System Architect Essentials

7.2

Student Guide

DOCUMENT: System Architect Essentials Student Guide
SOFTWARE VERSION: Pega 7.2
UPDATED: 02 07 2018

CONTENTS

COURSE INTRODUCTION

Before you begin

System Architect Essentials 7.2 overview

In this course, you learn the core principles of application development on the Pega 7 Platform. Business users and delivery team members use these principles to plan and deliver business applications faster and more accurately for maximum business value.

You learn how to perform the most common application development tasks to prepare yourself for your first Pega development project.

Objectives

After completing this course, you should be able to:

- Apply Pega's principles of application design and development to deliver business applications that are Built for Change™.
- Use Pega Express to model the life cycle of a case that mirrors the way business people think about how work is completed.
- Directly capture business objectives to help ensure that business requirements are accurately captured, and that business and IT stakeholders share a common understanding.
- Use Designer Studio to refine and enhance the case life cycle design.
- Identify the tasks and responsibilities of the system architect on a Pega Implementation.
- Configure a case and case processing behavior.
- Create data classes and properties for use in a Pega application.
- Automate decision-making throughout an application to improve process efficiency.
- Design responsive user forms for use on any platform or browser.
- Design reports to deliver key insights to business users.
- Incorporate and manage reference data to allow applications to adapt to changing business conditions.
- Test your application design to analyze rule behavior and identify configuration errors.

Intended audience

This course is for system architects who are responsible for developing business applications.

Prerequisites

To succeed in this course, you should:

- Know the business processes and policies used at your company.
- Have a basic understanding of business application development.
- Be familiar with project methodologies such as Scrum, RUP, or Waterfall.
- Have some experience developing software applications.

PEGA BUSINESS APPLICATION PLATFORM

The Pega Platform

Introduction to the Pega 7 Platform

In this lesson you will learn how the Pega 7 Platform is used to design and run Pega applications.

After this lesson, you should be able to:

- Explain the benefits of using a model-driven application design and development approach
- Describe the user roles and each role's high-level responsibilities associated with the Pega 7 Platform
- Describe the purpose of the run-time portals in Pega 7
- Describe the purpose of the design time portals in Pega 7
- Describe when to use Pega Express vs. Designer Studio

Pega 7 application platform

The Pega 7 Platform provides a single, unified platform with everything you need to build or modify enterprise applications. Business and IT stakeholders work together, using visual models to capture business requirements. With no coding required, Pega 7 automatically generates the application and its documentation.

Pega 7 provides a unified application development platform for building business applications. Traditional application development tools focus on creating individual business applications that must then be integrated with each other. For example, a company may need to integrate a website application, a mobile application, and a customer relationship management (CRM) application.

Each application has its own requirements, analytics, and business policies.

Existing business applications may be developed and maintained by separate groups that use different application development languages, tools, methods, and repositories.

Or an company might have a business rules engine to provide business logic, a development environment to integrate with external systems, or a document management system to manage case data. Unifying existing business applications might be unobtainable for a company without outside help. As a result, sharing and reusing application components when building business applications is difficult.

The Pega 7 Platform is unified. This means that all the functionality needed for business applications is configured using a consistent set of components that are defined and stored together in the company's systems of record. Each business application built on top of the Pega 7 Platform uses a common set of tools, a common vocabulary, and a consistent model to communicate, implement, and validate requirements. Business processes and policies, integrations with external systems, mobile interfaces, business performance monitoring, and reporting are all defined using a consistent, model-based application development approach. This makes sharing and reusing artifacts easier when building business applications.

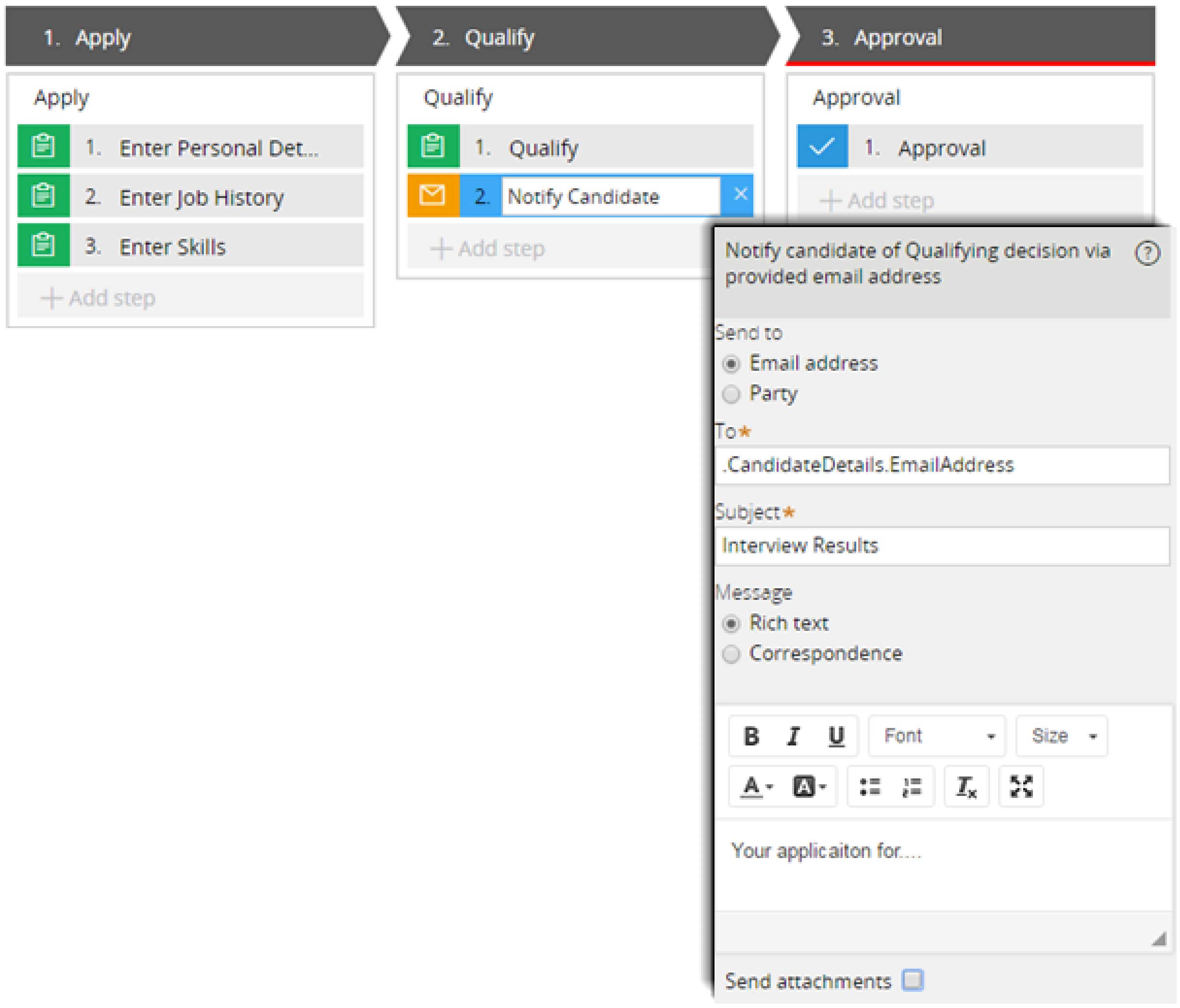
KNOWLEDGE CHECK

ANSWER What is the significance of a unified platform?

With a unified platform, you configure applications using a consistent set of components. The use of a consistent set of components reduces application development time.

Model-based application design and development

The Pega 7 application development platform uses a model-based approach to application development.



Notify candidate of Qualifying decision via provided email address

Send to
[✓ Email address]
Party

To*
.CandidateDetails.EmailAddress

Subject*
Interview Results

Message
[✓ Rich text]
Correspondence

Your applicaiton for....

Send attachments

You use visual, form-based definitions of application components in a model-based approach. No coding is required.

Application architects can see where application components are placed and how each piece is leveraged by the rest of the application. By improving the visibility architects have to the overall application design, all team members can communicate more effectively about the impact of new or modified requirements. Team members can also identify potential gaps. As a result, updating processes, user interfaces, and other business rules is easier.

KNOWLEDGE CHECK

ANSWER To ensure visibility among all team members to an overall application design, which application development approach does Pega 7 use?

Pega 7 uses a model-based application approach.

Pega 7 user roles

Building a successful business application requires collaboration between two parties: case participants and case designers.

Case participants
Case designers
Case worker
Case manager
Business architect
System architect

Case participants

Case participants are the business users of the application, processing and closing cases. Case participants are usually organized by roles. For example, in a credit card dispute case, the roles might include a customer service representative, a dispute agent, and a fraud investigator.

There are two groups of case participants:

Case workers are responsible for creating, viewing, and working on their own cases and assignments. A case worker cannot monitor or manage work among other case workers, or view work statistics.

Case managers are responsible for working on cases and monitoring work group status, goals, and deadlines. Case managers are also responsible for generating work manager reports.

Each case participant group uses a Pega run-time Pega portal specific to the group. Case workers use the Case Worker portal. Case managers use the Case Manager portal.

KNOWLEDGE CHECK

ANSWER If a case manager oversees the efforts of a case worker, what does a case worker do?

Case workers are responsible for creating, viewing, and working on their own cases and assignments.

Case designers

Case designers are part of the delivery team responsible for designing and building business applications. Case designers use Pega's design time portals, Pega Express and Designer Studio.

There are two groups of case designers:

Business architects (BAs), the first group of case designers, work with case participants and other stakeholders to define business objectives and application requirements. BAs plan application behavior to address the business objectives and requirements with specifications. These specifications describe how the application manages and automates work.

System architects (SAs), the second group of case designers, provide the technical skills needed to complete the application. SAs configure application assets such as User Interface (UI) forms, automated decisions, and correspondence. SAs then review the application with business stakeholders for approval.

The SAs and BAs work together to ensure the new application reflects business needs. SAs often prototype application features to help refine the specifications captured by the BAs. These prototypes help align the application with the business needs.

