supports independent review of the work item, allowing the reviewer to record a final verdict.
- A reviewer — an operator holding the `Work-.ReviewPolicyOverrides` privilege — who assesses the business exception and records, in the work item history, a verdict allowing normal flow processing to resume or end.

1. Define the business exception as a when condition rule

1. Identify the property or properties that are involved in the test.
2. Create a when condition rule that is true only when the property or properties assume an unlikely, suspect, or potentially erroneous values. Choose a condition that is rare; that for most work items is false at all times from initial entry through resolution.

2. Create a policy override review flow

1. Create a flow rule to support review of work items that become suspended when the business exception occurred. On the New dialog box, select `PolicyOverride` in the Template field. Design the flow to meet these criteria:

- Applies to the class of the suspended work item (or a parent class).
- Does not create a new work item.
- Does not change the work item status.
- Calls, as the last shape before the End shape, the standard flow `Work-.FinishPolicyOverride`.

This flow may be simple — containing only the Start, End Flow, and Subprocess shapes — or it may require multiple assignments performed by multiple operators. Use the standard flow `Work-.PolicyOverride` as a starting point.

In the SubProcess Properties panel, complete four flow parameters for the `Work-.FinishPolicyOverride` flow, to control the email correspondence it produces. You can send correspondence when the review results in a Deny verdict, and when it results in an Allow verdict.

3. Create an OnChange rule

The OnChange rule detects the business exception and starts the review flow.

1. Select the work type of the work items as the Applies To class.
2. Complete the OnChange Properties tab. In the When field, reference the when condition rule created in step 1 above.
3. In the Choose Action field, select `Suspend Work Object`.

4. Identify the review flow in the Policy Override flow to run field.
5. Enter text such as "Too many payees" or the key to a message rule in the Error Message field, characterizing the reason for the suspension.

4. Adjust the routing

By default, the `Work-.FinishPolicyOverride` flow creates an assignment on the worklist of the operator administrator@org.com, where org.com is the organization owning the work item.

1. Override the router activity `Work-.ToPolicyOverrideOperator` to route this assignment to a person (or one of a group) who is appropriate to act as a reviewer.

2. Ensure that the Operator ID of this person acquires the privilege `Work-.ReviewPolicyOverrides`. The standard role `PegaRULES:SysAdm4` conveys this privilege, but you can create other roles appropriate to your application and organization.

Use the Tracer tool to test such flows.

5. Ensure that reviewers complete their assignments

Each person who acts as a reviewer must review and act on assignments which have the status `Pending-PolicyOverride`. To verdict, they can choose `Allow` or `Deny` as the flow action.

If they choose `Allow`, processing resumes and any suspended assignments reappear in worklists or work queues. If they choose `Deny`, the work item status changes to `Resolved-Revoked`. Affected users may be notified by email, if configured in the flow. In the unusual case that two or more reviews are underway for one suspended work item, the status changes occur only after all the reviews are complete.

6. Periodically monitor outstanding flow errors

Suspended work items remain suspended until all open reviews are completed.

To see a report of suspended work items, in the header of Dev Studio, click **Configure > Case Management > Tools > Work Admin > Suspended Work in Current Work Pool**.

To see a list of assignments in suspense, in the header of Dev Studio, click **Configure > Case Management > Tools > Work Admin > Suspended Work Assignments**.

Note:

- This capability illustrates how procedural rules — such as the flows — can operate with declarative rules — such as the OnChange rules — to directly support business processing. Pegasystems identifies this distinctive combination as the Pro-Dex capability.
- The **Show Policy Override** button appears at run time on flows (created or updated in PRPC Version 4.2 SP6 and later) for users who have the `@baseclass.FlowViewProDex` privilege. This button does not appear on flow actions for which the Auto- generated HTML box on the HTML tab is not selected.

Process category

Declare OnChange rules *(on page 225)*

Declare Trigger rules

To define correlations between events in your case types, create Declare Trigger rules to cause an activity to run when a specified event takes place in a case. By creating Declare Trigger rules, you automate your business processes and flexibly respond to dynamic business needs.

Declare Trigger rules are a part of the declarative rules that implement the declarative programming paradigm in Pega Platform.

For each Declare Trigger rule, Pega Platform monitors database operations for objects that you specify as the Applies To class, and concrete classes that are derived from that class, such as a class that stores your case type. When a change affects a specified property, Pega Platform uses the Change Tracking feature to monitor updates to the property values and run triggers accordingly.

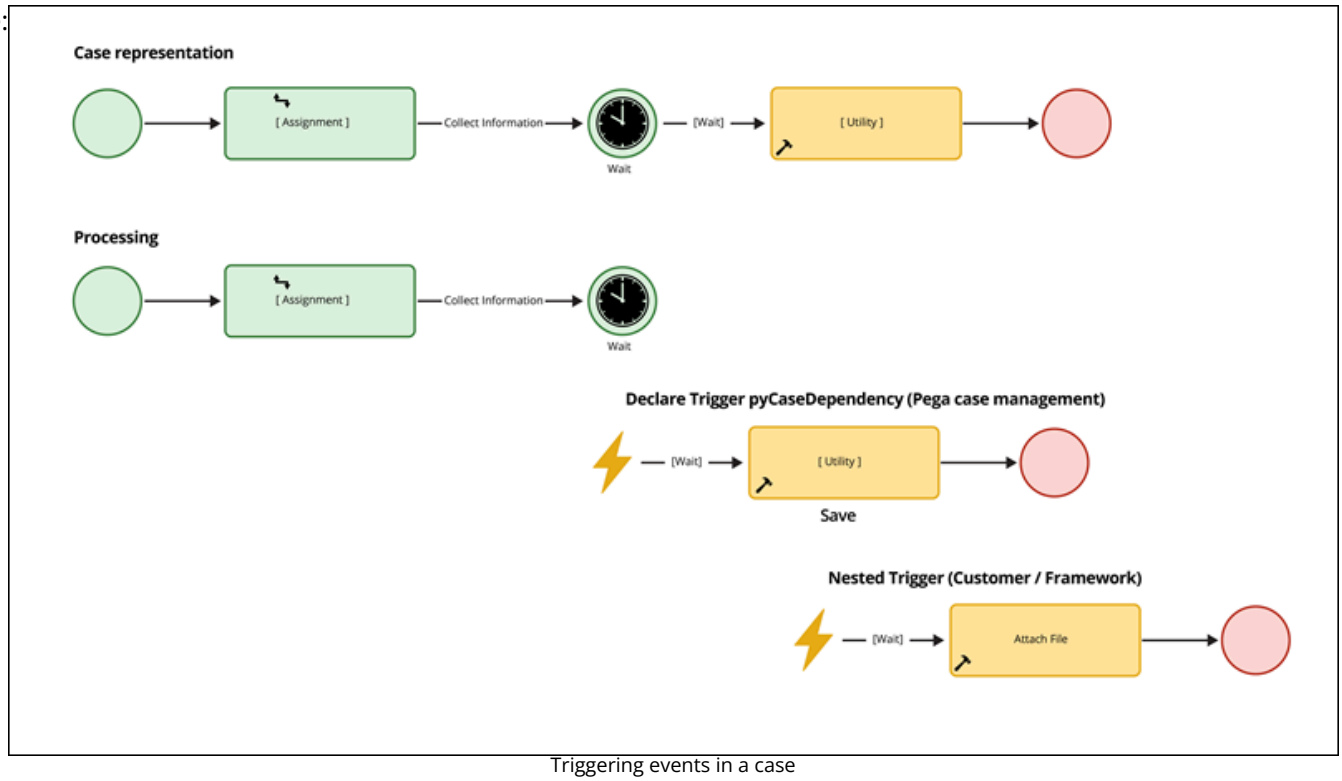
For example, a Declare Trigger rule can invoke an activity each time a user changes a postal code in their personal details. By changing their postal code, the user modifies a *Data-Party.pyPostalCode* property in an instance of the *Data-Party-Person* class. The activity that you specify in the Declare Trigger rule sends an email message to the customer service representative (CSR) whose territory includes the new address. Similarly, a Declare Trigger rule can implement a form of logging or audit history for a class by recording the date, time, and other facts. You can run Declare Trigger rules in the following scenarios:

- Page Save
  - In context
  - In background
- Page Delete
- Page Commit

Nested Declare Trigger rules

To resolve complex use cases, you can nest a Declare Trigger rule within another Declare Trigger rule. Nested Declare Trigger rules work in the context of the other Declare Trigger rule. For example, you can define a Declare Trigger rule that sends an email to a CSR when a customer updates their address details. Then you can create and nest another Declare Trigger rule that creates a document with the updated details after the case moves to the next stage. You can only nest rules of the *Save* and *Save and* types that run immediately. Nesting rules in the context of the *Commit* type rules might result in adding an excessive number of deferred operations during one database transaction. To avoid recursion during nesting of declare triggers, you can only nest a trigger in a trigger that is not already running on a page in the execution stack. For example, if a Page A save leads to a save of Page B and a trigger on Page B again leads to a save of Page A, the declare trigger does not run again on the Page A save.

The following figure presents how nested Declare Trigger rules start events in a case:



Triggering events in a case

To nest triggers, set the value of the *declaratives/nestedTriggersEnabled* dynamic system setting to true. For more information, see Creating a dynamic system setting *(on page)*.

Applying Declare Trigger rules

Immediately after you save the rule, your application runs a Declare Trigger rule according to your provided configuration.

Viewing rule history *(on page)*

Creating Declare Trigger rules

Define correlations between events in your case types by creating Declare Trigger rules. Declare Trigger rules run activities as a response to a specified event in a case. As a result, you provide flexible applications that precisely meet your business needs.For example, you can define a Declare Trigger rule that sends an email to a customer service representative (CSR) after a customer changes a postal address in a case.

Declare Trigger rules invoke activities when your application creates, updates, or deletes a class in the database. Operations on classes are related to actions in a case, such as saving a case or updating information in a case. You can also use Declare Trigger rules to track property changes, and for auditing purposes.

1. In the header of Dev Studio, click **Create > Decision > Declare Trigger**.
2. In the **Label** field, provide a descriptive name for the rule that you want to create.
3. In the **Context** section, specify where your application stores the rule:
    a. From the list of application layers, select an application to store the rule.
    b. In the **Apply to** field, enter the class to which the rule applies.
    At run time, the system monitors property values in this class and classes that inherit from this class.
    c. In the **Add to ruleset** list, select a ruleset and a ruleset version in which to store the rule.
4. Click **Create and open**.

    The rule form opens.

5. On the **Triggers** tab, in the **Trigger when an instance is** list, select a type of event to trigger the rule:

Choices	Actions
Run the rule when an application deletes an instance in the specified class	Select **Deleted**.
Run the rule when an application saves an instance of the specified class	Select **Saved**.
Run the rule when a Commit method occurs for a saved instance of the specified class	Select **Committed Save**.

Choices	Actions
Run the rule when a Commit method occurs for a deleted instance of the specified class	Select **Committed Delete**.
Run the rule when an application saves an instance of the specified class and specified property values in the class change	a. Select **Saved and**. b. In the **One of these properties was modified** section, in the **Property** field, enter a property that you want your application to track. c. **Optional:** To retain the initial value of the property that changes, in the **Copy Value To (optional)** field, enter a property that stores the initial value. Retaining the initial values is useful for auditing purposes and makes the previous property value available to the second and subsequent runs of the rule. Copying occurs only if the trigger activity runs, and after the trigger activity completes. d. To track changes to more properties, click **Add a row**, and then repeat steps 5.b *(on page 239)* and 5.c *(on page 239)*.

6. To run the rule only when a case meets specified conditions, in the **Condition** section, in the **When** field, enter a When condition rule.
   The system evaluates the conditions at run time and runs the Declare Trigger rule only when the When condition evaluates to true.
7. In the **Trigger activity** section, in the **Name** field, enter the activity that the rule invokes.
8. In the **Execute** list, determine how the activity runs:

- To run the activity during forward chaining, before the commit completes, select **Immediately**.
- To run the activity in a new requestor, outside of the context of forward chaining, select **In Background On Copy**.

> **Note:** Ensure that the activity that you select to run in the background includes the Commit method.

When you select an activity of a different type than *Trigger*, ensure that the preconditions and transitions in the activity do not use When condition rules and do not call functions. For more information about activities, see Creating an activity *(on page 100)*.

> **CAUTION:** Although your application can contain multiple *Trigger* activities for the one class that is specified in the **Apply to** field, you cannot control the order in which two or more triggered activities run. Create activities that provide correct results when running in any order, and when running either independently or simultaneously.

9. Click **Save**.

Declare Trigger rules *(on page 236)*
Viewing rule history *(on page)*
More about Declare Trigger rules *(on page 241)*

Declare Trigger form - Completing the Pages and Classes tab

Use this tab to list the clipboard pages referenced by name in the Triggers tab. See How to Complete a Pages & Classes tab *(on page)* for basic instructions.

Field	Description
Pages and Class-es	

Field	Description
Page Name	Optional. Enter the name of a page referenced on the Triggers tab. Optionally, add a row with the keyword `Top` as the page name, to identify a top-level page. The Top keyword allows you to use the syntax Top.propertyref to identify properties on other tabs of this rule form.
Class	Optional. Select the class of the page or pages that are referenced on the Triggers tab. You cannot use the keywords `$ANY`, `$NONE` or `$CLASS` here. The system completes the first row of this array, using the Applies To key part of the rule as the class, and leaving the Page Name field blank. Do not alter that row.
Page Context Data	

Page Context	Optional. Leave this blank if the properties watched by this rule (and recorded on the Triggers tab) are `Single Value` properties found on a top-level page.
	
	Otherwise, identify a `Page List` or `Page Group` property reference on the page. You can supply literal index values or property-reference index values. Omit any index to indicate that all values of the index are acceptable. These are acceptable:
	
	`Invoices.pyOrders(2).pyItems("Manuals").pyItemNames`
	
	`Invoices.pyOrders().pyItems().pyItemNames`

Class	Optional. Select the class of the page or pages that are identified in the Page Context field. You cannot use the keywords `$ANY`, `$NONE` or `$CLASS` here.

About Declare Trigger rules *(on page 236)*

Declare Trigger rules *(on page 236)*
Creating Declare Trigger rules *(on page 238)*
Viewing rule history *(on page)*
More about Declare Trigger rules *(on page 241)*

More about Declare Trigger rules

Create Declare Trigger rules to cause an activity to run when instances of a specific class are created, updated, or deleted in the database. This implements a form of forward chaining.

Trigger activities

The primary page passed to Trigger activities is the top-level page corresponding to the Applies To class of the rule.

An activity of type `Trigger` may alter properties, call functions and execute other rules, but do not perform database commits. Take care in declarative processing not to specify processing that produces infinite looping.

When you choose `In Background on Copy` in the Execute field, the triggered activity runs in a child requestor in parallel to the current requestor. This means that:

- The triggered activity does not appear in your Tracer session
- The triggered activity cannot access clipboard pages other than the primary page
- Within the triggered activity, the primary page has no name

Primary page

During execution of a Declare Trigger rule, the page on which the rule operates temporarily becomes the primary page. The page keyword `PRIMARY` and the results of the `tools.getPrimaryPage()` PublicAPI method reflect this change.

When the Declare Trigger rule execution completes, the primary page of the calling activity resumes as primary.

Testing and debugging

Using the Tracer, you can watch the evaluation of a Declare Trigger rule if the Execute field value is `Immediate`:

1. Start the Tracer and select a requestor session.
2. Click **Settings** and check the Declare Trigger box in the Event Types to Trace section.
3. Select the RuleSet that contains the rule to be traced.

History change auditing

Declare Trigger rules can automatically update the history of a work item, rule, or data object when certain properties change.

For work item change tracking, use the Field Auditing gadget, on the Work History landing page.

For data or rule tracking, see the Pega Community article *How to audit field-level changes to security rule and data instances*.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Application debugging by using the Tracer tool *(on page 347)*
Declare Trigger rules *(on page 236)*
Creating Declare Trigger rules *(on page 238)*
Viewing rule history *(on page)*

Declare Trigger rules

To define correlations between events in your case types, create Declare Trigger rules to cause an activity to run when a specified event takes place in a case. By creating Declare Trigger rules, you automate your business processes and flexibly respond to dynamic business needs.

Declare Trigger rules are a part of the declarative rules that implement the declarative programming paradigm in Pega Platform.

For each Declare Trigger rule, Pega Platform monitors database operations for objects that you specify as the Applies To class, and concrete classes that are derived from that class, such as a class that stores your case type. When a change affects a specified property, Pega Platform uses the Change Tracking feature to monitor updates to the property values and run triggers accordingly.

For example, a Declare Trigger rule can invoke an activity each time a user changes a postal code in their personal details. By changing their postal code, the user modifies a *Data-Party.pyPostalCode* property in an instance of the *Data-Party-Person* class. The activity that you specify in the Declare Trigger rule sends an email message to the customer service representative (CSR) whose territory includes the new address. Similarly, a Declare Trigger rule can implement a form of logging or audit history for a class by recording the date, time, and other facts. You can run Declare Trigger rules in the following scenarios:

- Page Save
    - In context
    - In background
- Page Delete
- Page Commit

Nested Declare Trigger rules

To resolve complex use cases, you can nest a Declare Trigger rule within another Declare Trigger rule. Nested Declare Trigger rules work in the context of the other Declare Trigger rule. For example, you can define a Declare Trigger rule that sends an email to a CSR when a customer updates their address details. Then you can create and nest another Declare Trigger rule that creates a document with the updated

details after the case moves to the next stage. You can only nest rules of the *Save* and *Save and* types that run immediately. Nesting rules in the context of the *Commit* type rules might result in adding an excessive number of deferred operations during one database transaction. To avoid recursion during nesting of declare triggers, you can only nest a trigger in a trigger that is not already running on a page in the execution stack. For example, if a Page A save leads to a save of Page B and a trigger on Page B again leads to a save of Page A, the declare trigger does not run again on the Page A save.

To nest triggers, set the value of the *declaratives/nestedTriggersEnabled* dynamic system setting to true. For more information, see Creating a dynamic system setting *(on page)*.

Applying Declare Trigger rules

Immediately after you save the rule, your application runs a Declare Trigger rule according to your provided configuration.

Viewing rule history *(on page)*

Creating Declare Trigger rules

Define correlations between events in your case types by creating Declare Trigger rules. Declare Trigger rules run activities as a response to a specified event in a case. As a result, you provide flexible applications that precisely meet your business needs.For example, you can define a Declare Trigger rule that sends an email to a customer service representative (CSR) after a customer changes a postal address in a case.

Declare Trigger rules invoke activities when your application creates, updates, or deletes a class in the database. Operations on classes are related to actions in a case, such as saving a case or updating information in a case. You can also use Declare Trigger rules to track property changes, and for auditing purposes.

1. In the header of Dev Studio, click **Create > Decision > Declare Trigger**.
2. In the **Label** field, provide a descriptive name for the rule that you want to create.
3. In the **Context** section, specify where your application stores the rule:
    a. From the list of application layers, select an application to store the rule.
    b. In the **Apply to** field, enter the class to which the rule applies.
    At run time, the system monitors property values in this class and classes that inherit from this class.
    c. In the **Add to ruleset** list, select a ruleset and a ruleset version in which to store the rule.
4. Click **Create and open**.

The rule form opens.

5. On the **Triggers** tab, in the **Trigger when an instance is** list, select a type of event to trigger the rule:

Choices	Actions
Run the rule when an application deletes an instance in the specified class	Select **Deleted**.
Run the rule when an application saves an instance of the specified class	Select **Saved**.
Run the rule when a Commit method occurs for a saved instance of the specified class	Select **Committed Save**.
Run the rule when a Commit method occurs for a deleted instance of the specified class	Select **Committed Delete**.
Run the rule when an application saves an instance of the specified class and specified property values in the class change	a. Select **Saved and**. b. In the **One of these properties was modified** section, in the **Property** field, enter a property that you want your application to track. c. **Optional:** To retain the initial value of the property that changes, in the **Copy Value To (optional)** field, enter a property that stores the initial value. Retaining the initial values is useful for auditing purposes and makes the previous property value available to the second and subsequent runs of the rule. Copying occurs only if the trigger activity runs, and after the trigger activity completes.

Choices	Actions
	d. To track changes to more properties, click **Add a row**, and then repeat steps 5.b *(on page 245)* and 5.c *(on page 245)*.

6. To run the rule only when a case meets specified conditions, in the **Condition** section, in the **When** field, enter a When condition rule.
   The system evaluates the conditions at run time and runs the Declare Trigger rule only when the When condition evaluates to true.
7. In the **Trigger activity** section, in the **Name** field, enter the activity that the rule invokes.
8. In the **Execute** list, determine how the activity runs:
   - To run the activity during forward chaining, before the commit completes, select **Immediately**.
   - To run the activity in a new requestor, outside of the context of forward chaining, select **In Background On Copy**.

   > **Note:** Ensure that the activity that you select to run in the background includes the Commit method.

   When you select an activity of a different type than *Trigger*, ensure that the preconditions and transitions in the activity do not use When condition rules and do not call functions. For more information about activities, see Creating an activity *(on page 100)*.

   > **CAUTION:** Although your application can contain multiple *Trigger* activities for the one class that is specified in the **Apply to** field, you cannot control the order in which two or more triggered activities run. Create activities that provide correct results when running in any order, and when running either independently or simultaneously.

9. Click **Save**.

Declare Trigger rules *(on page 236)*
Viewing rule history *(on page)*
More about Declare Trigger rules *(on page 241)*

Map Values

Use a map value to create a table of number, text, or date ranges that converts one or two input values, such as latitude and longitude numbers, into a calculated result value, such as a city name. Map value rules greatly simplify decisions based on ranges of one or two inputs. Use a map value to record decisions based on one or two ranges of an input value. A map value uses a one- or two-dimensional table to derive a result.

Through cascading — where one map value calls another — map values can provide an output value based on three, four, or more inputs.

Complete the Configuration tab before the Matrix tab.

Where referenced

Rules of five types can reference map values:

- In a flow, you can reference a map value in a decision task, identified by the Decision shape in a flow.
- In an activity, you can evaluate a map value using the Property-Map-Value method or Property-Map-ValuePair method.
- A Declare Expression rule can call a map value.
- A map value can call another map value.
- A collection rule can call a map value.

Access

Use the Application Explorer to access the map values that apply to work types in your application. Use the Records Explorer to list all the map values available to you.

After you complete initial development and testing, you can delegate selected rules to line managers or other non-developers. Consider which business changes might require rule updates and if delegation to a user or group of users is appropriate. For more details, see Delegating a rule or data type *(on page)*.

Category

Map values are part of the Decision category. A map value is an instance of the `Rule-Obj-MapValue` rule type.

Map Values

Create a map value by selecting `Map Value` from the `Decision` category.

Key parts:

A map value has two key parts:

Field	Description
Apply to	Select a class that this map value applies to. The list of available class names depends on the ruleset that you select. Each class can restrict applying rules to an explicit set of rulesets as specified on the **Advanced** tab of the class form. Map value rules can apply to an embedded page. On the Map Value form, you can use the keywords `Top` and `Parent` in property references to navigate to pages above and outside the embedded page. If you use these keywords, include the class and absolute name — or a symbolic name using `Top` or `Parent` — on the Pages & Classes tab.
Identifier	Enter a name that is a valid Java identifier. Begin the name with a letter and use only letters, numbers, and hyphens. See How to enter a Java identifier.

Rule resolution

When searching for instances of this rule type, the system uses full rule resolution which:

- Filters candidate rules based on a requestor's ruleset list of rulesets and versions
- Searches through ancestor classes in the class hierarchy for candidates when no matching rule is found in the starting class
- Finds circumstance-qualified rules that override base rules
- Finds time-qualified rules that override base rules

Map Values *(on page 247)*

Completing the Matrix tab

Map Value form - Completing the Matrix tab

This tab contains a table of one column (for a one-dimensional map value) or two or more columns (for a two-dimensional map value). The order of rows and columns is important. Rows are evaluated from left to right, and columns from top to bottom.

Complete the Configuration tab before updating the Matrix tab. Labels that you enter on the Configuration tab appear on the Matrix tab to guide your input.

To limit possible results to values in a fixed list of constant values, complete the Configuration tab before the Matrix tab.

Adding rows and columns

You can add new rows and columns by clicking Configure rows and Configure columns , respectively. You can also perform these actions from the Configuration tab.

Assessing completeness and consistency

Optionally, you can use these buttons to determine whether the map value is complete and consistent (based on a static evaluation).

Button	Results
Show Conflicts	Mark with a warning icon any cells of the matrix that are unreachable. For example, if two rows are identical, the second row can never evaluate to true and so cannot affect the outcome of the rule. Click the warning icon on a row to highlight with an orange background the other cells that cause a cell to be unreachable. The selected row is highlighted with a yellow background. A map value that contains no such unreachable rows is called consistent. Conflicts are also checked when you save the form, and when you use the Guardrails landing page to run the guardrails check for the application. Conflicts do not prevent the rule from validating or executing, but may indicate that the rule does not implement the intended decision.
Import	After you export a map value, you can make changes in the .xlxs file and import the updated file. The map value rule form is updated with the changes you made. You must import the same file that you exported. You can change the name of the exported file and import the renamed file. However, you cannot import a file different from the one you exported. The **Default** row and first rows are locked in the exported file. You cannot delete these rows, and you cannot insert rows when you select these rows. The **Default** column is locked in the exported file. You cannot delete this column, and you cannot insert columns when you select this column.
Export	Exports the map value in .xlxs format. After you make your changes and save this file, you can import it with your changes.

Button	Results
Show Completeness	Automatically add suggested rows that cover additional cases and reduce or eliminate the situations that fall through to the Default row . Suggested additions appear with a light green background. They are only suggestions; you can alter or eliminate them.

Entering row and column header

Each row and column has a header that defines both a label and a comparison. To create, review or update a row or column header:

1. Click **Configure rows** or **Configure columns**. A pop-up window appears.
2. Select a comparison operator: `<` `>` `=`, `>=`, `<=` or `is missing`. (If you omit an operator, the system assumes =.) Select `is missing` to detect that a property is not present — not that it is present but has the null value.
3. Enter an expression. For each expression, you can enter a literal value, a property reference or a more complex expression, including function calls.

The keyword `Default` always evaluates to true and appears as the final choice at the end of each row and column. You can complete values for the `Default` row or leave them blank.

Completing a cell

If you completed a list of literal constant values on the Configuration tab, select one of those values for each cell.

Otherwise, enter an expression in the cell — a constant, a property reference, a function call, or other expression. For guidance while entering expressions, click the **Expression Builder** to start the Expression Builder. (You can enter complex expressions and use the Expression Builder only if the Allowed to Build Expressions? check box is selected on the Configuration tab.)

If a cell is blank but is selected by the runtime evaluation, the system returns the null value as the value of the map value.

Cascading map values with Call

One map value cell can reference another map value as the source of its value. Type the word call followed by the name (the second key part) of another map value with the same first key part. SmartPrompt is available. Click the **Open** icon to open the other map value.

If, at run time, this map value executes in a backward-chaining mode (that is, the AllowMissingProperties parameter of the Property-Map-Value method is `True`), the called map value also executes in this mode.

Map Values *(on page 247)*
Building expressions with the Expression Builder *(on page)*

Completing the Configuration tab

Complete the fields on this tab to guide your inputs on the Matrix tab and define the possible values returned by this map value.

Security

The following options impact the initial presentation and available options on the Matrix tab.

For example, you can prevent users from accessing the Expression Builder or modifying the column layout of the map value. This helps you customize the development experience for delegated users, such as line managers, who may not require access to the full set of decision table options.

> **Note:** All users with a rule-editing privilege, including delegated users, can remove these restrictions.

Field	Description
Allow updating of the matrix configuration in delegated rules	Select this check box to allow users to modify the rows and columns of the Matrix tab. Clear this check box to prevent users from updating row or column configuration. Users with rule-editing privileges can still change values within the cells of the Matrix tab.
Allow use of the expression builder on the matrix view	Select this check box to allow access to the Expression Builder from any cell on the Matrix tab. Clear this check box to hide the Expression Builder icon. Users with rule-editing privileges can still add constants or property references in a row or column cell.

Input Rows

See Configuring rows and columns in a map value *(on page 258)*.

Input Columns

See Configuring rows and columns in a map value *(on page 258)*.

Results

Use the options in this section of the tab to define the possible values that this map value can return. You can also specify a list of preset properties that are calculated before the map value runs.

To define allowed results:

1. Enter a property or linked property name in the **Results defined by property** field.

   This property must use table validation *(on page)* because the table values are used to populate the **Result** field.

2. Select a value from the **Result** list.

   Alternatively, you can enter a string value without quotes to supplement the existing table values.

3. Define a list of **Target Property** and **Value** pairs that are set when the map value returns the corresponding **Result**.

   You can enter a constant, property name, or expression in the **Value** fields.

4. Repeat steps 2 and 3 as necessary.

At run time, the system sets target properties using the order you specify.

To define preset properties:

1. Enter a property name in the **Property** field.
2. Enter a constant, property name, expression, or input parameter in the **Value** field.
3. Click the **add icon** and repeat this process for as many properties as are required.

These properties are set before the rows and columns on the Matrix tab are processed.

About Map Values *(on page 247)*

Completing the Pages & Classes tab

Identify what is known about the class of each page that is referenced on other tabs. See How to Complete a Pages & Classes tab *(on page)* for basic instructions.

Field	Description
Page Name	Optional. Enter the name of a clipboard page referenced on the Matrix or Configuration tab. Optionally, add a row with the keyword `Top` as the page name, to identify a top-level page. The Top keyword allows you to use the syntax Top.propertyref on other tabs of this rule form to identify properties. Map value rules can apply to embedded pages that appear within top-level pages with various names. In such cases, you can use the keywords `Top` or `Parent` in the page name here.
Class	Optional. Select the class of that page.

About Map Values *(on page 247)*

More about Map Values

Map value rules can be updated as needed to reflect changing business conditions, without the need to ask a skilled developer to modify complex activities or other rules.

Uploading an Excel spreadsheet to start

If you have in advance an Excel spreadsheet in XLS file format that contains useful starting information for a map value, you can incorporate (or "harvest") the XLS file and the information it contains directly into the new rule.

Evaluating

Both rows and columns contain a Type field (set on the Headers tab). The system makes comparisons according to the data type you recorded on the Headers tab, converting both the input and the conditions to the specified data type.

At runtime, the system evaluates row conditions first from top to bottom, until one is found to be true. It then evaluates column conditions for that row, left to right, until one is found to be true. It returns the value computed from that matrix cell.

Where map values are used

You can reference map values in the following places:

- In activities that use the Property-Map-Value method or the Property-Map-ValuePair method. These methods evaluate a one-dimensional or two-dimensional map value, compute the result, and store the result as a value for a property.
- In other map values, through a `Call` keyword in a cell.

- Through a standard function `ObtainValuePair()` in the `Pega-RULES:Map` library.
- On the Rules tab of a collection.

Input parameters in called map values rules

If a map value is evaluated through a decision shape on a flow, or one of the two methods noted above, the input value or values may be literal constants or may be property references, recorded in the flow or in the method parameters.

However, if a map value is evaluated by a Call from a cell in another map value, the evaluation always uses the Input Property on the Header tab. Nothing in the Call can override this source.

Special processing with Declare Expression calls

When a Declare Expression rule has `Result of map value` for the Set Property To field, special processing occurs at runtime when a property referenced in the decision table is not present on the clipboard. Ordinarily such decision rules fail with an error message; in this case the Default value is returned instead. For details, see the Pega Community article *Troubleshooting: declarative expression does not execute when a decision rule provides no return value.*

Performance

The Pega Platform does not limit the number of nodes in a map value. However, as a best practice to avoid slow performance when updating the form and also avoid the Java 64KB code maximum, limit your map value rules to no more than 300 to 500 rows.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Parent class

Through directed inheritance, the immediate parent of the `Rule-Obj-MapValue` class is the `Rule-Declare-` class. However, despite the class structure, this rule type does not produce forward or backward chaining. Technically, it is not a declarative rule type.

Standard rules

The Pega Platform includes a few standard map values that you copy and modify. Use the Records Explorer to list all the map values available to you.

Applies To	Map Name	Purpose
`Data-`	`Set-Corr-Pref-er-ence`	Selects a correspondence type based on an input value. For example, if the input value is "Home Address", the correspondence type result is `Mail`. Allows outgoing correspondence to be sent based to on available addresses, for a `Data-Party` object.
`Work`	`Offi-cers`	Can associate an Operator ID or email addressee with officers based on the titles CEO, CFO, COO, and VP. (Cells are blank in the standard rule.)

Decision trees *(on page 186)*
Decision tables *(on page 174)*

More about Map Values

Map value rules can be updated as needed to reflect changing business conditions, without the need to ask a skilled developer to modify complex activities or other rules.

Uploading an Excel spreadsheet to start

If you have in advance an Excel spreadsheet in XLS file format that contains useful starting information for a map value, you can incorporate (or "harvest") the XLS file and the information it contains directly into the new rule.

Evaluating

Both rows and columns contain a Type field (set on the Headers tab). The system makes comparisons according to the data type you recorded on the Headers tab, converting both the input and the conditions to the specified data type.

At runtime, the system evaluates row conditions first from top to bottom, until one is found to be true. It then evaluates column conditions for that row, left to right, until one is found to be true. It returns the value computed from that matrix cell.

Where map values are used

You can reference map values in the following places:

- In activities that use the Property-Map-Value method or the Property-Map-ValuePair method. These methods evaluate a one-dimensional or two-dimensional map value, compute the result, and store the result as a value for a property.
- In other map values, through a `Call` keyword in a cell.
- Through a standard function `ObtainValuePair()` in the `Pega-RULES:Map` library.
- On the Rules tab of a collection.

Input parameters in called map values rules

If a map value is evaluated through a decision shape on a flow, or one of the two methods noted above, the input value or values may be literal constants or may be property references, recorded in the flow or in the method parameters.

However, if a map value is evaluated by a Call from a cell in another map value, the evaluation always uses the Input Property on the Header tab. Nothing in the Call can override this source.

Special processing with Declare Expression calls

When a Declare Expression rule has `Result of map value` for the Set Property To field, special processing occurs at runtime when a property referenced in the decision table is not present on the clipboard. Ordinarily such decision rules fail with an error message; in this case the Default value is returned instead. For details, see the Pega Community article *Troubleshooting: declarative expression does not execute when a decision rule provides no return value.*

Performance

The Pega Platform does not limit the number of nodes in a map value. However, as a best practice to avoid slow performance when updating the form and also avoid the Java 64KB code maximum, limit your map value rules to no more than 300 to 500 rows.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Parent class

Through directed inheritance, the immediate parent of the `Rule-Obj-MapValue` class is the `Rule-Declare-` class. However, despite the class structure, this rule type does not produce forward or backward chaining. Technically, it is not a declarative rule type.

Standard rules

The Pega Platform includes a few standard map values that you copy and modify. Use the Records Explorer to list all the map values available to you.