KNOWLEDGE CHECK

ANSWER

If business architects (BAs) gather business objectives and application requirements, for which tasks are the system architects (SAs) responsible?

SAs provide the technical skills needed to complete the application. They configure application assets such as User Interface (UI) forms, automated decisions, and correspondence. SAs then review the application with business stakeholders for approval.

Pega 7 user portals

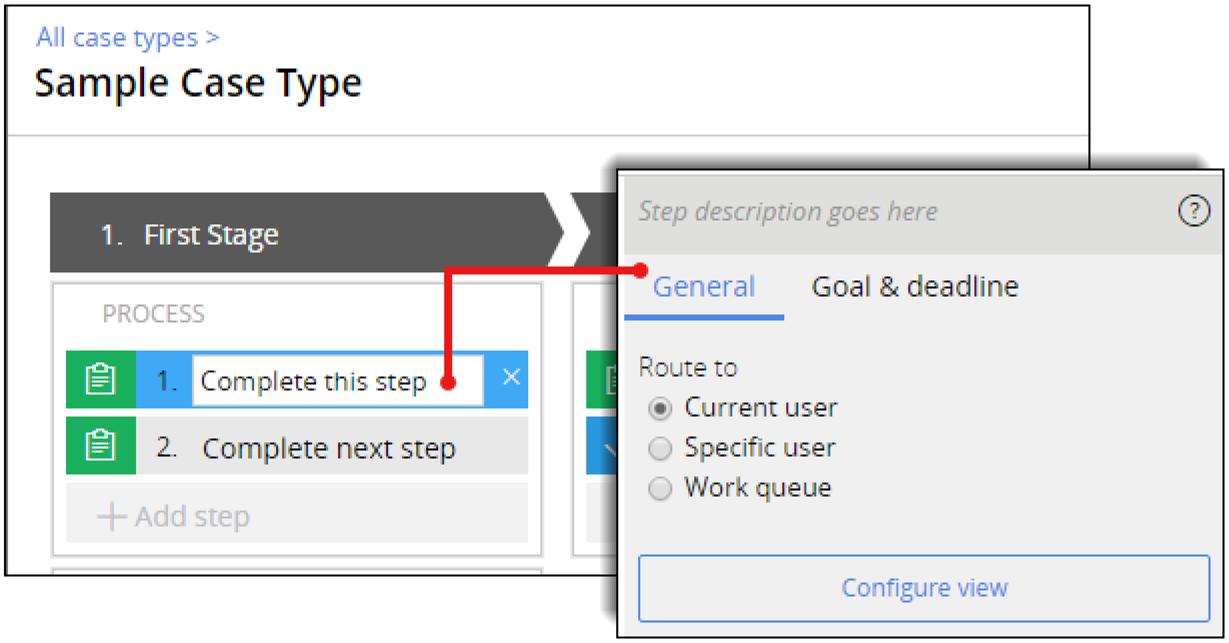
Pega 7 includes four user portals that provide intuitive, results-focused work spaces.

Design time portals

Application designers can use either of two design time portals, Pega Express or Designer Studio, to build applications that support a wide range of business objectives at all levels of complexity.

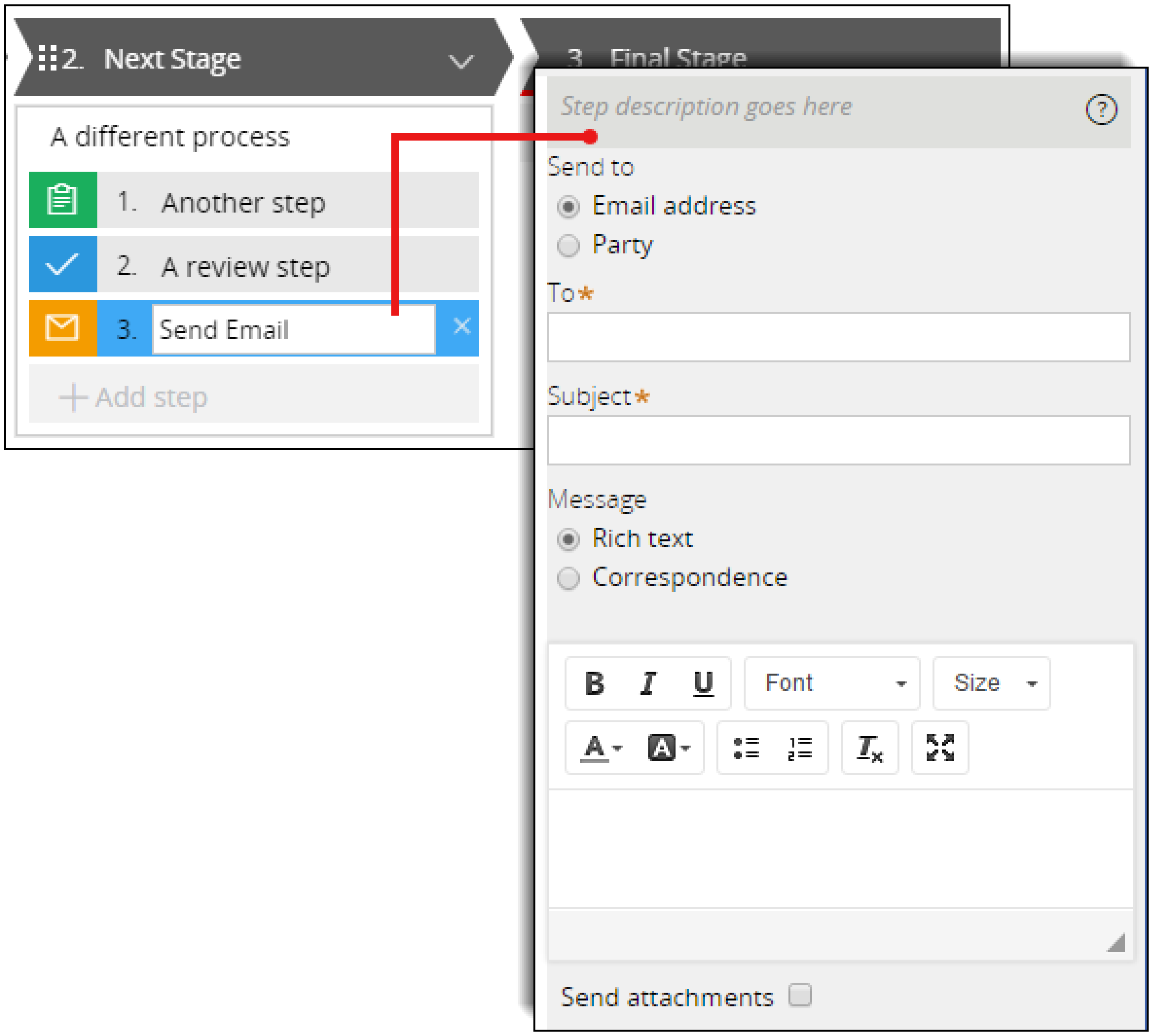
Pega Express

Pega Express is an accelerated application development environment that exposes key elements and features of the Pega 7 Platform. Use Pega Express to quickly build the case structure and process steps. Streamlined capabilities let you create a basic application that you can demonstrate to stakeholders and get their feedback.



Designer Studio

Designer Studio is a robust application development environment that exposes more advanced features of the Pega 7 platform. Use Designer Studio to refine the case life cycle design. For example, you can add predefined utilities used to automatically send an email.



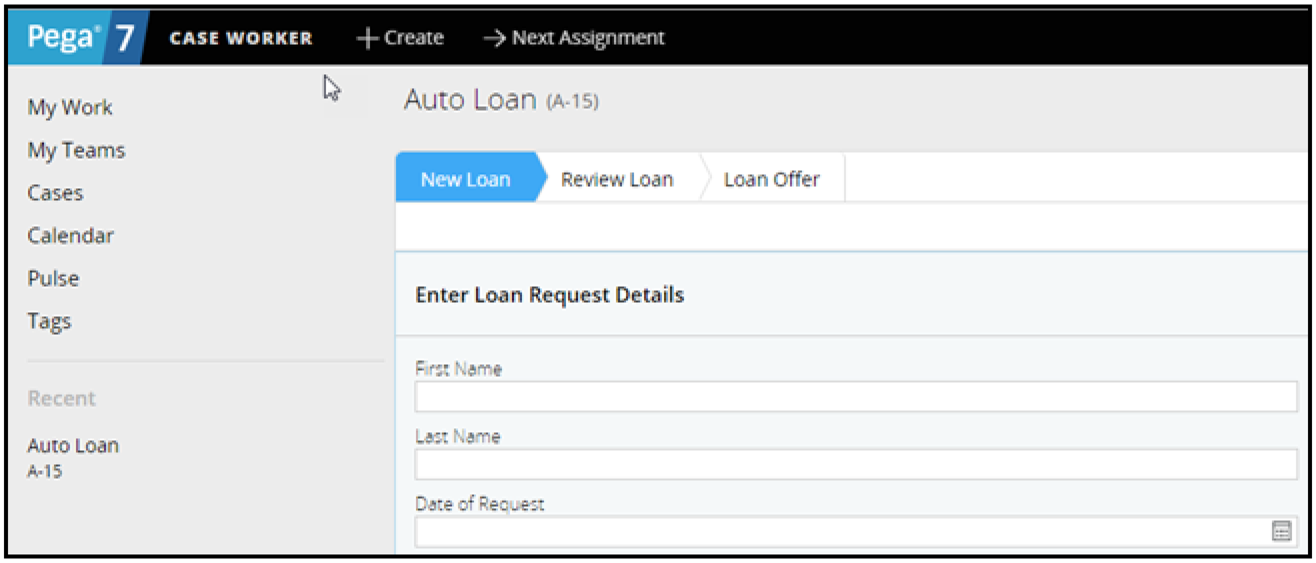
Design time users can toggle between Pega Express and Designer Studio.

Run-time portals

Case participants are assigned run-time portal access based upon each participant's role. Run-time portals users do not have design privileges.

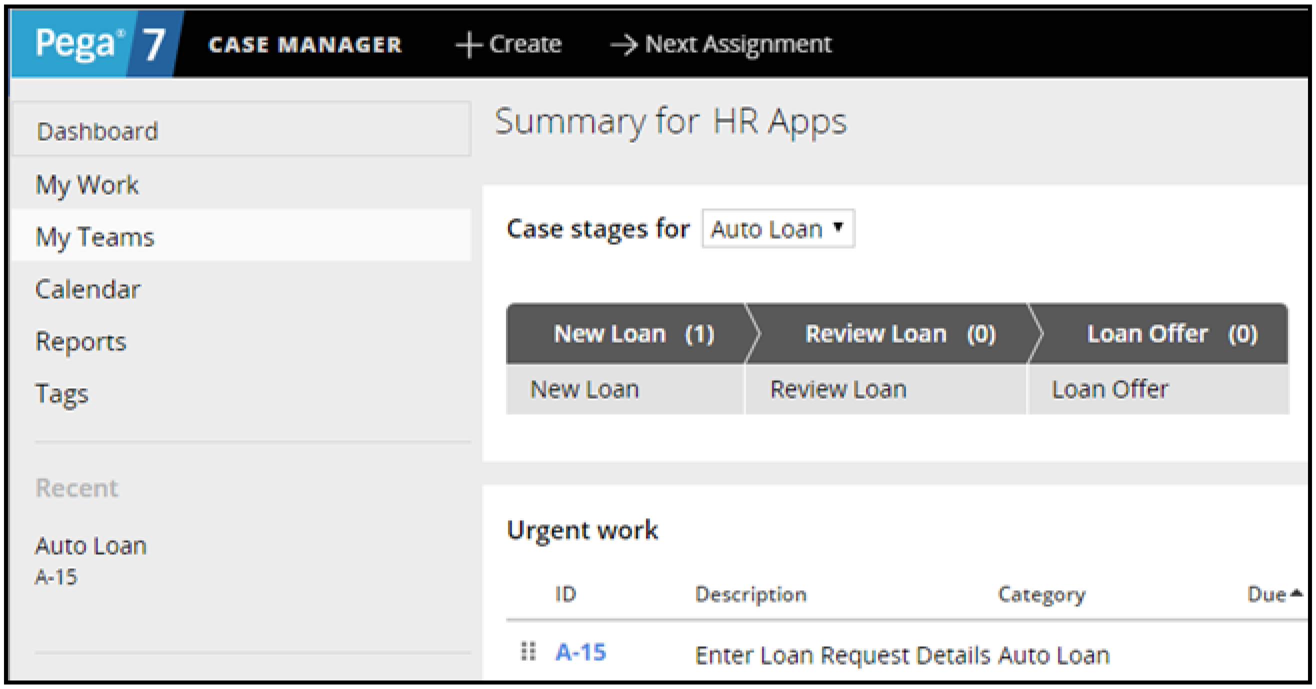
Case Worker portal

An end user with case worker privileges is able to access to the Case Worker portal. Case worker privileges enable the user to work on assigned cases.



Case Manager portal

An end user may be assigned case manager privileges.



ID	Description	Category	Due
A-15	Enter Loan Request Details Auto Loan		

Case manager privileges enable the user to manage their assigned cases and view the status of the cases assigned to all of their direct reports.

Pega Express

Pega Express is a design time portal that enables you to quickly create and run applications that model processes business users follow.

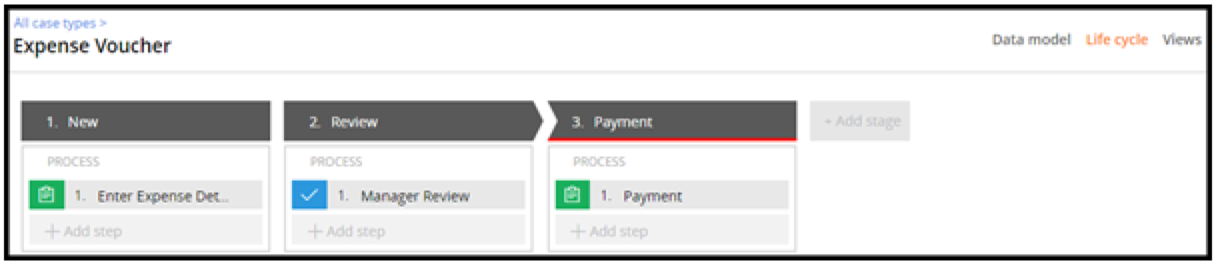
Pega Express allows you to:

- Create cases
- Create user views and fields
- Add or remove existing users from your application
- Define settings for theme, mobile apps, and other tools

In addition to the portal's design capabilities, you may also use the portal to run an application in **simulation mode**. This features allows you to test your design as a user would experience it. For example, you may add fields to a form that is presented to the business user. To test your updates, run Pega Express in simulation mode. This displays the form as a user would see it while working on a case. As a user, you enter information in the fields to ensure that your design works as expected.

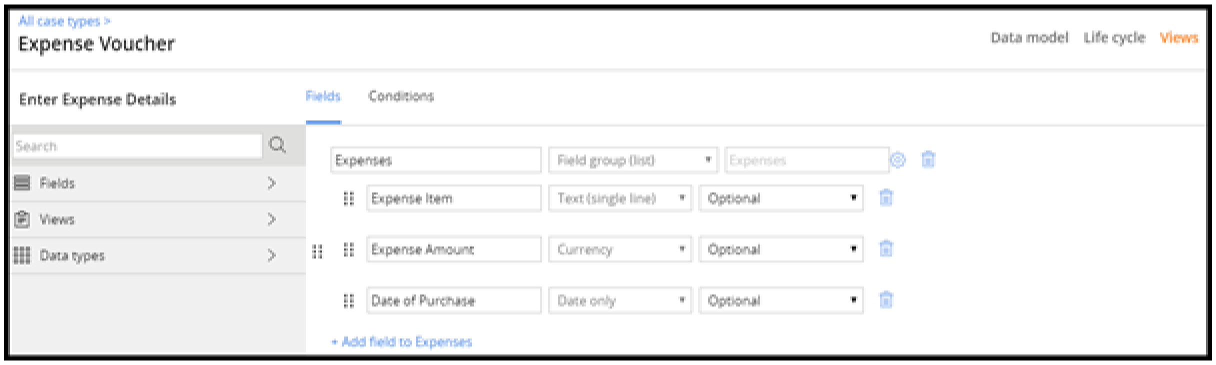
Designing with Pega Express

To create an application in Pega Express, start by building out the high-level case structure and processes.

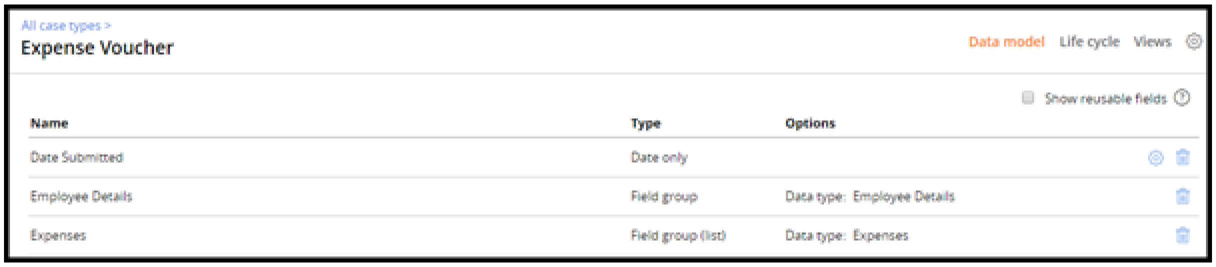


You define the phases in the life cycle of a case type, and the processes or work flows that users follow. By incorporating stages, processes, and steps into your case-type designs, you build robust business solutions in your application.

Next, you create forms that are associated with assignments or approval steps in the life cycle of a case.



By defining the fields that are displayed at run time, you can collect information from users or present case information for review.


All case types >
Expense Voucher

Name	Type	Options
Date Submitted	Date only	
Employee Details	Field group	Data type: Employee Details
Expenses	Field group (list)	Data type: Expenses

This allows you to build the data model in the context of the life cycle of the case.

Pega Express is ideal for initial Grooming/Elaboration sessions where business and IT stakeholders collaboratively define the primary case type life cycle. This may include importing existing specifications and linking them to process steps as you go, or defining new specifications in the context of the case life cycle.

KNOWLEDGE CHECK

ANSWER Pega Express allows you to run an application in __________ so that you can test your design as a user would experience using the application.

simulation mode

Using Pega Express to model the life cycle of a case

Using Pega Express, you can easily define the high-level case structure and steps of a case.

Transcript

Click **Cases**.

Click **Create new case type**.

Enter Auto Loan as case type name.

Click **Next**.

Click **+ Add field**.

Enter First Name and Last Name.

Click **Got it**.

Click **Life cycle**.

Click **Add stages**.

Enter New Loan as the name of the first stage.

Click **+ Add stage**, and then enter Loan Review as the name of the second stage.

Click **+ Add stage**, and then enter Loan Offer as the name of the final stage.

Click **Got it**.

Click **Add process?**.

Click **Add processes**.

Click **+ Add process**.

Enter Enter Loan Details in the first step of the first stage.

Click **+Add Process** in the second stage.

Rename the step to Review Loan Details

Click **+ Add process** in the third stage.

Rename the step to Present Loan Offer.

Click **Done**.

Click **+ New** and select the Auto Loan case.

Enter Irshad as the first name, and James as the last name.

Click **Done**.

Click **No, advance this case**.

Click **No, advance this case**, and then click **No, Advance this case** again.

You now have a basic case type, which you can further refine by adding user views and other details.