Applies To	Map Name	Purpose
`Data-`	`Set-Corr-Pref-er-ence`	Selects a correspondence type based on an input value. For example, if the input value is "Home Address", the correspondence type result is `Mail`. Allows outgoing correspondence to be sent based to on available addresses, for a `Data-Party` object.
`Work`	`Offi-cers`	Can associate an Operator ID or email addressee with officers based on the titles CEO, CFO, COO, and VP. (Cells are blank in the standard rule.)

Map Values *(on page 247)*

Unit testing a map value

You can test a map value individually, before testing it in the context of the application that you are developing. Additionally, you can convert the test run to a Pega unit test case. Testing a map value involves specifying a test page for the rule to use, providing sample values for required parameters, running the rule, and then examining the test results.

1. In the navigation pane of Dev Studio, click **Records > Decision > Map Value**, and then click the map value that you want to test.
2. Click **Actions > Run**.
3. In the **Test Page** pane, select the context and test page to use for the test:
    a. In the **Data Context** list, click the thread in which you want to run the rule. If a test page exists for the thread, then it is listed and is used for creating the test page.
    b. To discard all previous test results and start from a blank test page, click **Reset Page**.
    c. To apply a data transform to the values on the test page, click the data transform link, and then select the data transform you want to use.
4. Enter sample values to use for required parameters in the **Results** pane and then click **Run Again**.

    > The value that you enter and the result that is returned are the values that are used for the default decision result assertion that is generated when you convert this test to a test case.

5. **Optional:** To view the pages that are generated by the unit test, click **Show Clipboard**.
6. To convert the test into a Pega unit test case, click **Convert to Test**. For more information, see Creating unit test cases for rules *(on page)*.
7. **Optional:** To view the row that produced the test result, click a **Result Decision Paths** link.

Map Values *(on page 247)*
Unit testing individual rules *(on page 345)*
Using the Clipboard tool *(on page 374)*
Opening a unit test case *(on page)*
Application debugging by using the Tracer tool *(on page 347)*

Configuring rows and columns in a map value

Complete the fields in the Input Rows and Input Columns sections of the Configuration tab to guides your inputs on the Matrix tab of a map value.

Evaluation of a map value can be based on the value of properties (specified here as the Row Property and Column Property), or on the value of parameters specified in a method.

If you leave the Property fields blank, the method must specify parameter values that match or are converted to the Data Type values on this tab.

When the Property fields are not blank but the activity step used to evaluate the rule specifies a parameter, the parameter value in the activity step is used, not the property value.

Input Rows

Field	Description
Row Parameter	
Data Type	Select `String`, `integer`, `double`, `Boolean`, `Date`, or `DateTime` to control how the system makes comparisons when a row parameter is supplied. It uses the Java `compareTo( )` method when comparing two dates or two strings. For example, if the method parameter is "007" and the Data Type is `String`, then a comparison of "007" < "7" is true. If the method parameter is "007" and the Data Type is `Number`, then the comparison of "007" < "7" is false. For Booleans, only the "=" comparison is available. The Data Type field is ignored (and becomes display-only on the form) when the Row Property property is the source of a value for the map value. Comparisons in that case depend on the type of that property.

Field	Description
Row Property	
Property	Optional. If this map value is to obtain the row input value from a property, select or enter a property reference or linked property reference. If you leave this blank, the calling method must supply a parameter value for the row. For a map value that is "called" by another map value, this field is required.
Label	Enter brief text that becomes a row name on the Matrix tab.

Input Columns

Select `none` as the Column Parameter Data Type when defining a one-dimensional map value.

Complete these optional fields to define a two-dimensional map value, which can be evaluated by the Property-Map-ValuePair method.

Field	Description
Column Parameter	
Data Type	Select `String`, `integer`, `double`, `Boolean`, `Date`, or `DateTime` to define a two-dimensional map value and to control how the system makes comparisons when a column parameter is supplied. It uses the Java `compareTo()` method when comparing two dates or two strings. To create a one-dimensional map value, select `none`. For Booleans, only the "=" comparison is available. The Data Type field is ignored (and becomes display-only on the form) when the Column Property property is the source of a value for the map value. Comparisons in that case depend on the type of that property.
Column Property	

Field	Description
Property	Optional. If this map value is to obtain a column input value from a property, select or enter a property reference or linked property reference. If you leave this blank but use a two-dimensional matrix, the calling method must supply a parameter value for the column. For a two-dimensional map value that is called by another map value, this field is required.
Label	Enter brief text that becomes a column name on the Matrix tab.

About Map Values *(on page 247)*

Unit testing a map value

You can test a map value individually, before testing it in the context of the application that you are developing. Additionally, you can convert the test run to a Pega unit test case. Testing a map value involves specifying a test page for the rule to use, providing sample values for required parameters, running the rule, and then examining the test results.

1. In the navigation pane of Dev Studio, click **Records > Decision > Map Value**, and then click the map value that you want to test.
2. Click **Actions > Run**.
3. In the **Test Page** pane, select the context and test page to use for the test:
    a. In the **Data Context** list, click the thread in which you want to run the rule. If a test page exists for the thread, then it is listed and is used for creating the test page.
    b. To discard all previous test results and start from a blank test page, click **Reset Page**.
    c. To apply a data transform to the values on the test page, click the data transform link, and then select the data transform you want to use.
4. Enter sample values to use for required parameters in the **Results** pane and then click **Run Again**.

    > The value that you enter and the result that is returned are the values that are used for the default decision result assertion that is generated when you convert this test to a test case.

5. **Optional:** To view the pages that are generated by the unit test, click **Show Clipboard**.
6. To convert the test into a Pega unit test case, click **Convert to Test**. For more information, see Creating unit test cases for rules *(on page)*.
7. **Optional:** To view the row that produced the test result, click a **Result Decision Paths** link.

Map Values *(on page 247)*
Unit testing individual rules *(on page 345)*
Using the Clipboard tool *(on page 374)*

Opening a unit test case *(on page)*
Application debugging by using the Tracer tool *(on page 347)*

When condition rules

A When condition rule evaluates a Boolean logic statement that involves comparisons among field values, to return true or false. As a result, you deliver flexible software that adjusts to changing business circumstances.

For example, a When condition rule returns true if a job applicant has more than ten years of experience, and if the applicant speaks French fluently. As a result, an application performs a specific action, such as starting an optional process in a case.

You can apply When condition rules in multiple places in your application. To create more sophisticated logic, you can create multiple conditions, and then apply grouping that meets your business requirements.

Elements of When condition rules

When condition rules are Boolean expressions that consist of three elements:

- An input value to compare at run time, for example, the value of a field that a user completes.
- A comparator, for example `is greater than` or `is equal to`.
- A value to compare against the input value from the condition. This value can be a constant that you enter or a reference to another field.

Input for When condition rules

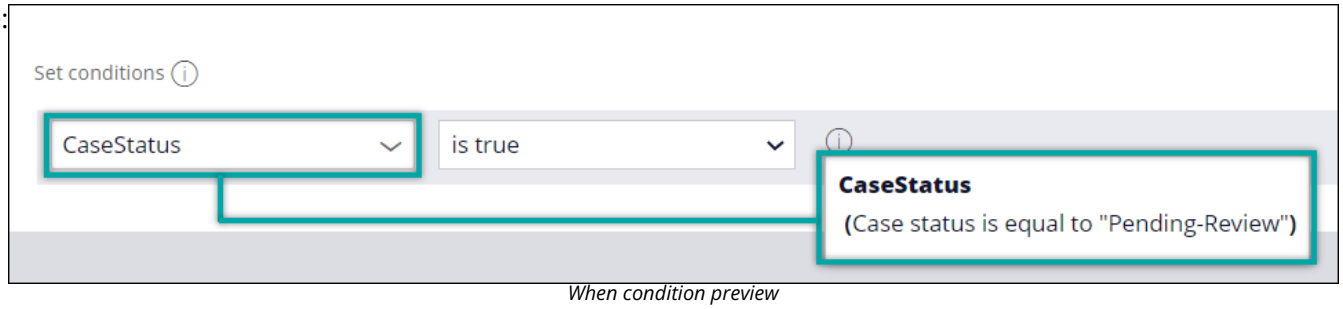
As an input value for a When condition rule, you can use any of the following information:

- A field value
- A configuration
- Another When condition

As you build a When condition by applying another When condition, you define whether the input When condition needs to evaluate to true or false at run time instead of selecting a comparator and a test value.

For example, you can create a When condition rule `CaseStatus is true`, where `CaseStatus` is another When condition defined as `CaseStatus is equal to "Pending-Review"`. In this scenario, `CaseStatus is true` evaluates to true if the case status at run time is `Pending-Review`. For more intuitive development experience, you can easily

preview the When condition that you apply as an input value, as shown in the following figure:



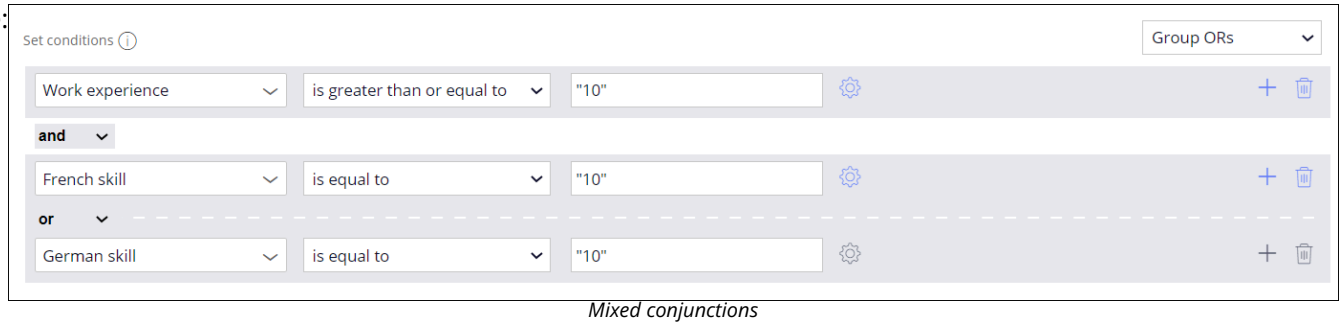
When condition preview

Grouping conditions

You can group conditions by applying AND and OR conjunctions. When you apply AND conjunctions, all grouped conditions need to evaluate to true for a When condition rule to return true. For example, in an application to review job candidates, if you create the conditions `French skill is equal to "10"` and `Work experience is greater than or equal to "10"`, and you use an AND conjunction, both conditions need to evaluate to true for the When condition to return true. If only one condition evaluates to true, for example, if a candidate has more than ten years of work experience, but the French skill is below 10, overall the When condition evaluates to false.

If you group conditions by using an OR conjunction, any grouped condition needs to evaluate to true for the When condition rule to return true. For example, if you create the conditions `German skill is equal to "10"` and `French skill is equal to "10"` and you use an OR conjunction, the When condition evaluates to true if a candidate fluently speaks German or French.

You can mix comparators to meet your business requirements. For example, you can create a When condition rule that determines that a job candidate needs to have at least ten years of work experience and needs to be fluent in either French or German, as shown in the following figure:



Mixed conjunctions

To resolve complex business scenarios, you can use advanced logic to group conditions by entering a logic string. When you apply advanced logic, every condition has an integer assigned that you use in the logic string. In logic strings, you can subgroup conditions and apply nested levels of grouping. A sample logic string is `1 AND (2 OR (3 AND 4))`. In advanced logic, you can also use the `NOT` conjunction.

Referencing When condition rules

Rules of many other rule types can reference When condition rules. For example, you can reference When condition rules in activities, processes, and through the `<pega:when>` JSP tag in HTML and XML rules in your applications based on Theme UI-Kit.

For example, you can use a When condition rule to define when an optional process starts in a case. For more information, see Conditionally starting a process *(on page)*.

Delegation

After you complete initial development and testing, you can delegate selected When condition rules to line managers or other non-developers. Consider which business changes might require rule updates and if rule delegation to a user or group of users is appropriate.

For more information, see Delegating a rule or data type *(on page)*.

Conditions in activity steps

Each activity step can reference one or several When condition rules, as preconditions for a method, or transitions between the completed method and the next step.

As a precondition, a When condition rule determines whether the method in the activity step runs. Referenced in a transition, a When condition determines which activity processing runs after the completion of the current activity step.

For more information, see Creating an activity *(on page 100)* and Configuring steps in an activity *(on page 101)*.

Standard function

To run a When condition rule in an expression or in the context of Java code, call the standard function *callwhen()* in the Pega-RULES *Default* library.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

Category

When condition rules are part of the Decision category. A When condition rule is an instance of the *Rule-Obj-When* rule type.

Performing unit testing on a When rule *(on page 269)*

Creating a When rule

Evaluate a Boolean logical statement that involves comparisons among values of properties, to return true or false, by creating a When rule.For example, you can create a When rule to prompt a user to enter a delivery address only when the user specifies that the delivery address varies from the address of residence.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **When**.
3. On the **When** tab, click **Create**.
4. In the **Label** field, enter a description of the When rule.
   Create names of when condition records that logically follow the word when.
5. **Optional:** To enter a when expression that evaluates whether a value is true or false, click **Show additional options**, and then enter the expression.
6. **Optional:** To enable editing the rule in the legacy mode, click **Show additional options**, and then select the **Use legacy authoring mode** checkbox.
   By default, you edit conditions in the condition builder. For more information, see Defining conditions for a When rule *(on page 264)*.
7. In the **Context** section, select an application to which the rule applies.
8. In the **Apply to** field, select a class to which the rule applies.
9. In the **Add to ruleset** field, select a ruleset for the rule.
10. **Optional:** To enable traceability of the rule, in the **Current work item** section, enter the work item to associate with the rule, for example, a bug ID.
11. Click **Create and close**.

When condition rules *(on page 261)*

Defining conditions for a When rule

Enter or revise a When rule that can be expressed as a single Boolean expression or the conjunction (AND or OR) of one or more Boolean expressions.

1. In the navigation pane of Dev Studio, click **Records > Decision > When**.
2. From the list of When instances, select a When rule that you want to edit.
3. On the rule form, click the **Conditions** tab.
4. In the first field, select a value or a property from the list.
5. In the second field, select a comparator from the list.
6. In the third field, enter a value that you want to compare with the value from the first field.

   The system compares the value of this field to the value of the corresponding field or condition in the first column.

7. **Optional:** To specify additional conditions and define the relationships between them by performing the following actions:

    a. Click the **Add a row** icon and add as many conditions as you need.

    b. In the drop-down lists between each pair of conditions, specify how these conditions relate to each other.
    You can group conditions with **AND** or **OR** operators.

    c. In the drop-down list in the upper right corner, specify how you want to evaluate condition groupings
    - **Group ANDs** - Select this option if you want conditions linked with the **AND** operator to be evaluated as a group, and conditions linked with the **OR** operator to be evaluated individually. With this grouping, the grouping of condition 1 **AND** condition 2 **OR** condition 3 **OR** condition 4 is evaluated as (1 AND 2) OR 3 OR 4. That is, either conditions 1 and 2 must both be true, or either one of conditions 3 and 4 must be true.
    - **Group ORs** - Select this option if you want conditions linked with the **OR** operator to be evaluated as a group, and conditions linked with the **AND** operator to be evaluated individually. With this grouping, the grouping of condition 1 **AND** condition 2 **OR** condition 3 **OR** condition 4 is evaluated as 1 AND (2 OR 3 OR 4). That is, both condition 1 and one of conditions 2, 3, and 4 must be true.
    - **Use advanced logic** - Select this option if you want to define a mix of grouping by the **AND** and **OR** operator. Use the **Logic string** field to specify the condition grouping.

    > **Note:** If you change a condition that uses advanced logic back to **Group ANDs** or **Group ORs**, all the groupings that you previously defined are reset.

    > **Tip:** Conditions that are evaluated as a group are displayed on a single block of gray background.

8. Click **Save**.

Building expressions with the Expression Builder *(on page)*
When condition rules *(on page 261)*
Creating a When rule *(on page 264)*

Editing a When rule in the legacy mode

Apply functions and more advanced logic in When rules by editing the conditions in the legacy authoring mode.

By default, when you create and edit When rules, you use the condition builder view. For more information on how to define conditions, see Defining conditions for a When rule *(on page 264)*.

1. In the navigation pane of Dev Studio, click **Records > Decision > When**, and then open the rule that you want to edit.
2. On the rule form, double-click the link below the **When** node.
3. In the first field, enter a property, a literal constant, or a function call.
4. In the second field, select a comparator.
5. In the third column, provide a value that you want to compare with the value from the first field:
    - Enter the value.
    - Click **Select values**, and then select the value.
6. **Optional:** To configure advanced conditions, press the Down arrow button, and then select a condition from the list.
7. Click **Submit**.
8. **Optional:** Develop the rule by clicking **Actions**, and then performing one of the following procedures:
    - To update the condition, click **Edit**.
    - To add another condition, click **Insert condition**.
    - To create a group of nested conditions, click **Insert group**.
    - To delete the expression node, click **Delete**.
9. Click **Save**.

> The system displays the conditions and logical operators that you enter on the **Advanced** tab in the **Condition** array and the **Logic String** field.

When condition rules *(on page 261)*

Configuring advanced options of a When rule

Define complex logic strings and use functions in conditions by configuring advanced options of a When rule.

For example, if you create conditions `1`, `2`, `3`, and `4`, you can define an advanced logic string for the conditions, such as `1 AND (2 OR (3 AND 4))`.

> **Note:** As a best practice, use the **Conditions** tab to configure and update the rule. Your configuration populates the conditions array and the **Logic String** field on this tab. If you add, change, or delete rows, or edit the **Logic String** field, you can no longer use the **Conditions** tab. Updating values in conditions or using the **Options** area does not disable that tab.

1. In the navigation pane of Dev Studio, click **Records > Decision > When**, and then open a rule that you want to edit.
2. On the rule form, click the **Advanced** tab.
3. In the drop-down list, select an expression type.
4. Enter a label for the expression, for example, A.
5. Complete an expression.
6. If you want to add more expressions, click **Add condition**, and then repeat steps 3 *(on page 267)* trough 5 *(on page 267)*.
7. In the **Logic string** field, enter the Boolean logic operations performed on the Conditions array to evaluate if the condition is true or false at run time.
   If you created the conditions on the **Conditions** tab, the system populates this field by using the AND and OR operators, in addition to the group hierarchies that are defined on the condition tree.

   You can use `and`, `or`, and `not` in the statement. Use parentheses to control the order of evaluation.

   > When you save the rule, if the form contains only one test row, the system inserts the label for that test row in the **Logic string** field. If you have more than one row, the system requires all rows to be true. If you want a different outcome, revise and save the statement again.
8. **Optional:** In the **Advanced options** section, in the **Allowed functions** field, select a function alias that restricts the available conditions to the one in this field by clicking **Add function**.
   If you add multiple functions, the top entry is the default one.
9. Click **Save as**.

When condition rules *(on page 261)*
Creating a When rule *(on page 264)*

Specifying pages and classes of a When rule

Ensure that a When rule accesses or updates information on clipboard pages by specifying the page name and class of the pages. At run time, these pages contain the properties that are referenced on the other tabs of a When rule.

Define the pages and classes of a rule to:

- Set up rule prompting during rule configuration.
- Configure rule validation.
- Apply default class for list elements.

For more information about pages and classes, see Defining the pages and classes of a rule *(on page)*.

1. In the navigation pane of Dev Studio, click **Records > Decision > When**, and then open the rule that you want to edit.
2. On the rule form, click the **Pages & Classes** tab.
3. In the **Page name** field, enter the name of the page that contains the property that is referenced in this rule.
4. In the **Class** field, select the class of the page.
5. **Optional:** To add more pages and classes, click **Add item**, and then repeat steps 3 *(on page 268)* and 4 *(on page 268)*.
6. Click **Save**.

When condition rules *(on page 261)*
Creating a When rule *(on page 264)*

More about When Condition rules

When condition rules help you build application logic so that you can deliver software that dynamically responds to changing business scenarios. When condition rules compare the value of one property reference against a constant, or against the value of another property reference. If you define more than one comparison, you can combine the results with `AND`, `OR`, and `NOT` operations to determine the final true/false outcome.

Conditions in activity steps

Each activity step can reference one or several When condition rules, as preconditions for a method, or transitions between the completed method and the next step.

As a precondition, a When condition rule determines whether the method in the activity step runs or not. Referenced in a transition, a When condition determines what activity processing runs next after the completion of the current activity step.

For more information, see Creating an activity *(on page 100)* and Configuring steps in an activity *(on page 101)*.

Standard function

To run a When condition rule in an expression or in the context of Java code, call the standard function *callwhen()* in the Pega-RULES *Default* library.

You can view the generated Java code of a rule by clicking **Actions > View Java**. You can use this code to debug your application or to examine how rules are implemented.

When condition rules *(on page 261)*
Creating a When rule *(on page 264)*

Performing unit testing on a When rule

For more efficient and detailed debugging, you can test a When rule individually before testing it in the context of the application that you develop. You can also convert the test into a Pega unit test case to validate application data by comparing the expected property values to the actual values that the test returns.In a continuous delivery environment, Pega unit testing gives you feedback on the quality of your applications to ensure that you can quickly identify and correct issues.

By default, when you run the When rule, the result assertion uses the input value that you enter and the result that the test returns. The system generates the assertion when you convert this test to a test case.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **When**.
3. In the list of When rule instances, click the rule that you want to test.
4. In the rule form header, click **Actions > Run**.
5. In the **Run** window, in the **Run context** pane, in the **Thread** list, select the thread in which you want to run the rule.
6. In the **Page** list, select whether to copy parameter values from an existing page or to create an empty page:
    - To use parameter values from a clipboard page in the selected thread, click **Copy existing page**, and then, in the **Page to copy** field, select the page that you want to use.

      For example, you can select a clipboard page to test specific values.
    - To start a test from a page that contains no parameter values, click **Empty test page**.

> **Note:** The system runs the rule instance on the *RunRecordPrimaryPage* clipboard page, regardless of the page that you select from this list. If you convert this test run to a test case and the *RunRecordPrimaryPage* clipboard page requires initial values, configure the

clipboard to populate the page with initial values. For more information, see Setting up your test environment *(on page)*.

7. **Optional:** To apply a data transform to copy values for the test, select the **Apply data transform** checkbox, and then select the transform to apply.
8. **Optional:** To change the values that the rule uses during the test, enter the new values on the main test page.
9. Start the test by clicking **Run**.
10. **Optional:** To convert the test run into a Pega unit test case, click **Convert to test**, and then configure the test case.
    For more information, see Creating unit test cases for rules *(on page)*.
11. **Optional:** To view the pages that the unit test generates, click **Clipboard**.
    For more information, see Clipboard pages created by the Run Rule feature *(on page 383)*.

When condition rules *(on page 261)*
Unit testing individual rules *(on page 345)*
Using the Clipboard tool *(on page 374)*
Viewing test case results *(on page)*
Application debugging by using the Tracer tool *(on page 347)*

Debugging When rules with the Tracer tool

If your When rule returns unexpected results, and you cannot determine the problem by running the rule and examining the clipboard pages, run the Tracer tool. Using Tracer, you can watch each step in the evaluation of a When rule, and as a result, you can find and then fix issues in your application faster.For example, you can use Tracer if a When rule returns false but you expect the rule to return true.

1. In the footer of Dev Studio, click **Tracer**.
2. In the header of the **Tracer** window, click **Settings**.
3. In the **Events to trace** section, select **When rules**:
   - To trace events when the rule starts running, select the checkbox in the **Start** column.
   - To trace events when the rule ends running, select the checkbox in the **End** column.

   You can select both **Start** and **End** checkboxes.
4. In the **Rulesets to trace** section, select the ruleset that contains the rule that you want to trace.
5. **Optional:** To avoid memory use from tracing events that occur in other rulesets, clear the checkboxes for other rulesets.
6. Click **OK**.
7. Return to Dev Studio, and then run the When rule.

> **Note:** Keep the **Tracer** window open by minimizing the window.

8. In the **Tracer** window, watch the Tracer output as the rule runs.
9. **Optional:** To obtain more information, click a row in the Tracer output.
10. Analyze values and information present while the When rule runs, and then adjust relevant elements of your application.
    For more information, see Defining conditions for a When rule *(on page 264)*.

Performing unit testing on a When rule *(on page 269)*

Defining conditions in the condition builder

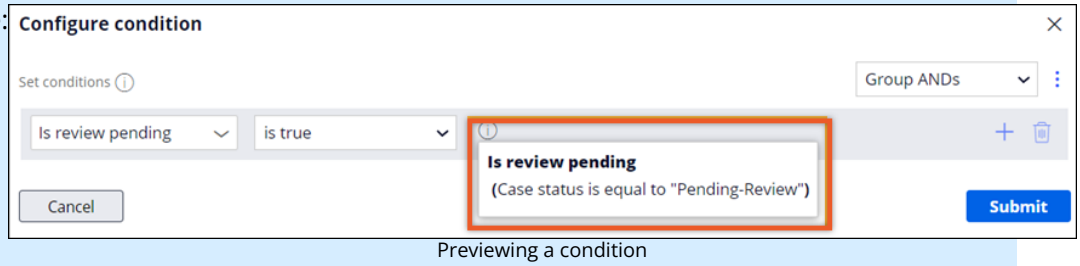
Use the condition builder to create conditions that define the behavior of your application, or to use for propositions evaluated by a proposition filter. You can save custom conditions to the condition library for future use.

1. In App Studio, navigate to a condition builder that defines the application element that you want to edit.
2. **Optional:** For proposition filters, to remove all previously added criteria from the condition builder, click **Actions > Clear all criteria**.
3. In the **Configure condition** section, in the list of values, select a value to evaluate:
   - To select a configuration that defines application behavior at run time, click **Configurations**, and then select the configuration from a configuration set.
   - To select a field from a data model in your application, click **Fields**, and then select the field.
   - To select a question to evaluate the answer at run time, click **Questions**, and then select the question.
   - To select a prediction, click **Predictions**, and then select the prediction.

     If the prediction has a binary outcome, for example, the probability of a fraudulent claim, the following fields are available:
     - **Probability** (value between 0 and 1, for example, 0.01)
     - **Segment** (cutoff probability labels, if defined in the prediction, for example, **Abnormal** and **Normal**)

     If the prediction has a continuous outcome, for example, future credit card transactions, the **Value** field is available and contains the predicted value.

- To select a when condition from your application, click **When conditions**, and then select the condition.

> **Tip:** To view the logic of the When condition that you want to use, click **Preview this condition**, as in the following figure:
>
> 
>
> Previewing a condition

4. In the comparator list, select a comparator.
5. In the value field, enter or select a value that your application compares with the field in the data model or a when condition.
6. **Optional:** To add more conditions, click the **Add a row** icon, and then repeat steps 3 *(on page 271)* through 5 *(on page 272)*.
7. If you add multiple conditions, between the rows, select the **and** or **or** operator to define how to evaluate the conditions.
   You can group conditions using the **and** or **or** operators.

   > If you select **and**, the condition evaluates to true when all of the rows evaluate to true. If you select **or**, the condition evaluates to true if at least one of the rows evaluates to true.

8. In the list in the upper-right of the page, select one of the following options to specify how to evaluate condition groupings:

   - To evaluate the conditions that are linked with the **AND** operator as a group, and to evaluate the conditions that are linked with the **OR** operator individually, click **Group ANDs**.

     With this grouping, the grouping of condition 1 **AND** condition 2 **OR** condition 3 **OR** condition 4 is evaluated as (1 AND 2) OR 3 OR 4. That is, either conditions 1 and 2 must both be true, or either one of conditions 3 and 4 must be true.

   - **Group ORs** - Select this option if you want conditions linked with the **OR** operator to be evaluated as a group, and conditions linked with the **AND** operator to be evaluated individually.

With this grouping, the grouping of condition 1 **AND** condition 2 **OR** condition 3 **OR** condition 4 is evaluated as 1 AND (2 OR 3 OR 4). That is, both condition 1 and one of conditions 2, 3, and 4 must be true.

- To define a grouping by both the **AND** and **OR** operators, click **Use advanced logic**, and then specify the condition grouping in the **Logic string** field.

> **Note:** If you change a condition that uses advanced logic for the **Group ANDs** or **Group ORs**, all the groupings that you previously defined are reset.

> **Tip:** Conditions that are evaluated as a group are displayed on a single block of gray background, so that you can identify them more easily.

9. **Optional:** To reuse the condition in the future, save the condition to the library:
   - For conditions in a workflow, click **More > Add to when condition library**.
   - For conditions in a proposition filter, click **Actions > Save to library**.

> **Note:** The **Save to library** option is not available if the conditions include either Strategy rules or when conditions with parameters. The option is also disabled for proposition filters which include default criteria that are created using the **Use advanced logic** option.

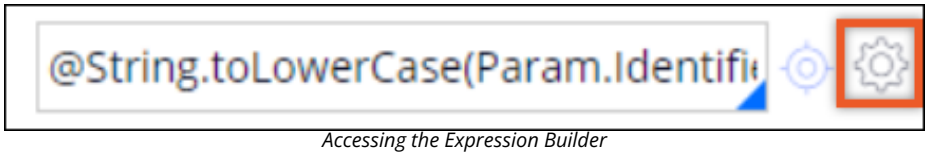
> The conditions are saved to a new when rule in the top level class of the proposition filter. The system automatically registers the when rule as a relevant record for use in the condition builder.

Conditionally starting a process *(on page)*
Defining conditions for skipping a stage *(on page)*
Displaying optional actions conditionally *(on page)*
Displaying supporting processes conditionally *(on page)*
Adding decisions to processes *(on page)*

Expression Builder

The Expression Builder is a tool that guides you in authoring arithmetic and logical expressions in a natural language format. For example, you can compute an employee's salary by defining a calculation that uses properties, such as the number of years employed, employee review rating, and job grade level.

You can open the Expression Builder in multiple locations throughout Dev Studio by clicking the **Build an expression** icon. For example, you can enter expressions in activities, data transforms, constraints, and most other places that compute a value at run time.



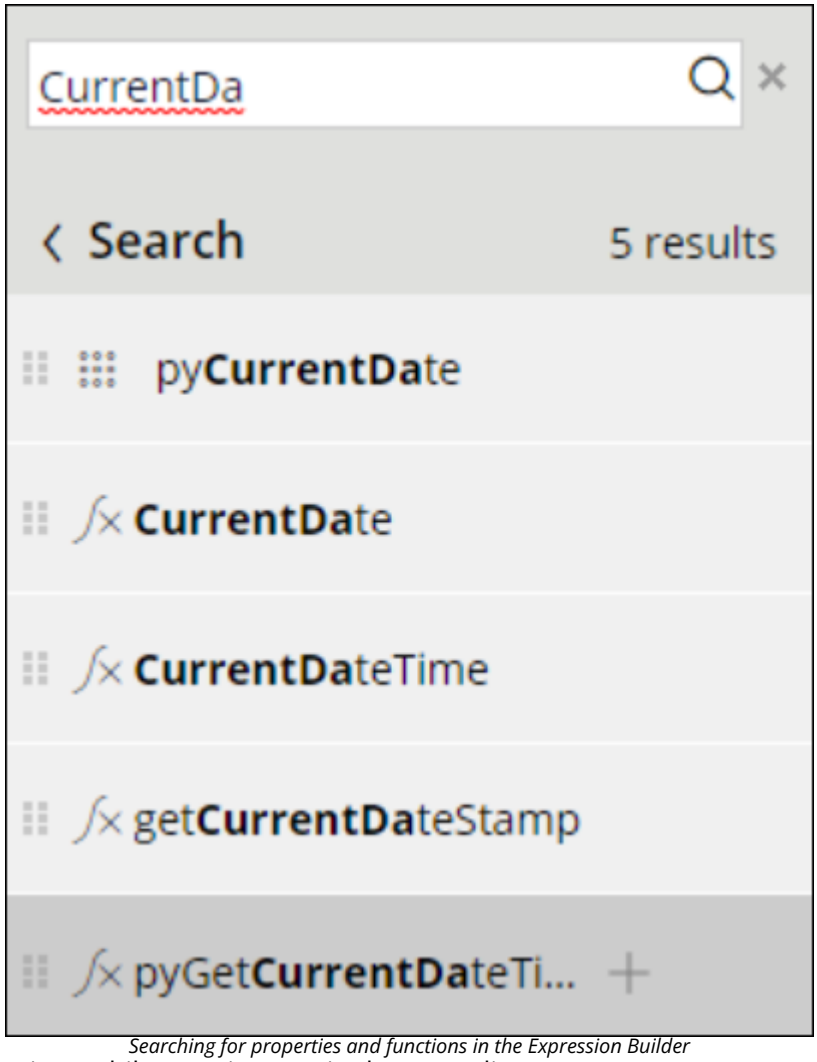
Accessing the Expression Builder

For more information about using the tool, see Building expressions with the Expression Builder *(on page 276)*.

The Expression Builder provides the following features:

- Build expressions in an integrated text editor
- Add functions and properties into the text editor
- Search for properties and non-internal functions.