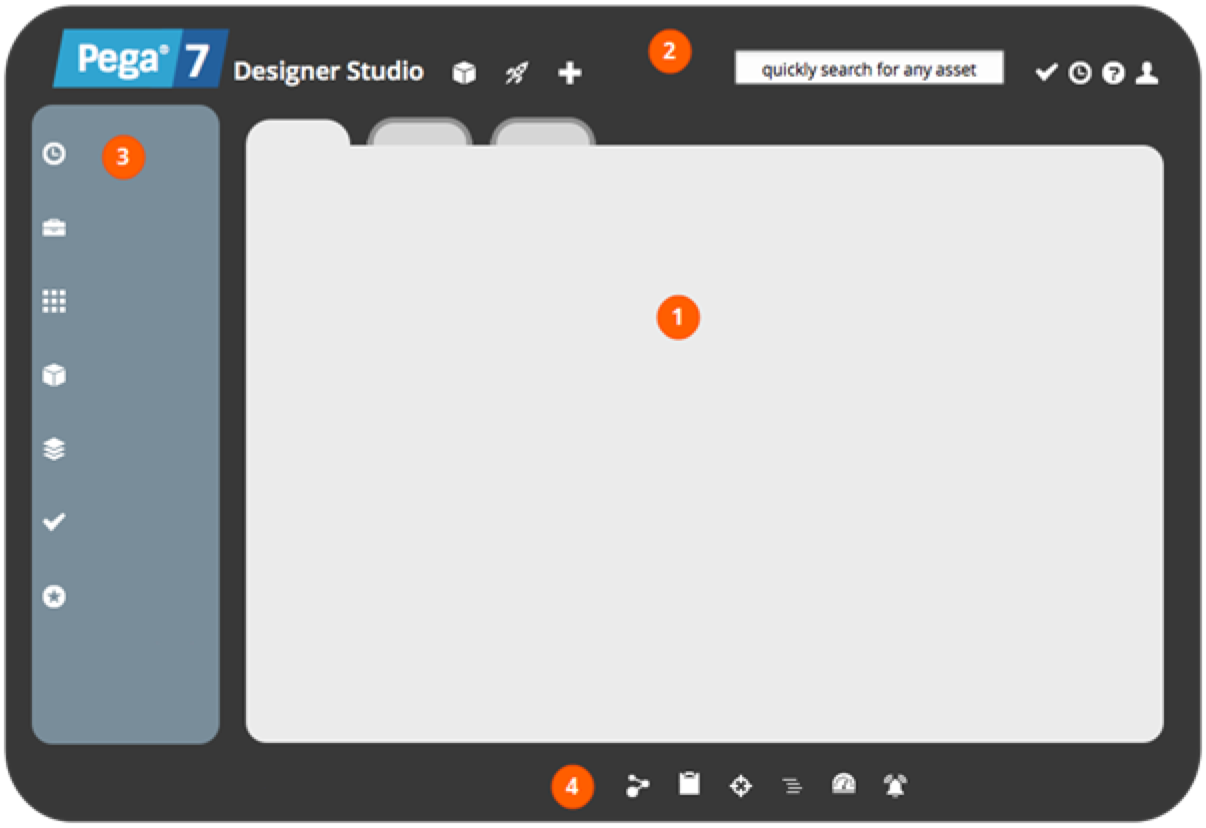
Designer Studio

Designer Studio is a powerful design-time portal for architects who wish to build and extend Pega 7 applications.

After you create initial case life cycle designs in Pega Express, move to Designer Studio to extend the application and refine your designs.

Designer Studio user portal consists of a:

1. Work area
2. Header bar
3. Explorer area
4. Developer toolbar



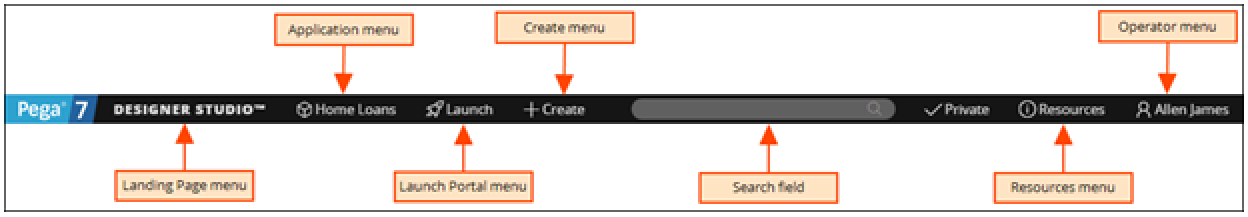
Designer Studio work area

Use the work area to view landing pages. A landing page presents information or tools that help developers to build an application or view specific information on the application. You may open, review, and edit application artifacts such as requirements, case types, user views, data models, and reports. Multiple items open in the work area display in separate tabs.

Designer Studio header

The Designer Studio header provides tools to create and manage application assets.

Use the header to create cases, search for records and launch secondary portals.

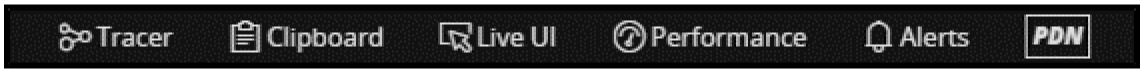


Designer Studio explorer area

The Explorer area appears as a panel on the left side of the Designer Studio and provides navigation to specific record types.

Icon	Explorer	Purpose
🕒	Recent	Display and access up to the last 20 recently opened records, wizard items, instance lists, landing pages.
💼	Cases	Open and review case types in the current application. The tree structure helps you identify parent-child relationships. Advanced options allow you to edit case types and create new ones.
▦	Data	Review data types in the current application and the data pages associated with them. You can filter the results by application or applies to class.
⬢	App	Review or open the records that belong to the current and built-on applications. The tree structure organizes rules by class, category, rule type, and instance.
≋	Records	Open a list of records in the system organized by category and type.
✓	Private	Review your checked-out rules.
★	Favorite	Review and update your personal or access group favorites.

Designer Studio developer toolbar



The Developer toolbar helps users debug applications, tune performance and quickly analyze the composition of user interface (UI) components.

Use the:

- Tracer tool to debug rule execution.
- Clipboard tool to view data in memory.
- Live UI tool to identify user interface elements.
- Performance tool to analyze application performance.
- Alerts tool to view system alerts generated by Pega.

KNOWLEDGE CHECK

ANSWER Use the __________________________ to open, review and edit application artifacts such as requirements, case types, user views, data models and reports.

Work area

Using Designer Studio to refine the life cycle of a case

Using Designer Studio, you can easily refine the high-level case structure and steps of a case.

From Pega Express, you can seamlessly to Designer Studio to further refine the life cycle of your case.

Designer Studio provides the same case life cycle view as Pega Express.

You can further refine the basic case life cycle by adding stages that represent exceptions to the normal life cycle of a case.

Add process steps to the alternate stages. Pega 7 provides predefined utilities for actions such as sending an email.

Easily reorder steps in a process.

Save your changes and continue refining the case life cycle design.

Easily edit user views.

Add new data fields.

Your changes are immediately available.

Seamlessly transition back to Pega Express. Changes you make in Designer Studio are available in Pega Express.

Principles of application development

Introduction to principles of application development

In this lesson, you will learn five key principles of application development.

Objectives

At the end of this lesson, you should be able to:

- State the importance of using Pega's Directly Capture Objectives™ approach to managing requirements
- State the benefits of using Pega's Situational Layer Cake™ architecture to design an application
- State the benefits of using a model-driven application design

Capture objectives directly in the application

Business application development teams can find it difficult to communicate business requirements.

There may be no common language between the business and IT stakeholders. And, there may not be a common view of the business goals.

Often, business stakeholders are not sure of what their business needs are. When this happens, IT stakeholders find it difficult to get the details they need. Business and IT stakeholders must share a common understanding of the business requirements. You also need a way to ensure business requirements are current and available to all stakeholders

In Pega 7, you capture business requirements directly in the application. This practice is called Directly Capture Objectives, or DCO.

DCO ensures business and IT stakeholders share a common understanding of the business requirements. DCO also ensures the business requirements are up-to-date and available to everyone.

Build multi-dimensional applications

The critical dimensions of any business are product, region, channel, and customer.

When you conduct business in different countries, you must manage the regulations of each jurisdiction, and the cultural differences in each region. When you sell multiple products through multiple channels, you must manage the business rules for selling each product in each channel separately. When you sell to different types of customers, you must manage each customer's expectations and preferences.

With some application development platforms, you must create separate copies of the application for each product, region, or channel. Or, you must create an application that treats all business transactions the same, regardless of the business context. The result is enterprise applications that are hard to maintain, and even harder to change

Pega uses a unique application architecture called a situational layer cake.

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

Pega's unique approach to enterprise application architecture - the Situational Layer Cake - can help turn the complexity of an ordinary enterprise application into a simple and coherent end-to-end customer experience.

Use a model-driven application design

Ask any business person to explain their needs for an enterprise business application.

As they explain their needs, you will notice they do not dive into the details about any particular part of the application. And, they do not discuss the behind-the-scene technologies needed to make the business application useful.

When business people explain their needs for an enterprise business application, they describe the major steps of how work gets done. They talk in terms of a case and a desired outcome, and the stages that case may go through until the desired outcome is achieved.

What they describe is the life cycle of a case.

To be effective, your application design and development efforts must match the way business people naturally talk about their work. A case, and its life cycle, is the central metaphor in Pega's model-driven approach to building business applications.

Rather than drawing complex end-to-end diagrams, Pega 7 allows you to build a visual representation of the life cycle of a case essentially building the skeleton on which you hang the more detailed processes. This allows you to establish a business view of the case before debating the details.

Best practices and Guardrails

Introduction to best practices and guardrails

In this lesson, you will learn best practices for designing and building applications.

Some best practices help you deliver business application development projects successfully. Other best practices help you design business applications with fewer defects. Following best practices increases your chances of overall project success.

Objectives

At the end of this lesson, you should be able to:

- Explain the purpose and benefits of best practices.
- Identify Pega best practices
- Identify the most important best practices when building a Pega application
- Explain how Pega guardrails indicate flaws in an application design
- Describe the information provided in guardrail warnings
- Locate guardrail warnings in Designer Studio

Purpose of best practices

Best practices are well-defined methods or techniques that lead to desired results. Every best practice has at least one goal. A best practice is proven to make progress towards achieving its goals. If an organization follows best practices, it can predict a desired result with minimal problems or complications.

Best practices are used in everyday life. For example, when preparing a meal, a best practice is to always use a potholder when touching a hot pot. The goal is to not burn yourself, and using a potholder is a proven way to prevent you from burning yourself.

As you design and build your application, establish and follow best practices to increase your chances of project success. Look for existing best practices used by your organization and by Pega. Select existing best practices or create new best practices to meet your goals. Applying best practices increases the likelihood that your application is delivered on time and meets all of its design requirements.

Tips for establishing best practices

The standards for establishing best practices can vary, depending upon the needs of the organization and who is making the choices. You can use several criteria to help choose best practices that will deliver measurable and predictable performance improvements for your application development projects.

Is the best practice appropriate for your organizational goals?

To be effective, best practices must address the specific goals of your organization. For example, if an organizational goal is to ensure the protection of sensitive customer information, a best practice to outsource developers may not be appropriate — the developers may also be working for the organization's competitors.

Does the best practice fit with the structure of your organization?

If a best practice places authority in a single person or part of the organization, it does not provide value if your project teams are supposed to be able to make their own decisions.

Do you have the necessary resources to use the best practice?

Understanding what a particular best practice requires in regard to resources — whether money, personnel, or skills — is essential. Make sure that your organization can provide those resources before committing to a specific best practice.

Is the best practice cost-effective?

If a best practice works well for other organizations but requires an unacceptable amount of money or time to reproduce in your organization, it will be hard to justify the use of that best practice.

Pega's best practices for project success

With experience from thousands of Pega project implementations, Pega has defined best practices that are key to delivering successful Pega projects. Using some of the best practices identified below can help increase your chances of project success.

Leverage DCO to improve product quality

Directly Capture Objectives (DCO) enables a project team to directly enter business requirements for an application into Pega.

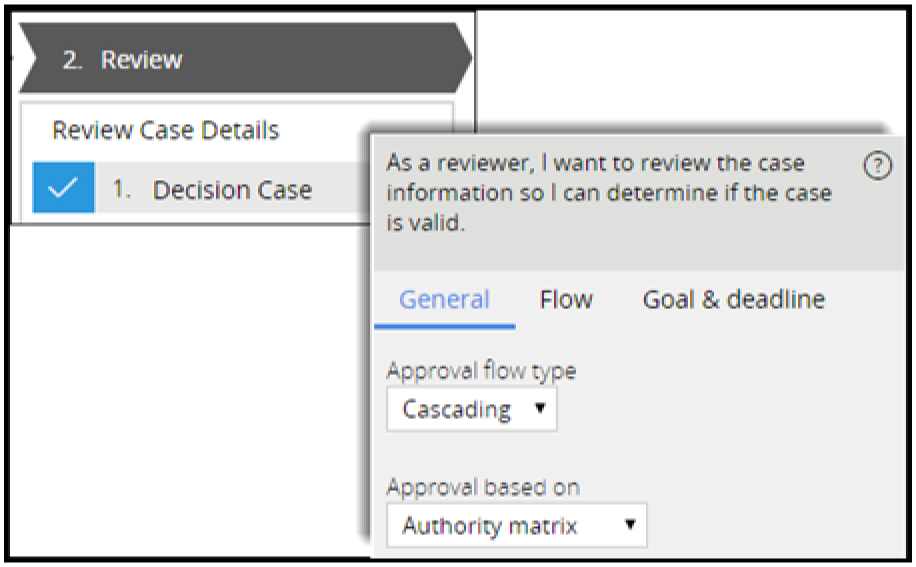
DCO helps eliminate translation errors, saves the team time and effort, facilitates direct engagement of business and IT resources around visible working models, and enables project participants to optimally review work progress.

Pega recommends that all projects leverage DCO as a core part of the delivery process.

Use standard Pega capabilities

Pega 7 has many features and capabilities built into the product. Use Pega capabilities, which have been tested and proven reliable.

For example, assume part of a case's life cycle requires increasing levels of review. Rather than building a custom review process, use an approval process available in Pega 7. You can design the life cycle of a case to support your requirements in a fraction of the time needed to build custom processes.



As a reviewer, I want to review the case information so I can determine if the case is valid.

Approval flow type: Cascading

Approval based on: Authority matrix

Iterate and test as you build

Use the most agile, iterative delivery model that your organization can adopt. First, separate large applications into smaller, more manageable components. For example, instead of building the complete application and then testing the completed application all at once, build and test individual processes incrementally. Then, demonstrate completed features to interested parties who can provide feedback.

Begin testing early in the project life cycle to drive higher levels of product quality. Test each new feature or capability to make sure it works. Then, test the system for processing issues that may affect performance. Also, check to see if the new features work together without error. Finally, have analysts test the application to make sure the application meets the requirements and business objectives.

Communicate project progress at all levels

Regular communication among all project participants helps teams focus on the right issues in a timely manner. Pega recommends the following:

- Daily standup meetings to set priorities for the day, ensure alignment, and eliminate any blockers.
- Weekly project updates to track issues and update the status report. Flag conditions that need immediate attention to keep projects moving on schedule.
- Biweekly or monthly meetings to review the entire project and determine whether the application conforms to the original design objectives. Promptly incorporate feedback into project design, and revise project plan if necessary.

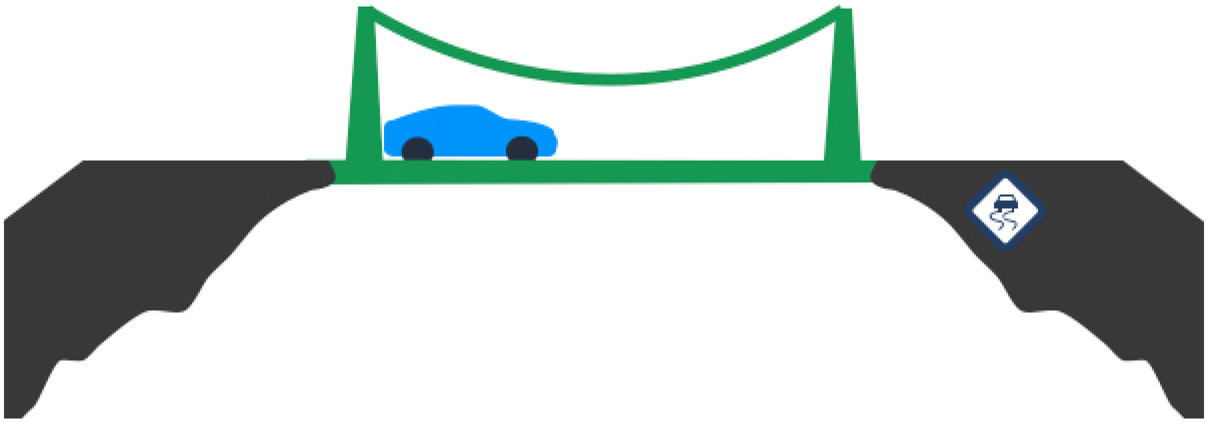
Ensure project team members are certified

Project success depends on a complete and capable team. As a guideline, Pega recommends that all team members hold the appropriate certifications for their roles.

Pega recommends that business architect and project management resources pass the Certified Business Architect exam. All developers should, at a minimum, pass the Certified System Architect exam.

Follow Pega guardrails

Pega guardrails help ensure that you use best practices for configuring Pega applications. Pega's powerful capabilities offer many possible approaches to create a specific design requirement. Not all of those approaches are the best solution. Pega provides standard guardrail capabilities that enable development team members to track compliance with Pega best practices. Compliance with the guardrails results in applications that are easier to maintain and upgrade, and have significantly fewer defects than non-compliant applications.



Collaborate with everyone invested in the success of the project

Collaborate with individuals interested in making your project succeed. Bring business users, business analysts, and system architects together so they can share their unique skills and viewpoints. For example, business users and business analysts who have in-depth knowledge of the business process can help capture accurate requirements as well as design and optimize business processes. System architects can help provide guidance on the best way to implement a business requirement.

KNOWLEDGE CHECK

What are the possible consequences of not following best practices and guardrails when designing and building an application on the Pega 7 platform?

When not following best practices, you may spend more time re-creating existing functionality, or debugging a component that is not well designed. There is also the risk that you will implement a feature that does not work correctly.

Pega guardrails for application design

Following best practices when designing and building an application is important. To help ensure your project's success, Pega guardrails help you and your team track whether an application conforms to Pega's best practices.

Following best practices can help you deliver applications that require less maintenance, have fewer defects, and can be easily upgraded compared to applications that deviate from best practices.

As you work with Pega 7 you will see messages.

These messages are important; they are known as guardrails.

Guardrails help to ensure that the application you create follows the known best practices for Pega 7 development.

When creating an application it is always a good idea to follow the best practices for whatever technology we are working with. What happens if we don’t follow known best practices?

By not following best practices we add increased risk that our application development effort could go off course and not. The application does not work as well as it could or should. It can also make the application difficult to maintain and update.

It is also possible that our application can get out of control and crash the project entirely.

Guardrails are best practices and guidance about situations that contain risky conditions or that might result in an undesirable outcome.

Guardrails ensure you and your team use Pega 7 the right way and help you avoid troublesome situations.

PROTOTYPING AN APPLICATION WITH PEGA EXPRESS

Designing a case life cycle

Introduction to designing a case life cycle

Effective case management requires individual contributors to complete steps in a coordinated manner so they can resolve cases efficiently. A case life cycle design is a visual model used to define the major steps of how work gets done. Case life cycle design provides a more natural, business-friendly way to develop an application.