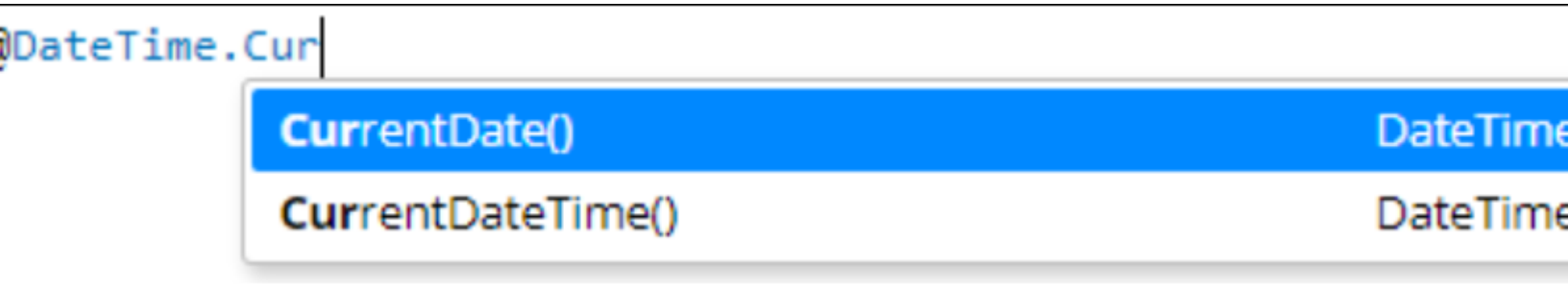
The following figure shows the list of results for a keyword search:



Searching for properties and functions in the Expression Builder

- Select properties and functions while entering text in the text editor

The following figure shows some example

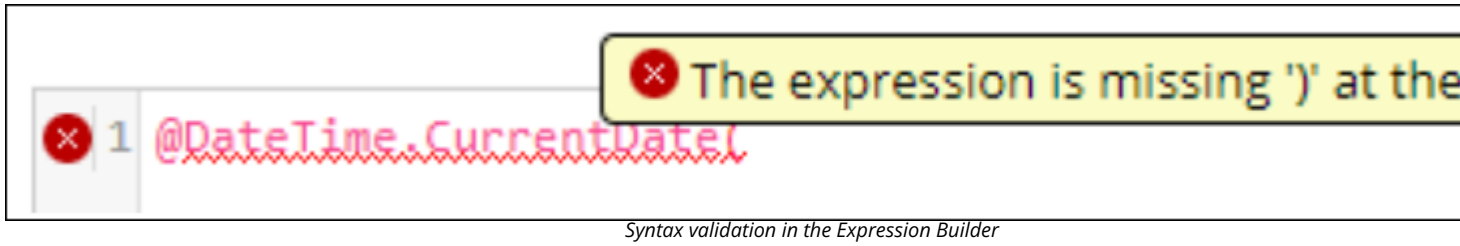


Selecting properties and functions from a text editor prompt

- A navigation pane that displays functions in available libraries and properties within a selected page
- Automatic bracket matching

- Syntax validation

  The following figure shows an example of syntax



Syntax validation in the Expression Builder

Building expressions with the Expression Builder *(on page)*

Expressions *(on page 281)*

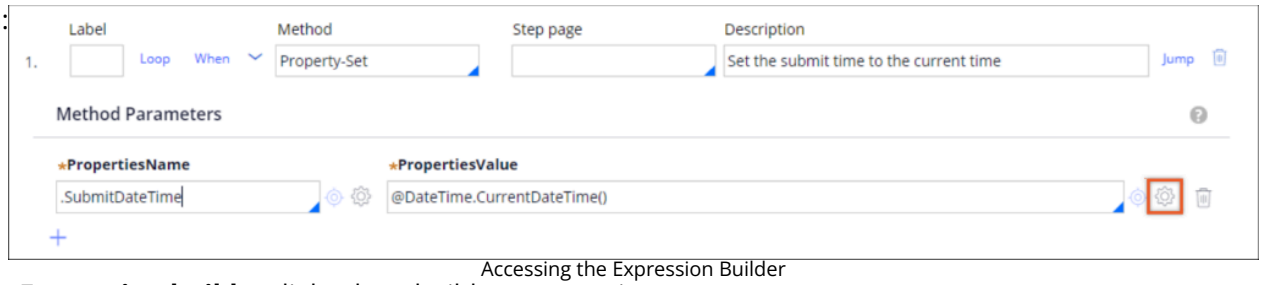
Building expressions with the Expression Builder

Create calculations that save you time by automatically providing values in your application by authoring expressions in the Expression Builder. The Expression Builder eases the construction of expressions in your application by providing a large input area, prompting for functions, prompting for properties, and adding the ability to test your results.

For more information about the Expression Builder, see Expression Builder *(on page 274)*.

An expression is a single text element that, after evaluation at run time, usually returns a single value. Use expressions to perform operations on data, such as performing mathematical operations, comparing date and time values, manipulating text, transforming data, and converting data types. For example, you can create an expression to calculate the total salary of an employee using their base salary, a bonus that depends on their years of experience, and another bonus dependent on the number of certifications that the employee has.

1. In Dev Studio, navigate to a place in your application where you want to build an expression.
   The following figure shows the location of the **Build an expression** icon in a rule form:



Accessing the Expression Builder

2. In the **Expression builder** dialog box, build an expression:

Choices	Actions
Enter the expression manually	In the text field, enter an expression.

Choices	Actions
	You can include combinations of constants, functions, arithmetic operators, logical operators, comparison operators, and property references or property references. For more information, see Examples of expressions *(on page 282)*. **Tip:** When you press the Period (.) key, the Expression Builder prompts a list of the fields available in the data page in which context you currently work. When you press SHIFT+2 (@), the Expression Builder prompts a list of the available functions.
Browse for and apply a function	a. In the header of the **Expression builder** dialog box, click **Browse**. b. In the navigation pane, click **Functions** to expand a list of available function libraries. c. Click a function library to expand the available functions. d. Click the plus icon to add the function to your expression. When you add the function to the text area, the function does not include arguments. **Tip:** Hover over a function to see a tooltip with information about each function, as

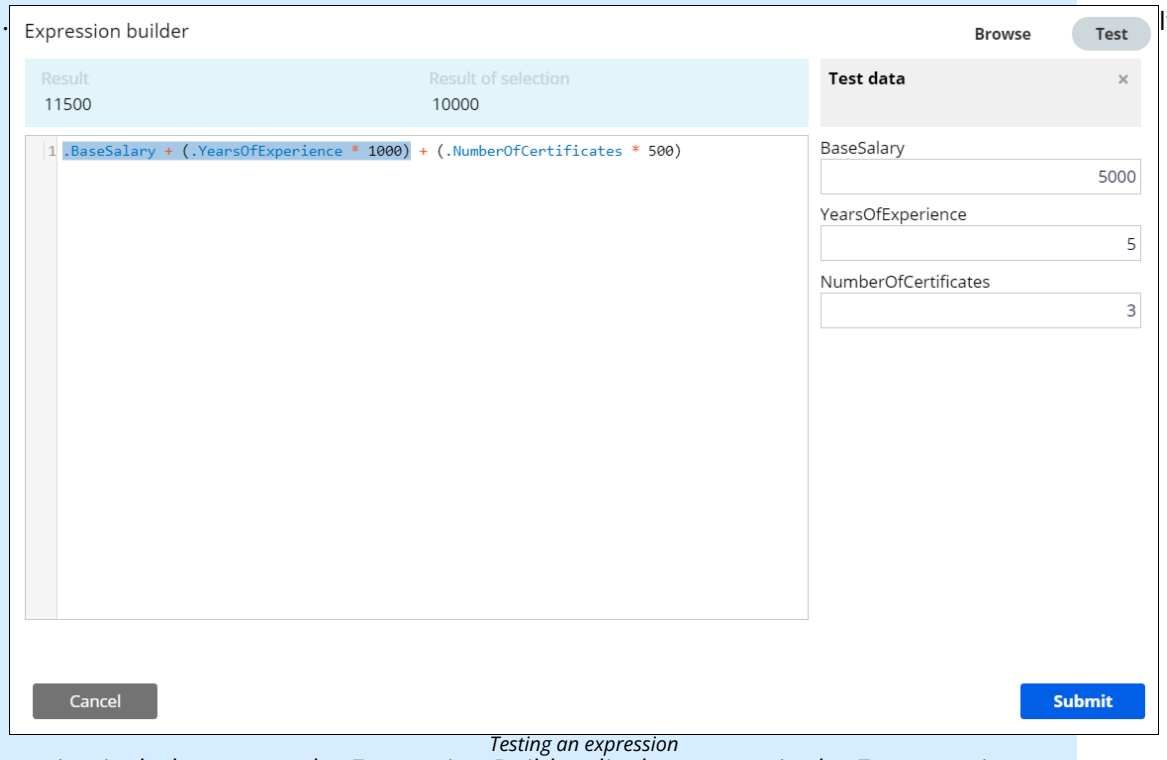
Choices	Actions
	shown in the following figure:  *Sample function* **Note:** The Expression Builder displays only non-internal functions. To display all functions in the system, see Displaying all functions in the Expression Builder *(on page 294)*. e. If the function requires an argument, enter the argument value.
Browse for and apply a property	a. In the header of the **Expression builder** dialog box, click **Browse**. b. In the navigation pane, click **Properties** to expand a list of available properties organized by context. You can select properties from the following types of pages: **Current Page** Includes a list of properties from your current case context. **Custom Pages** Includes properties from custom pages defined on

Choices	Actions
	the **Pages & Classes** tab of the rule that you edit. **Data Pages** Includes a list of properties available from application data pages. **Parameter Page** Includes properties defined as parameters on the rule that you edit. c. Click the selected context to expand the available properties. Some areas, such as data pages or custom pages, might include nested data pages. d. Drag the property to the text field to add the property to your expression. **Tip:** Hover over the property to view additional information, such as the property name, property label, and the class that includes the property.

3. **Optional:** To validate the correctness of the expression, test the expression:
   a. In the header of the **Expression builder** dialog box, click **Test**.
   b. If your expression includes properties or parameters, in the **Test data** pane, enter the values that you want to use for testing.

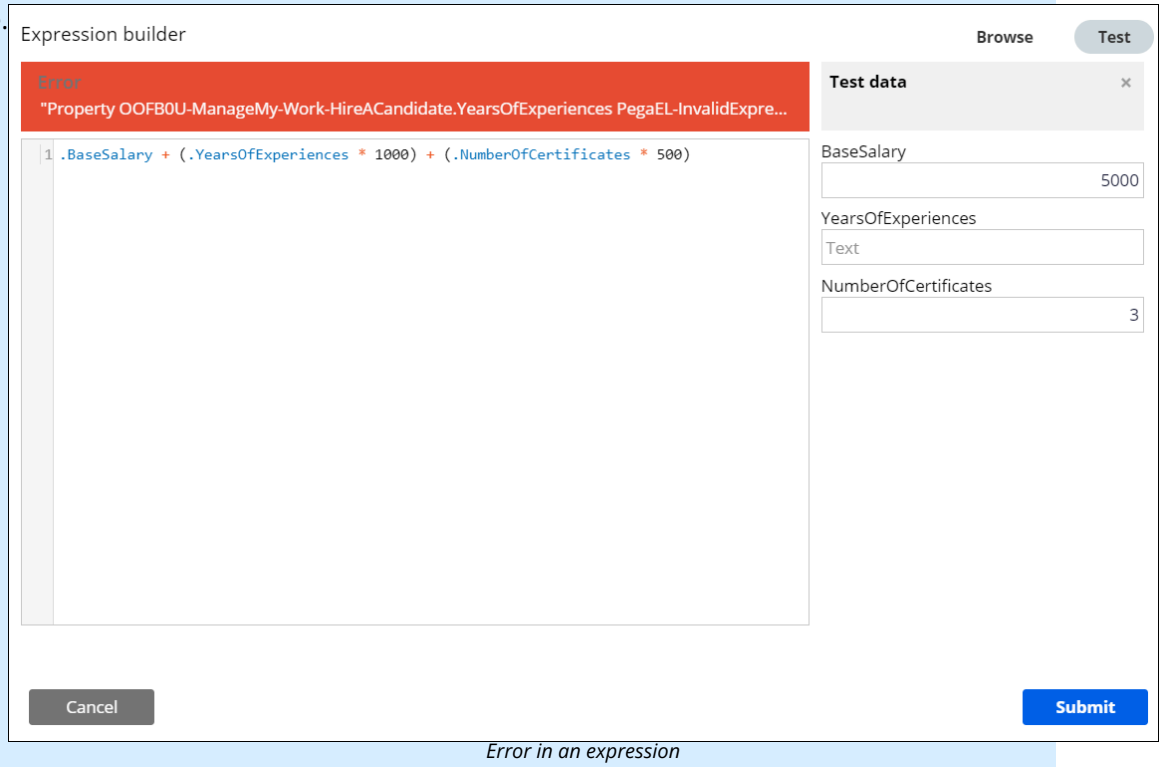
The **Result** section, which is displayed with a blue background in the following figure, includes the results of the computation. You can also test a part of the expression by selecting a fragment that you want to test, as shown in the following figure. If you test a part of the expression, the result appears in the **Result of selection**

section.



Testing an expression

If the expression includes errors, the Expression Builder displays errors in the **Error** section, as shown in the following figure. You can hover over the section to view the full error message.



Error in an expression

4. Click **Submit**.

The value of the expression becomes available in the rule form.

Expressions

An expression is a single text element that when produces a string value. Users create both SQL expressions (or functions) for use in report definitions and Java expressions for use in activities and other rules.

For information about SQL functions, see Report Definitions — Using the Calculation Builder *(on page)*.

The information in this topic discusses creating Java expressions.

Expressions in Pega Platform look similar to formulas in Microsoft Excel but are based on Java language conventions.

You can type in an expression, or (in many places) use the Expression Builder *(on page)* to guide you in entering an expression. Expressions can include constants, property references, operators for arithmetic and logical operations, parentheses to control the order of evaluation, and functions.

Expressions can perform operations on data, including:

- Performing mathematical operations
- Comparing property values
- Manipulating text
- Transforming data
- Converting data types (casting)

> **Note:** Expressions, which are used in many situations, differ from Declare Expression rules, which create expressions that are evaluated automatically. Declare Expression rules contain expressions, but so do many other rule types.

Expressions in rule forms

You can enter expressions in activities, data transforms, Constraints rules, and most other places that compute a value at run time.

In rule forms, the sorts of expressions you can enter into fields in the forms depends on the defined behaviors of the rule form's fields and what sorts of entries they allow. The field in which an expression appears determines whether the expression is part of:

- The right-hand side of an assignment to a property
- The computation of a value

- A Boolean (or logical) expression (yielding a true/false value)
- A property reference containing an expression as a subscript

You can also use expressions in most fields that allow a constant value. Similarly, you can use expressions to supply the subscript value of a list or group element within a property reference.

You cannot use expressions in fields that require property references as the destination of assigning a value. For example, you cannot use an expression in the Target column of a Set action in a data transform or in the Property-Set method parameter array.

Expressions in HTML and XML streams

Through the <when> JSP tag and Java scriptlets (or the<when> directive and Java directive), you can include expressions in HTML-based rules or XML Stream rules. You can use these in HTML rules, XML Stream rules (`Rule-Obj-XML` rule type) and correspondence rules (`Rule-Obj-Corr` rule type).

Building expressions with the Expression Builder *(on page)*
Examples of expressions *(on page 282)*

Examples of expressions

Expressions provide a wide range of computations. Some examples of expressions are:

Constants

An expression can be as simple as a single literal constant.

```
"Good Evening"

142

true

20050705

0x143F871A
```

Single property references

An expression can reference a single property, identifying the page on which it is found. In the context of an activity, an expression can reference a parameter or local variable.

```
.Price

parameters

MortgageLoan.Application.ZIPCode

primary.pyLabel
```

Aggregate property references

You can identify aggregate properties or parts of aggregate properties. (The property mode of the target must match the result of the expression.)

```
MortgageLoan.Application.Address(4)

Globe.Division(7).Unit("West")

Invoices.pyOrders(2).pyItems("Manuals").pyItemNames
```

Linked property references

You can identify properties accessed through linked properties, using the syntax `.property1.property2.`, where property1 is the linked property and property2 is a property reference in the linked class.

> **Note:** Note the initial period character before the property name. (When no initial period is present, the system assumes that property1 is a page name.)

```
.pyUpdateOperator.pyLabel

.pyUpdateOperator.pySkillsPrimary(6)
```

Arithmetic, logical, and comparison operators

You can use most Java operators for arithmetic, string operations, comparisons, and conditions. Use parentheses to control the order of evaluation.

```
.Price * (1+(.Tax/100)) + ShipInfoPage.ShippingCost

.pyEffortActual >= .pyEffortForecast

3.14159*.radius *.radius
```

Function calls

Your expressions can call built-in functions, functions in standard libraries, and custom functions.

```
@SUM(.SubComponents().Price)

@Pega-RULES:MapTo.Function(argument1, argument2)
```

All combined

Expressions can incorporate all the elements described in this topic together:

```
@SUM(.SubComponents(38+.Offset).Price)
```

Expressions *(on page 281)*

Constants in expressions

An expression can consist of only a constant — also called a literal value — or can include constants, operators, property references and functions.

Use these guidelines to enter constants. The value you type may depend on the Type of the property. Include all quote characters as shown. (Don't use Microsoft Word Smart Quote characters; use only the ASCII double quote character `"` to identify constants.)

Type	Instructions	Example
`True-False`	Enter "true" or "false" or "T" or "F." In most cases, such properties appear as a check box or radio buttons. Avoid the null value "" for a `TrueFalse` property; in many but not all settings, it is the same as "false".	`"true"`

`Text`	Enter two double quotes for the null string.
	
	For normal text, enter printable characters (other than double quotes) between two double quotes. You can include single quotes within the double quotes.
	
	Use the two character sequence \" to include a double quote.
	
	Use Java character escapes to include tab characters (\t), backspace (\b), new-line (\n), form feed (\f), carriage return (\r), backslash (\\), and \zzz for the Latin-1 character identified by the octal value zzz, between 000 and 377.
		`""`
		`"Hello World"`
		`"It's a beautiful day"`
		`"Make\tmy\tday."`

`Iden-tifi-er`	Enter a text value not containing double quote characters, newline characters, or carriage return characters.	`MarketPrice`
		`ProductID`
		`<X-95$?? >`

`Inte-ger`	Enter digits 0-9 and an optional plus or minus sign. Do not include commas or a decimal point.
	
	Enter pairs of hex digits (0 to F) preceded by 0x. for a hexadecimal value.
		`-443`
		`1`
		`432329894`
		`0xFFF4`
		`0x143F871A`
		`014371`

`Deci-mal`	Enter any number of decimal digits, a single comma or period, and additional decimal digits. Don't include more than one punctuation mark. Don't start a value with a period; use a leading zero.	`437.1` `3.14159`

Type	Instructions	Example
`Password`	Use any non-blank printable character. Don't include spaces or control characters. Case is significant.	`4X9KK6u`

`Date`	When a date constant appears alone in an expression, include double quotes to prevent the value from being interpreted as an integer.
	
	Enter two or four digits for the year, two digits for the month, two digits for the day.
	
	If the year contains two digits, the system uses an algorithm to determine a "reasonable" century based on the current date. During 2006, the system prefers dates between 1997 and 2097.
	
	The valid expression
	`12/10/2006`
	is an expression involving two divisions and three integers, whereas
	`"12/10/2006"`
	is an American-style date.
	
	The empty string and "" are equivalent to "19700101" or January 1, 1970. In settings where this is not desirable, processing can test for and block this date value.
		`"19990512"`
		`"20021231"`
		`"990101"`

`Double`	Enter an optional sign, digits, decimal point, digits, the letter E and an exponent. All parts of the literal are optional except for the decimal point and one digit.	`123.45`
		`0.04`
		`6.02E23`

`DateTime`	Enter a value using this pattern: `yyyyMMddtHHmmss.SSS zzz`. Enter: • Two or four digits for the year • Two digits for the month • Two digits for the day • The single `T` (or t or period . or comma ,) • Two digits for the hour • Two digits for the minutes • Two digits for the seconds • A period • Three digits for the milliseconds	`20060415T115959.123 GMT`

Type	Instructions	Example
	• One space • The literal `GMT` for the zone. The year, month, and day are required. Separate the date and time portions of the value with either `T`, `t`, a single space, or a punctuation mark such as a period or a comma. If you include milliseconds, separate the seconds from the milliseconds with either a period (`.`) or a comma (`,`). Use the GMT time zone only. The time portion is optional. If you omit the time portion, follow the guidelines above for Date constants, by enclosing the value within parentheses. If the year contains only two digits, the system uses an algorithm to determine a "reasonable" century based on the current date. During 2007, the system prefers dates between 1998 and 2098.	
`Time of Day`	Enter two digits for the hour, from 00 to 23. Optionally, enter two digits for the minute, from 00 to 59. If you include six digits, the system interprets the last two as seconds. You can include an optional colon to separate the segments of this value. Do not enter a time zone.	`000000` for midnight `12:00:00` for noon `23:59:59` for one second before midnight

Expressions *(on page 281)*
Data transforms *(on page 140)*

Functions in expressions

Functions extend the power of Pega Platform expressions. An expression can contain many function calls, including calls to built-in functions, standard functions, and custom functions that you create. Use the Function Builder to help find the function that you need.

This topic relates to creating and using Java functions. For information about creating and using SQL functions in Report Definition rules, see About Report Definitions *(on page)* and Report Definition Rules — Using the Calculation Builder *(on page)*.

Expressions *(on page 281)*

Methods for calling a function

The following forms call a function.

Fully qualified function call

The fully qualified syntax for calling a function identifies both the ruleset and the library:

```
@(RuleSet:libraryname).functionname(arg1, arg2... argn)
```

For example:

```
@(MyInventory:Sort).CardType("Gold")
```

Using fully qualified references eliminates the risk of function overrides by a library or ruleset.

By fully qualifying a ruleset, standard ruleset resolution does not apply. Double-check your requirements before using this type of function call.

> **Note:** A deprecated syntax used before Version 04-01 is still available as an alternative to using a fully qualified function call:

```
lib(ruleset:libraryname).functionname(arg1, arg2... argn)
```

When using this syntax, "ruleset" may be omitted (defaults to `Pega-RULES`), and "libraryname" may be omitted (defaults to `default`).

Library qualified function call

The library qualified syntax for calling a function identifies the library name:

```
@LibraryName.FunctionName(arg1, arg2... argn)
```

Using library qualified references provides a more readable syntax than the fully qualified syntax, and prevents accidental library conflicts that can occur with unqualified calls.

For example, if two libraries with the same name are located in different rulesets, when using library qualified notation, the library from the ruleset listed higher in the operator runtime ruleset list is selected.

Additionally, using this notation helps to avoid multiple "suitable found" instances, such as when the same function exists in two different libraries but in the same ruleset.

With library qualified references, standard ruleset resolution applies, allowing the function to be resolved from any ruleset. This provides the capability to override a function in a different ruleset name, similar to how other rule types can be overridden.

Unqualified function call

The unqualified syntax for calling a function calls a function rule:

```
@FunctionName(arg1, arg2... argn)
```

The unqualified syntax is the simplest to read, but does not identify a ruleset or a library for the function. Your system may contain many functions that match the function name.

> **Note:** When using this syntax, if another developer creates a function with the same name and signature in another library or ruleset, their call may be affected.

Pega Platform uses your ruleset list to find the best one to execute, using the following algorithm:

1. Collect a list of the `Rule-Utility-Function` rules belonging to the rulesets listed in the user's ruleset list that have the same name and same number of parameters.
2. Exclude functions defined in any library named default. (The use of the default library is deprecated.)
3. If the list is empty, no such functions exist. The system reports an error.
4. If more than one such function exists (because the same name is overloaded), select the best match by comparing the data types of the parameters, using the promotion rules for Pega Platform data types.
5. If the `Rule-Utility-Function` rule has not supplied data types for its parameters or return value, use standard Java promotion rules to match parameters. As a last resort, substitute `double` for the preferred `BigDecimal` type if necessary to get a match.
6. If more than one function matches the selected name and fully decorated signature, then the function from the higher listed ruleset is executed.

Java call

Functions when compiled extend the Java class:

`com.pegarules.generated.` *myruleset_mylibrary*

where the ruleset name and library name are converted to lowercase. To call a function from within Java, use the following syntax:

```
result = com.pegarules.generated.
myruleset_mylibrary.MyFunction( params )
```

using the exact case for the function name.

For example, to call the CardType() function in the Sort library of the MyInventory ruleset, use:

```
myinventory_sort.CardType("Gold")
```

Pega Platform constructs the Java class name for a `Rule-Utility-Library` by concatenating the ruleset name, an underscore and the library name, with the resulting string converted to lowercase and characters not valid in a Java identifier *(on page)* translated to an underscore.

Built-in and When() functions

Four functions are part of the expression language. All other names following an at-sign are references to function rules, instances of the `Rule-Utility-Function` rule type. An expression can use any of these built-in functions:

Syntax	Description
These functions are part of the expression language. Their names are reserved and cannot be overridden. These functions may only be invoked using the preferred ('@') syntax.	
`@if( boolean expression, result1, result2)`	Evaluates the condition in parameter one and then, if true, evaluates and returns parameter two, otherwise evaluates and returns parameter three. Equivalent to the Java ternary operator "?:." Only the one appropriate result parameter is evaluated.
`@and( e1, e2, ...)`	Logical AND operation of the (arbitrary number of) parameters (Evaluated left-to-right, stopping when the result is known)

Syntax	Description
`@or( e1, e2, ...)`	Logical OR operation of the (arbitrary number of) parameters (Evaluated left-to-right, stopping when the result is known)
General purpose	
`@when(name)`	Evaluates a when condition rule in the context of the primary page of the rule in which the expression appears. The parameter can be an expression that evaluates to a text value.
`@when(name, pagename)`	Evaluates the specified when condition rule using the supplied `ClipboardPage` PublicAPI object (not a page name) as the primary page for evaluation. Either parameter can be an expression that evaluates to the correct data type. **Note:** The primary page for evaluation is supplied as a `ClipboardPage` object, not a page name, so that embedded pages (or pages whose name is not known) can be evaluated.

You can use a `Value List`, `Value Group`, or `Page Group` property as an argument to the @SUM(), @MIN(), @MAX() and @AVERAGE().

Arithmetic functions

These basic arithmetic functions are available.

Syntax	Description
Scalar arithmetic	
`@abs(num)`	Absolute value
`@sqr(num)`	Square
`@sqrt(num)`	Square root

Syntax	Description
`@exp(num)`	Exponential
`@ceil(num)`	Ceiling, least integer not greater than
`@floor(num)`	Floor, greatest integer not less than
`@round(num)`	Rounded to the nearest integer
Aggregate arithmetic	
`@sum(num1, num2)`	Sum
`@max(num1, num2)`	Maximum
`@min(num1, num2)`	Minimum
`@mode(num1, num2)`	Modal value
`@stddev(num1, num2)`	Standard deviation
`@average(num1, num2)`	Average

You can use a `Value List`, `Value Group`, or `Page Group` property as an argument to the @SUM(), @MIN(), @MAX() and @AVERAGE().

DateTime functions

The following functions operate with date and time values. A `DateTime` property type represents an instant in time while `Date` and `TimeofDay` property types represent values. Adding (the BigDecimal/double values corresponding to) a `Date` and a `TimeofDay` together similarly represents an instant in time, but in an unknown or arbitrary time zone.

Syntax	Description
These Rule-Utility-Function rules are similar to the corresponding functions in Microsoft Excel.	

Syntax	Description
@date(year, month, day)	Returns a BigDecimal value (Pega Platform Date) corresponding to the i parameters specified. The year is a four digit year, and is not adjusted fo "base date." The parameters are interpreted leniently (as in Microsoft Ex through the use of the Java Calendar class.
@datevalue(string)	Returns a BigDecimal value (Pega Platform Date) corresponding to the parsed value of the string specified. Only the date portion of the string is considered.
@day(double) @day(BigDecimal))	Returns the int value corresponding to the day of the month (1 to 31) wh the double parameter is interpreted as a Pega Platform Date.
@month(double) @month(BigDecimal)	Returns the Java int value corresponding to the month of the year (0 to 1 when the double parameter is interpreted as a Pega Platform Date. The corresponding Excel function also returns 0 to 11.
@today() @today(selection)	Returns the BigDecimal value (Pega Platform Date) corresponding to the current date.
@weekday(double) @weekday(BigDecimal)	Returns the int value corresponding to the day of the week (Sunday = 1 t Saturday = 7) when the double parameter is interpreted as a Pega Platfo Date.
@year(double) @year(BigDecimal)	Returns the int value corresponding to the year when the double parame ter is interpreted as a Pega Platform Date.
@hour(double) @hour(BigDecimal)	Returns the Java int value corresponding to the hour of the day (0 to 23) when the double parameter is interpreted as a Pega Platform Time of Da
@minute(double) @minute(BigDecimal)	Returns the Java int value corresponding to the minute of the hour (0 to when the double parameter is interpreted as a Pega Platform Time of Da
@second(double) @second(BigDecimal)	Returns the Java int value corresponding to the second of the minute (0 59) when the double parameter is interpreted as a Pega Platform Time o Day.
@time(hour, minute, second)	Returns a BigDecimal value (Pega Platform Time of Day) corresponding the int parameters specified.
@timevalue(string)	Returns a BigDecimal value (a Pega Platform Time of Day value) corresponding to the parsed value of the string specified. Only the time portio of the string is considered.
The following functions facilitate conversion between BigDecimal/double	

Syntax	Description
values derived from DateTime properties and Date or Time of Day properties. Use these functions to adjust the double value resulting from the sum of a Date and Time of Day, or the double value resulting from a DateTime property into a (possibly different) known time zone. The return type of these functions is the same as the type of the first argument.	
@toGMT(double, zone) @toGMT(BigDecimal, zone)	Returns a double/BigDecimal value (Pega Platform DateTime) in the GM time zone corresponding to the double value interpreted in the “zone” ti zone. Use to convert the sum of a Date and Time of Day to a DateTime. I zone is null, then the time zone of the requestor is used.
@toLOCAL(double, zone) @toLOCAL(BigDecimal, zone)	Returns a double/BigDecimal value that is the result of treating the doub value as a Pega Platform DateTime (in the GMT time zone) and convertin to the time zone. You can then treat the resulting value as Date and Time Day values in the specified time zone. If zone is null, then the time zone the requestor session is used. For example, if bostonTime is a double va that represents a date and time in the Eastern Standard Time zone, you find the corresponding time in India by using: toLOCAL(toGMT(bostonTi “EST”), “IST”).

Calling an activity in an expression

The standard function `callActivity( )` in the Pega-RULES `Utilities` library calls an activity, but returns only void, not a value result. You can execute an activity in an expression, but only for its side effects. Identify the primary page, the activity, and the parameter page. For example:

```
@Pega-RULES:Utilities.callActivity(pyWorkPage, MyActivity, tools.getParameterPage());
```

Calling an overloaded function

Pega Platform allows multiple function rules to be defined with the same name, in the same ruleset, version, and library, if they have different signatures. The signature of a function is computed automatically from the name, the position and data types of parameters, and the return type. Functions that have multiple variants are called overloaded.

When calling an overloaded function, you do not need to specify which variant to use. The system determines which to use based on the types and positions of parameters you supply.

For example, assume one function named Round is defined with two variants:

- Round(double: d) — Returns an int by rounding the input value d, a double
- Round(double: d, places: n) — Returns a double by rounding the value to n decimal places

If your call includes two parameters, the system uses the second variant.

Displaying all functions in the Expression Builder

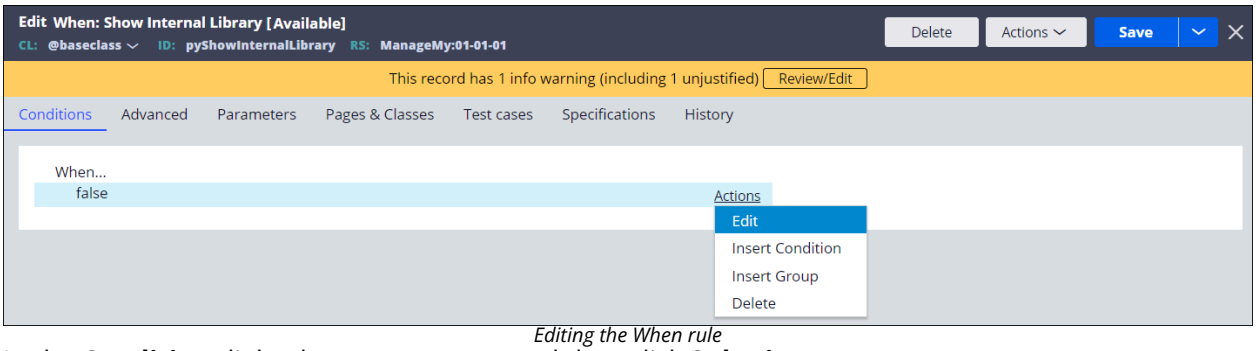
Build advanced expressions that automate calculations in your application by displaying internal functions and libraries from your entire system in the Expression Builder. By default, the Expression Builder displays only non-internal libraries and functions. You can display all functions in the system by overriding the *pyShowInternalLibrary* When rule setting.

Internal functions are not displayed by default, because they are subject to change without notice. Use internal functions with caution. A function is internal if the *pyMethodStatus* property of the library that stores the function has the value of `Internal`. To check the *pyMethodStatus* property, view the XML or the clipboard page of the rule. For more information, see Viewing the XML of a rule *(on page)* and Using the Clipboard tool *(on page 374)*.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **When**.
3. In the list of instances of When rules, click the **pyShowInternalLibrary** rule.

> **Tip:** To find the rule faster, you can filter the rules by name.

4. In the rule form header, click **Save as**.
5. On the **Save As When: Show Internal Library** form, in the **Context** section, in the **Add to ruleset** list, select your application ruleset and ruleset version.
6. Click **Create and open**.
7. On the **Conditions** tab, in the **When...** section, click **false**.
8. Click **Actions > Edit**, as shown in the following figure:



Editing the When rule

9. In the **Condition** dialog box, enter `true`, and then click **Submit**.
10. Click **Save**.

> When you author expressions in the Expression Builder, you can browse through all functions and libraries in the system.

Related information

When condition rules *(on page 261)*

Defining URL patterns for work items

Share your work items with users of the same application by defining URL patterns for the work items in your application, for example, cases, reports, landing pages, or other work items, such as *sitemap.xml*. Simple, meaningful URLs make collaboration more efficient for users. For example, they can bookmark a case for quick access, and then send a direct link to the case to other users.

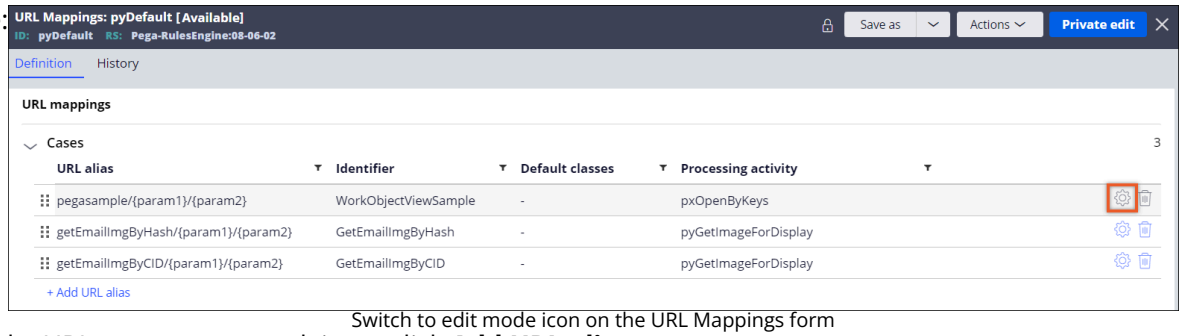
Before you define URL patterns for work items, your application must contain a URL Mappings rule. Each application can have only one URL Mappings rule, whose name is always *pyDefault*. You define URL patterns by editing this rule.

1. Create a URL Mappings rule in your application:

> **Note:** If your application already contains a URL Mappings rule, go to step 2 *(on page 295)*.

   a. In the header of Dev Studio, click **Create > Technical > URL Mappings**.
   b. In the **Context** section, in the **Add to ruleset** list, select the name and version of a ruleset to store the URL Mappings rule.
   c. Click **Create and open**.
   d. Click **Save**.
2. In the navigation pane of Dev Studio, click **Records**.
3. Expand the **Technical** category, and then click **URL Mappings**.