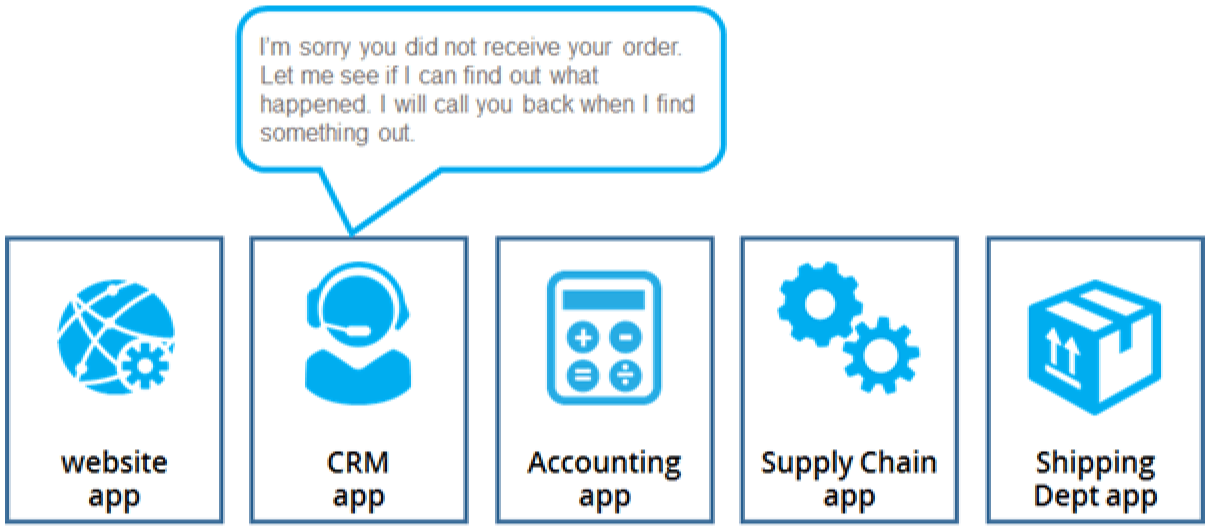
Objectives

At the end of this lesson, you should be able to:

- Explain the purpose of case life cycle design
- Describe the elements of a case life cycle
- Explain the relationship between case types, stages, and process steps
- Create a case type using Pega Express
- Add stages and process steps to a case type using Pega Express

Case life cycle design

Business applications are the foundation of every organization. Business applications automate work that must be completed to achieve specific business outcomes. For example, opening a new account, filing an accident claim, or ordering merchandise online.

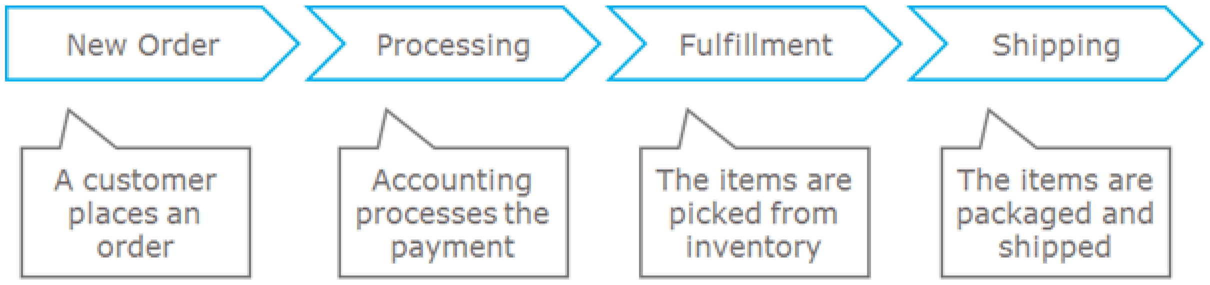

I'm sorry you did not receive your order. Let me see if I can find out what happened. I will call you back when I find something out.
website app
CRM app
Accounting app
Supply Chain app
Shipping Dept app

Traditional business applications are based on individual transactions and are built as standalone applications for different departmental functions.

These separate applications make it hard for business users to complete work in a coordinated manner. When working with separate applications, business users lack the visibility they need to effectively achieve business outcomes.

A business view of work

To help business users effectively achieve business outcomes, business users need a way to work in a coordinated manner. Business applications should function in the same way business users naturally think about and describe their work.


New Order
Processing
Fulfillment
Shipping
A customer places an order
Accounting processes the payment
The items are picked from inventory
The items are packaged and shipped

Business users often think of the work they do — investigating a fraud claim, processing a loan application — as a case.

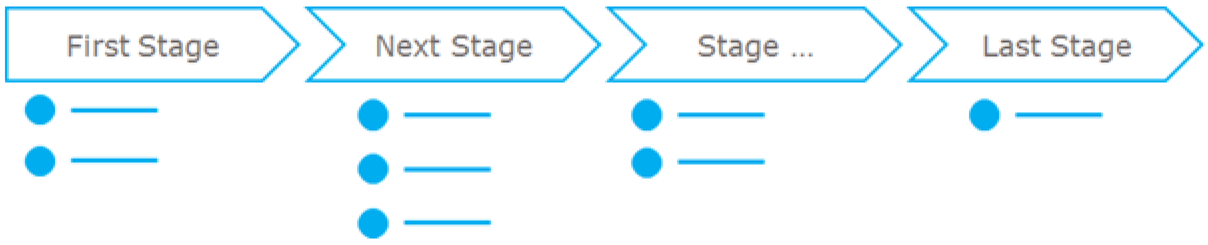
A **case** is work that delivers a business outcome. The outcome of a case is a meaningful deliverable provided to a customer, partner, or internal stakeholder. A deliverable can be processing an auto-insurance claim, onboarding a new mortgage, or designing and releasing a new product. In all these examples, the work to be done can be defined in terms of its resolution (the insurance claim is paid, the new account is opened, or the new product is released).

Business users understand, and work with the case as it moves from one person to the other, from one part of the organization to another. What business users are describing is the life cycle of a case — how they manage the case as it is opened, worked on, and resolved.

Design your application using a business view of work

A case is the central metaphor in Pega's model-driven approach to application design.

Case life cycle design is a modeling technique Pega uses to describe, in business terms, how a business application should work. Case life cycle design allows business users to see, and interact with a case in the same way they think about it.



Cases are organized into high-level milestones, known as stages. **Stages** are the first level of organizing the different tasks, or processes required to complete work associated with a case. Stages help to organize the life cycle of a case around the key events that support the primary goal of the case.

Stages are further organized into **processes** which define one or more paths the case must follow. Processes contain a series of actions, or steps that must be completed to resolve the case.

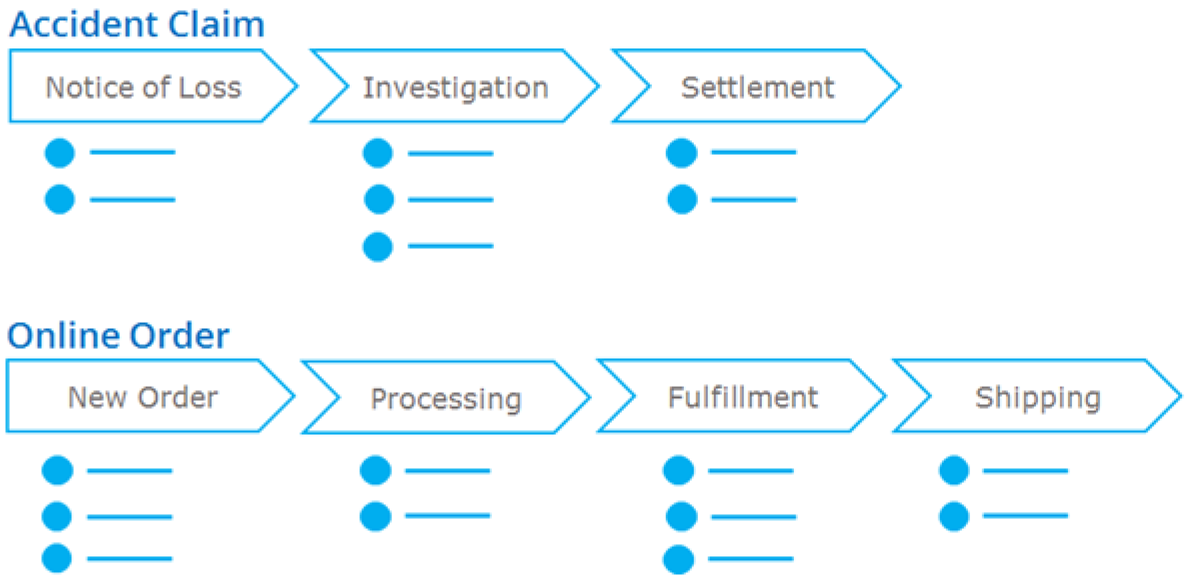
KNOWLEDGE CHECK

ANSWER A _______________ delivers a meaningful business outcome to a customer, partner, or internal stakeholder.

case

Case types

A **case type** is an artifact in Pega used to define the tasks and decisions needed to resolve a case.



Each case type captures the life cycle of a specific type of case, from creation to resolution.

Guidelines for identifying and naming case types

Case types are generally named after the case they represent. For example, a case type used to process loan applications for a new vehicle might be named *Auto Loan*, while a case type used to process mortgage applications might be named *Home Loan*.

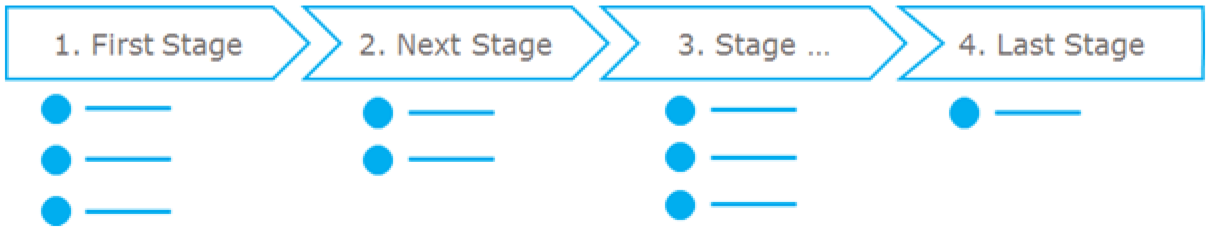
The name given to the case type is usually only one or two words. For example, a case type used to open new accounts might be named *New Account*.