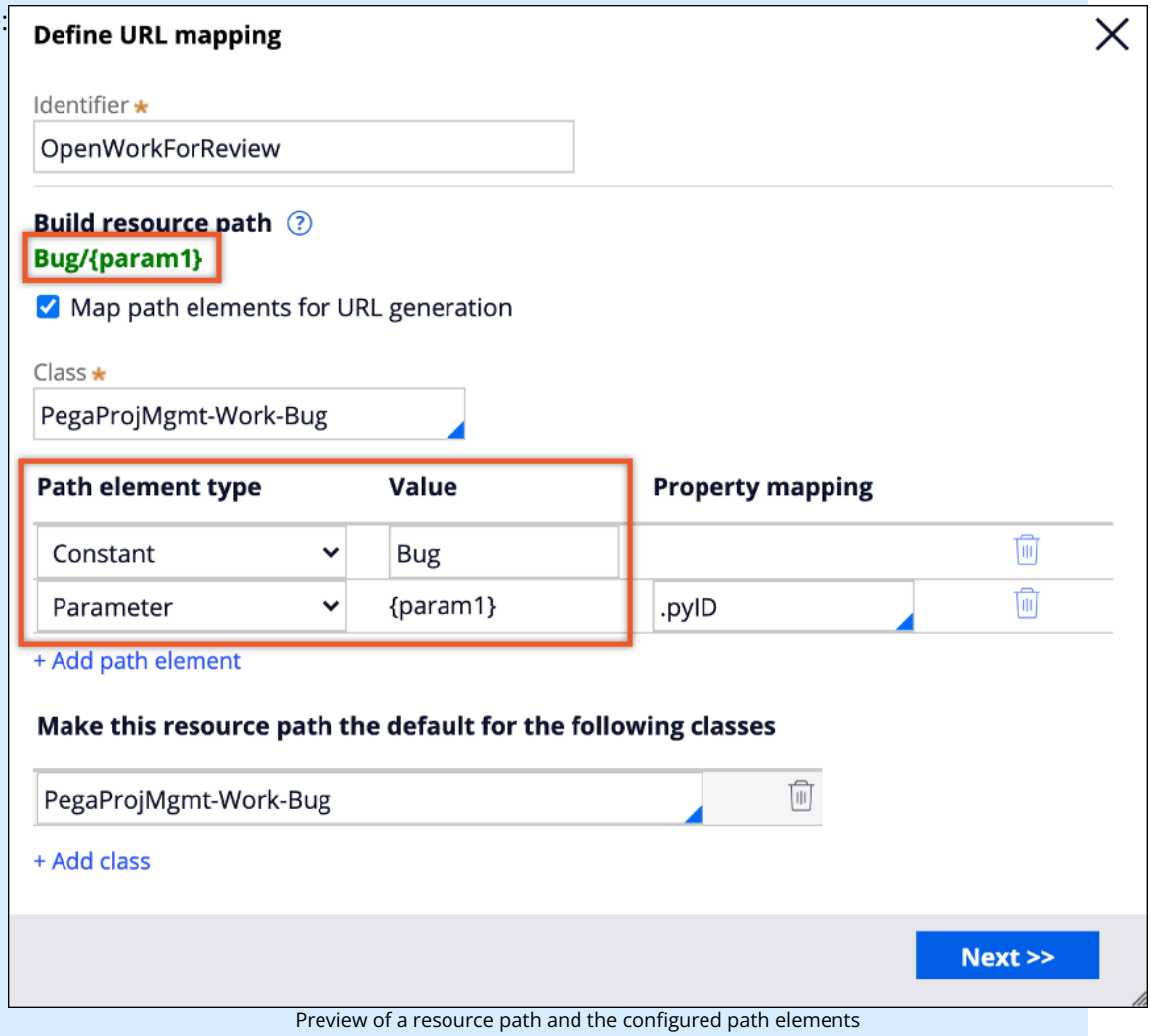
4. In the list of URL Mappings instances, click *pyDefault*.
   Each application has only one valid URL Mappings rule, whose name is always *pyDefault*.
5. On the **Definition** tab, expand the category of work items for which you want to define the URL pattern.
6. Start configuring a URL pattern:
   - To modify a pattern, click the **Switch to edit mode** icon, as shown in the following figure:



Switch to edit mode icon on the URL Mappings form

   - To add a URL pattern to a work item, click **Add URL alias**.
7. In the **Identifier** field, enter a name for the URL alias that uniquely identifies it in all patterns in the rule.
8. **Optional:** To generate a valid URL for an object, apply the URL pattern to the object:
   a. Select the **Map path elements for URL generation** checkbox.
   b. In the **Class** field, specify the class of the object for the properties mapping.
9. In the **Path element type** column, select **Constant**.
   The first path element must be a constant.
10. In the **Value** field, enter text that is a constant element in the URL pattern.
11. **Optional:** To add more path elements to the URL pattern, click **Add path element**.
    a. To define text, in the **Path element type** column, select **Constant** and then, in the **Value** field, enter the text.
    b. To define a parameter, in the **Path element type** column, select **Parameter**.
    c. If you selected **Map path elements for URL generation** in step 8 *(on page 296)*, in the **Property mapping** column, enter a clipboard property to map to the URL path element.

Note: You can preview the URL pattern that you create below the **Build resource path** label, as shown in the following figure:



Preview of a resource path and the configured path elements

12. **Optional:** To make this URL pattern the default for objects in specific classes, in the **Make this resource path the default for the following classes** section, click **Add class**, and then in the text field, enter the name of the class.
    The class can be the same class that you entered in step 8 *(on page 296)*, or any of the child classes.

Note: The option is available only if you selected **Map path elements for URL generation** in step 8 *(on page 296)*.

13. Click **Next**.

14. In the **Edit processing activity** dialog box, in the **Class** field, enter the class of the processing activity for the URL pattern.
15. In the **Activity** field, specify the processing activity that controls running the URL.

    > The system identifies activity parameters and displays them in the table.

16. In the **Value** column, enter the values for the processing activity parameters.
    Map your URL path elements (parameters) to the processing activity parameters. For example, in step 11 *(on page 296)*, if you defined a URL path element as the {param1} parameter, in the **Value** field for an activity parameter, enter `{param1}`.
17. **Optional:** To pass an additional parameter to the activity that the current activity invokes, click **Add parameter**, and then repeat step 16 *(on page 298)*.
18. Click **Finish**.
19. **Optional:** To add a URL pattern to another category, expand the category, and then repeat steps 6 *(on page 296)* through 18 *(on page 298)*.
20. On the **URL Mappings** form, click **Save**. (Note that if you make any changes to an activity referenced in a URL Mapping, you must re-save the individual URL Mapping. This step is necessary irrespective of any change in the individual URL Mappings rule.)

Related information

URL Mappings troubleshooting FAQs

Developing applications in branches

Your teams can develop multiple features simultaneously without overwriting work and causing conflicts by implementing branched application development. During branched development, developers can make changes in one ruleset that affect other developers only after you merge the changes into a target ruleset. As a result, you speed up application development and can clearly analyze what changes your application includes.

For example, by using branches, two teams can work simultaneously on different application features, such as case type creation and notifications. Each team can make can make changes that do not affect the changes of the other team during the development process, even if the features use the same base rulesets. When the development process is complete, you can review branches to resolve any conflicts, and then merge the results into a target ruleset in order to make the new features available in the final version of your application.

To learn more about branched application development, see the following articles:

Implementing branched application development

For faster delivery of your products, configure your application development to use branches. By implementing branches, team members can work simultaneously on multiple features, without the risk of interrupting work of different team members.For example, team members can develop different service-level agreement rules or work on multiple bugs in parallel by using separate branches.

To use branches in your application, create a development application that is built on your base application. You create and save rules in your development application, which results in creating branch rulesets. You can then work with branches in several ways, such as creating branch reviews and merging branches.

To configure your application to use branches and branch rulesets, complete the following steps:

1. Create a development application that is built on your base application.
   For more information, see Creating a development application *(on page 299)*.
2. Add branches to your application, into which you can save the rules that you develop.
   For more information, see Adding development branches to your application *(on page 300)*.
3. Work with branches in several ways that meet your business requirements. For example, create a review, merge branches, or delete branches from the system.
   For more information, see Branch operations *(on page 303)*.

Setting branch development preferences *(on page 303)*
Branches and branch rulesets *(on page 324)*

Creating a development application

To speed up application development, you can support branched development by creating a development application that is built on the production application. By providing branches for development, you ensure that different teams and team members can work on different features simultaneously, without a risk of creating errors and conflicts.For example, different team members can work on bug fixes that correspond with two separate features but refer to the same base rule. During development, conflicts might occur. Edits made by one team member might impact the work of other developers. Creating a development application is not mandatory but can provide an uninterrupted development process, especially for large or multiple development teams that work on the same application.

For branched development, create a new application by copying an existing application. As a result, the names of classes and rulesets remain unchanged. Consequently, you can conveniently merge your changes after the development in a branch is complete.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. In the rule form header, click the Down arrow next to the **Save** button, and then click **Save as**.

3. In the **Application Record Configuration** section, in the **Label** field, enter a name of your development application.
   Team application names typically reflect the base application and team name or the focus of the team.
4. Create a unique application rule by changing the identifier:
   a. In the **Identifier** section, click **Edit**.
   b. In the dialog box that is displayed, in the text field, enter a new identifier.
   c. Click **OK**.
5. **Optional:** To create another version of a development application that already exists, in the **Version** field, enter a new version number.
6. In the **Context** section, in the **Add to ruleset** list, select a ruleset to store your edits.
   The ruleset that the system prompts by default is the ruleset in your production application. If you select a different ruleset, you are unable to add your changes directly to the production application.
7. Click **Create and open**.

Building your first application *(on page 16)*

Adding development branches to your application

To avoid conflicts and errors when team members work simultaneously, add development branches to your application so that team members can work simultaneously on multiple features without risking conflicts and errors that can arise from working on the same rules.For example, one team member can fix bugs in the application UI while another developer fixes invalid service-level agreement rules. If both rules refer to the same base rule, working in separate branches prevents conflicts during development.

1. In the header of Dev Studio, click the name of the application, and then click **Definition**.
2. In the **Development branches** section, click **Add branch**.
   You can add branches that contain rulesets only from your current application.
3. In the **Add a Branch ID** dialog box, in the **Branch name** field, enter a branch name:
   - To create a new branch, enter a unique name that starts with a letter.

     Ensure that the branch name reflects the purpose so that your development team can easily identify the correct branch in which to save changes during application development.
   - To reuse an existing branch from your system, press the Down arrow key, and then select a branch that you want to use.
4. Click **Submit**.
5. **Optional:** If you have multiple branches, you can specify how the system selects the rules for rule resolution by reordering the list of branches.
   For more information, see Reordering branches *(on page 307)*.

The system selects the rules from the top branches first.

6. **Optional:** If you use Live UI, to facilitate application development so that less technical users can modify common properties of a control or the layout of rules in the branch, set up and manage a run-time branch.
   For more information, see Enabling run-time branching and editing *(on page)*.
7. Create rules and add them to your branch.
   The system automatically creates rulesets where you save your rules. For more information about working with rules in branches, see Rule development in branches *(on page 301)*.

> When you develop your application, you can select the branch and ruleset into which you want to save new rules.

Branch operations *(on page 303)*

Final rules and utility finctions in branched development

To add rules to branches, when you create a rule, you can select the branch where you want to save your rule. Branch rulesets are automatically created.

Working with rules in the branch ruleset is the same as in standard rulesets, except for the following differences:

- Final rules – You can typically modify final rules only in the ruleset and class in which they are declared final. This limitation means you cannot normally copy final rules to another ruleset that either has the same Applies To class or is located within the subclass of the ruleset of the final rule. For branch rulesets that are branched from the originating ruleset of the final rule, this limitation is removed. You can check out a final rule from its original ruleset into a branch ruleset that is branched from the owning ruleset. For more information, see Checking out a rule to a branch *(on page)*.
- Utility functions – Some function references in the system use the ruleset name as part of the reference. To allow for rules to appropriately reference functions that exist in rules in a branch ruleset, the system's function lookup considers the branches in your ruleset list when resolving the references.

Branch operations *(on page 303)*

Developing branches with libraries and functions

Libraries hold custom functions for your application, which can supplant and extend standard functions. For example, the OperatorHasPrivilege function verifies that a user has a specific privilege.

However, you cannot create libraries in branch rulesets. Libraries are based on the ruleset name only, not the ruleset version. Because you must merge branches into a particular ruleset version, you cannot create libraries directly in branch rulesets.

To use libraries and functions in branches, perform the following steps:

1. Create a library and save it in the base ruleset.
2. Create the functions that the library contains and save them into a branch ruleset.
3. Merge the branch so that libraries and functions are contained within the branch.

Final rules and utility finctions in branched development *(on page 301)*
Merging branches into target rulesets *(on page 312)*

Adding system branches

You can add branches that already exist on your system to your application. You cannot add a branch that contains branched versions of a ruleset that is not in your application stack.

1. In the navigation panel, click **App**, and then click **Branches**.
2. Right-click the application to which you want to add a system branch and select **Add branch from this system**.
3. Press the **Down Arrow** and select the system branch that you want to add.
4. Click **Submit**.
5. If you are using multiple branches, reorder the list of branches so that it matches the order in which rules should be resolved.
   For more information, see Reordering branches *(on page 307)*.
6. Create rules and add them to your branch.
   When you create rules, you can select the branch and ruleset into which you want to save them. Rulesets are automatically created. For more information, see Final rules and utility finctions in branched development *(on page 301)*.

Adding development branches to your application *(on page 300)*

Creating branch rulesets

Provide your development team with an ability to work on multiple features in parallel by creating branch rulesets. When you save rulesest in different branches, team members can work on isolated rulesets without impacting development of other features. As a result, you avoid overwriting results and generating errors.

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

    > The system autopopulates the version and description of your ruleset.
3. Select the **Branch RuleSet** check box.
4. In the **Branch ID** list, select a branch to store your ruleset.
5. In the **Ruleset to branch** list, select a ruleset that you want to branch.
6. Click **Create and open**.

> The system creates a new ruleset version and saves it in the branch.

Related information

Adding development branches to your application *(on page 300)*

Branch operations

After you create branches and develop rules in branch rulesets, you can work with branches in a number of ways. For example, you can create branch reviews with other users, delete branches form the system, and lock branches before you merge them.

Branches and branch rulesets *(on page 324)*
Integrating with file and content management systems *(on page)*

Setting branch development preferences

Define branches where you want to save the changes that you make in App Studio, so that you can work on a feature without affecting other parts of your application.

For example, you can create a sandbox branch where you can develop and test different rules, and then merge or discard your changes. You can also create multiple branches to work on different features or bugs within one application.

1. In the header of Dev Studio, click the **Toggle branch development** icon.
2. In the header of App Studio, click the **Toggle branch development** icon.
3. In the **Branch development preferences** dialog box, toggle the branch development on.
4. **Optional:** From the list, select an application layer where you want to make changes.
5. Define the branch to save your changes:

- From the list, select a branch.
- If you need a new branch, click **Create a new branch**, and then complete the dialog box.

6. Click **Submit**.
   By default, Dev Studio sets the context of new and updated rules to the highest branch in your application.

Final rules and utility finctions in branched development *(on page 301)*

Viewing branch quality and branch contents

You can view information about your branch, such as the rules that it contains and whether branches have been reviewed. You can also view quality metrics such as guardrail warnings for information about the health of your branch.

1. In the Dev Studio navigation pane, click **App**, and then click **Branches**.
2. Click a branch, and then click the **Content** tab to view the rules that the branch contains.
   You can also see if a branch has been reviewed and can open a review in the branch.
3. Click the **Branch quality** tab to view guardrails, merge conflicts, and information about the unit tests that are configured on the rules in the branch.

Branches and branch rulesets *(on page 324)*
Understanding unit test cases *(on page)*

Understanding branch quality metrics

You can view quality metrics to monitor the health of your branch.

Branch quality metrics include the following information:

Guardrails

Monitor the number and severity of guardrail violations that were found in your branch. By investigating and resolving the violations, you can improve your branch's overall quality.

The **Guardrails** section displays the following information:

- **Weighted score**: The score calculated by comparing guardrail warnings (weighted by severity and justification) against the total number of rules in the branch.
- **Warnings**: The total number of guardrail warnings in the branch.
- **Severe**: The number of severe guardrail warnings in the branch.

For more details about individual guardrail violations, select the **Warnings** tab.

Test coverage

Check how many of your branch rules are covered by tests of any kind.

The **Test coverage** section displays the following information:

- **Rules covered**: The percentage of the branch rules covered by tests.

For more details about individual rules, select the **Uncovered rules** tab.

Unit testing

Monitor unit test results and see how many of your branch rules have unit test cases written for them.

The **Unit testing** section displays the following information:

- **Rules with unit tests**: The percentage of the branch rules with unit tests.
- **Test pass rate**: The percentage of unit tests written for the branch rules that passed.

For more details, select the **Failed unit tests** and **Rules without unit tests** tabs.

Merge conflicts

Investigate any potential merge conflicts to identify problems with your branch as early as possible.

The **Merge conflicts** tab displays the following information:

- **Total**: The total number of potential merge conflicts between the branch and the main application.
- **Name**: The name of a rule that is causing a potential merge conflict.
- **Rule type**: The type of rule that is causing a potential merge conflict.
- **Base rule**: The fallback rule that is selected by the conflicted rule's resolution.

Warnings

Investigate individual guardrail violation warnings to identify problems related to your branch.

The **Warnings** tab displays the following information:

- **Total**: The total number of guardrail warnings in the branch.
- **Severe**: The number of severe guardrail warnings in the branch.
- **Moderate**: The number of moderate guardrail warnings in the branch.
- **Rule name**: The name of a rule that is causing a guardrail warning.
- **Rule type**: The type of rule that is causing the guardrail warning.

- **Ruleset**: The ruleset that contains the rule that is causing a guardrail warning.
- **Severity**: The severity status of the guardrail warning.
- **Warning type**: The type of the guardrail warning.
- **Justified**: The justification status of the guardrail warning.
- **Warning message**: The guardrail warning message.
- **Introduced by**: The user who introduced the rule that is causing the guardrail warning.

Uncovered rules

Investigate individual rules that are still uncovered by tests of any kind.

The **Uncovered rules** tab displays the following information:

- **Executable rules**: The total number of executable rules in the branch.
- **Uncovered rules**: The number of branch rules that are not covered by tests.
- **Rule name**: The name of a rule that is not covered by unit tests.
- **Rule type**: The type of rule that is not covered by unit tests.
- **Class**: The class of the rule that is not covered by unit tests.
- **Ruleset**: The ruleset that contains the rule not covered by unit tests.

Failed unit tests

Determine the causes of individual unit test failures.

The **Failed unit tests** tab displays the following information:

- **Total unit tests**: The total number of unit tests in the branch.
- **Failed unit tests**: The number of failed unit tests in the branch.
- **Test case name**: The name of a failed unit test case.
- **Rule type**: The type of rule that failed the unit test case.
- **Rule name**: The name of the rule that failed the unit test case.
- **Class**: The class of the rule that failed the unit test case.
- **Last run**: The time when the failed unit test case was last run.
- **Result**: The result of the failed unit test case.
- **Run history**: The run history of the failed unit test case.

Rules without unit tests

Investigate the details of rules that are supported by unit testing but that still do not have unit test cases written for them.

The **Rules without unit tests** tab displays the following information:

- **Supported rules**: The number of branch rules that are supported by unit tests.
- **Rules without unit tests**: The number of supported rules without unit tests.
- **Rule name**: The name of the rule without a unit test.
- **Rule type**: The type of rule without a unit test.
- **Ruleset**: The ruleset that contains the rule without a unit test.

Viewing branch quality and branch contents *(on page 304)*
Justifying an application warning *(on page 341)*
Base rules *(on page)*

Reordering branches

If you use multiple branches in your application, ensure that the order of the branches in the application rule form matches the order in which rules should be resolved for this application.

1. To open the application rule form, click the application name in the Dev Studio header, and then click **Definition**.
2. On the **Definition** tab, in the **Development branches** branches section, drag a branch to move it. When you log in to this application, the system assembles the ruleset list and branch rulesets according to the sequence in which the branches are listed.
3. **Optional:** If you are using Live UI, you can set up and manage a run-time branch so that less technical users can modify common properties of a control or a layout on rules in the branch. See Enabling run-time branching and editing *(on page)* for more information.
4. Click **Save**.

Adding development branches to your application *(on page 300)*
Branches and branch rulesets *(on page 324)*

Locking a branch

Before you merge a branch, lock the branch after development is complete so that no further modifications can be made to the rulesets within the branch. When you lock a branch, all rulesets within the branch are locked.

1. Check in any rules that are checked out. You cannot lock a branch if any rules are checked out.
2. In the navigation panel, click **App**, and then click **Branches**.
3. Right-click the branch that you want to lock and select **Lock**.
4. Enter a password to lock the branch.
5. Reenter the password to confirm it.
6. Click **Lock**.

Branches and branch rulesets *(on page 324)*

Packaging a branch

To retain a copy of your branch, you can package the contents of a branch in a .jar file. Data instances that are tagged with the rulesets that are associated with the branch and history record are included in the package.

1. In the navigation panel, click **App**, and then click **Branches**.
2. Right-click the branch that you want to package and click **Package**. A browser window displays the progress of the packaging process.
3. Click the link in the browser window to save the .jar file to a local directory. If any rules in the branch rulesets are checked out, the latest version of the checked-in rule is included in the package.

Branches and branch rulesets *(on page 324)*

Removing a branch from an application

When you remove a branch, you remove an association between a branch and an application. Removed branches and their rulesets are not available when you create or copy records in the application.

> **Note:** Removing a branch does not delete its contents from the system. To delete a branch from the system, see Deleting a branch from the system *(on page 308)*.

1. In the navigation panel, click **App**, and then click **Branches**.
2. Right-click the branch that you want to remove from your application and click **Remove from application**.

Branches and branch rulesets *(on page 324)*

Deleting a branch from the system

You can delete an entire branch and its contents from the system. You can also package the branch in a JAR file and download it before deleting it.

1. Check in any records that are checked out.
2. In the navigation panel, click **App**, and then click **Branches**.
3. Click the branch that you want to delete.
4. Click **App > Delete from system**.

5. **Optional:** If there are rulesets in the branch, and you want to save them in a JAR file that you can download, select the **Save branch contents in a JAR fie for download** check box.
6. Click **Delete** in the confirmation dialog box.
7. Click **Done**.

Branches and branch rulesets *(on page 324)*

Branch reviews

To increase the quality of your application, you can create reviews of branch contents to improve branch quality by, for example, ensuring that the rules are guardrail-compliant. You can assign branch reviews to other users, use Pulse to collaborate on reviews, and close reviews after you have addressed any issues.

If you have a Default email account configured, you are notified when reviews are assigned to you and when reviews are closed. For more information, see Configuring outbound email in Dev Studio *(on page)*.

If Pulse notifications are configured, you can receive emails when a new comment is added or when a reply is posted to a new comment. For more information, see Configuring Pulse email notifications *(on page)*.

You can override the following when rules to allow merges when branches are unlocked or not reviewed:

- pyAllowMergeWhenNotReviewed: Override to disallow merges when branches have not been reviewed.
- pyAllowMergeWhenUnlocked: Override to disallow merges when branches are not locked.

In addition, if you use branches in a continuous integration (CI) environment, you can configure a pipeline on the CI tool to perform a branch review.

Branches and branch rulesets *(on page 324)*

Creating a branch review

You can create a review for a branch so that you and other review team members can collaborate to improve the branch quality before you merge the branch. Only one branch review can be open at a time.

If you have a Default email account configured, you are notified when reviews are assigned to you and when reviews are closed. For more information, see Configuring outbound email in Dev Studio *(on page)*.

Complete the following steps:

1. As a best practice, check in any rules that are checked out.
2. In the navigation panel, click **App**, and then click **Branches**.
3. Click the branch for which you want to create the review.
4. On the **Content** tab, click **Create review**.
5. In the **Create Review** dialog box, in the **Objective** field, enter the purpose of the review.
6. In the **Due** field, specify the date on which the review should be completed.
7. In the **User ID** field, press the Down Arrow and select the user ID of a reviewer.
8. Add more reviewers by clicking the **Plus** sign and repeating the previous step.
9. Click **Create**.

Branch reviews *(on page 309)*
Branches and branch rulesets *(on page 324)*

Modifying a branch review

For open branch reviews, you can modify the list of users who are assigned to the review.

1. In the navigation panel, click **App**, and then click **Branches**.
2. Click the branch for which you want to modify the branch review.
3. On the **Content** tab, in the **Current review** section, click the review to open it.
4. Select **Actions > Manage review team**.
5. Add or remove reviewers, as appropriate.
6. Click **OK**.

Branch reviews *(on page 309)*
Branches and branch rulesets *(on page 324)*

Deleting a branch review

You can delete both open and closed branch reviews.

1. In the navigation panel, click **App**, and then click **Branches**.
2. Click the branch for which you want to delete the review.
3. On the **Content** tab, in the **Current reviews** section, click the review to open it.
4. Click **Actions > Delete**.

Branch reviews *(on page 309)*
Branches and branch rulesets *(on page 324)*

Closing a branch review

You can close a branch review after all comments have been addressed.

If you have a Default email account configured, you are notified when reviews are assigned to you and when reviews are closed. For more information, see Configuring outbound email in Dev Studio *(on page)*.

Complete the following steps:

1. In the navigation panel, click **App**, and then click **Branches**.
2. Click the branch for which you want to close the review.
3. On the **Content** tab, in the **Current review** section, click the review to open it.
4. Click **Actions > Close**.
5. A dialog box is displayed if branches in the rule are checked out. Click **Yes** to close the review.
6. Click **Submit**.

Branch reviews *(on page 309)*
Branches and branch rulesets *(on page 324)*

Reopening a branch review

You can reopen a branch review after the review was closed, for example, if a reviewer inadvertently closed it.

1. In the navigation panel, click **App**, and then click **Branches**.
2. Click the branch for which you want to reopen the branch review.
3. On the **Content** tab, in the **Previous reviews** section, click the review to open it.
4. Select **Actions > Reopen**.
5. In the dialog box, click **Yes** to confirm that you want to reopen the review.

Branch reviews *(on page 309)*
Branches and branch rulesets *(on page 324)*

Reviewing branches

If you are a branch reviewer, you can open reviews and inspect the rules within the branch improve the quality of the branch, for example, before you merge its contents. You can also communicate with other users in Pulse to work collaboratively on branch reviews.

If you have a Default email account configured, you are notified when a review is assigned to you and when reviews are closed. For more information, see Configuring outbound email in Dev Studio *(on page)*. If

Pulse notifications are configured, you can receive emails when a new comment is added or when a reply is posted to a new comment. For more information, see Configuring Pulse email notifications *(on page)*. Complete the following steps:

1. In the navigation panel, click **App**, and then click **Branches**.
2. Click the branch that has the review that you want to open.
3. On the **Content** tab, in the **Current review section**, click the review to open it. You can perform the following actions:
    - Filter the list of rules in the branch by rule type using the drop-down list.
    - Click **General comments** to post and view Pulse comments about the branch.
    - Click a rule to collaborate with other users in Pulse about the specific rule. You can open the rule form on this page.

Branch reviews *(on page 309)*
Branches and branch rulesets *(on page 324)*

Merging branches into target rulesets

When rule development in a branch is complete, make the changes available by merging branches into a target ruleset of the development application. As a result, your team provides required solutions in a timely and efficient way, without the risk of overriding or losing work.

As a best practice, the system administrator creates a new ruleset version for the base ruleset, independent of the wizard. Individual teams develop rules in their specific branches, and then merge those branches into the existing base ruleset version that the administrator provides. As a result, administrators have better control over versions.

1. **Optional:** To save time and resolve any potential issues before merging rulesets, check whether any conflicts might occur.
   For more information, see Viewing branch information *(on page 304)*.
2. **Optional:** To ensure that branches include only changes that you want to merge, lock a branch after development is complete.
   For more information, see Locking a branch *(on page 307)*.
3. In the navigation pane of Dev Studio, click **App**.
4. In the App Explorer, click the **Branches** tab.
5. Select branches that you want to merge:

Choices	Actions
Merge a single branch	a. Right-click the branch, and then click **Merge**. b. In the **Merge branch** dialog box, click **Proceed**.
Merge multiple branches	a. Right-click your application name, and then click **Merge multiple branches**. b. In the **Select Branches to Merge** dialog box, select the branches that you want to use. c. Click **OK**.

6. If conflicts or warnings occur, on the **Merge Branches** tab, review information about any issues. You must resolve conflicts before you can merge branches. For more information, see Conflicts and warnings in the Merge Branches wizard *(on page 313)*.
7. In the **Target ruleset** section, in the list of versions, select the base ruleset version into which you want to merge rulesets:
    - To create a new ruleset version in the base ruleset during the merge, select **Create new version**.
    - To reuse an existing ruleset, select the ruleset version.
8. Click **Merge**.

    When a merge is complete, you can view the details about the outcome, the number of merged rules, and the source and target rulesets.

Branches and branch rulesets *(on page 324)*

Conflicts and warnings in the Merge Branches wizard

The Conflicts and Warnings window displays an expandable list of rules that the Merge Branches wizard has identified as having conflicts, warnings, or both. To open this window, click the displayed number of conflicts and warnings on the main wizard screen.

Note: The displayed report of conflicts and warnings depends on the target ruleset version selected for the merge. For example, merging a rule into target ruleset version 01-01-01 might display no conflicts and one warning, while merging that rule into target ruleset version 01-01-02

might display one conflict and no warnings. You must select the target ruleset version for the merge to get an accurate report of the potential conflicts and warnings.

When there are resolved conflicts, the report refreshes to indicate how many are resolved.

Merging branches into target rulesets *(on page 312)*

Conflicts in the Merge Branches wizard

Before the wizard proceeds with the merge, you must indicate in this window that you have resolved all conflicts reported by the wizard. Conflicts occur for the following situations:

Conflict situation	Example of how this occurs
The rule in a lower base ruleset version has been recently updated.	Base ruleset XYZ has two locked ruleset versions, 01-01-01 and 01-01-02. TeamA branches XYZ into their Story-A branch, and TeamB branches XYZ into their BugFix-1 branch. TeamB's work is a bug fix for version 01-01-01, and they complete their work and merge their BugFix-1 branch with the changed rules for base ruleset XYZ into 01-01-01. Then TeamA starts the wizard to merge their Story-A branch by creating a new version for base ruleset XYZ. The wizard reports a conflict because the rule was updated in ruleset version 01-01-01 (a lower version) by TeamB prior to TeamA's merge.
The rule in the selected target (base) ruleset version has been recently updated.	Base ruleset XYZ has two locked ruleset versions, 01-01-01 and 01-01-02. TeamA branches XYZ into their Story-A branch, and TeamB branches XYZ into their Story-B branch. TeamB completes their work and merges their Story-B branch into 01-01-02. Then TeamA starts the wizard to merge their Story-A branch into 01-01-02 also. The wizard reports a conflict because the rule was updated in the same ruleset version 01-01-02 (TeamA's selected target version) by TeamB prior to TeamA's merge.
The same rule exists in multiple branches that are merged using the wizard to merge multiple branches.	Base ruleset XYZ has one version, 01-01-01. TeamA has branches Story-A and Story-B, which have the same rule. Story-A is the topmost branch in the hierarchy. Story-A and Story-B are being merged using multibranch merge. The wizard reports a conflict for Story-A branch because Story-B has the same rule and will be merged before Story-A. To resolve this conflict, move all your changes for this rule from the Story-B branch to the rule in Story-A.

Resolving conflicts in the Merge Branches wizard *(on page 315)*
Conflicts and warnings in the Merge Branches wizard *(on page 313)*
Merging branches into target rulesets *(on page 312)*

Resolving conflicts in the Merge Branches wizard

To resolve the conflicts for a rule, complete the following steps:

1. Examine the conflicts reported for the rule in the **Conflicts and Warnings** window.
2. Click **Compare** next to the first conflict to examine the differences between your branch copy of the rule and the one in the base ruleset. Alternatively, click the ruleset name to open the base rule and directly examine it.
3. Review and resolve all of the differences between the base rule and your branch rule.
    - If the changes are minor (such as a typo), open your branch rule and apply the same changes to it so that it matches the base rule.
    - If the changes are more complex, or if they conflict with the choices or logic in your branch rule, contact the individual who modified the base rule and negotiate what the final set of changes should be and the testing procedure for testing them. After reaching a mutual decision, make the agreed-upon changes in your branch rule and test.
4. When changes to your branch rule are completed and tested, select the Mark as Resolved check box to indicate you have resolved the conflict.

> After all conflicts reported in the **Conflicts and Warnings** window are marked resolved, you can click **OK** to close the window and return to the Merge Branches wizard to continue with the merge process.
>
> In the main screen of the wizard, you can see if all conflicts are resolved by verifying if the number of Resolved conflicts matches the number of reported conflicts for all of the branch rulesets.

Conflicts in the Merge Branches wizard *(on page 314)*
Comparing rule versions *(on page)*

Addressing warnings in the Merge Branches wizard

Warnings are primarily informational and are provided to make you aware of how merging your branch might impact other teams or might not behave exactly as you think it will.

Warning situation	Example of how this occurs	Recommended steps
A rule with the same keys as the branch rule exists in a higher ruleset version in the target ruleset.	You are merging the branch rule into target ruleset version 01-01-01, and the corresponding base rule also exists in target ruleset version 01-01-02.	1. Click **Compare** next to the warning to examine the differences between your branch copy of the rule and the one in the other base ruleset version. Alternatively, click the ruleset name next to the caution icon to open the other rule and directly examine it. 2. Review all differences between the rules, and use the Related Rules display to assess the impact of the differences. If the features in the rule in the higher ruleset version does not apply to your target ruleset version, and your changes in the lower target are not present in, or will not impact, the one in the higher ruleset version, it is likely safe to proceed with your merge. Merge into the lower ruleset version. The changes in the rule will work at that ruleset version level. Those applications that use the higher ruleset versions might not see your merged changes. For changes in your branch rule that you determine need to be ported to the rule in the higher ruleset version, define a new development application and matching branch using the higher ruleset version, and copy the base rule from the higher ruleset version into that branch ruleset. Make your changes in that branch, test, and merge to the higher ruleset version.

Warning situation	Example of how this occurs	Recommended steps
The corresponding base rule exists in another branch ruleset in the system.	Another team has branched the same base ruleset that you are merging, and has a copy of the base rule in their branch ruleset.	1. Click **Compare** next to the warning to examine the differences between your branch copy of the rule and the one in the other branch. Alternatively, click the ruleset name next to the caution icon to open the other rule and directly examine it. 2. Review all differences between the rules, and use the Related Rules display to assess the impact of the differences. 3. Contact the owner of the other branch, or the author of the other branched rule, and describe the changes you made to the rule in your branch and how they can test for the changes. Best practice is they would validate your changes after your merge your branch and they obtain your changes and include them in their branch. 4. Complete the merge of your branch.
The corresponding base rule exists in another branch ruleset, which is branched from a different ruleset than your branch ruleset.	In this situation, the rule exists in another branch ruleset which will later be merged into a different ruleset than the one the corresponding base rule is in. A team has branched a base ruleset that is different than the one you are merging, and has copied the base rule that corresponds to your branch rule into their branch ruleset.	1. Click **Compare** next to the warning to examine the differences between your branch copy of the rule and the one in the other branch. Alternatively, click the ruleset name next to the caution icon to open the other rule and directly examine it. 2. Review all differences between the rules, and use the Related Rules display to assess the impact of the differences. 3. Contact the owner of the other branch, or the author of the other branched rule, and discuss why a rule with the same name exists in two different rulesets, and determine whether there are reasons for both to exist or whether one should be removed.