Use names that are relevant and meaningful to the business users. For example, a case type used to process auto accident claims might be named *Accident Claim*.

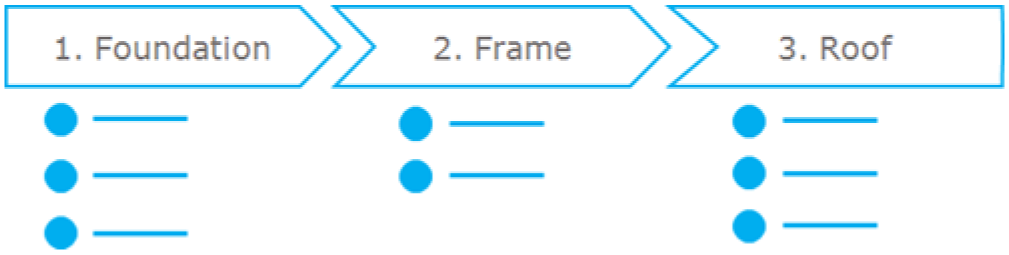
Use a singular context when naming case types. For example, a case type used to process a fraudulent credit card charge might be named *Fraud Claim*.

Stages

A stage is the first level of organizing work in your case type. It contains the workflows, or processes, that users follow before they can move a case to the next phase in the case life cycle.



By capturing business requirements in stages, you get a sense of what needs to be done first, what must happen in sequence, and what can happen in parallel during case processing. In a case life cycle, stages that lead to the expected outcome are called **primary stages**. The sequence of primary stages is often referred to as the **happy path**.



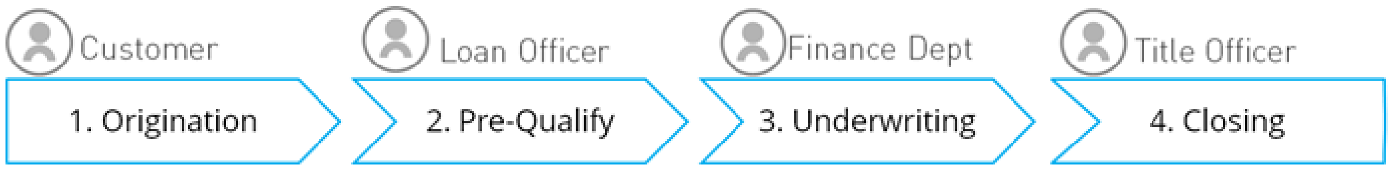
As an example, consider the construction of a new home. If you were asked to organize the tasks required to build a house into key phases of construction, you might organize the tasks in a series of three stages. The foundation of any building is always the first phase. Then the house itself — the frame — is constructed. Finally, the roof is added.

Each stage represents a distinct phase of the home construction case life cycle. In the home construction example, foundation, framing, and roofing are primary stages that lead to the completion of the house which is the business outcome.

Guidelines for defining stages

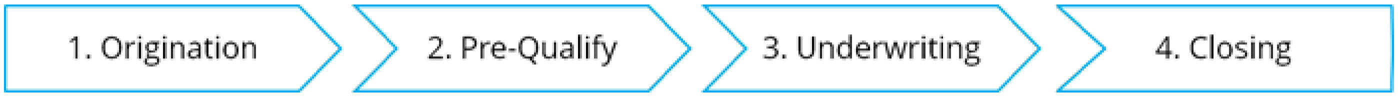
To define stages, consider the following guidelines.

Organizing stages



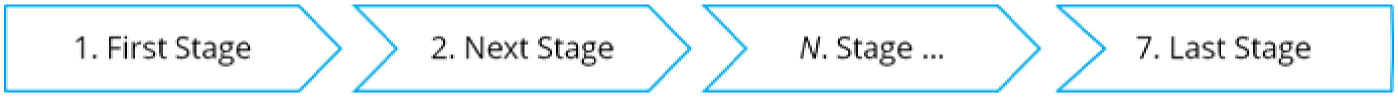
Stages typically represent the transfer of the case from one authority to another, or from one part of the organization to another. Stages may also represent a significant change in the status of the case.

Naming stages



Use names that are most meaningful and relevant to the business users. Use a noun, or noun phrase, to describe the context of the stage. As much as possible, try to limit the stage name to no more than two words.

Number of stages



Consider limiting the number of stages in any given case type to 7, plus or minus 2. If you find yourself needing more than 10 or so stages, consider combining one or more stages, or using a separate case type.

On the minimum side, do not be concerned if a case has only one or two stages. Focus on maintaining a maximum number of stages in any given case type and the minimum will work itself out.

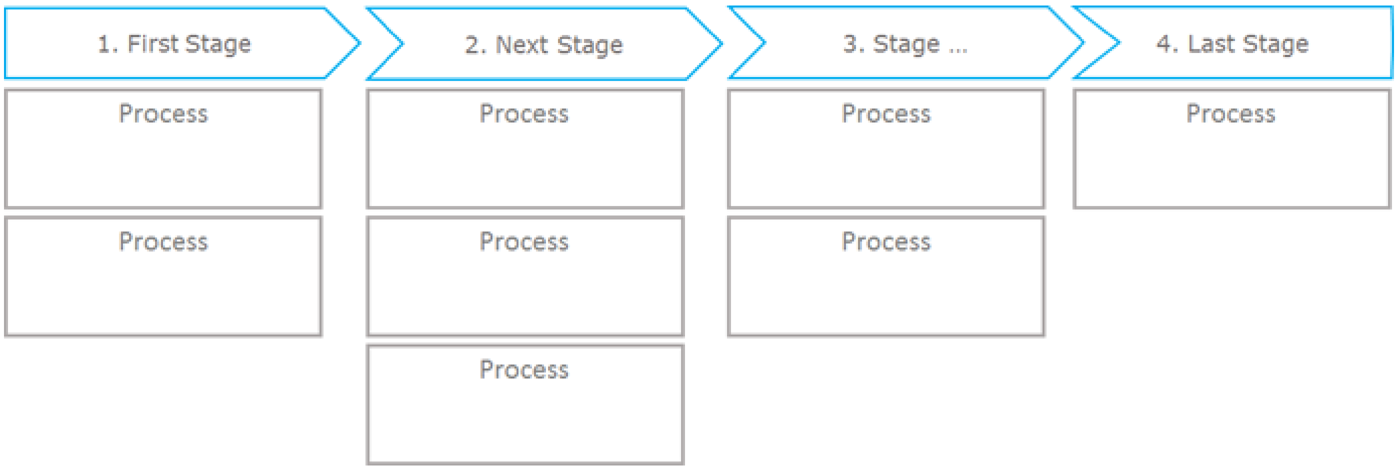
KNOWLEDGE CHECK

ANSWER A _______________ is used as a first level of organizing work in a case.

stage

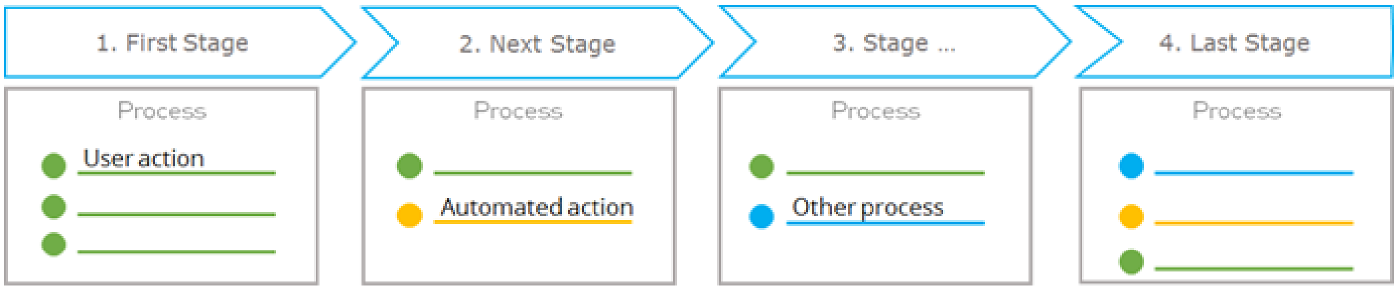
Processes

In a case life cycle, **processes** are organized within stages and define one or more paths the case must follow. You add a process to a stage in a case type to define a set of tasks that users accomplish as they work on a case.



Process steps

A process contains a series of tasks, or steps. A step can be an action that a user performs, an automatic action performed by the application, or other processes that contain a separate set of actions.

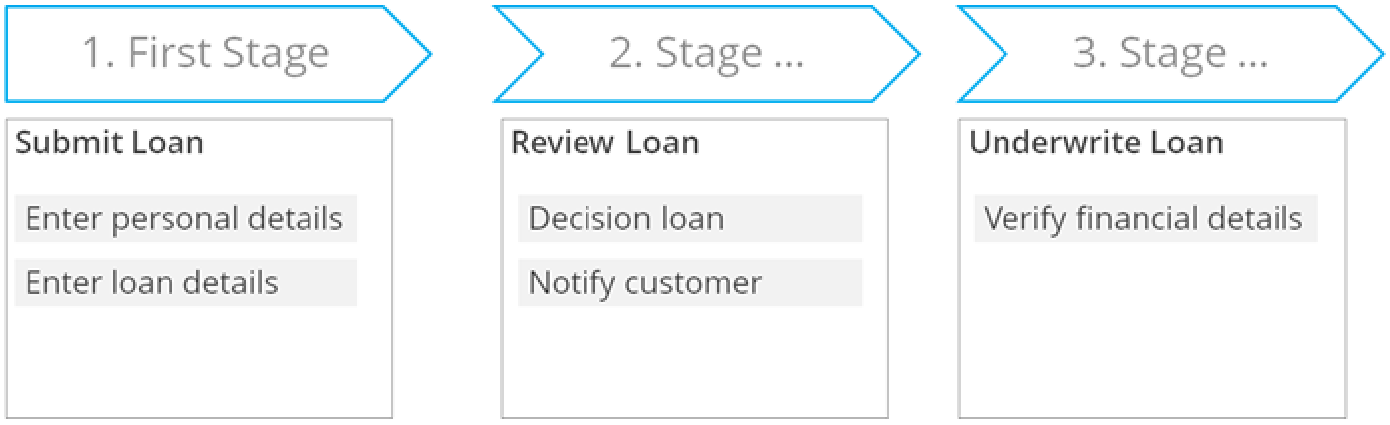


By organizing related tasks into processes, you can control how, when, and by whom work is performed in each stage of the case life cycle.