Warning situation	Example of how this occurs	Recommended steps
		• If the base rulesets are for different purposes and will not be run concurrently, then continue with your merge. • If the teams agree the rulesets are to be combined, negotiate an appropriate conclusion and perform agreed-upon changes. Depending on the outcome, complete your merge if appropriate.
A rule with the same keys has been updated somewhere else in the system.	A rule exists in the system that has the same keys as the rule you are merging, and the other rule has been updated.	1. Click **Compare** next to the warning to examine the differences between your branch copy of the rule and other rule. Alternatively, click the ruleset name next to the caution icon to open the other rule and directly examine it. 2. Review all differences between the rules, and use the Related Rules display to assess the impact of the differences. 3. Contact the owner of the other ruleset, or the author of the other rule, and discuss why a rule with the same name exists in two different rulesets, and determine whether there are reasons for both to exist or whether one should be removed. • If the base rulesets are for different purposes and will not be run concurrently, then continue with your merge. • If the teams agree the rulesets are to be combined, negotiate an appropriate conclusion and perform

Warning situation	Example of how this occurs	Recommended steps
		agreed-upon changes. Depending on the outcome, complete your merge if appropriate.

Conflicts and warnings in the Merge Branches wizard *(on page 313)*

Branch merge process customization

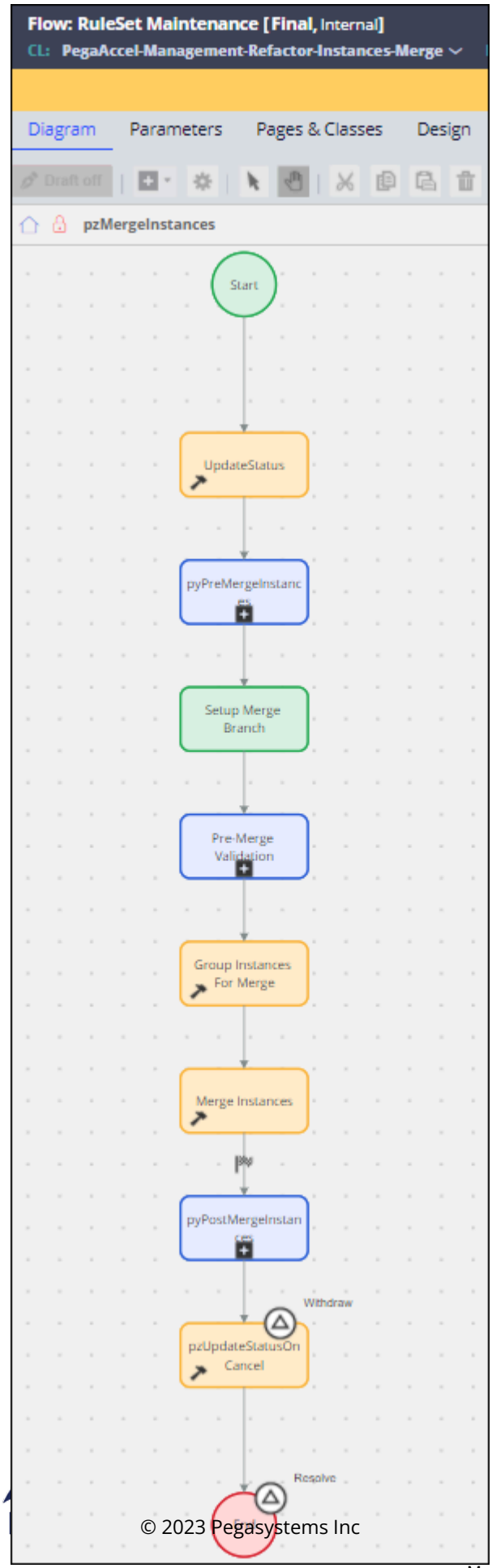
You can enhance the merge process for your development team in various ways to comply with your organization's policies and procedures. By using extension points, you customize the standard branch merge process. For example, you can add postprocessing behavior that sends an email notification to the project lead with the status of the merge process.

The merge process and its extension points

When development in a branch is complete and stable, you can use the Merge Branches wizard to merge the changes in one or more branches to the base rulesets.

> **Tip:** As a best practice, use the customizable and model-driven pipelines of Deployment Manager to validate a branch before a merge. For more information, see Deployment Manager for model-driven DevOps *(on page)* and Start a Deployment Manager deployment in a distributed, branch-based environment by using the Merge Branches wizard.

The merge process is defined by the standard flow rule *pzMergeInstances* in the *PegaAccel-Management-Refactor-Instances-Merge* class.



pzMergeInstances process

For preprocessing and postprocessing behavior, the process has two extension points (as subprocesses), to which you can add custom steps.

By saving a copy of the standard rule and customizing your copy, you can override the generic behavior. For more information about making a copy of the standard rule, see the Creating a class to organize your customizations *(on page 322)* section.

The standard merge process provides extension points, at which you can provide new behavior:

pyPreMergeInstances (subprocess)

To add preprocessing behavior. By customizing a copy of *pyPreMergeInstances*, you can provide processing that occurs before the Merge Branches wizard starts, such as verifying that a user's operator ID is performing the merge instead of a generic account ID.

pyPostMergeInstances (subprocess)

To add postprocessing behavior. By customizing a copy of *pyPostMergeInstances*, you can provide processing that occurs after the Merge Branches wizard completes, such as sending an email notifying the team about the status of the merge.

pyValidateMergeOptions (activity)

To customize validation on the inputs into the user interface of the Merge Branches wizard.

You can also modify the UI for the merge process by overriding the following sections and when rules in the Merge Branches wizard:

pyMergeInstancesRuleSetListFooter (section)

To customize the pre-merge screen by adding fields or information to the bottom of the merge wizard.

pyConfirmContent (section)

To customize the confirmation screen. You can modify this section to show a message to the user after the user submits the merge or when a user returns the merge work object to review.

pyRestrictedTargetOptions (when rule)

To apply custom logic for merge options, such as Hide Merge Options, which determines which target rulesets and passwords are visible.

pyMergeButtonDisabled (when rule)

To apply custom logic for when the merge is enabled on the UI, such as the Disable Merge Button option.

Creating a class to organize your customizations

The recommended action before extending the merge wizard is to create a class to store all your extensions. By creating a class to add your customizations, you can expose different merge behaviors and enable users to fall back to an existing behavior.

> **Tip:** You can create a new component to expose your merge wizard customizations, which provides a way to package and share your customization between different applications. For more information, see Components *(on page 87)*.

To safely organize your customizations in a class in which you can override the extension points, ensure that:

- The class is a child of the *PegaAccel-Management-Refactor-Instances-Merge* class.
- You update the *pySetCustomMergeExperience* activity to conditionally set the class of the primary page by using the page-set-class method.
- If multiple merge wizard implementations are required, include this logic in the highest ruleset version available.

For more training materials, see the Merging a development branch in 8.6 challenge on Pega Academy.

For an example of the enhancements that you can make in the branch merge process by customizing this process, see Modifying postprocessing behavior in the branch merge process *(on page)*.

Related information

Branches and branch rulesets *(on page 324)*
Merging branches into target rulesets *(on page 312)*

Branches and unlocked rulesets

Based on your development model, you can build your application through branched development or by using unlocked rulesets. Before you select an approach, analyze which option suits your requirements best.

Working in branches

Branches allow developers to work in separated areas, which helps avoid the problem of overwriting someone else's changes and minimizes the occurrence of conflicts and errors. When a developer or developers make changes in a branch, these changes have no effect on other branch rulesets or base

application rulesets. As a result, developers can work on multiple features in parallel, even if the features use the same base rules. This saves time and accelerates application development.

As a best practice, to start branched development, create team applications by copying your base application. When every team has a separate application, implementing and testing changes is more convenient compared to a scenario in which multiple teams work on the same ruleset in multiple branches. For more information, see Creating a development application *(on page 299)*.

When the development is complete, teams merge changes to the base application. By merging, developers can select only those changes that they intend to implement in the base application, and discard other rules. Merging provides greater control over changes that developers implement in the base application. For more information, see Merging branches into target rulesets *(on page 312)*.

Working in unlocked rulesets

Unlocked rulesets are available to multiple developers at the same time. When developers work in an unlocked ruleset, the implemented changes immediately affect other developers who use the affected rules in the same ruleset. This approach might lead to work overwrites, errors, and unexpected results. A solution to minimize the risk of overwrites is rule checkout. When a developer checks out a rule, the rule is unavailable to other developers for editing. However, after implementing their changes, a developer checks in a rule, and only then do the changes affect other rules in the ruleset. For more information, see Checking out a rule *(on page)* and Checking in a rule *(on page)*.

Application development for larger projects

The most common scenario is application development that involves multiple teams working on multiple features. In such scenarios, branched development is the most suitable choice. Developers can implement changes simultaneously, because changes in one branch have no effect on other branch rulesets, which minimizes errors and work overwrites. Parallel development of multiple features saves time and helps developers create software faster.

Branched development involves merging, which developers can use to select rules to include in the base application, and remove unnecessary rules. Also, validation of new features in an environment that does not include the ongoing work of other developers is more effective and reliable. With branched development and merging, the process of reverting changes is easier than when teams work in unlocked rulesets. Merging is also applicable in situations in which changes require approval, for example from a manager or a team lead.

Rule checkout during branched development is a best practice. By checking out rules, developers avoid situations in which two or more team members edit the same rule. Rule checkout also creates a rule history, which developers can use to analyze changes in the rule, or to revert the rule to a specific state.

Branched development supports continuous integration and continuous delivery (CI/CD) pipelines, so that developers deliver well-tested software through an agile and iterative development process. For more information about CI/CD pipelines, see Understanding continuous integration *(on page)* and Understanding continuous delivery *(on page)*.

Application development for a single developer and small-scale teams

When only one person works on an application, or the project involves a small and communicative team of up to four or five members, branched development and rule checkout are not required, because the implemented changes do not affect other developers. For a small development team, communicating the changes that the developers intend to implement is crucial to avoid work overwrites and errors. This business scenario is untypical and usually applies in situations when a developer builds an application for learning purposes, or creates an application as a citizen developer. Typically, citizen developers have limited technical background and focus on more granular applications. Similar scenarios include the very early stages of application development process, when a client and a software company define requirements for a new application.

> **Note:** Rule checkout creates a history of all changes that occur in a rule. Developers can use this history to analyze the edits in a rule, or to roll back the rule to a specific state. As a best practice, check out the rules that you want to edit every time you intend to edit them.

Working in unlocked rulesets is a good approach for development projects that include very limited CI/CD pipelines. As a best practice, if you intent to move an application to a production environment, implement branched development first.

Branches and branch rulesets

Branches help you manage work in development environments in which multiple teams contribute to a single application. You use branches to develop software simultaneously in a version-controlled environment. For example, a team can develop a feature in one branch while another team develops another feature in a different branch, even if the teams share the same rulesets.

When you use branches, different development teams can work without interfering with changes that other teams make in an application, even if multiple teams share one base ruleset. For example, Team A can develop service-level agreements for a case type while Team B fixes UI bugs in the application. When you provide separate branches for both teams, you minimize the risk of errors and work overwrites that might arise when both teams work on the same ruleset simultaneously.

To organize rules better during your development process, you can categorize rules into branch rulesets. Branch rulesets have the following characteristics:

- Branch rulesets are branched from rulesets in your application.
- Branch rulesets contain rules that are in active development in an associated branch.
- Branch rulesets have a naming convention that includes a ruleset ID, the word `Branch`, and a branch name, for example, `SLAS_Branch_TeamABranch`, where SLAS is a ruleset ID, `Branch` indicates a branch ruleset, and `TeamABranch` is a branch name.

Branching is especially useful in large development projects when multiple teams work simultaneously on the rules in an application, and members of one team view each others work, but isolate their development changes from other teams. Consequently, rule changes that one team makes do not affect the other teams until the changes are stable, conflicts are resolved, and the approval is granted to make the new improvements available to the entire development project team.

Branched development includes the following main tasks:

1. You create a development application that your team can use to develop new features. You create a development application by copying an existing application to ensure that the names of classes and rulesets remain unchanged.

   For more information, see Creating a development application *(on page 299)*.

2. You add development branches to the development application. Teams use branches to develop features independently.

   For more information, see Adding development branches to your application *(on page 300)*.

3. As a best practice, you create a branch review to ensure that rules in your branches are guardrail-compliant and meet your business objectives.

   For more information, see Branch reviews *(on page 309)*.

4. You make new features available in your application by merging changes from branches into a target ruleset. You now can resolve conflicts and address warnings that might arise among different branched versions of the same ruleset.

   For more information, see Merging branches into target rulesets *(on page 312)*.

Branches and unlocked rulesets

An alternative to branched development is working in unlocked rulesets. Unlocked rulesets are suitable for projects that typically involve one team, and team members immediately need to view changes that any team members make. When a developer saves a change in an unlocked ruleset, the change immediately affects the work of other developers on that application.

During branched application development, a change that a developer makes in a branch ruleset affects the work that other developers do on the same development application only after merging changes into a target ruleset.

For more information, see Branches and unlocked rulesets *(on page 322)*.

Adding development branches to your application *(on page 300)*
Understanding continuous integration and delivery pipelines *(on page)*
Troubleshooting tools and techniques *(on page 340)*

Delivering application documentation

Deliver application documentation to stakeholders for transparent application development, and to end users to ensure that they have enough information to independently use your software.

Explore the following topics to learn more about application documentation:

Documenting your application

Create documentation to engage stakeholders and give guidance to the end users and developers who interact with your application. You can generate a physical document or provide embedded information, based on your audience.

Use the following techniques to document your application:

Creating a report *(on page)*
Creating advanced reports *(on page)*

Creating project documents for stakeholders

Create an application overview to educate stakeholders about the features in your application and to review the development status of your project. To provide most relevant information, customize the application overview by including only specific chapters.

1. In the navigation pane of App Studio, click **Overview**.
2. In the **Application documents** section, click **Export**.

3. **Optional:** To modify the outline, in the **Generate overview document** window, click **Show outline**, and then:
    - To include a chapter in the outline, select the check box next to the name of the chapter.
    - To change the order of the items in the outline, move the chapters.
4. Click **Generate**, and then perform one of the following actions:
    - To save the document locally, click **Download**.
    - To change the content of the document, click **Regenerate document**, and then perform steps 3 *(on page 327)* and 4 *(on page 327)*.

Adopting feature-driven development *(on page)*

Exporting a data model into a document

Share information about a data model in your application with stakeholders and developers so that they can quickly check if the data model includes all relevant information.

For example, if the aim of your application is to investigate various loan requests, you can quickly check the information that is collected for every case type, such as *Mortgage request* or *Car loan request*.

1. In the navigation pane of App Studio, click **Overview**.
2. Click **Actions > Export data model**.
3. In the **Export data model** dialog box, select a format for the file.
4. Click **Generate**.

Adopting feature-driven development *(on page)*

Contents of application overview and data model documents

Provide stakeholders with high level information about your application by generating application overview and data model documentation. Customize the document to include only relevant information.

You can include the following details in the application overview:

- Case types
    - Case type details
        - Graphic representation of stages in case types
        - Case wide optional processes
        - Case wide optional actions
    - Stages in case type details
        - Statuses associated with each stage
        - Processes associated with each stage

        - Stage wide optional processes
        - Stage wide optional actions
        - Conditions for skipping a stage
    - Data types
        - Systems of records
        - References between data types and case types
    - Roles
        - Number of operators for each role
        - Web channels available for each role
        - Pages available for each role for each web channel
    - Channels and interfaces
        - Roles that use a specific channel as a default channel
        - Roles with access to a specific channel
    - Features that are defined in the application

A data model document contains the following information that you can export to `.xml` and `.pdf` files:

- All data types and case types with descriptions
- References between data type and case types
- System of records for each data type
- Fields for each data type and case type with the following details:
    - Label
    - ID
    - Type
    - Field group type, if applicable
    - List items, if applicable
    - **Is calculated** field
    - Calculation details, if calculated
    - Description

Related information

Exporting a data model into a document *(on page 327)*

Creating project documents for stakeholders in Dev Studio

Create project documents to educate stakeholders about the features in your application and to review the development status of your project.

1. **Optional:** Align the look and feel of your document with your brand.
    - Changing the style of a project document *(on page 329)*
    - Changing the structure of a project document *(on page 329)*
2. In the header of Dev Studio, click **Configure > Application > Tools > Document**.
3. In the **Select a document type** section, choose a document type:
    - To generate a document that provides high-level information, such as the application description, case types, data model, and roles, click **Application overview**.
    - To generate a document that provides detailed information on application data, click **Data model**.
    - To generate a document that provides traceability from unfinished features to open work items, and a brief product overview, click **Gap analysis**.
4. To control which features you document, select a filter option from the **Features to include** list.
5. To exclude a chapter from the document, clear the check box next to the chapter name.
6. Click **Generate document**.
7. Click **Document generation completed**.
8. Click the document name to open the document in Microsoft Word.

Creating a guide for application developers and administrators *(on page 330)*
Creating guidance for developers *(on page)*

Changing the style of a project document

Use HTML styles to align the look and feel of project documents with your brand.

1. Use search to find and open the standard `@baseclass.pyRTADocStyle` HTML rule.
2. Click **Save as > Specialize by class or ruleset**.
3. In the **Add to ruleset** field, select a ruleset and version in your application.
4. Click **Create and open**.
5. On the **HTML** tab, enter your custom styles in the `<style>` element.
   For example, you can change the font, font size, or colors.
6. Click **Save**.

Changing the structure of a project document

Use a template to change align the structure of project documents with your brand.

1. Use search to find and open the standard **`pyNextGenDocTemplate`** file.
2. Click **Download file**.
3. In Microsoft Word, change the structure of the template.

For example, you can insert a page or add a header for your company logo.

4. Save the `pyNextGenDocTemplate.docx` file to your local machine.
5. On the File form, click **Upload file**.
6. Click **Choose File**, and then navigate to your local file.
7. Click **Open**.
8. Click **Upload file**.
9. Click **Save**.

Creating a guide for application developers and administrators

Create an application guide to direct users through a set of tasks. By providing instructional information, you can help users learn about the functionality that your application provides, configure complex settings, or extend features with limited assistance.

You add structure to an application guide by organizing related tasks into chapters. Each task can include instructional text and a link to supporting information, such as an external document or a landing page.

For example, you can create an application guide for migrating locally stored information to a database server. In the chapter for configuring your database, you can add tasks for creating tables, views, and users. In the task for creating users, you can supplement the instructions with a document that lists database privileges.

Creating project documents for stakeholders in Dev Studio *(on page 328)*
Creating guidance for developers *(on page)*

Creating an application guide

Create an application guide to direct users through a set of tasks. By providing instructional information, you can help users learn about the functionality that your application provides, configure complex settings, or extend features with limited assistance.

For more information, see Creating a guide for application developers and administrators *(on page 330)*

Adding a chapter to an application guide

To organize related tasks, add a chapter to an application guide.

Tip: To support translation, use a paragraph to define your chapter description. For more information, see Creating a paragraph *(on page)*.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, click **Add Chapter**.
3. In the **Chapter name** field, enter text that describes the purpose of this chapter.
4. Provide a chapter description.
    - To reference a paragraph:
        - In the **Description** list, select **Paragraph**.
        - In the **Paragraph** field, press the Down Arrow key, and then select the paragraph that describes the chapter.
    - To manually enter a description:
        - In the **Description** list, select **Rich Text**.
        - In the rich text editor, enter and format your chapter description.
5. Click **Submit**.
6. Click **Save**.

Documenting your application *(on page 326)*

Adding a task to a chapter in an application guide

To provide step-by-step information in an application guide, add a task to a chapter.

> **Tip:** To support translation, use a paragraph to define your task description. For more information, see Creating a paragraph *(on page)*.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, expand a chapter in the **Chapter name** column.
3. Click **Add task**.
4. In the **Task name** field, enter text that describes the outcome of the task.
5. Provide instructions to complete the task in the task description.
    - To reference a paragraph:
        - In the **Description** list, select **Paragraph**.
        - In the **Paragraph** field, press the Down Arrow key, and then select the paragraph that describes the task.

- To manually enter a description:
    - In the **Description** list, select **Rich Text**.
    - In the rich text editor, enter and format your task description.

6. Click **Submit**.
7. Click **Save**.

Creating project documents for stakeholders in Dev Studio *(on page 328)*

Adding supporting information to an application guide

Add supporting information to an application guide to direct users to the resources that relate to their current task.

Use the following techniques to add supporting information to an application guide:

Documenting your application *(on page 326)*

Linking a URL to an application guide

Link a URL to your application guide to integrate a task with an external resource.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, expand a chapter in the **Chapter name** column.
3. Click the **Gear** icon next to a task name.
4. In the **Action type** list, select **Open URL**.
5. In the **Display text** field, enter a label for the link that users click in the task.
6. In the **URL** field, enter the web address of your supporting information.
7. Click **Submit**.
8. Click **Save**.

Documenting your application *(on page 326)*

Linking a rule or data instance to an application guide

Link a rule or data instance to your application guide give users direct access to internal resources.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, expand a chapter in the **Chapter name** column.
3. Click the **Gear** icon next to a task name.

4. In the **Action type** list, select **Open rule/data**.
5. In the **Display text** field, enter a label for the link that users click in the task.
6. In the **Rule Type** field, enter the class that defines the type of rule that you are linking.

   For example, all data transforms are instances of the `Rule-Obj-Model` class.
7. In the **Instance Name** field, use one of the following methods to select your rule.

   - Press the Down Arrow key, and then select the fully qualified name of a rule.
   - Enter the name of a data instance.
8. Click **Submit**.
9. Click **Save**.

Creating a guide for application developers and administrators *(on page 330)*

Calling a guided tour from an application guide

Call a guided tour from your application guide to make a task more interactive.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, expand a chapter in the **Chapter name** column.
3. Click the **Gear** icon next to a task name.
4. In the **Action type** list, select **Launch tour**.
5. In the **Display text** field, enter a label for the link that users click in the task.
6. In the **Action to take** field, select **Start tour**.
7. In the **Tour applies to** field, press the Down Arrow key, and then select the class to which the guided tour applies.
8. In the **Tour name** field, press the Down Arrow key, and then select the name of a guided tour.
9. **Optional:** Change the look and feel of the tour stops in the guided tour.
   a. Select the **Override default tour stop container template** check box.
   b. In the **Template name** field, press the Down Arrow key and then select the name of a section that defines the layout and controls for each tour stop.
10. Click **Submit**.
11. Click **Save**.

Creating a guide for application developers and administrators *(on page 330)*

Calling a wizard from an application guide

Call a wizard from your application guide to provide well-defined steps for a complex task.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, expand a chapter in the **Chapter name** column.
3. Click the **Gear** icon next to a task name.
4. In the **Action type** list, select **Run wizard**.
5. In the **Display text** field, enter a label for the link that users click in the task.
6. In the **Wizard** field, press the Down Arrow key, and then select the class of your wizard.
7. Click **Submit**.
8. Click **Save**.

Creating a guide for application developers and administrators *(on page 330)*

Linking a help topic to an application guide

Link a standard help topic to your application guide to integrate a task with supporting information.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, expand a chapter in the **Chapter name** column.
3. Click the **Gear** icon next to a task name.
4. In the **Action type** list, select **Open help URL**.
5. In the **Display text** field, enter a label for the link that users click in the task.
6. In the **Relative path** field, enter the file path to the help topic, relative to the base path that you provide on the System Settings landing page.
7. Click **Submit**.
8. Click **Save**.

Creating a guide for application developers and administrators *(on page 330)*

Linking a landing page to an application guide

Link a landing page to your application guide to give users direct access to internal tools.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, expand a chapter in the **Chapter name** column.
3. Click the **Gear** icon next to a task name.
4. Add a link to the task.

> **Tip:** For an example configuration of the following fields, open the `Pega-Landing.pyMainMenu` navigation rule by searching for it or by using the Application Explorer.

a. In the **Action type** list, select **Open landing page**.

b. In the **Display text** field, enter a label for the link that users click in the task.

c. In the **Action** field, select **Display**.

d. In the **Name** field, enter a label for the tab that displays the landing page.

e. In the **Class** field, press the Down Arrow key, and then select a class that defines harnesses.

f. In the **Harness Name** field, select the harness that the landing page displays.

g. Decide which tab has focus when users open the landing page.

h. Enter the tab name in one of the following fields, based on the tab's position in the landing page hierarchy:

- **Level A**
- **Level B**
- **Level C**

5. Click **Submit**.
6. Click **Save**.

Creating a guide for application developers and administrators *(on page 330)*
Adding action sets to a control *(on page)*
Control form - Actions tab *(on page)*
Available UI actions *(on page)*

Linking a custom section to an application guide

In addition to other types of supporting information, such as URLs and help topics, you can link a section to your application guide to change the presentation or available actions in a task.

1. Open an application guide by searching for it or by using the Application Explorer.
2. On the **Definition** tab, expand a chapter in the **Chapter name** column.
3. Click the **Gear** icon next to a task name.
4. In the **Action type** list, select **Custom section**.
5. In the **Action value** field, press the Down Arrow key, and then select a section in the `Embed-Guide-Task` class path that is not in a `Pega-` ruleset.

> **Tip:** To support users in App Studio, ensure that the first action in your section calls the `pega.desktop.closeAllDocuments` API.

6. Click **Submit**.
7. Click **Save**.

Creating a guide for application developers and administrators *(on page 330)*
Adding action sets to a control *(on page)*
Control form - Actions tab *(on page)*
Available UI actions *(on page)*

Adding an application guide to a portal

Add your application guide to one or more portals so that users can access it.

1. In the Dev Studio header, click the name of your current application, and then click **Definition** to open the Application form.
2. Click the **Documentation** tab.
3. If there are no guides defined in the Application Guides section, click **+ Add guide**.
4. In the field that is displayed, press the Down Arrow key, and then select the name of your application guide.
5. In the **Available in** column, click **portals**.
6. Click **+ Add portal**.
7. In the field that is displayed, press the Down Arrow key, and then select the name of a portal.
8. Click **Submit**.
9. On the Application form, click **Save**.

Creating a guide for application developers and administrators *(on page 330)*

Describing features to end users

You can use a guided tour to showcase application features in real time. By providing information in a sequence of pop-up windows, you can help users discover functionality that is available on their current screen.

For example, you can create a tour that gives example use cases for the different tools in a web channel.

Guided tours *(on page)*
Documenting your application *(on page 326)*

Creating content for a guided tour

You can use sections to create the content for each pop-up window, or tour stop, in a guided tour. By using a common format, you can associate your content with application features more quickly.

1. Create a section that contains the text for the tour stop.

   You can add action controls to the section, however, all elements must be read-only. For more information, see Creating sections *(on page)*.

2. Find a section in a form or portal, that contains the application feature for your tour stop.
   - Search for the section by name.
   - In the Application Explorer, navigate to the section by class.
   - Use Live UI to inspect sections on the screen.
3. On the Section form, click an element, such as a field or layout.
4. If you are in full section editing mode, click the **Gear** icon.
5. In the **Cell properties** dialog box, enter a unique string identifier in the **Tour ID** field.
6. Click **Submit**.
7. Click **Save**.

The Live UI tool *(on page)*
Documenting your application *(on page 326)*

Associating content with a feature in a guided tour

You can associate content with a feature to create a tour stop in a guided tour.

1. Find and open a guided tour, by searching for it or by using the Application Explorer.
2. If there are no guided tours available, create one in the **User Interface > Guided Tour** rule category.

   For more information, see Creating rules *(on page)*.

3. On the **Definition** tab, click **+ Add tour stop**.
4. In the **Tour stop anchor** list, select a type of identifier, and then enter an ID in the text field that identifies an application feature.

- **Tour ID** - A unique string that you set on the **General** tab of the **Cell Properties** panel on the Section form.
- **Custom css selector** - A string that you define, which contains a pattern for finding elements on the screen.

5. In the **Title** field, enter the text that is displayed above the content for your tour stop.
6. In the **Tour stop section** field, press the Down Arrow key, and then select the section that defines your content.
7. In the **Show when** list, select an option to control when the tour stop is displayed at run time.
8. If your tour stop relies on the presence of a specific element, select a type of identifier, and then enter an ID for that element in the fields that are displayed.
9. **Optional:** To change the order of your tour stop, drag it to a different position on the form.
10. Click **Save**.

Creating project documents for stakeholders *(on page 326)*

Integrating a guided tour with your application

You can use one or more controls to integrate a guided tour with your application. By defining the conditions that start, stop, or restart your tour, you can respond to the actions that users take on the screen.

1. Add a control to the section that starts the tour.
    a. Find and open a section to start the guided tour, by searching for it or by using the Application Explorer.
    b. On the **Design** tab of the Section form, add any field, or control, to the section. For more information, see Creating sections *(on page)*.
    c. Click the field to edit it.
    d. If you are in full section editing mode, click the **Gear** icon.
    e. In the **Cell properties** dialog box, click **Change**, and then click **Other** in the **Custom** control category.
    f. In the autocomplete field, enter one of the following controls:
        - `pxGuidedTourAutoStart` – Starts the tour one time per user.
        - `pxGuidedTourAutoStartOncePerSession` – Starts the tour one time per session, to support users who share the same log-in credentials.

2. Associate the control with your guided tour.
    a. In the **Cell properties** dialog box, click **Parameters**.
    b. In the **TourClass** field, press the Down Arrow key, and then select the class of your guided tour.
    c. In the **TourName** field, press the Down Arrow key, and then select the name of your guided tour.
    d. **Optional:** To start the tour every time that the control is displayed, select the **AutoLaunchAlways** check box. This option is available for the `pxGuidedTourAutoStart` control only.
3. Click **Submit**.
4. Click **Save**.
5. **Optional:** To support restarting the guided tour, add actions to other elements on the screen.
    a. Follow step 1 to open the **Cell Properties** dialog box for another section or another element in the same section.
    b. Click **Actions**, and then click **Create an action set**.
    c. Click **Add an event**, and then click the type of event, such as mouse click, that initiates the action.
    d. Click **Add an action > All actions > Manage guided tour**.
    e. In the **Action to take** list, select **Start tour**.
    f. In the **Tour applies to** field, enter the class of your guided tour.
    g. In the **Tour name** field, enter the name of the guided tour.
    h. **Optional:** To restart the tour from the beginning instead of the last visited tour stop, clear the **Always continue where left off** check box.
    i. Click **Submit**.
    j. Click **Save**.

Adding actions to a control *(on page)*

Describing features to end users

You can use a guided tour to showcase application features in real time. By providing information in a sequence of pop-up windows, you can help users discover functionality that is available on their current screen.

For example, you can create a tour that gives example use cases for the different tools in a web channel.

Guided tours *(on page)*
Documenting your application *(on page 326)*

Troubleshooting tools and techniques

Maintain good quality and health of your application by using troubleshooting tools and techniques that Pega Platform provides. You can view application metrics to locate any issues, and then fix them, so that you deliver top quality application.

Related information

Designing applications for reuse and extension *(on page 74)*

Improving your compliance score

Follow development best practices to improve your compliance score. By eliminating risks, such as custom code or degraded performance, you can improve quality and resolve issues before your application goes into production.

To maintain your application quality, try to achieve a compliance score of 90 or greater. If your application's score becomes less than 80, take immediate action to address the issues.

The Compliance score landing page includes also a score for a Security Checklist. For more information about tasks that you complete for the Security Checklist, see Security Checklist *(on page)*.

Compliance score logic *(on page 344)*

Monitoring project compliance *(on page 344)*

Viewing application quality metrics *(on page)*

Analyzing a compliance score

Analyze your compliance data to find the root cause of a low compliance score. By taking action when your application's score is less than 80, you can improve application quality more quickly.

1. Find your compliance score.
    a. In the header of Dev Studio, click **Configure > Application > Quality > Guardrails > Compliance Score**.
    b. In the **Compliance score** section, review your weighted score.
2. **Optional:** To broaden the scope of your compliance data, include built-on applications.
    a. Click **application**, and then select one or more built-on applications.
    b. Click **Apply**.
3. Identify the highest priority issues that affect your score.
    a. Click the **Compliance Details** tab.
    b. In the **Warning impact** section, click the values in the **Resolve now** column.
    c. Review the list of rules with severe warnings.

   d. In the **Application risk introduced by operator** section, take note of the user with the highest risk percentage.
   e. Click the percentage, and then review the list of rules with warnings that the user added to your application.
4. Identify the types of rules with the most warnings.
   a. Click the **Warning Summary** tab.
   b. In the **# Rules with warnings** column, review the distribution of warnings across the different types of rules.
   c. Expand the row for the rule type with the highest number of warnings.
   d. **Optional:** For more information about a specific rule, click a rule name in the **Rule** column.
5. Review how your compliance score has changed over time.
   a. Click the **Compliance Score** tab.
   b. In the **Rules modified on or after** field, enter a date, and then click **Apply date filter**.

Monitoring project compliance *(on page 344)*
Compliance score logic *(on page 344)*

Resolving an application warning

Resolve warnings to improve application quality and ensure that you follow development best practices.

1. In the header of Dev Studio, click **Configure > Application > Quality > Guardrails > Compliance Score**.
2. In the **Compliance score** section, click the number in the **Rules with unjustified warnings** field.
3. In the **Rule name** column, click a rule to open it.
4. In the header of the rule form, click **Review/Edit**.
5. Review the warning message to understand which best practice your application does not follow, how it affects your compliance score, and what to do next.
6. Click **OK**.
7. Update the rule by following the guidance from the warning message.
8. Click **Save**.

Justifying an application warning

Justify a warning to explain why your application is exempt from a best practice, or guardrail. By documenting which warnings are intentionally unresolved, you can improve your compliance score.

1. In the header of Dev Studio, click **Configure > Application > Quality > Guardrails > Compliance Score**.
2. In the **Compliance score** section, click the number in the **Rules with unjustified warnings** field.
3. In the **Rule name** column, click a rule to open it.

4. In the header of the rule form, click **Review/Edit**, and then review the warning to confirm that your application can tolerate the associated risk.
5. Click **Add Justification**.

> **Note:** You cannot justify some warnings, because the risk that they introduce is too severe. For these cases, consider resolving the warning instead.

6. In the field that is displayed, enter text that explains why your development team chooses to leave this warning unresolved.
7. Click **OK**.
8. Click **Save**.

Resolving an application warning *(on page 341)*

Reviewing user interface components

Find out which UI elements in your application are high-maintenance. By analyzing the reports on the UI issues landing page, you can learn how to more efficiently render process-heavy components, and save development time during future updates.

1. In the header of Dev Studio, click **Configure > Application > Quality > App Studio Compliance > User Interface**.
2. In the **Built on applications** section, select one or more applications whose UI components you want to review.
3. Click **Apply**.
4. Review the report and follow the suggestions for updating UI components.
5. **Optional:** To open a section that contains UI issues in a new tab, expand the relevant report header, and then click the section name.

Related information

Improving your compliance score *(on page 340)*

Exporting compliance data

Export compliance data to create reports that you can download and distribute later. By manually collecting data, you can control which metrics to share with stakeholders.

1. In the header of Dev Studio, click **Configure > Application > Quality > Guardrails**.
2. To export your compliance score:
    a. Click the **Compliance Score** tab.
    b. **Optional:** To capture data from a specific point in time, select a date from the **Rules modified on or after** field, and then click **Apply date filter**.
    c. Click **Export to PDF**.
3. To export the distribution of warnings by type, age, and owner:
    a. Click the **Compliance Details** tab.
    b. **Optional:** To capture data from a specific point in time, select a date from the **Rules modified on or after** field, and then click **Apply date filter**.
    c. Click **Export to PDF**.
4. To export a list of rules with warnings:
    a. Click the **Warning Details** tab.
    b. **Optional:** To capture rules from a specific class, click **classes**, enter a class name in the field that is displayed, and then click **Apply**.
    c. Click **Export to Excel**.

Scheduling a compliance report

Schedule a compliance report to automatically inform stakeholders about changes to your application's compliance score over time.

1. In the header of Dev Studio, click **Configure > Application > Quality > Guardrails**.
2. Based on the type of data that you want to include in your report, click one of the following tabs:

    Compliance Score

    Summarizes the factors that contribute to your compliance score

    Compliance Details

    Categorizes the warnings in your application by type, age, and owner

    Warning Summary

    Depicts the distribution of warnings in your application by severity and rule type

    Warning Details

    Lists more information about the rules that introduced the warnings

3. Click **Schedule report**.
4. In the **Email Scheduling** dialog box, select the **Enable email notifications** check box.
5. In the **Send the report** list, select the frequency at which your application sends the report data in an email.
6. Click the **Add** icon.

7. In the field that is displayed, press the Down Arrow key, and then select the name of a stakeholder.

   If the stakeholder is not in the list, enter an email address in the field instead.

8. Click **Submit**.

Exporting compliance data *(on page 342)*

Monitoring project compliance

Monitor the collective compliance score for your application and built-on applications to determine how well your project follows development best practices.

1. In the header of Dev Studio, click **Configure > Application > Quality > Dashboard**.
2. On the **Compliance Score** tab, click **Enable historical compliance score**.
3. Periodically review the graph that is displayed, to monitor how your score changes from week to week.

Log files tool *(on page 384)*

Compliance score logic

A standard formula determines your compliance score. By understanding the logic and variables in this formula, you can focus on the development changes that improve application quality the most.

The following figure presents the logic that assigns weights to the warnings in your application.

$$\max\left(0{,}1 - \left(\frac{10Sev1u + 5Sev2u + 2Sev1j + Sev2j}{Total\ Remaining\ Rules}\right)\right) \times 100$$

Variable definitions:

Sev1u

Severe, unjustified warnings

Sev2u

Moderate, unjustified warnings

Sev1j

Severe, justified warnings

Sev2j

Moderate, justified warnings

The sum of the variables is divided by the number of rules with informational warnings or no warnings. The result is a positive, integer percentage.

Compliance scores apply only to the rules in your application and imported rules that you resave. Property rules and all rules in Pega* rulesets do not affect your compliance score.

Related information

Analyzing a compliance score *(on page 340)*

Improving your compliance score *(on page 340)*

Application Guardrails landing page

Use the Application Guardrails landing page to find the warnings in your application and learn how they affect your compliance score.

For more information, see Improving your compliance score *(on page 340)*.

UI guardrail issues landing page

The UI guardrail issues landing page helps you find high-maintenance UI elements, and learn how to convert them to a more modern and efficient rendering.

Application components that do not comply with the user interface guardrails might require additional development time during updates. You can use the UI guardrail issues landing page to check the condition of your application and better prepare it for future changes.

The UI guardrail issues landing page displays a list of violations in the selected application. The list groups similar issues together, and each grouping contains advice on how to resolve the items in each category.

For more information, see Reviewing user interface components *(on page 342)*.

Related information

Improving your compliance score *(on page 340)*

Unit testing individual rules

An incorrect rule configuration in an application can cause delays in case processing. To avoid configuration errors such as incorrectly routed assignments, unit test individual rules as you develop them. To expedite future rules testing, you can create reusable test cases from the unit test.

You can test a rule with test data that you provide by clicking **Actions** > **Run** on the rule form toolbar. For some rule types, such as binary file rules, Pega does not provide an option for unit testing. If the rule cannot be unit tested, the **Run** option is not available.

The appearance of the **Run Rule** window varies across rule types, so how you run a rule varies by its type. In general, however, the rules run by using data from a test page that you define for the test.

Tasks involved in defining the test page include the following.

1. Selecting a method for creating the test page – You can copy values from a thread of an existing clipboard page to the test page, create a new test page, or reset the values of an existing test page. For more information about clipboard pages, see Clipboard tool *(on page 373)*.
2. Applying data transforms – For a reusable and expedited method of making decisions and calculating values, you can apply data transforms to set values for the test page. For example, to unit test a decision table, you can create a data transform to provide values for the properties evaluated by the table, rather than manually entering values when you run the rule. For more information about data transforms, see Data transforms *(on page 140)*.
3. Manually entering test data – In some cases, you can manually enter values to use. If you enter values for a test, the values that you enter override values on the test page.
4. Specifying how service rules run – For services, you also specify whether the service rule is to run in your session or is to run as a newly created service requestor. If the service is configured to run as an authenticated user, you are prompted for a user name and password.

> **Note:** To test a circumstance rule, ensure that the circumstances are correct for the rule. Otherwise, the system tests the base rule.

When you run the rule, the system uses rule resolution. If you unit test a rule, and there is a higher version of the rule, the system runs the higher version.

After you run the test, you can also convert the test to a reusable test case that you can run at any time. For more information about using unit test cases, see Understanding unit test cases *(on page)*.

Troubleshooting newly created rules

If there are issues with copying a rule or data instance, see the following information for troubleshooting help.

ALLEP: Make this a task with xrefs to suggested things to look at

- If your newly created rule is not being executed, it might be related to the rule resolution algorithm which operates differently on different rule types. See the help topics *Completing the Create form* for each rule type to understand specifics of rule resolution for that type.
- Many standard rules have Work- as the **Apply to** key part, but you cannot create additional rules with Work- as the **Apply to** key part. When you're copying such a rule, the Save As form defaults the name of the container class for the current work pool as the **Apply to** class.
- The Save As form is restricted to those users who hold the privilege `@baseclass.ToolbarFull` or `@baseclass.ToolbarSaveAs`.
- The availability of your new rule is not specified on the Save As form.
- The status of your new rule may be automatically cleared if the following criteria are met:
    - The original rule status is not internal
    - One of the following conditions apply
        - The specified ruleset is not a branch of the original rule
        - You are changed the Apply To class
        - The original rule has circumstances defined
- You can specialize a rule or create a circumstance of it.

Copying a rule or data instance *(on page)*
Setting rule status and availability *(on page)*

Application debugging by using the Tracer tool

To provide a top-quality experience for your software users, test and debug your applications by using the Tracer tool. You can test and debug multiple resources, for example, activities, data transforms, decision rules, service rules, parse rules, and processes. You can also troubleshoot offline-enabled applications from a browser by testing scripts when the application is running.For example, you can trace the events that are related to queue processing to ensure that your application correctly queues tasks to process in the background. As a result, you increase the efficiency of your application and save time because you avoid further debugging that untraced events might cause.

You can select which rulesets, rules and events to trace; set breakpoints and watch variables; trace reference properties; select which requestor session to trace, and add custom events to Tracer output.

You can also trace services and listeners anywhere in the cluster by using the RuleWatch trace.

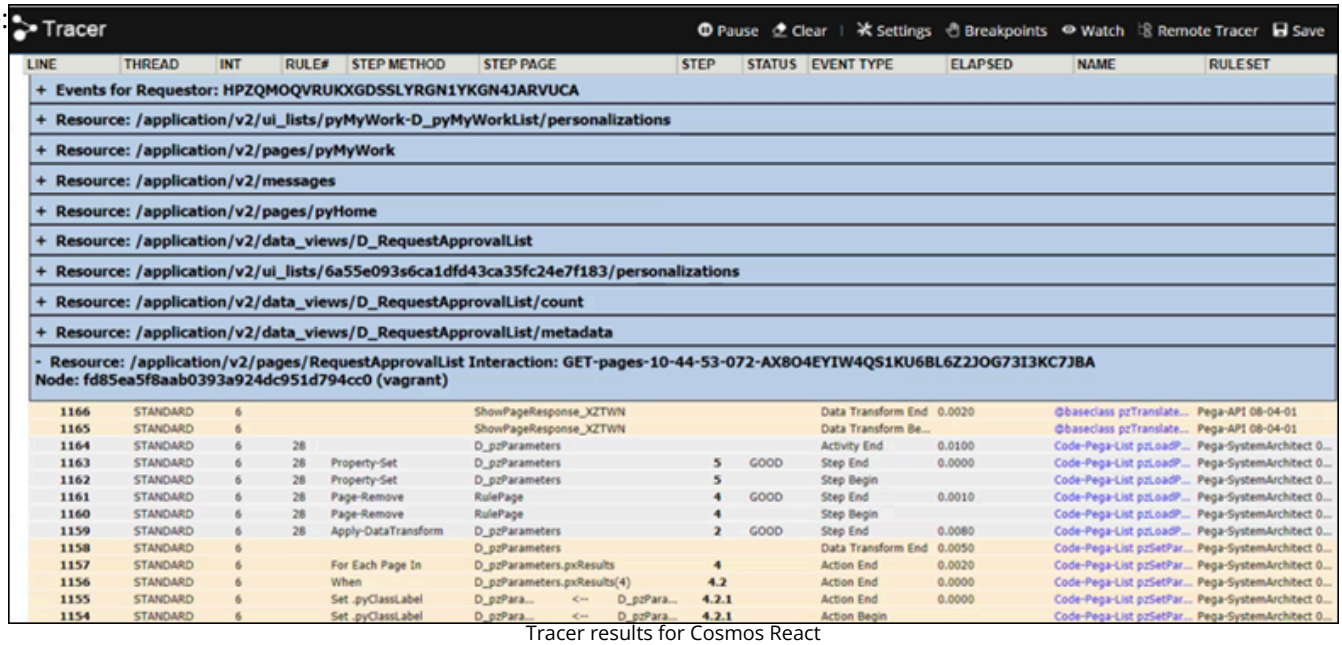
When you set up your trace conditions and start tracing, you can view the Tracer output in the Tracer window or save the Tracer output to your local system, so that you can analyze your system offline or share the results with other developers.

Note: When using this tool for Customer Service chat interactions, be aware that multiple requestors can serve a single browser session.

Note: The Tracer tool is not available during a test coverage session.

Tracer for Cosmos React applications

Apart from the standard Pega Platform applications and your custom applications, Tracer also supports applications that you build by using the Cosmos React. Because Cosmos React applications are stateless and rely on DX APIs, the Tracer results window reflects the scheme in which your application interacts with a browser during a browser session. For greater clarity, the results window groups events by request ID, because multiple requests can be active simultaneously, as in the following example:



Tracer results for Cosmos React

For relevant training materials, see the Debugging application errors module on Pega Academy.

Hybrid mode *(on page)*

Cluster-wide tracing of service rules *(on page 366)*

Estimating test coverage *(on page)*

Tracer best practices

Use Tracer best practices to make tracing easier and more efficient.

Trace only one requestor session per operator and node

Do not attempt to start the Tracer when another requestor session on the same Pega Platform node is using the same Operator ID, or is running the Tracer.

Turn off HTTP compression

Ordinarily, Pega Platform uses HTTP compression to reduce the length of text messages sent to Internet Explorer. HTTP compression is also used in SOAP messages. This might complicate Tracer debugging because compression makes it difficult to interpret the messages.

To disable HTTP compression, set the value of the *prconfig/enable-compression/default* Tracer dynamic system setting to `false` and its owning ruleset to *Pega-Engine*. This change takes effect the next time your system starts. For more information, see Creating a dynamic system setting *(on page)*.

Reset this value to enable compression when debugging is complete.

Domain Name System lookups required

In some Pega Platform installations, the `ClientHostName` value appears only as a numeric Internet Protocol (IP) address. The true `ClientHostName` is visible only when the application server is enabled to perform Domain Name System lookups:

- For Tomcat-based systems, set the parameter `enableLookups` (in the `server.xml` file) to "true" to allow DNS lookups.
- For Oracle Weblogic, add the parameter `ReverseDNSAllowed="true"` to the **`config.xml`** file to allow DNS lookups.

Duplicate trace error

If the node hosting your trace session fails and you attempt to trace the same service again, you might encounter a duplicate trace error. Clear stale trace sessions using Admin Studio.

Tracer event queue overflows *(on page 361)*
Adjusting the buffer size of the Tracer header *(on page 360)*
Managing requestors *(on page)*
Reducing Tracer event output *(on page 359)*
Configuring dynamic system settings *(on page)*

Tracer disk usage FAQ

For effective debugging, Tracer creates a file with the details of traced events so that you can make an informed decision about processes in your application. When you trace a high number of events, the file size can become extensive. To avoid issues with disk usage, Tracer sets a limit on the file size. As a result, tracing your application has no impact on running of your system.

To which deployments of Pega Platform does the limit apply?

The limit on disk usage applies to on-premises Pega Platform deployments, Pega Cloud deployments, and Pega Platform deployments on client-managed clouds.

What is the disk usage limit?

The disk usage limit is 10 GB per node. The limit is the same for all nodes in a system, and the disk limit check occurs at the node level. Multiple sessions on one node share the disk usage limit.

In a sample scenario, four Tracer sessions run on two nodes. Sessions S1 and S2 run on node N1, and sessions S3 and S4 run on node N2. The disk usage limit defines Tracer behavior in the following way:

- When the session S1 reaches the limit of 10 GB, S1 stops and the system deletes the respective file of event details.
- Sessions S2, S3, and S4 continue to run.

What happens when the file size reaches the disk usage limit?

When the event file reaches the size of 10 GB, Tracer behaves in the following way:

- Tracing of the session that reaches the disk usage limit stops.
- The system deletes the event file. You can then only download error event messages.
- In the Tracer window, you can view the header of the previous Tracer session without the full event details.

Does the system reclaim the disk space after stopping a Tracer session?

Yes, the system reclaims the disk space after the file size in a session reaches 10 GB.

Does the Tracer stop running after reaching the disk usage limit?

Yes, the Tracer stops a tracing session that reaches the disk usage limit. For service rule tracing, the system generates a disk write event. Consider a scenario in which sessions S1 and S2 run on the same node. The node already uses 9 GB of disk space. If session S1 tries to use another 1.5 GB, the system stops session S1. Session S2 continues to run.

How does service rule tracing work with the disk usage limit?

Service rule tracing continues to run even after the event files reach the size of 10 GB, however, the system deletes the file that exceeded the limit and you can view only event error messages.

Can I increase or decrease the disk usage limit for Tracer?

Yes, you can change the disk usage limit by creating and then editing the *tracer/disk/maxusage* dynamic system setting. The minimum value for the disk usage limit is 5 GB.

> **Note:** Change the default limit value only in justified situations, for example, when you do not have 10 GB free space in your system, or when 10 GB per node is not sufficient for your scenario.

For more information about dynamic system settings, see Configuring dynamic system settings *(on page)*.

Does the system delete the event file when the file size reaches 10 GB?

Yes, the system deletes the event file after the file size reaches 10 GB. The Tracer file that you can later download contains only error messages.

Does each Tracer session have a separate event file?

Yes, each Tracer session has a separate event file. If you have multiple Tracer sessions that run on one node, the total maximum size of all event files is 10 GB.

How does the system maintain the disk usage limit?

The system maintains the disk usage limit at the node level. Multiple Tracer sessions share the usage when running on one node.

In a sample scenario, sessions S1 and S2 run on node N1, while session S3 runs on node N2. Session S1 occupies 9 GB in an event file, and session S2 uses 700 MB. Session S2 needs 200 MB to write details about the next event to trace. The system allows the action because the total space that sessions S1 and S2 need in the files is less than 10 GB. Next, session S2 tries to write details about the next event that require another 200 MB. The system stops session S2 and sends the message that the file size reaches the Tracer disk usage limit for node N1. The system deletes the event file for session S2 and users can then only view the error messages. In the meantime, session S3 tries to write event details that require one GB in the event file. The system allows the action because every node has a separate limit of 10 GB.

What happens if a Tracer session starts and a browser crashes, or the Tracer does not receive a close event before the event file size reaches 10 GB?

The Tracer session remains active and writes events to the file. However, the system treats the session as stale and deletes the session automatically after an hour. You can change the timeout value after which the system deletes stale sessions by editing the *tracer/session/timeout* dynamic system setting.

> **Note:** The *tracer/session/timeout* dynamic system settings affects also pause sessions. The system deletes paused sessions after the timeout value passes.

For more information about editing dynamic system settings, see Configuring dynamic system settings *(on page)*.

What can I do when the file size constantly reaches the 10 GB limit, even if I do not run any Tracer sessions that use disk space?

Verify with your system administrator that you close all Tracer sessions that are not necessary at the moment. The list of operators that run Tracer sessions is available in the error message after reaching the disk usage limit. Also, ensure that you select only necessary rules and events to trace instead of tracing all available events. If you do not need to view clipboard page content, selecting the **Abbreviate Events** check box in the Tracer settings drastically reduces the disk usage.

How can I reclaim disk space when another operator starts a Tracer session and forgets to close the session?

You can check the other operators who run Tracer sessions. The list of operators is available in the error message that you receive after the event file size reaches the disk usage limit. You can also close Tracer sessions from Admin Studio.

For more information, see Managing requestors *(on page)*.

Related information

Managing requestor pools *(on page)*
Monitoring system health *(on page)*
Viewing Tracer results in the Tracer window *(on page 367)*
Offline debugging by using Pega-TracerViewer *(on page 370)*
Configuring trace conditions *(on page 353)*
Tracing services *(on page 365)*
Managing system resources *(on page)*

Configuring trace conditions

Before you run the Tracer tool, optionally configure trace conditions, based on the activity, data transform, rule, and so on, that you are testing and debugging. If you do not configure trace conditions, the default conditions are used.

1. On the developer toolbar, click **Tracer**, or click **Actions > Trace** on an activity, data transform, or service rule form.
2. **Optional:** Configure Tracer settings *(on page 354)*. Select which events, event types, rulesets, and pages to trace, break conditions, and options for the amount of output that you want the Tracer to generate.
3. **Optional:** Send the Tracer output to a file *(on page 361)*. This option is useful if you want to work offline with the Tracer results instead of using the Tracer window.
4. **Optional:** Set breakpoints *(on page 362)* to stop the Tracer at specific steps in activities.
5. **Optional:** Set watch variables *(on page 363)* to detect when processing has changed the value of a property.
6. **Optional:** Select a requester session *(on page 364)* other than your own current session to trace. By default, the Tracer traces your own requester session.

Tracing and capturing events *(on page 364)*
Tracer best practices *(on page 348)*

Configuring Tracer settings

Maintain the quality of your application and quickly locate any issues by tracing selected events with the Tracer tool. Analyze the Tracer results conveniently by defining what information you want to include in the output. For example, you can specify when you want the Tracer to pause processing, or how many events you want to display in the output.

1. In the footer of Dev Studio, click the **Tracer** icon.
2. On the **Tracer** toolbar, click **Settings**.
3. In the **Tracer Settings** window, in the **Events to trace** section, select check boxes of events that you want to include in the Tracer output:
    - To trace when an event starts, select the check box in the **Start** column.
    - To trace when an even ends, select the check box in the **End** column.
4. In the **Break conditions** section, select the conditions under which you want the Tracer to pause processing:
    - To pause processing on the first Java exception, select **Exception**.
    - To pause processing when a step ends with a Fail status in the *pxMethodStatus* property value, select **Fail Status**.
    - To pause processing when a step ends with a Warn status in the *pxMethodStatus* property value, select **Warn Status**.

    > **Note:** If you build your application on Cosmos React, Tracer does not support break conditions.

5. In the **General options** section, select the options that determine the quantity of output in specific conditions:
    - To include the Java class details from those properties in the Tracer output, if processing has properties of mode Java Pages, select **Expand Java Pages**.

      This option might significantly slow processing.
    - To capture the status of local variables for Begin and End events of activities and their steps, select **Local Variables**. Tracer creates an output row each time the system sets the value of a local variable in an activity.

      For local variables with a value of Java null, the Tracer output displays the string `null – No Value` for the value of the local variable.
    - To reduce the performance impact by limiting the amount of clipboard detail for each row in the Tracer window, select **Abbreviate Events**.
6. In the **Event types to trace** section, select event types that you want the Tracer to monitor.

7. **Optional:** To add additional event types, in the **Event Type** field, enter the event type, and then click **Add**.
8. In the **Rulesets to trace** section, select the rulesets that you want to trace.
9. **Optional:** To trace additional pages, in the **Pages to trace** section, in the **Page name** field, enter a page name, and then click **Add**.

   While the Tracer session runs, the Tracer watches the pages that you add. If the Tracer locates the pages, links to their content display in the **Properties on Page** window when you open an event from the Tracer output.

   > **Note:** When you add multiple pages, the Tracer might work slower.

   The option to add pages is unavailable if you select **Abbreviate Events** in the **General options** section.
10. In the **User interface** section, in the **Max Trace Events to Display** field, enter the number of trace events to display in the output.
11. Click **OK**.

Troubleshooting newly created rules *(on page 346)*

Tracer events to trace

You can select the following events to trace in the **Tracer Settings** dialog box. When selected, the tracer output includes these events. Events are most relevant for debugging activities rather than flows or declarative rules.

Events to Trace	Description
Access Deny rules	Displays a line at the start of each run of an Access Deny rule.
Activities Start	Displays a line at the start of each activity.
Activities End	Displays a line at the completion of each activity.
Activity Steps Start	Displays a line at the start of every activity step and step iteration.
Activity Steps End	Displays a line at the completion of every activity step.
Data Transforms Start	Displays a line at the start of each data transform.

Events to Trace	Description
Data Transforms End	Displays a line at the completion of each data transform.
Data Transform Actions Start	Displays a line at the start of each data transform action.
Data Transform Actions End	Displays a line at the completion of each data transform action.
Exception	Displays a line for every Java exception. In most cases, processing continues after the excepti
Push notification	For Subscriptions, Tracer displays the following information: • Notification Channel • Subscription Class • Subscription Condition • Subscriber Data For Publish, Tracer displays the following information: • Notification Channel name • Source Page • Published Page • Stack trace of publish
When Rules Start	Displays a line at the start of every when condition evaluation, including preconditions and tr tions.
When Rules End	Displays a line at the completion of every when condition evaluation, including preconditions transitions. If selected, Tracer output shows the results of all when rule executions from all ru regardless of your selected rulesets.

Tracing and capturing events *(on page 364)*
Configuring Tracer settings *(on page 354)*
Configuring trace conditions *(on page 353)*

Tracer event types to trace

You can trace the progress of various events that occur in the system by selecting event types to trace in **the Tracer Settings** dialog box. Examples of rules that produce events are flows, declarative rules, and decision trees rules. You can also trace events related to services and database operations.

For declarative rules, rows of the Tracer output show the operation of the declarative network, that is, the forward and backward chaining computations that occur automatically.

Label	Description
Event Type	Enter an additional event type as instructed by Pegasystems Global Customer Services and click **Add** to enable tracing for this type.
ADP Load	Trace the background thread (requestor) that loads a declare page asynchronously.
Adaptive Model	Trace Adaptive Models.
Alert	Include Alert log messages produced by your Thread in the Tracer output.
Asynchronous Activity	Trace asynchronous activity.
Auto Populate Properties	Trace auto-populated properties.
Automation	Trace automation rules.
CaseType	Trace case type rule calculations.
DB Cache	Show hits and misses for the rule cache and the conclusion cache.
DB Query	Show each SQL query sent to the PegaRULES databases and its results, including expansion of the BLOB column.
Data Pages	Show data page activity, including instantiations from properties, calls to data sources, and responses to those calls.
Declare Collection	Trace collection rules.
Declare Constraint	Trace Constraints rules.
Declare DecisionMap	Trace map values (start only).
Declare DecisionTable	Trace decision tables (start only).
Declare DecisionTree	Trace decision trees (start only).

Label	Description
Declare Expression	Trace Declare Expression rules.
Declare Index	Trace Declare Index rules.
Declare OnChange	Trace Declare OnChange rules.
Declare Trigger	Trace Declare Trigger rules.
Flow	Trace a flow. Rows of the Tracer output identify each flow start and end, each flow shape start and end, and the connectors (arrows on the flow diagram).
Interaction	Start and end of a user interaction, as defined by the Performance tool.
Linked Page Hit	Trace operations on linked pages that cause retrieval of the target page from a cache.
Linked Page Miss	Record cache misses for linked properties.
Locking	Trace locking operations. Tracer output shows each lock acquired and each lock released.
Log Messages	Record each run of the Log-Message method that has the SendtoTracer parameter selected.
Parse Rules	Trace Parse XML, Parse Structure, and Parse Delimited rules. Other parse rule types are not traced.
Predictive Model	Trace predictive model rules.
Query resolution	Trace report queries to identify queries that run against the database instead of Eleasticsearch.
SOAP Messages	Trace detailed SOAP request and response messages for SOAP, SAP, and .NET connectors and services.
Scorecard	Trace scorecard rules.
Services	Trace service rules and service activities.
Strategy	Trace strategy rules.

Label	Description
Stream Rules	Trace all types of stream rules.

Tracing and capturing events *(on page 364)*
Configuring trace conditions *(on page 353)*
Configuring Tracer settings *(on page 354)*
About Collection rules *(on page)*
Property form: Completing the General tab - Value modes *(on page)*
Log-Message method *(on page)*
Tracing services *(on page 365)*

Rulesets to trace

You can select the rulesets to trace in the **Tracer Settings** dialog box. The dialog box lists all of the rulesets that you can access, in the order they appear in your ruleset list, based on your access group and other sources.

The Tracer always enforces ruleset access controls. Whether you trace your own session or another user's session, Tracer output includes only information about the execution of rules in the ruleset that you selected. The Tracer window does not identify gaps that were caused by this restriction.

The `Pega-` rulesets listed last are base Pega Platform rulesets. You can select them for tracing, but the Tracer might produce large quantities of output. You can trace flows in your application rulesets without selecting the `Pega-ProCom` ruleset or other base rulesets.

The rulesets that you select do not affect Tracer output for when rules. If you selected the **When rules End** option in **Events to Trace**, Tracer shows the outcome of every rule that you execute, in any ruleset.

Tracing and capturing events *(on page 364)*
Configuring Tracer settings *(on page 354)*
Configuring trace conditions *(on page 353)*

Reducing Tracer event output

You can reduce the amount of clipboard detail that is sent to the Tracer by using the **Abbreviate Events** option. Reducing the amount of output improves the performance of the requestor session that is being traced so that elapsed time statistics during tracing are closer to the normal values, that is, the values when processing is not being traced.

When you abbreviate events, some watch variables might not operate correctly, and the **Pages to Trace** and **Local Variables** features are disabled. Do not abbreviate events if debugging requires these features.

1. On the developer toolbar, click **Tracer**, or click **Actions > Trace** on an activity, data transform, or service rule form.
2. Click **Settings**.
3. In the **GENERAL OPTIONS** section, select **Abbreviate Events**.
4. Click **OK**.

Setting the number of lines to display in Tracer *(on page 360)*

Setting the number of lines to display in Tracer

You can change the number of lines that are displayed in the Tracer window to manage network and workstation load. By default, the Tracer displays the 500 most recent trace lines. You can set a higher value, which requires additional workstation memory. If you only want to view Tracer output as an XML file and save it to your local system, set the number of lines to display to zero (0) to reduce the network and workstation load.

1. On the developer toolbar, click **Tracer**, or click **Actions > Trace** on an activity, data transform, or service rule form.
2. Click **Settings**.
3. In the **USER INTERFACE** section, set **Max Trace Events to Display** to the number of lines to display. When Tracer output reaches this limit, the oldest lines are dropped from the window to allow newer lines to be displayed.
4. Click **OK**.
5. Restart the Tracer tool for the changes to take effect.

Reducing Tracer event output *(on page 359)*

Adjusting the buffer size of the Tracer header

You can adjust the buffer size of the Tracer header to increase the limit for unprocessed events. By default, the system saves up to 50,000 items for unprocessed events during a Tracer operation. If the buffer exceeds this limit, Tracer processing ends.

1. Configure the *prconfig/tracer/queue/header/limit/default* dynamic system setting with the owning ruleset *Pega-Engine* and the value of a nonnegative integer.
   For more information, see Creating a dynamic system setting *(on page)*.
2. Stop and restart (or redeploy) the server node to enable the change.

Reducing Tracer event output *(on page 359)*
Tracer best practices *(on page 348)*

Updating dynamic system settings by using Java methods *(on page)*
Configuring dynamic system settings *(on page)*

Tracer event queue overflows

Each Tracer session uses a file buffer that holds up to 500 events not yet processed for display. Some complex tracing situations reach this limit and additional events are discarded.

The following message is displayed:

```
Event queue is overflowing on server, events will be discarded.
```

To avoid this situation, select fewer rulesets or event types, and then repeat the operation.

If this message occurs often, you can increase the file buffer size so that it can contain more events. If the problem persists after you have set a larger buffer size, your application may contain an infinite loop.

Various conditions can cause the event queue to fill up. For example, if the Tracer and the traced requestor session are running on one client workstation, then both sessions share the normal limit of two HTTP operations to the server. If the traced operations involve many HTTP operations, then the Tracer contends with other browser operations for access to the server. This contention can cause the Tracer to drain the event queue more slowly, causing an event queue overflow.

Adjusting the buffer size of the Tracer header *(on page 360)*
Reducing Tracer event output *(on page 359)*
Tracer best practices *(on page 348)*

Sending Tracer output to a file

You can send the Tracer output to a file if you want to work offline with the Tracer results instead of viewing the output interactively in the Tracer window. In addition, you can suppress the event window to reduce network traffic and workstation load when you only want to view the output offline.

By default, the system saves the output as an .xml file. When tracing in a cluster, and more than one node reported events for a given RuleWatch, the system saves the output as a .zip file that contains an .xml file for each node that reported events. If only one node reported events, then the system saves the output as an .xml file.

For information about viewing the output file and to download a Windows viewer for Tracer output .xml files, see the Pega Community article *Using the Tracer tool to summarize Tracer XML output in Pega 7*.

1. On the developer toolbar, click **Tracer**, or click **Actions > Trace** on an activity, data transform, or service rule form.
2. Click **Settings**.
3. **Optional:** To suppress the event window, in the **USER INTERFACE** section, set **Max Trace Events to Display** to 0.
4. Click **OK**.
5. Perform the work that you want to trace.
6. Click **Save**. The Tracer output file is saved in the local Downloads folder.

Configuring trace conditions *(on page 353)*
Tracing and capturing events *(on page 364)*

Setting breakpoints in the Tracer tool

You can set breakpoints to pause processing for specific steps in activities. When you set a breakpoint for an activity, the Tracer window acquires focus when you or the selected requestor connection starts the activity.For example, if a step in an activity triggers a data transform, you can set a breakpoint before the step so that you can analyze and understand how the system updates the values in the data transform.

> **Note:** If you build your application on Cosmos React, Tracer does not support breakpoints. Changing configurations for breakpoints in Cosmos React applications might cause issues.

> **Note:** You cannot view the clipboard when the system reaches a breakpoint during processing because the requestor that the Tracer traces pauses.

1. In the footer of App Studio, click the **Open runtime toolbar** icon, and then click the **Tracer** icon.
2. On the **Tracer** toolbar, click **Breakpoints**.
3. In the **Class Name** field, select the Applies To class that stores an activity to trace.
4. In the **Activity Name** field, select an activity.
5. In the **Where to Break** list, select one of the following options:
   - To stop processing when the activity starts, select **1**.
   - To stop processing before a specific step, select the step number.
   - To stop processing before every step, select **all steps**.
6. Click **Set Break**.
7. **Optional:** To add more breakpoints, repeat steps 3 *(on page 362)* through 6 *(on page 362)*.

8. **Optional:** To remove a breakpoint, in the **Remove** column, select the check box, and then click **Remove**.
9. Click **Close**.

Configuring trace conditions *(on page 353)*
Tracing and capturing events *(on page 364)*

Setting watch variables in the Tracer tool

Set a watch variable to detect when processing changes the value of a property. After you set a watch variable, you can minimize the Tracer window and continue working. Focus returns to the Tracer window when the property value changes.For example, if you watch a case status, you can see who changes the status of a particular case,and when. You can use this information when a case unexpectedly fails to successfully reach a resolution.

Note: If you build your application on Cosmos React, the Tracer does not support watch variables. Changing configurations for watch variables in Cosmos React applications might cause issues.

1. On the developer toolbar, click **Tracer**, or click **Actions > Trace** on an activity, data transform, or service rule form.
2. In the **Tracer** toolbar, click **Watch**.
3. In the **Page Name** field, enter the name of the page that stores the property to watch, or enter a parameter for the parameter page.
4. Enter the property name without a period in front of it in the field, or select to watch this page for page messages.
5. Select variables to watch:
    - To watch a single property, in the **Property Name**, enter the property to watch.
    - To watch the page for page messages, select the **Or Page Messages**.
6. Click **Set Watch**.
7. To watch more properties, repeat steps 3 *(on page 363)* through 6 *(on page 363)*
8. **Optional:** To remove a watch variable, select the check box in the **Remove** column, and then click **Remove**.
9. Click **Close**.

Configuring trace conditions *(on page 353)*
Tracing and capturing events *(on page 364)*

Selecting a session to trace

By default, the Tracer traces your own requestor session. You can trace another session by using the Remote Tracer. The Remote Tracer lists every requestor that is connected to your Pega Platform server.

The first letter of the session ID identifies the requestor type:

- H – Internet Explorer-based users
- B – Background users such as agents, service requestors other than for Service Portal rules, and daemons
- A – External applications
- P – Access through Service Portlet rules

You cannot trace listener processing because listeners operate as Java threads, not full requestor sessions. Use remote logging to debug listeners.

1. On the developer toolbar, click **Tracer**, or click **Actions > Trace** on an activity, data transform, or service rule form.
2. Click **Remote Tracer**.
3. Select the session that you want to trace.
4. Click **Okay**.

Configuring trace conditions *(on page 353)*
Tracing and capturing events *(on page 364)*

Tracing and capturing events

After you have configured the trace conditions, you can begin tracing and capturing events. As you perform the work that you want to trace, the Tracer displays the traced events according to the selected trace conditions. Each event is a row in the Tracer window.

1. On the developer toolbar, click **Tracer** or click **Actions > Trace** on an activity, data transform, or service rule form.
2. Click **Play** to start tracing.
3. Move or minimize the Tracer window so that you can see the Dev Studio to perform the work that you want to trace.
4. Perform the work that you want to trace, or when tracing a requestor session other than your own, wait for the system to perform the work in that session.
5. If you have set a breakpoint for an activity, and the activity starts, the Tracer window acquires focus. Click **Continue** to resume tracing. Tracing automatically resumes after one hour.
6. **Optional:** To remove a breakpoint:

a. Click **Breakpoints**.
b. Select the **Remove** checkbox.
c. Click **Remove**.

7. If the value of a watch variable changes, the Tracer window acquires focus and the Property Inspector is displayed in the lower left of the window. To view the full Property Inspector display, place your pointer at the line until a yellow arrow appears and drag the dispaly up. The following fields are displayed:

   Reference – Specifies the page and property being watched.

   Old Value – If you clicked **Continue** and the Tracer finds that the value has changed, the previous value is displayed.

   Current Value – Displays the current value of the property.

8. Click **Continue** to resume tracing. Tracing automatically resumes after one hour.
9. To end the watch:
   a. Click **Watch**.
   b. Select the **Remove** check box.
   c. Click **Remove**.
10. Click **Pause** to stop tracing.

Configuring trace conditions *(on page 353)*
Tracer best practices *(on page 348)*

Tracing services

You can use the Tracer to monitor any active requestor session. However, a service usually runs in a new requestor session with a requestor ID that is not created until the service begins processing. At that point, the processing that the service performs in that requestor session occurs so quickly (in less than one second), that it can be hard to catch the event to trace it.

In this situation, it is recommended that you use the Trace Open Rule feature to trace the service rule. You can use this feature to trace a service request that is invoked from an external client application.

The ability to trace services comes with a performance impact even when you are not actively tracing, therefore it is recommended that you disable service tracing in production environments and during performance testing unless you are troubleshooting a related issue.

Service rule tracing is only enabled in environments where the *trace/cluster/ServiceRuleWatchMaxProductionLevel* Dynamic System Settings is greater than or equal to the production level. By default, *trace/cluster/ServiceRuleWatchMaxProductionLevel* is set to 4, which means that the

ability to trace service rules would not work in a production level 5 environment until *trace/cluster/ServiceRuleWatchMaxProductionLevel* is increased to 5.

- Owning Ruleset: `Pega-RulesEngine`
- Setting Purpose: `trace/cluster/ServiceRuleWatchMaxProductionLevel`
- Value: `<greater than or equal to the production level, for example, 5>`

To use the Trace Open Rule feature to trace and debug a service rule:

1. Open the service rule by performing one of the following tasks:
    - In the developer toolbar of Dev Studio, click **Tracer**.
    - In the developer toolbar of App Studio, click **Tracer**.
2. Click **Settings** to open the Tracer Settings window.
3. In the **Event Types to Trace** section, select **Services**.
4. In the **Rulesets to Trace** section, select the rulesets that contain the service rule and service activity.
5. Click **OK** to close the Tracer Settings window.
6. Run the rule.
7. Watch the Tracer window as the rule runs. The window includes lines for the start and end of each service rule execution, lines for the start and end of request mapping, lines for the start and end of response mapping, and (if these options are selected in the Tracer Settings panel) lines for the start and end of parse rules, XML Stream rules, and HTML rules.

> The Tracer offers the following trace options for services:
>
> - Services – The Tracer adds steps for when the service invocation begins and ends. Nested within those steps are entries that show when the inbound data mapping begins and ends, and when the outbound data mapping begins and ends.
> - Parse Rules – The Tracer adds steps when a parse rule (delimited, structured, or XML) begins and ends processing.
> - Stream Rules – The Tracer adds steps when an HTML rule or XML Stream rule begins and ends processing.

Configuring trace conditions *(on page 353)*
Cluster-wide tracing of service rules *(on page 366)*

Cluster-wide tracing of service rules

You can trace service rules across a cluster. Cluster-wide tracing makes it easier for you to troubleshoot services in clustered configurations where a load balancer dynamically assigns service requests to nodes in

the cluster. This functionality is especially useful for debugging mobile applications, RESTful Web Services, and other stateless service executions.

When tracing across the cluster, the trace feature only captures one trace at a time, which might result in some concurrent service executions not being traced. All other concurrent service executions in the cluster continue unimpeded. When the trace in progress completes, the next new service execution in the cluster is traced. For example, you might see traces for invocation one on node A, invocation two on node C, and invocation four on node B. This selective tracing approach ensures that tracing with a breakpoint will not stop all service executions across the cluster.

You can selectively disable cluster-wide tracing based on production level to eliminate performance impact in production by using the `trace/cluster/ServiceRuleWatchMaxProductionLevel` dynamic system setting. Set the value to the maximum production level for which you want to allow RuleWatch tracing for Rule-Service-data instances. The Trace option will not be shown in the **Actions** menu for the rule form on systems with a production level higher than the dynamic system setting's value.

> **Note:** Only one user can trace a service rule at any given time. If a user is tracing a service rule on one node and another user tries to trace the same service rule on either the same node or a different node, an error message is displayed.

Tracing services *(on page 365)*

Viewing Tracer results in the Tracer window

To maintain the quality of your application, you can use the Tracer window to interact with Tracer results. Each row in the window represents an event. The Tracer records selected events from the rule executions, database operations, and other event types that you select when you configure the Tracer.

The tracer tool also supports stateless applications that you build on Cosmos React UI and that rely on Digital Experience (DX) APIs. In Cosmos React UI applications, multiple API requests can be active simultaneously during a single browser session. For greater clarity, the Tracer window categorizes the events by request ID.

1. In the footer of Dev Studio, click the **Tracer** icon.
2. In the **Tracer** window, analyze the Tracer output:
    - To learn more about the event and view a Java stack trace if an exception occurred, click the value in the **Line**, **Rule#**, or **Step** column.
    - To view the properties that were on the step page when the step began, click the value in the **Step Page** column.

- To open the corresponding rule, click the value in the **Name** column.
- To view the contents of the parameter page, click the value in the **Line** column, and then click **Parameter Page Name**.
- To view the contents of the primary page, click the value in the **Line** column, click **Primary Page Name**.

Hybrid mode *(on page)*
Troubleshooting newly created rules *(on page 346)*

Tracer results window

The Tracer helps you debug your applications and improves the quality of your software for users. By analyzing results in the Tracer window, you can obtain more information about how your application runs in the system, as well as find and fix any issues.

The way the tracer results window groups information depends on the type of your application. If you use a standard Pega Platform application, the header row of the tracer results window contains the Pega node hash and computer name. For cluster-wide tracing of service rules, the tracer displays a separate section for each node that reported events.

For applications that rely on the Cosmos React, the Tracer results window groups the data by request ID. Because in the Cosmos React applications multiple requests can be active simultaneously, grouping events by requests ID provides more clarity, so that you can find relevant information faster.

The tracer records selected events from the rule executions, database operations, and other event types that you select when you configure the Tracer. Each row represents an event and is color-coded in the following way:

- Gray rows represent activity processing.
- Orange rows represent events from flow, decision, or declarative rules.
- Light blue rows represent PegaRULES database and cache operations.

The following table contains columns that you find in the Tracer results window. The values of some columns change depending on the type of event that you trace.

Col-umn	Description
Line	Number of events traced, where 1 marks the oldest event.
Thread	The Thread object for the step.

Column	Description
Int	The interaction number for the step. The system starts tracing the total number at login and resets upon logout.
Rule#	Count of distinct activities that are traced. The count is not reset to zero if you clear all the events. When a single activity runs again later, the previously assigned number is repeated. Rules other than activities are not assigned a number.
Step Method	For an activity, the value indicates the method that you associate with a step. For a declarative rule or decision rule, the value indicates the start or end of a computation. For a when condition rule or Boolean expression, the value identifies the rule name or the expression. When you trace a database operation, this column lists the specific operation performed on the database, for example commit, insert, or update.
Step Page	Name of the step page. If the Step Page column of this step is blank, the window displays the `=unnamed=` value. Step pages that have messages set on them, such as error messages, have a bright orange background. When you trace a database operation, this column lists the database table affected by the database operation in the Step Method column.
Step	Step number of this step. If two or more rows appear with the same step number, an iteration is in process at that step.
Status	Status of the method in the step that the system derives from the *pxMethodStatus* property, such as `Good`, `Fail`, or `Warn`. A red background marks steps that fail and that are not addressed by a transition. `Exit Iteration` marks the end of an iteration step. In this context, a red Fail row indicates an unhandled exception condition. If a method returns a `Fail` status but the step contains a transition, the Tracer row displays the status as `Good` and has a gray background. This behavior is consistent with the processing status that the next activity step to run perceives. The `Good` status reflects that the activity notes any existing error condition.

Column	Description
	When you trace a database operation, this column lists the number of bytes involved in a write to the database.
Watch	Properties for which you set up a watch variable.
Event Type	Type of event or rule, such as `Step Begin`, `Step End`, `Activity End`, `Constraint`, `Expression`, `DecisionTree`, or `MapValue`. The `Begin` and `When End` events identify the start of a when condition rule or a similar test, such as in a precondition or transition.
Elapsed	For `Step End` and `Activity End` rows, the value defines the elapsed time in seconds for the step. A Tracer operation might decrease the time interval value.
Name	Full name of the rule that Tracer traces, with all key parts. When you trace a database or cache operation, the **Name** and **RuleSet** columns are combined. The text in this combined column provides details on the rules that the system searches for in the cache, or on the size of the database operation.
Ruleset	Ruleset and version that contains the rule that Tracer traces.

Hybrid mode *(on page)*
When condition rules *(on page 261)*
Decision trees *(on page 186)*
Decision tables *(on page 174)*
Creating an activity *(on page 100)*
Viewing Tracer results in the Tracer window *(on page 367)*

Offline debugging by using Pega-TracerViewer

The Pega-TracerViewer tool presents and summarizes Tracer output data, that is saved in an .xml file, in an interactive table or tree format. By using TracerViewer, you can analyze the Tracer data offline to understand what happens on your system, determine where performance bottlenecks occur, and diagnose additional issues.

Note: The Pega-TracerViewer requires Java for launch and is provided as-is in an open source GitHub repository. The viewer is not a licensed Pegasystems product. Ensure that you download and use the latest version of the Pega-TracerViewer tool. The `README.md` file might include

information about additional prerequisites, for example a Java version that the latest Pega-TracerViewer requires. For more information, see Pega-TracerViewer.

After you run the Tracer tool on Pega Platform™, save the output as an `.xml` file to your local system, and then open it in the viewer.

Key features of Pega-TracerViewer

By using the TracerViewer, you can perform the following actions:

- View the Tracer data offline after the test is complete.
- Organize the data in a table or in tree views.
- Sort the data, for example, by the slowest call.
- Search the Tracer tool output data.
- View Performance tool statistics for each interaction.
- View alerts and SQL statements in the context of where and when they occur.
- Group traced events by a key, such as a unique SQL statement, stream, activity, or method.

Starting with the 3.3 version, the TracerViewer supports standard Pega Platform applications that rely on Cosmos React. The output depends on the type of application that you trace. For Cosmos React, the output additionally includes DX API interaction IDs and DX API paths.

Related information

Tracer best practices *(on page 348)*
Log files tool *(on page 384)*
My Alerts display *(on page 390)*

Debugging applications by using Pega-TracerViewer

Fix issues, prevent bottlenecks, and analyze events in your application by using the Pega-TracerViewer tool. You can generate Tracer output in an `.xml` format so that you can continue debugging offline or share the results with other developers.

The viewer runs as an executable file from a directory on your operating system.

Obtaining and starting Pega-TracerViewer

1. Download the latest version of the viewer from https://github.com/pegasystems/pega-tracerviewer/releases.
2. Extract the files into a directory on your local operating system.

Ensure that you maintain the viewer's directory structure.

3. Open the `README.md` file for instructions on how to launch the Pega-TracerViewer.

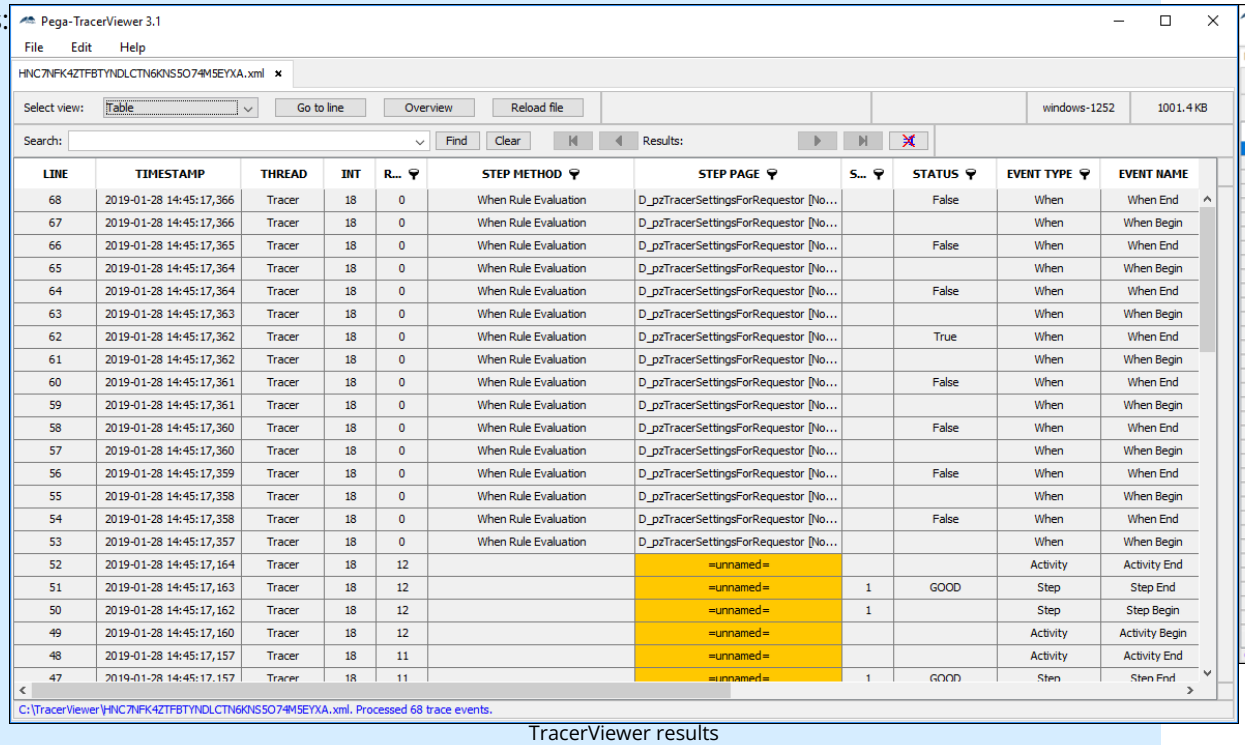
Viewing Tracer output data in the TracerViewer

4. Start the Tracer tool and configure the tool to capture the information about the areas that you want to investigate, for example, the event types to trace.
   For more information, see Configuring Tracer settings *(on page 354)*, Setting breakpoints in the Tracer tool *(on page 362)*, and Setting watch variables in the Tracer tool *(on page 363)*.

> **Note:** If you plan to analyze the results in the tree view, in the Trace Options window, for events such as activities that have Begin and End options, either select or clear both check boxes to ensure that the viewer nests the events correctly in the tree view.

5. From Pega Platform, run the process that you want to investigate.
6. In the **Tracer** window, click **Save** to download the .xml file to your downloads directory.
7. In the header of the TracerViewer, click **File > Load Pega Tracer File** File.
8. In the file directory window, navigate to the .xml file with the Tracer output, and then open the file.

You can view the Tracer results in offline mode, as displayed in the following figures:



TracerViewer results

Related information

Tracer best practices *(on page 348)*
Log files tool *(on page 384)*
Viewing and resolving errors *(on page 396)*

Clipboard tool

Every connected Pega Platform requestor (including browser-based users, even unauthenticated guest users) has an associated temporary memory area on the server known as the clipboard. The clipboard has a hierarchical structure, consisting of nodes known as pages, most of which have a name and an associated class. Pages act as buffers or temporary copies of object instances (of that class) that are copied from, or might later be stored into, the Pega Platform database or another database.

Use the Clipboard tool when developing and debugging to:

- Examine property values and messages associated with them
- Find the information necessary to reference a property — the page and property names
- Quickly update, add, or delete page properties
- Quickly start activities
- Search for properties or property values in all or selected pages

> **Note:** When using this tool for Customer Service chat interactions, be aware that a single browser session can be served by multiple requestors.

Access the Clipboard tool by clicking **Clipboard** in the Dev Studio developer toolbar. As you interact with the Clipboard tool at your workstation, you see a static snapshot copy of your clipboard, which resides in memory on the server.

Required privileges

The Clipboard tool is available only to users who have access to the `@baseclass.clipboardViewer` privilege. Action menu items that update the clipboard contents are available only to users who hold the `@baseclass.clipboardViewerUpdate` privilege.

The standard access role `PegaRULES:SysAdm4` provides these privileges.

Clipboard tool for debugging

Several methods manipulate the clipboard. As you execute activities that affect your clipboard, you can examine the results with the Clipboard tool.

For example, when you execute an activity that contains a Page-New method, you can see the resulting named top-level page in the User Pages section. If you execute an activity that uses the Property-Set method (on a named clipboard page), you can see the new values.

Mobile Clipboard

Pega Mobile offers a clipboard view similar to the clipboard that you use on your Apple Safari desktop emulator in Dev Studio. Useful for trouble-shooting and debugging your mobile app, the clipboard is grouped by User, Declare, Linked Property, and System pages.

The Mobile Clipboard is available only to users who have access to the `@baseclass.clipboardViewer` privilege. The access roles `PegaRULES:SysAdm4` and `PegaRULES:ProArch4` provide these privileges.

In the Safari desktop emulator, access the clipboard by using a tap and hold gesture on the mobile app. Display the clipboard for the main content panel and modal/overlay windows. Tap and hold does not display the clipboard for the main navigation panel, the toolbar menu, or a menu bar in a section. You can also add the clipboard as a navigation menu item.

See the Pega Community article *Mobile Clipboard*.

Application debugging by using the Tracer tool *(on page 347)*

Using the Clipboard tool

The Clipboard tool displays two panels.

- The left panel displays, for a selected thread, the page structures in a hierarchical tree format. As you interact with Dev Studio, you create new Threads in a variety of ways. These include starting user portals, opening rules or landing pages in portal tabs, or creating work items in a process or a wizard.
- The right panel contains the names and values of Single Value, Value List, and Value Group properties and messages for one page selected in the left panel.

To interact with the clipboard:

1. Select a Thread from the Clipboard header. Every Thread in your requestor session has a clipboard context. Each open tab on Dev Studio has a dedicated Thread. Thread names correspond to their tab names; work item Threads are identified by their IDs. By default, the thread for the selected tab appears when you open the clipboard.
2. Select an aggregate object in the left panel, which may be of mode Page, Page List, Page Group, Value List, or Value Group.
3. Review or interact using menus and links in either panel.

Review page structure in the left panel

The left panel presents the entire clipboard for the currently selected Thread as a tree, growing from the left.

Click a name or expand arrow to display any node in the left panel to view the page structure it contains. Click again or click a collapse arrow to hide) the pages within a page.

The clipboard contains three groups of pages:

- `User Pages` – Top-level pages created by your normal processing, sorted alphabetically by page name. For more information on adding user pages to facilitate testing, see Adding a user page to the clipboard *(on page 381)*.
- `Data Pages` – Read-only data pages (created if necessary when accessed, defined by data pages rules).
- `Linked Property Pages` – Read-only pages retrieved because a harness, section, flow action or other rule contained a property reference based on a linked property. These pages are read-only, and never updated. They are removed automatically whenever your requestor performs any Commit operation.
- `System Pages` – Top-level created pages created during authentication. These correspond to the organization, division, access group, application rule, and organization unit of the operator. None of the pages are available to guest users.

Three single, system-maintained pages are always present and are displayed when you click the **Show Advanced** link at the bottom of the tree:

- `pxThread` – Known as the thread page, it provides the context of clipboard.
- `pxRequestor` – Known as the requestor page, it contains information about your access roles, RuleSet list, and HTTP protocol parameters.
- `pxProcess` – Known as the process page, it contains information about the server's Java Virtual Machine and operating system.

Each top-level page may contain other pages that in turn may contain other pages, and so on. For most pages, the class of the page appears in parentheses after the page name and a single space. (For group and list properties, additional parentheses appear around subscripts.)

Every embedded page is defined by a property. Properties of mode `Page`, `Page Group`, and `Page List` may appear as embedded on the clipboard. Pages may contain properties (of any mode) and messages.

The branch structure of the tree in the left panel directly corresponds to the fully qualified name of a property reference. For example, the property reference:

```
pyWorkPage.pxFlow(VacationRequest).pyConfirmationNote
```

identifies a `Single Value` property named `pyConfirmationNote` embedded on a page named `pxFlow("VacationRequest")`, an element of a `Page Group` named `pxFlow` on the top-level user page named `pyWorkPage`.

Square brackets and the term [Refers to ..] indicate a reference property. The text in the square brackets identifies the non-reference page.

Using the Tools menu

Click the Tools menu on the Clipboard header to do the following:

- Select **Analyze** to review a tabular report showing the approximate size in kilobytes of each clipboard page, the page name and class, the number of accesses (read or update), and date and time of the last access, and passivation history. If the *Collect Details* mode is enabled, the display contains stack traces showing how pages were created or deleted and historical information about pages no longer on the clipboard.
- Select **Collect Details** to enable, or disable, automatic collection of detailed clipboard size and access information. A check mark next to this menu option indicates that data collection is active. A pop-up window confirms your menu action. While enabled, select the *Analyze Clipboard* option to review the size and use of each page on the clipboard, including pages that were removed during the data collection period.

> **Note:** To eliminate unnecessary processing, disable this facility except when you need the additional detail it provides.

Using right-click command menu

You can perform the following right-click commands on a page you select in the tree. Options that update the clipboard are available only to users who have the hold the `clipboardViewerUpdate` privilege.

Command	Description
Refresh Page	Select to cause the Clipboard tool to access and redisplay only a single aggregate property and its elements.
Delete Page	Select to delete a page. Confirm the deletion in the resulting dialog box. **Note:** You cannot delete a data page.
Show XML	Select to display the page's underlying XML. Use this command to see `pz` properties and other clipboard entries that aren't truly properties.
Show JSON	Select to display the page's underlying JSON, which is sent to the mobile client.
Execute Activity	Select to test a rule without the need to create a test activity that creates the pages and properties expected as starting conditions for the rule. The selected page becomes the primary page for the activity. An Execute Activity dialog appears and displays the class of the selected page in the Page Class field. 1. Enter the Activity Name key part. 2. Click **Get Params** to enter parameter values. 3. Click **Execute Activity** to run it.

Review property values in the right panel

When you select a page in the left panel, all `Value List`, `Value Group`, and `Single Value` properties on that page appear in the right panel, sorted by property name. Property messages appear in red text.

Click a property name to open its form in Dev Studio.

Values of `TextEncrypted` properties are encrypted. They appear as blank in this display.

The following options are available in the right panel.

Option	Description
Refresh	Click to cause the Clipboard tool to access and redisplay the properties on the current page.
Edit	Select to add a single value property to a page, modify an editable property value, or delete an editable property from the page. When you select **Edit** the following occurs: • Editable property values appear in text boxes • The Add option appears above the properties list • The Delete icon appears at the end of a row containing an editable property When you are finished with your edits, click **Save** to keep your updates and save the page to the memory, or click **Cancel** to cancel them. **Note:** You cannot edit a read-only data page outside of the data page load and post-activity process.
Add	To add a single value property to a page, select the Edit option, select a row in the list, and click **Add**. Enter the property name and value in the Add properties dialog box. **Note:** You cannot add an embedded property.
Save	This option is available when you click **Edit**. Click **Save** to save your updated page to the memory. **CAUTION:** Pega Platform does not validate properties on the page. This action can introduce invalid data into the memory.
Delete	This option is available when you click **Edit**. Select a row containing an editable property and click to delete it from the page.

Option	Description
Discard	This option is available when you click **Edit**. Click **Discard** to cancel your updates.
Actions	Click to select the *Show XML*, *Show JSON*, or *Execute Activity* commands as described previously.

⚠ **CAUTION:** Changing your clipboard contents or structure may affect the integrity of your system or your application results. Change clipboard values with this tool only in a debugging situation. Deleting properties or saving altered pages may introduce invalid data into the memory.

Page and property messages

Clipboard page names with errors appear in red text on the left panel. Click the name to view messages associated with properties on this page in the right panel.

A page message is a text clipboard value that is generated by the system and associated with a page. Similarly, a property message is a text clipboard value that is generated by the system and associated with a property. These messages can convey error conditions, progress, or exceptions to a user.

Although messages appear as values on the clipboard, they are not defined through properties. Generally, a clipboard page containing messages cannot be saved, because typically, the message indicates that the page, or a property on it, is invalid. The value of a property might not meet the requirements of a permanent instance of the page's class because of missing or incorrect data.

Property messages are associated with a single property and value. They might indicate that the property value is not valid. Page messages are associated with an entire page.

When a workstation user submits an HTML form, previous page messages corresponding to the input are cleared, and property messages are cleared for any value that changed.

Properties and pages not found on the clipboard

The following items are not visible on the clipboard:

- Properties with Internal status (properties with names that start with `pz`). These standard properties support internal Pega Platform operations. To view these properties you can override the `@baseclass.pyShowInternalProperty` when rule or right-click in the left panel and select *Show XML*.
- Primary pages of some activities. The parameter page of an activity. Use the Tracer to view these pages.
- Properties of mode `Java Object`, `Java Object Group`, and `Java Object List`.

Note: The `Application` page contains much, but not all, of the properties that make up the requestor's application rule.

Clipboard tool *(on page 373)*
Adding a user page to the clipboard *(on page 381)*
Page names and reserved pages *(on page)*

Setting property values with the Clipboard tool

Maintain the best quality of your application by testing it with the Clipboard tool. To test or debug your application, set values for the properties that your application logic uses, and then check whether the application logic behaves as expected.For example, to test the logic of a child case type, before you finish creating its parent case type, you can provide a sample of the data that the child case type normally inherits from its parent.

For relevant training materials, see the Clipboard data module on Pega Academy.

1. In App Studio, navigate to an element that you want to test.
2. In Dev Studio, navigate to an element that you want to test.
3. In the footer of App Studio, click **Toggle runtime toolbar > Clipboard**.

   The **Clipboard Viewer** window opens. The navigation pane lists pages in the memory that are associated with the thread that you select in the **Thread** list.
4. In the footer of Dev Studio, click **Clipboard**.

   The **Pega Clipboard** window opens. The navigation pane lists pages in the memory that are associated with the thread that you select in the **Thread** list.
5. In the navigation pane, click the page that stores the values that you want to edit.
6. In the **Clipboard page** section, click **Edit**.
7. In the Value column, provide a value for the property that you want to test or debug.

8. Click **Save**.
9. Populate the process that you want to test or debug with the new clipboard data.
10. Verify that the field values match the test values.

Clipboard tool *(on page 373)*
Using the Clipboard tool *(on page 374)*

Adding a user page to the clipboard

Quickly exercise your application logic in different scenarios by manually adding user pages in the Clipboard tool to mock the state of the application. For testing purposes, you can add user pages manually when your application is still in development. As a result, development efforts can continue in parallel rather than in sequence, which helps you deliver working software more rapidly.For example, in an insurance claim case, you want to test a process that includes a Load data page step, but the case currently lacks a data model. In this situation, you can use the Clipboard tool to create mock pages to verify that the process runs correctly against a proposed model.

You add a user page during application development in the following situations:

- To avoid building logic into your application that provides mock values for application validation.
- To manually test your application against a data structure that is not available yet.

For relevant training materials, see the Clipboard data module on Pega Academy.

1. In App Studio, navigate to an element that you want to test.
2. In the footer of App Studio, click **Toggle runtime toolbar > Clipboard**.

   > The **Pega Clipboard** window opens. The navigation pane lists pages in memory that are associated with the selected thread.

3. In the **Thread** list, select the thread in which you want to add a user page.
4. In the navigation pane, right-click **User Pages**, and then click **Add Page**.
5. In the **Add New Page** dialog box, in the **Page Name** field, enter a name for the new page.
6. **Optional:** To create a list of properties based on a specific class, in the **Page Class** field, enter or select the class to which the user page belongs.
   Otherwise, the page that you create is classless.
7. **Optional:** To provide values in the user page by using a data transform, in the **Data Transform** field, specify the data transform that supplies the initial values for the user page.

8. If the data transform sends parameter values to the user page, you can provide these values by clicking **Get Params**, and then entering the parameter values that you want to set in the appropriate fields.
9. Click **Submit**.

Clipboard tool *(on page 373)*
Using the Clipboard tool *(on page 374)*
Data pages *(on page)*

Measuring clipboard size

The clipboard display shows the contents of the clipboard, but not its size in bytes. Large clipboards can affect performance because memory in the Java Virtual Machine (JVM) supporting the Pega Platform holds the clipboards of all requestors.

You can use the Performance tool to see the size of your clipboard in bytes, or to track the growth and contraction of your clipboard over time.

1. In the developer toolbar, select **Performance**.
2. Click **Add reading with Clipboard Size**.

Use Admin Studio to determine the size of each requestor's clipboards and the size of each page. For more information, see Managing requestors *(on page)* and the Admin Studio help.

1. Click **Admin Studio > Resources > Requestors**.
2. In the options menu, click **Analyze clipboard**.

Clipboard tool *(on page 373)*
Using the Clipboard tool *(on page 374)*

Indirect pages

An indirect page is a page that the system finds by searching the clipboard at run time. The page reference to an indirect page starts with the keyword `prompt` followed by the page name.

> **Note:** This type of page is deprecated but still supported for rules that used this feature prior to Pega 7.1.

With a Call or Branch instruction, the calling activity can identify parameters to the target activity by using indirect page references. For example, if you identify a page as promptALPHA, the system searches the clipboard for pages named ALPHA.

You cannot use symbolic page names (such as `primary` or `param`) after the keyword `prompt`; an explicit page name is required.

You can use indirect pages in these rule types:

- Activities
- Correspondence
- Harness
- HTML
- JSP
- Paragraph
- Parse Infer
- Parse Structured
- XML

On the Pages & Classes tab of these forms, indicate that a page referenced in the rule is an indirect page by setting the Mode field to `prompt`.

Page names and reserved pages *(on page)*

Viewing clipboard pages created by unit testing a rule

After running a rule, you can open the Clipboard tool and examine the output as it appears on the resulting clipboard pages.

The following clipboard pages are created when you unit test a rule:

- RuleToRun — The clipboard representation of the rule that you tested.
- runRulePage — Holds the output from the rule.
- temp_ pages — Pages created or copied by the Run Rule feature when it ran the rule. The names of these pages begin with the literal temp_.
- pySimulationDataPage — For service rules, a page of the helper class `Data-Admin-IS-ClientSimulation`. Contains information about the simulated request and response.

Unit testing individual rules *(on page 345)*

Troubleshooting long-lived clipboard pages

Clipboard pages that remain unused for long periods could indicate a design or implementation flaw that can hurt performance. For example, the application created a page but neglected to use the Page-Remove method to remove the page after it was no longer needed.

When building or testing an application, you can discover whether your own processing has created such "orphan" or "near-orphan" page.

1. Select **Clipboard** from the developer toolbar in Dev Studio.
2. Once the clipboard is open, select **Tools > Analyze**.
3. In the resulting grid of data, look in the **Est Size (KB)**, **Last Passivation**, and **Last Activation** columns to identify any pages that were automatically passivated but never reactivated.
4. Research whether such pages are needed at all, and whether they can be removed at specific points in the application, to reduce clipboard size.

Understanding passivation and requestor time-outs *(on page)*
Reporting on passivated requestors *(on page)*

Log files tool

In the Log files tool, you can view or download the current log files on the server node that you currently access. Analyzing log files helps you make informed decision when you manage your applications, as well as identify any issues. As a result, you deliver an application that runs correctly and avoid exposing your users to errors.

> **Note:** If you are a Pega Cloud customer, you can download the log files by submitting a service request (SR). For more information, go to Pega Support.

For on-premises environments, the available log files depend on the contents of the `prlog4j2.xml` file for the current node. The log files can include:

PEGA

A text file that contains warnings, errors, and information messages about internal operations.

ALERT

Performance-related alerts that are triggered by prconfig settings, or implicit, default values. A text file with fields separated by a single asterisk character.

ALERTSECURITY

Alerts identified by the prefix SECU that suggest incorrect configuration of the Internet Application Composer facilities, or overt attempts to bypass system security features through URL tampering. For more information, see *Performance alerts, security alerts, and AES* on Pega Community.

BIX

Files that were created during an extract of operation rules by the optional Business Intelligence Exchange product.

CLUSTER

A file that includes information about the setup and run-time behavior of the cluster, and other information provided by the clustering technology.

DATAFLOW

A file that includes data flow events in the system, such as a number of running threads, a number of newly created threads, or a number of stopped threads.

MOBILE

A text file that includes information about running of your mobile application. You can retrieve logs for a user that you select and specify how much information you want to include in the file. For example, the file can include information about failures during logging to a mobile application in an offline mode.

Failures of data synchronization of your mobile application are part of the PEGA0066 alert. For more information, see *PEGA0066 alert: Mobile App Data-Sync Failure* on Pega Community.

PegaRULES-SecurityEvent

A file that contains information about security-related events that occur in the system, such as login failure or a password change. When security-related events occur in unusually large numbers or in suspicious patterns, they might represent a security issue.

For relevant training materials, see a Reviewing log files module on Pega Academy.

Viewing and resolving errors *(on page 396)*
Application debugging by using the Tracer tool *(on page 347)*

Viewing logs

Analyze information about your system by viewing log files. Log files gather data about internal operations, system performance, or security so that you can make informed decisions when you manage your applications.

The log files match the current operator ID for all the sessions on the server node.

1. In the header of Dev Studio, click **Configure > System > Operations > Logs**.
2. On the **Logs** tab, in the **Log utilities** section, click **Log files**.
3. In the **Log File Download** window, click a file name to view one of the logs.
   The initial display shows log file entries for your operator ID, 25 lines at a time.
4. **Optional:** To adjust the display to your needs, change or remove the log filters:
   - To change the number of lines, in the **Lines Per Page** field, enter a new value, and then click **Apply**.
   - To change the number of pages that the window displays, in the **Number of Pages Presented**, enter a new value, and then click **Apply**.
   - To filter the logs by a user, in the **Filter by** field, enter an operator ID, and then click **Apply**.

     If you clear the **Filter by** field, the results include logs for all users in the system.
5. After you analyze the logs, return to the list of logs by clicking **Back**.

My Alerts display *(on page 390)*

Downloading log files

To view your log files later or to share the logs with other team members, download the log files. As a result, you can analyze system performance at a time that is convenient for you.

1. In the header of Dev Studio, click **Configure > System > Operations > Logs**.
2. On the **Logs** tab, in the **Log utilities** section, click **Log files**.
3. In the **Log File Download** window, in the **Download** column, select the type of file that you want to save on your machine:
   - To download a log as a text file, with `.txt` as the file type, click **text**.
   - To download a log as a `.zip` file, click **zip**.
4. In the **Sign in** dialog box, enter the authentication information for your application server, and then click **Sign in**.
5. In the dialog box, select where you want to save your file, and then click **Save**.

Viewing logs *(on page 385)*

Viewing log files in an external log viewer

You can view log files using an external log viewer, such as Kibana. An external log viewer can help you visualize logs and monitor potential issues.

The external log viewer must be installed and its URL configured on the **Resource URLs** tab of the System Settings landing page before you can use it.

1. Click **Configure > System > Operations > Logs**.
2. Click **External Log Viewer**.

> **Note:** If the log viewer has not been configured, click **Configure URL** to specify the log viewer's URL.

Log levels for log categories

You can select which log messages your log files include by creating custom log categories and associating log levels with your categories.

The log level of a category determines the type of messages that the log file includes. When you create a log category and set its default log level, these selections apply to all nodes in the cluster and they persist until they reset. The default log level applies to all the loggers that you associate with your category.

Log levels

When you choose a log level, messages from that level and above are written to the log file. For example, if the log level is set to ERROR, the log file includes log messages with a severity of ERROR and FATAL. The following list orders log levels from highest (most severe) to lowest (least severe):

OFF

The log file does not include any messages.

FATAL

The log file includes messages about severe errors that can cause the application to terminate.

ERROR

The log file includes messages about serious errors that might allow the application to continue running.

ALERT

The log file includes Pega Platform-specific messages that indicate that a performance threshold has been exceeded, or that an event that affects the performance has occurred.

WARN

The log file includes messages about situations that might have an adverse performance implication.

INFO

The log file includes messages about a run-time events, such as startup or shutdown, that have occurred.

DEBUG

The log file includes messages about informational events that are useful for debugging.

ALL

The log file includes messages for all levels.

> **Note:** Adjust a log level to your specific needs. The log level that is too verbose may cause performance issues.

You can apply the same log levels to individual loggers. For example, if you want a log category that groups loggers that correspond with agents to work on the FATAL level, but you need one logger within this category to work on the ALL level, you can change the level of the individual logger.

Creating log categories in Dev Studio

Structure log messages into categories by creating custom log categories. Grouping loggers into categories makes it easier to manage log levels of loggers, because you do not have to remember the names of individual loggers.

1. In Dev Studio, click **Records > SysAdmin > Log Category**, and then click **Create**.
2. Enter a short description and a logger category name, and then click **Create and open**.
3. Enter a description of your log category.
4. From the **Log Level** dropdown list, select a default log level for your category.
   The default log level determines which log messages are written to the log file and applies to all the loggers associated with the category. For more information, see Log levels for log categories *(on page 387)*.
5. Associate loggers with your category by clicking **+Add logger** and entering logger names in the **Logger Name** field.
6. Click **Save**.

Renaming Pega logs

If two or more Pega Platform servers are installed in a single application server instance, they write lines to a common Pega log file. To prevent this from occurring, you can change the name or path of the Pega log for a node by modifying the `prlog4j2.xml` file.

In the following example, the log file name starts with PEGABLUE:

```
filePattern="${sys:pega.tmpdir}/PegaBLUE-%d{MM-dd-yyyy}-%i.log.gz">
```

Viewing logs *(on page 385)*

Rolling a log file

You can update the **`prlog4j2.xml`** file or dynamic system setting to cause a new log file to be created at the start of each day or on a periodic basis, rather than only at startup. This is known as "rolling" the log file.

Pega Platform uses the RollingRandomAccessFileAppender from Log4j 2 as the default file appender. You can use a different log file appender. For more information, refer to the Log4j 2 documentation.

Displaying node type in the log

You can optionally choose to display the system node type in the Pega Platform log.

The system node type is available for display in the Pega Platform log, although by default it is not shown. To display the node type in the Pega Platform log, configure the PatternLayout for your log to render the nodeType property. For more information, refer to the Apache Log4j 2 documentation for PatternLayout.

Generating logs for autopopulated properties

For more efficient debugging of autopopulated properties, generate logs that contain events connected with specific properties that you select. Consequently, the troubleshooting process is better targeted and you can more quickly identify and fix issues in your application.Autopopulated properties hold metadata on clipboard pages and are declarative, which means that the system triggers autopopulated properties automatically. Debugging autopopulated properties can be challenging without specific information about metadata that the properties store, so creating logs that include this information can accelerate the troubleshooting process.

You can generate logs for autopopulated properties by setting the logging level to `DEBUG`. For more information about logging levels, see Log levels for log categories *(on page 387)*.

1. In the navigation pane of Admin Studio, click **Resources > Log categories**.
2. In the **Category name** column, locate the `pxAutoPopulate` category.

> **Tip:** To find the category faster, you can enter a logger of the *pxAutoPopulate* category in the **Search categories by loggers** field. A sample logger of the *pxAutoPopulate* category is *autopopulate.lifecycle*.

3. In the row of the `pxAutoPopulate` category, click **More > Change log level**.
4. In the **Modify log level** dialog box, in the **Log level** list, select **DEBUG**.
5. In the **Reset in (hours)** list, select the number of hours that pass before the log resets.
6. **Optional:** To filter the results that the log file includes, define filter criteria:
    - To generate logs only for a specific requestor session, in the **Operator** field, enter the operator ID for which you want to generate logs.
    - To restrict logs to properties that are associated with a specific rule type, in the **Rule Type** list, select a rule type that you want to use.
    - To restrict logs to properties that are part of a specific class, in the **Class** field, enter the class that you want to use.
    - To restrict logs to properties that are associated with a specific rule, in the **Rule** field, enter a rule that you want to use.
7. In the **Auto Populate Property Name(s)** field, enter the name of the property for which you want to generate logs.

> **Tip:** You can generate logs for multiple properties by entering the properties' names in a comma-separated list. If you skip specifying autopopulated properties, the system generates logs for all autopopulated properties in the current cluster.

8. Click **Submit**.

My Alerts display

You can view the current Alert log on the server node produced by your own requestor session by using the My Alerts display. Select **Issues** from the Dev Studio developer toolbar.

> **Note:** This display is limited to the current Alert log; alerts in older logs are not visible. Depending on your log file system settings, a new log may start each time a node is started, at a specific time each day, or on some other basis.

The following summary information is displayed for each alert:

- **Date and Time** – Date and time of the alert, converted from GMT to the time zone of the server.
- **Alert Type** – A text description of the alert type. For example, the text `BrowserInteraction` corresponds to alert type PEGA0001, interaction times. Other types are `Database` (`PEGA002` to `PEGA007`) .
- **Value** – The measured value of the Key Performance Indicator, recorded in seconds, as a count, or in bytes.

- **Interaction** – The sequence number of the browser-to-server interaction since the requestor session began, as indicated by the Performance tool display.
- **Work Pool** – Work pool that the requestor is using, or `none`.
- **Last Input** – A portion of the URL received in the most recent interaction. Typically the text Stream= or Activity= and the name of an activity or stream rule run by the URL.
- **First Activity** – For alerts from interactive requestors, the first activity or stream run by this requestor in the current interaction. (This activity or stream may or may not have caused the alert.)

The alerts are ordered by date and time; the first row is the most current. Click the arrow in the first column to expand the row to view more details about the alert.

> **Note:** When using this tool for Customer Service chat interactions, be aware that a single browser session can be served by multiple requestors.

Display options

By default, 20 alerts are displayed on the page.

- Click a numbered link, **next>**, or **<previous** to scroll back and forth through the list.
- Click **This Session Only** to limit the display to alerts produced by your current requestor session.
- Click **All My Sessions** to include alerts from the current session and other sessions with your current Operator ID.
- Click **Performance** or **Security** to toggle the display between security-related alerts (SECnnnn) and performance-related events (PEGAnnnn).
- Click the **Options** link to view or set log filtering criteria. You can set the following options:
    - **Lines per page** – Enter the number of alerts to display on a page. The valid values are 1 through 200.
    - **Number of pages presented** – Set a maximum number of pages to present as numbered links, between 2 and 20.
    - **Filter by** – Optional. Enter a text string to limit the display to only alert lines containing an exact match anywhere within the line. Leave blank for no filtering. Case is not significant. For example, enter Smith@Alpha.com to find lines containing this value, or containing SMITH@alpha.com.
    - Click **Apply** to update the display with the new filter criteria.

Parsing the alert log

You can import the alert log into Microsoft Excel for viewing, searching, sorting, or other analysis using the Text Import Wizard in Microsoft Excel. Fields in the alert log are delimited by an asterisk.

1. Start Microsoft Excel.
2. Click **File > Open**.
3. Navigate to the `PEGA-ALERT-ZZZZ.log` file and click **Open**.
4. Click Delimited in the **Original data type** group.
5. Click **Next**.
6. Select **Other**.
7. Enter a single asterisk as the delimiter.
8. Click **Next**.
9. Click **Finish**.

Alert format *(on page 393)*
Summarizing the alert log *(on page 393)*

Suppressing sensitive data in alerts

The alert log identifies the names of activities and other rules that have been executed. By default, PEGA0002 to PEGA0007 alerts also show parameter values and SQL statements containing property values. You can configure the `prconfig/suppressInserts/default` and `prconfig/includeParameterPage/default` dynamic system settings (DSS) so that parameter values and other potentially sensitive data values are omitted in prepared statement inserts:

1. Add the following DSS:
   - *prconfig/alerts/database/operationTimeThreshold/suppressInserts/default* with the value of `true` and the owning ruleset *Pega-Engine*
   - *prconfig/alerts/general/includeParameterPage/default* with the value of `false` and the owning ruleset *Pega-Engine*

   For more information, see Creating a dynamic system setting *(on page)*.
2. Stop and restart or redeploy the system.

Alerts *(on page 393)*
Parsing the alert log *(on page 391)*
Summarizing the alert log *(on page 393)*
Alert format *(on page 393)*
Configuring dynamic system settings *(on page)*

Summarizing the alert log

You can parse, consolidate, and summarize alert logs using the PegaRULES Log Analyzer or services such as Elastic Stack.

For information on the PegaRULES Log Analyzer, see the Pega Community articles *Understanding the PegaRULES Log Analyzer* and *How to use the PegaRULES Log Analyzer*.

Alerts *(on page 393)*
Parsing the alert log *(on page 391)*
Suppressing sensitive data in alerts *(on page 392)*
Alert format *(on page 393)*

Alerts

You can view or download the current alert log to see important notifications about the functioning of your system.

Threshold values in dynamic system settings (DSS) affect the operation and criteria of many alerts.

If an article about an alert lists default `prconfig.xml` settings, as a best practice, you can change those settings by using DSS. For example, to enable the PEGA0001 alert, you create the *prconfig/alerts/browser/interactionTimeThreshold/enabled/default* DSS, and then set its value to `true`. For more information, see Creating a dynamic system setting *(on page)*.

Note: These changes take effect the next time your system starts.

For a list of performance and security alerts in Pega Platform, see Alerts overview.

My Alerts display *(on page 390)*
Alert format *(on page 393)*
Parsing the alert log *(on page 391)*
Suppressing sensitive data in alerts *(on page 392)*
Summarizing the alert log *(on page 393)*
Configuring dynamic system settings *(on page)*

Alert format

Each line of the log contains multiple fields, delimited by a single asterisk character. Each line contains the following fields. The first column of the table is the Excel column after you have imported the file into Excel.

Column	Field	Description
A	Time stamp	Date and time of the alert event in UTC format.
B	Version	Displays the header version for the alert. For example, "8".
C	Message ID	Identifier of alert type, for example `PEGA0011`.
D	KPI Value	Observed duration in milliseconds or count of value, known as the key performance indicator.
E	KPI Threshold	Threshold value for this alert type, in milliseconds or count.
F	ServerID	Node ID for the node that is the source of the alert event.
I	Requestor ID	Session identifier; the first letter (A, B, H, or P) identifies the requestor type. • A - requestor is being used by an application • B - batch requestor used by agent processing • H - requestor is being used by a user (HTTP interaction) • P - requestor is used for portlet support
J	User ID	Operator ID of requestor, or `none`.
K	Work pool	Work pool that the requestor uses, or `none`.
L	Rule application name version	Rule application and version used at the time of the alert.
M	Encoded Ruleset	Hash code identifying the requestors ruleset list.
N	Checkout Enabled	`Y` indicates that the activity in the Activity field is in a personal ruleset; that a developer has checked out the activity.
O	Interaction number	For alerts from browser sessions, the interaction number (as it appears on the Performance tool display).
P	Correlation ID	ID used to filter messages arriving on the request queue. By default, the requestor ID.

Column	Field	Description
Q	Sequence	Sequence number for this alert in the log; unique within this log.
R	Thread	Internal name of the PRThread instance, or NA if this alert is not from a Thread.
S	Pega Thread Name	The name of the Pega thread on which the alert happened.
T	Logger	Java class that produced the alert instance.
U	Stack	An indicator in the engine process that shows the processing state when the message occurred. (Not available for all alerts.)
V	Last Input	For alerts from interactive requestors, a portion of the URL received in the most recent interaction. Typically the text Stream= or Activity= and the name of an activity or stream rule run by the URL.
W	First Activity	For alerts from interactive requestors, the first activity or stream run by this requestor in the current interaction. (This activity or stream may or may not have caused the alert.)
X	Last Step	If the First Activity field identifies an activity, this field identifies the step number last completed, when available.
Y	Reserved	Reserved.
Z	Reserved	Reserved.
AA	Reserved	Reserved.
AB	Reserved	Reserved.
AC	Trace List	Colon-delimited string containing the most recent 50 operations of this requestor before the alert event. Each operation consists of two subfields in the format number:value:.
AD	PAL data	For some Message ID types, a set of selected of Performance tool statistics at the time of the alert. A colon de-

Column	Field	Description
		limited string in the format propertyname:value:. Values are in milliseconds, bytes, or counts.
AE	Primary page class	The class of the primary page at the time of the alert.
AF	Primary page name	The name of the primary page at the time of the alert.
AG	Step page class	The class of the step page at the time of the alert.
AH	Step page name	The name of the step page at the time of the alert.
AI	Pega stack	The Pega stack at the time of the alert.
AJ	Parameter page	The clipboard page data found on the parameter page at the time of the alert.
AK	Line	A text description of the alert.

Parsing the alert log *(on page 391)*

Viewing and resolving errors

Errors appear if you save a form and the system detects one or more errors.

You must correct any errors before you can save the form by completing the following steps:

1. Use the information displayed in the error section of the header to understand what the issue is and how to resolve it.
2. Use the indicator icons to quickly see which tab has an issue and what fields on that tab are involved in the error condition.
3. Make your changes to the affected areas on the form and click **Save**.
   The header automatically updates to reflect the new error count. After you resolve all errors, errors no longer appear in the header, and you can save the form.

Viewing in-progress and completed wizards

You can find all wizards that have been started on the **All Wizards** landing page. You can use this landing page to clean up incomplete wizards.

1. In Dev Studio, click **Configure > Application > Tools > All Wizards**.
2. Enter keywords in the fields to filter results. By default, only unfinished wizards are listed.
3. Click a wizard name to open it.
4. Follow the steps of the wizard to resolve it.

Creating connector simulators

With the Connect Simulator functionality, test connect rules in flows and from data pages before the actual data source is implemented, or when it is not available.

To simulate a connector:

1. Create an activity that sets values to the properties that are to be returned by the connector. For more information, see Creating an activity *(on page 100)*.
2. On the **Service** tab of the **Connector** rule form, click the **Simulations** button to access the **Simulations** form.
3. On the **Simulations** form, click **Add a row**.
4. To enable simulation, in the **Simulation Activity** field, enter or select an activity.
5. Select either the **Global** or **User Session** options.
6. Click **Submit**.

   When flow execution reaches the connector shape, the activity is called.

Disabling simulations

When you no longer need to simulate the connector, you can temporarily ignore all simulators by accessing the **Connect Simulator** window and clicking the **Clear Local** or **Clear Global** buttons.

You can also disable or enable simulations on the Connectors landing page.

Connect Simulator Properties

Field	Description
Simulation Activity	Activity name (second key part) of an activity that simulates the results of the connector rule identified by the previous fields in this row.
Global	Enables system-wide use of the simulator whenever flow execution reaches the connector shape. When enabled along with a User Session, the User Session overrides the Global simulator.
User Session	The simulation occurs only when the flow is started by the logged-in user. The User Session overrides a Global simulator.
	Remove the Simulation Activity.
Clear Local	Click to clear the User Session option from all Simulation Activities.

Field	Description
Clear Global	Click to clear the Global option from all simulation activities. This option is not available if the production level of the system is 5.
Apply Changes	Click to apply changes to the Connect Simulator.
Close	Click to close the Connect Simulator form.

Example

For an example of a connector simulation, see the Pega Community article *How to simulate a SOAP connector*.

Related information

Integration Resources category *(on page)*

The DateTime parse tester tool

Use the Date/Time Parse Tester tool to test whether a text input matches the DateTime patterns used by Pega Platform, which are based directly on Java standards.

1. Use the Application Explorer to open the standard activity `Code-Pega-Parse.DTParseTester`.
2. Click **Actions > Run**. A test input form appears.
3. In the **Page** list, click **Empty test page**.
4. Leave the parameter values blank. Click **Run**.
5. Complete the input fields on the form:

Field	Description
Date/Time Pattern	Enter a pattern. The count of pattern letters identifies a format: • G — Era designator, for example AD • y — Year, for example 2018 • M — Month in year, for example July or 07 • d — Day in month, for example 10 • h — Hour in A.M./P.M. format, 1 to 12 • H — Hour in day format, 0 to 23 • m — Minute in hour, for example 30 • s — Second • S — Millisecond

Field	Description
	• E — Day in week, for example Tuesday • Z — Time zone, for example Pacific Standard Time
Test Case	Enter text that may contain a date or date/time formatted with the pattern.
Locale	Select: • Use default locale — The current locale of your requestor session is used. • Specify a locale — Select a locale from a list.
Timezone	Select: • Use default timezone — The current locale of the server node is used (not the time zone in your workstation or operator ID instance) • Specify a timezone — Select a locale from a list.

6. Click **Test Parsing**.
7. The system searches the Test Case text for a conforming date or date/time value. If found, it converts the value into a Pega Platform internal format and displays the results.

Internationalization and localization *(on page)*

About the bulk Revalidate and Save tool

You can force revalidation in bulk of selected rules or data instances of a single type.

For example, you can revalidate all operator ID instances, or all flows in a specific RuleSet and version.

The Revalidate and Save tool is useful in various situations, including the following:

- After you import a .zip file archive of rules that may depend on rules previously present, or that were exported from a lower-version Pega Platform system.
- To confirm that rules pass current validation requirements.
- To propagate the effects of changes you made to some rules to other rules that reference those rules.

This operation is similar to opening and then saving each selected rule or data instance. If an instance fails validation, the previous version is unchanged.

Every saved rule or data instance was previously validated at least once, at the time it was most recently saved. The instance was valid at that time in the environment, system, and context in which it was saved. Revalidation is desirable to confirm that nothing in the current environment makes the object invalid.

If an instance passes validation, the History tab is updated with your operator ID, the current date and time, and current system in the Updated row. For rule instances, a history detail instance is added with your memo and the notation Bulk Update.

Starting the bulk revalidation tool

To revalidate rule or data instances:

1. In Dev Studio, click **Configure > System > Release > Upgrade > Validate**.
2. Complete the form. See Help — Using the Revalidate and Save tool *(on page 402)* for instructions.
3. Click **Run** to begin revalidation.
4. As it executes, the Revalidate and Save tool marks valid rules (or data instances) with a checkmark; rules not valid are marked with a red X. Hold the mouse pointer over the red X to see the error message. Click the **Open** icon to open the instance.

Note:

- If an instance fails validation, the previously saved instance remains and is unmodified.
- Depending on the number of rules or data instances selected, this processing may require minutes or hours to complete.
- Note any rule or data instances that fail validation. After making corrections, you can click **Clear Status** and rerun the test.
- For some rule types, validation of an instance may fail because another instance is invalid. Running the Revalidate and Save tool a second time may allow instances that failed the first time to validate.
- You can revalidate rules that belong to a locked RuleSets version, excluding RuleSet versions that form the Pega Platform product. The History tab and history details are updated to reflect a successful revalidation; other aspects of the rule (as visible on the form) do not change.
- Some rules may present warning messages. Warning messages do not indicate that the rule failed to validate or save. Use the Guardrails landing page to see rules that have warnings.
- For an automated approach, see the Pega Community article *How to revalidate all rules in a RuleSet*.

Validation tool

Use the Validation tool to check the validity of rules in your application. This tool ensures that all rules defined in an application are valid and contain valid rule references based on a specified application context.

The Validation tool can help you improve the quality of your application and should be used regularly. It is available for rulesets using both Application Validation (AV) mode and Ruleset Validation (RV) mode. For more information, see Ruleset validation modes *(on page)*.

Use the Validation tool:

- To identify how changes to some rules will impact other rules that reference those rules. For example, if you delete or change the return type of a rule by running the Validation tool, you will know how many rules became invalid by that change.
- To confirm that rules pass current validation requirements
- After critical changes or milestones, such as:
    - Changes to a ruleset list in the application rule
    - Changes to a built-on application
    - Before lock/export
    - After import
    - Regularly during development

How the Validation tool works

The Validation tool is available only to users who have access to the `ApplicationValidation` privilege. The access role `PegaRULES:SysAdm4` provides these privileges. The tool will quickly validate the rules in your application, ruleset, or ruleset version without re-saving them. Only rules that can be called at runtime are validated; withdrawn and circumstanced rules are not validated.

To run the Validation tool

1. Click **Configure > Application > Tools > Validation**.
2. Select the application or individual ruleset and (optionally) the version you want to validate from the **Select Validation** drop-down menu.
3. Click **Run Validation** to begin validation.

A progress bar appears, indicating:

- **Total rules** – Number of rules associated with application
- **Processed Rules** – Number of rules validated at given point in time
- **Invalid Rules** – Number of invalid rules found at given point in time

Click **Cancel** at any time to stop the execution of the validation process.

You can only run the Validation tool for the selected application. To run for another application, click the **Reset Validation** button.

Once validation is complete, a results grid displays a list of any rules that failed validation, along with their associated error messages.

- Expand a rule name to display the rule's validation error.
- Click the Rule Name to open the rule.

> **Note:** When you open the rule from the results grid, it will not contain the error messages from the validation process. To re-validate the rule, save the rule.

Using the Revalidate and Save tool

You can identify the class of the instances (such as rules or data instances) to be revalidated, and optionally a RuleSet and Version, and list instances that meet these criteria. You can also bulk validate the rules you select.

Completing the form

Complete this form to identify the objects to be revalidated.

Label	Description
Class	Select the type of object to be listed.
RuleSet Name	Optional. If you chose a rule type, you many select a RuleSet to restrict which the rules are listed.
RuleSet Version	If you selected a **RuleSet Name**, select to limit the rules to be listed to a single version. Select a version number.

Label	Description
During Update Move to?	If you selected a version number, select to cause the revalidated rules to be placed in a higher version number. Rules that fail validation are not altered. Select the new version number, from those already defined for the selected RuleSet.
Memo	Enter a brief text memo describing the purpose of this bulk revalidation. This memo appears in the detailed history of instances that are successfully revalidated.
List	Click to list the objects that meet the criteria you have entered.
Run	Click to begin processing the selected instances.
Close	Click to close the Revalidate and Save form.

Controls

Columns of the list include an **Update?** check box, the key of the object, a RuleSet and version (for rules), and the status. Check at least one box before starting the bulk validation:

- Select the **Update?** check box for a row to include this row in the validation.
- Click **Check All** to check the Update? box in all rows.
- Click **Uncheck All** to clear the Update? box in all rows.
- Click **Inverse** to invert the Update? box value in all rows.

Click **Run** to begin revalidation. Click **Stop** to pause revalidation. Results are presented in the **Results** column as a check mark or red X. Hold the mouse pointer over the red X to review at least one error.

Click the **Open** icon in any row to review or correct any that fail.

After making corrections, you can click **Clear Status** and rerun the validation test.

Results

A dynamic display identifies the key of the object being processed; a counter shows total rows and rows that failed validation.

This operation is similar to opening and saving each rule or data instance. (In addition, some but not all RuleSet prerequisites and class restrictions are checked.) If an instance fails validation, the previous saved version is unchanged.

If an instance passes validation, the **History** tab is updated with your operator ID, the current date and time, and current system in the **Updated** row. For rules, a history detail instance is added with your Memo and the notation `Bulk Update`.

Warning messages may appear. Warning messages do not indicate that the rule failed to validate or save. Use Guardrails landing page to see all warnings.

You can revalidate rules that belong to a locked RuleSets version, excluding RuleSet versions that form the Pega Platform. The **History** tab and history details are updated to reflect the successful revalidation; other aspects of the rule (as visible on the form) do not change.

About the bulk Revalidate and Save tool *(on page 399)*

Application Structure landing page

The Application Structure landing page helps you to manage and understand the rulesets, rules, access groups, and operators that make up your Pega Platform application. Options on this page display referencing applications, allow you to invite collaborators to the development process, and provides access to other applications on the system (for operators with permission to do so).

Access this landing page from Dev Studio by clicking **Configure > Application > Structure**.

RuleSet Stack tab

Access the RuleSet Stack tab by clicking **Configure > Application > Structure > RuleSet Stack**.

This tab lists the rulesets and ruleset versions that make up your current application including those inherited from any built-on applications. It also allows you to lock versions and roll (increment) them to higher versions.

The display is organized by Current Application followed by the Built on Application(s) and is filtered by ruleset and version, lock status, total number of rules, and number of checked out rules.

To open an application rule in the list, click its name.

Field	Description
RuleSet	The rulesets and versions in the application. To open a ruleset, click its name.
Lock	Contains an icon indicating the lock status of ruleset versions for the application. • — Indicates that the version for any given ruleset is locked. Also indicates a locked application rule. • — Indicates that the version for any given ruleset is unlocked. Also indicates an unlocked application rule.

Field	Description
	• — Warns you that a locked ruleset version has lower unlocked versions. Click the icon to view a list that shows you which versions are unlocked. To lock an unlocked version, select it in the list, and its rule form opens where you can lock the version. As a best practice, leave only the highest version of a ruleset unlocked. If the warning condition exists, you cannot roll (increment) a lower unlocked version using Lock and Roll functionality. • — Warns you that this ruleset version does not exist but appears in your application rule. To correct, remove the version. Click an icon to open a window that displays a list of versions, and for each version, the lock status and the number of total and checked out versioned and non-versioned rules, including ruleset, class, and organization instances.
Checked Out	The total number of checked-out versioned and non-versioned rules.
All Rules	The total number of versioned and non-versioned rules. Click a number to open a window listing the rule types and counts comprising the total. Click a row (including Total) in the list to drill down to a list of rule instances comprising a rule type.
Lock and Roll	Click to display the Application Lock and Roll window where you can review, lock current ruleset versions, and roll (increment) versions in a single step. You can optionally update your current application rule to reflect the new versions, or create a new application rule that contains the new ruleset versions.
Package and Export	For the Current Application only. Click to start the Application Package wizard, to create a product rule for this application.

In the window, select the application ruleset versions you want to lock and roll to a higher version.

Field	Description
Lock	Select the check box to indicate you want to lock the ruleset version and roll to a new version. Locking prevents a developer from saving new rule instances to this version, or updating or deleting existing instances. A icon appears if there is a condition that might prevent you from locking the ruleset version. For example: A higher locked version exists or a version currently has rules checked out. A icon appears if the ruleset version is already locked.

Field	Description
RuleSet Version	The ruleset and current version. Click the **+** icon next to the Prerequisites label to display the Current and New prerequisites for the version. To open the rule form for a current ruleset and version, click its name in the Current list. When you lock a ruleset version and select the Roll check box, you can modify the new prerequisites. By default, they are the same as the current ones. You can select different prerequisites, add additional ones, and delete ones from the New list. To add another ruleset version to the list, click the **Add a row** icon. To open the rule form for a ruleset in the New list, click the **Open the Rule Form** icon next to its name. To delete a ruleset from the New list, click the **Delete** icon next to its name.
Password	Appears when you select the Lock check box. Enter the password for the ruleset version.
Roll	Displays when Lock is selected. Used to roll the ruleset to a higher version.
Roll to Version	Displays the version that the ruleset will be rolled to unless you indicate another version number. Defaults to the next logical number in the ruleset numbering sequence.
Description	Current description of the ruleset version.
Run	Select an option to run against the selected ruleset versions. • **Do not update my application** — Locks and rolls the versions without updating the application ruleset list. • **Update my Application to include the new RuleSet Versions** — Locks and rolls the versions and updates the current application ruleset list. Prompts with an application description (current by default, which you can edit). Also prompts for a password if the current application is locked. Enter the application password. • **Create a new version of my Application** — Locks and rolls the versions and creates a new application rule. Prompts with the new application version (one increment higher by default, which you can edit), and with an application description (current by default, which you can edit). Also prompts for a password if the current application is locked. Enter the application password.

Application Structure landing page *(on page 404)*

RuleSet Prerequisites tab

Access the RuleSet Prerequisites tab by clicking **Configure > Application > Structure > RuleSet Prerequisites**.

This tab displays a tree view of rulesets and versions and their prerequisites in the current application.

- Click the **+** icon to expand the tree to show the prerequisite rulesets and versions. A version in blue text indicates that it is listed as a top-level version in the application ruleset list.
- Click the name of a ruleset in the tree to open and view its ruleset form.

Note: Rulesets using Application Validation mode do not have prerequisites.

Application Structure landing page *(on page 404)*
Application Validation mode ruleset *(on page)*

Referencing Applications tab

Access the Referencing Applications tab by clicking **Configure > Application > Structure > Referencing Applications**.

This tab lists other applications that reference the current application.

Field	Description
Name and Version	This (referencing) application name and version.
Access Groups	The total number of access groups associated with this application.
Operators	The total number of operators associated with the access groups.
RuleSet	This application ruleset.

Application Structure landing page *(on page 404)*

Access Groups & Users tab

Access the Access Groups & Users tab by clicking **Configure > Application > Structure > Access Groups And Users**.

Use this tab to:

- See the list of access groups that provide access to the current application.
- See the list of users that belong to an access group.
- Invite people to collaborate in developing this application.

Field	Description
Invite Collaborator	Click to invite people (via email) to collaborate in developing the application. For example, a Lead Business Analyst can invite other business analysts to edit and add specifications and requirements to the in-progress application. They might also invite subject matter experts to review the specifications. Upon clicking the link, the Invite Collaborator window opens for you to specify the information needed to send email invitations to the invitees. See Invite Collaborator window — Inviting collaborators.
Access Group	The name of an access group referenced by the application. Select a name to see the list of users that reference that access group, and so have that level of access into the application.
Operator Name	The names of operators who reference the access group.
Operator ID	The IDs of operators who reference the access group.

Application Structure landing page *(on page 404)*

Other Applications tab

Access the Other Applications tab from the header of Dev Studio by clicking **Configure > Application > Structure > Other Applications**.

Administrators (operators with the `PegaRULES:SysAdm4` role or a role based on that role) can use this tab to lock and roll applications in the Pega Platform system other than those in the ruleset stack of the current application. For example, when upgrading a Pega Platform it is considered best practice to lock all applications and create new ruleset versions before resuming development.

This page displays all applications in the system that contain an unlocked ruleset version and lists the rulesets and ruleset versions that make up each application including those inherited from any built-on applications.

To open an application rule in the list, click its name.

Field	Description
RuleSet	The rulesets and versions in the application.
Lock	Contains an icon indicating the lock status of ruleset versions for the application. • — Indicates that the version for any given ruleset is locked. Also indicates a locked application rule. • — Indicates that the version for any given ruleset is unlocked. Also indicates an unlocked application rule. • — Warns you that a locked ruleset version has lower unlocked versions. Click the icon to view a list that shows you which versions are unlocked. To lock an unlocked version, select it in the list, and its rule form opens where you can lock the version. As a best practice, leave only the highest version of a ruleset unlocked. If the warning condition exists, you cannot roll (increment) a lower unlocked version using Lock and Roll functionality. • — Warns you that this ruleset version does not exist but appears in your application rule. To correct, remove the version. Click an icon to open a window that displays a list of versions, and for each version, the lock status and the number of total and checked out versioned and non-versioned rules including ruleset, class, and organization instances.
Checked Out	The total number of checked-out versioned and non-versioned rules.
All Rules	The total number of versioned and non-versioned rules. Click a number to open a window listing the rule types and counts comprising the total. Click a row (including Total) in the list to drill down to a list of rule instances comprising a rule type.
Lock and Roll	Click to display the Application Lock and Roll window where you can review, lock current ruleset versions, and roll (increment) versions in a single step. You can optionally update your current application rule to reflect the new versions, or create a new application rule that contains the new ruleset versions.

In the window, select the application ruleset versions you want to lock and roll to a higher version.

Field	Description
Lock	Select the check box to indicate you want to lock the ruleset version and roll to a new version. Locking prevents a developer from saving new rule instances to this version, or updating or deleting existing instances. A ⚠ icon appears if there is a condition that might prevent you from locking the ruleset version. For example: A higher locked version exists or a version currently has rules checked out. A 🔒 icon appears if the ruleset version is already locked.
RuleSet Version	The ruleset and current version. ⓘ **Note:** Rulesets using Application Validation mode do not have prerequisites. Click the **+** icon next to the Prerequisites label to display the Current and New prerequisites for the version. To open the rule form for a current ruleset and version, click its name in the Current list. When you lock a ruleset version and select the Roll check box, you can modify the new prerequisites. By default, they are the same as the current ones. You can select different prerequisites, add additional ones, and delete ones from the New list. To add another ruleset version to the list, click the **Add a row** icon. To open the rule form for a ruleset in the New list, click the **Open the Rule Form** icon next to its name. To delete a ruleset from the New list, click the **Delete** icon next to its name.
Password	Appears when you select the Lock check box. Enter the password for the ruleset version.
Roll	Displays when Lock is selected. Used to roll the ruleset to a higher version.
Roll to Version	Displays the version that the ruleset will be rolled to unless you indicate another version number. Defaults to the next logical number in the ruleset numbering sequence.
Description	Current description of the ruleset version.
Run	Select an option to run against the selected ruleset versions.

Field	Description
	• **Do not update my application** — Locks and rolls the versions without updating the application ruleset list. • **Update my Application to include the new RuleSet Versions** — Locks and rolls the versions and updates the current application ruleset list. Prompts with an application description (current by default, which you can edit). Also prompts for a password if the current application is locked. Enter the application password. • **Create a new version of my Application** — Locks and rolls the versions and creates a new application rule. Prompts with the new application version (one increment higher by default, which you can edit), and with an application description (current by default, which you can edit). Also prompts for a password if the current application is locked. Enter the application password.

Application Structure landing page *(on page 404)*