Guidelines for defining processes

Naming processes and steps

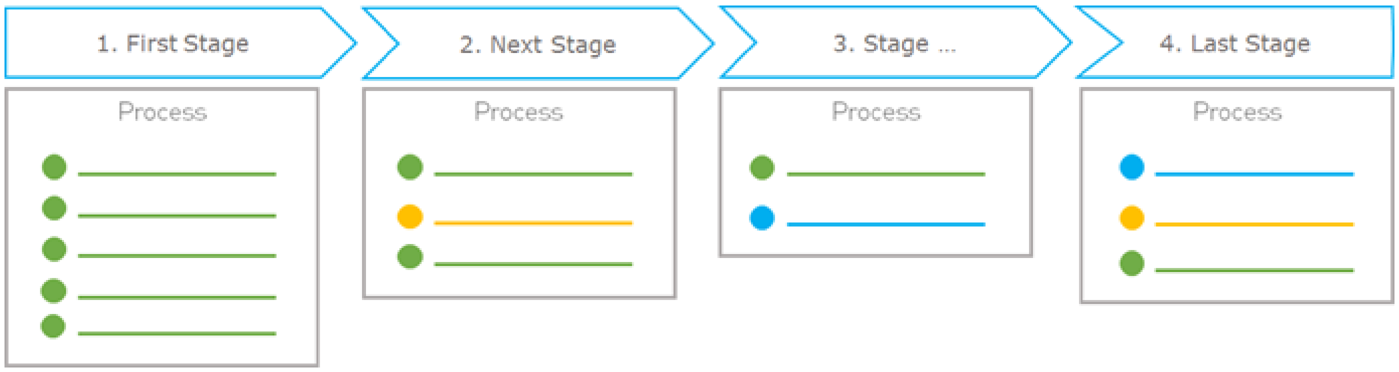
When naming processes and steps, use a “verb + noun” naming convention (ex. perform “this action” on “this object.”)



Consider every process as a distinct action taken to help resolve a case. Every process should have a goal that can be expressed as a singular outcome.

Organizing process steps

Consider limiting the number of steps in each process to five, plus or minus two.



If more than seven obvious steps are needed, consider breaking down some steps into other processes.

KNOWLEDGE CHECK

ANSWER In Pega 7, ________________ are organized within stages and define one or more paths the case must follow.

processes

Assigning work

Introduction to assigning work

In most business applications, more than one user works on a case until the case is resolved. Effective case design includes routing the right information to the right individuals at the right time.

Assignments allow you to determine the order in which users perform different tasks. The order in which the assignments are completed is managed through routing. Correctly routing work allows the right decisions to be made in a timely manner.

In this lesson, you will learn how to route an assignment to the correct user.

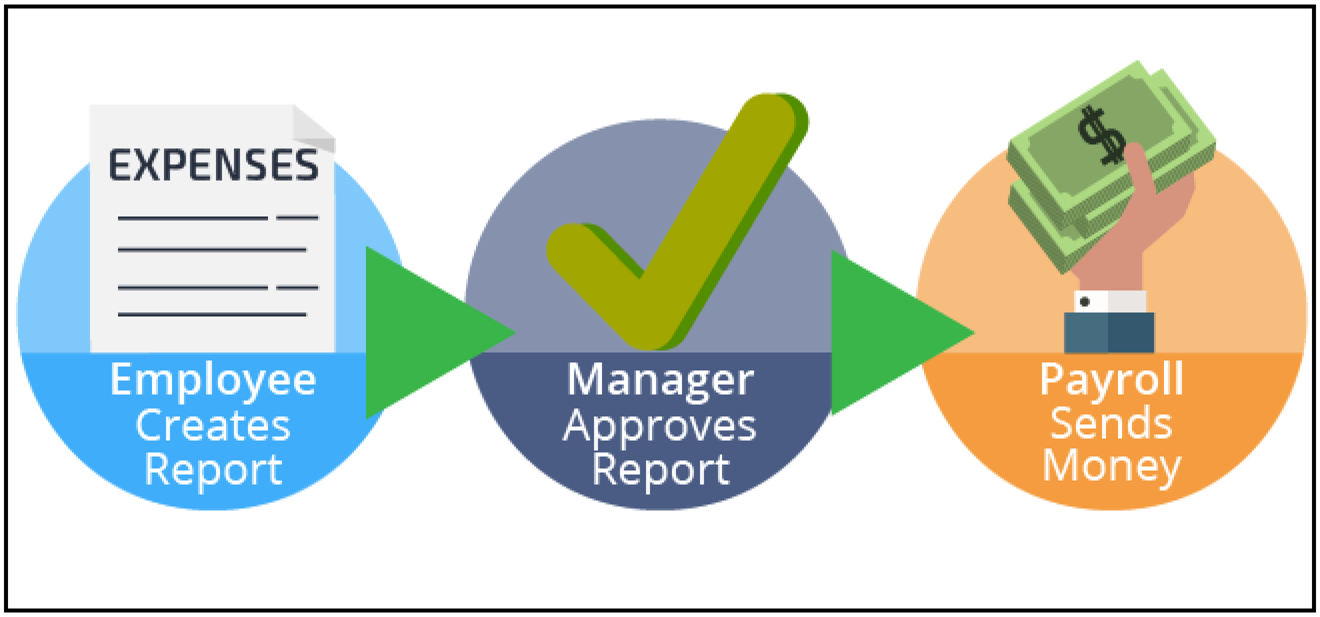
Objectives

At the end of this lesson, you should be able to:

- Describe the role of routing in a case
- Assign work to case participants

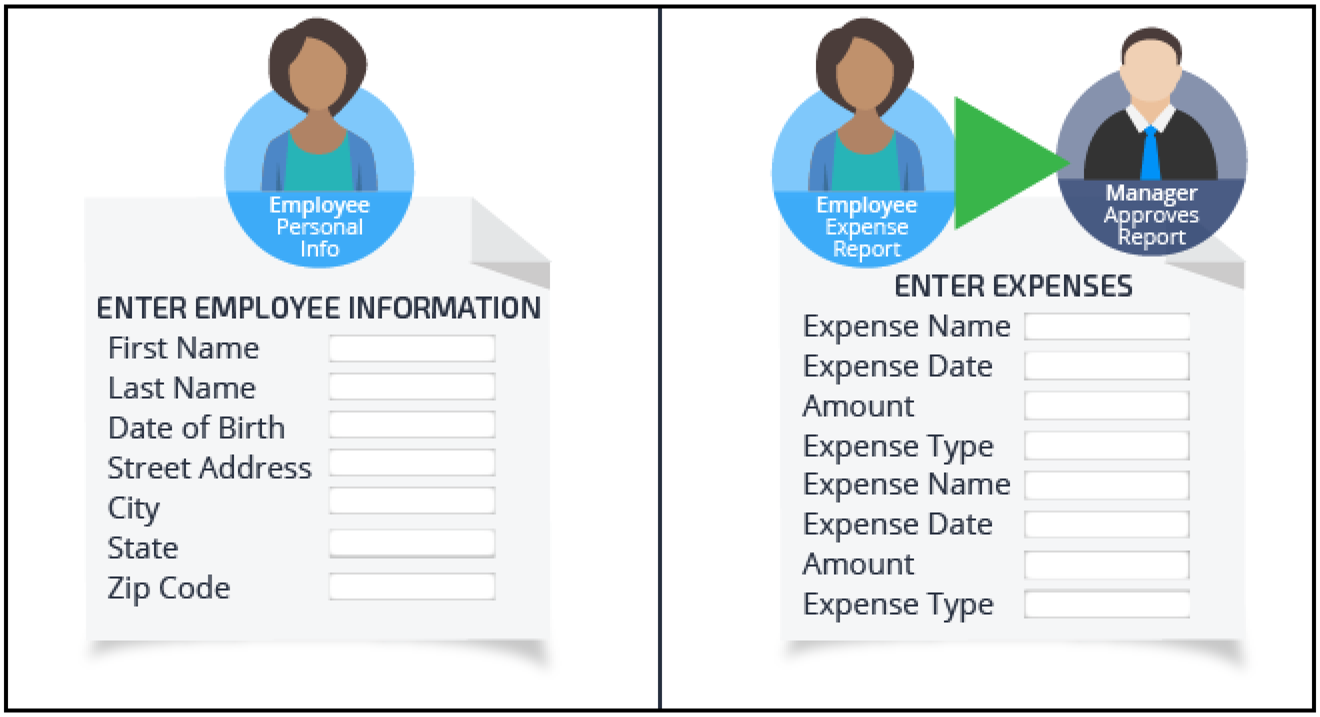
Assignment routing

When processing a case, it is common that more than one person completes works on the case. For example, when creating an expense report, an employee creates the report, a manager approves it, and payroll sends the money. That's three sets of people working on the same case.



As part of modeling a process you define where the work should go. Assignment steps define the work to do. The question you need to ask yourself when designing your assignment is: who should do the work?

You route the task to a single user if the current user should perform the task or you know the user to route the work to. For example, you would route to the current user if you have several data collection screens since the current user would likely perform them all. You would route to a specific user in the situation where you have an expense report approval process. The user who starts the expense report can't approve their own expenses. Instead you route the approval task to the manager of the employee to approve it.



You route a task to a group of users if a set of users could complete the task and it does not matter which user completes the task. For example, after a user completes an auto insurance claim, it does not matter which claim processor reviews the claim. The task can be routed to a work queue where claim processors go to get claims.

Assigning work to case participants

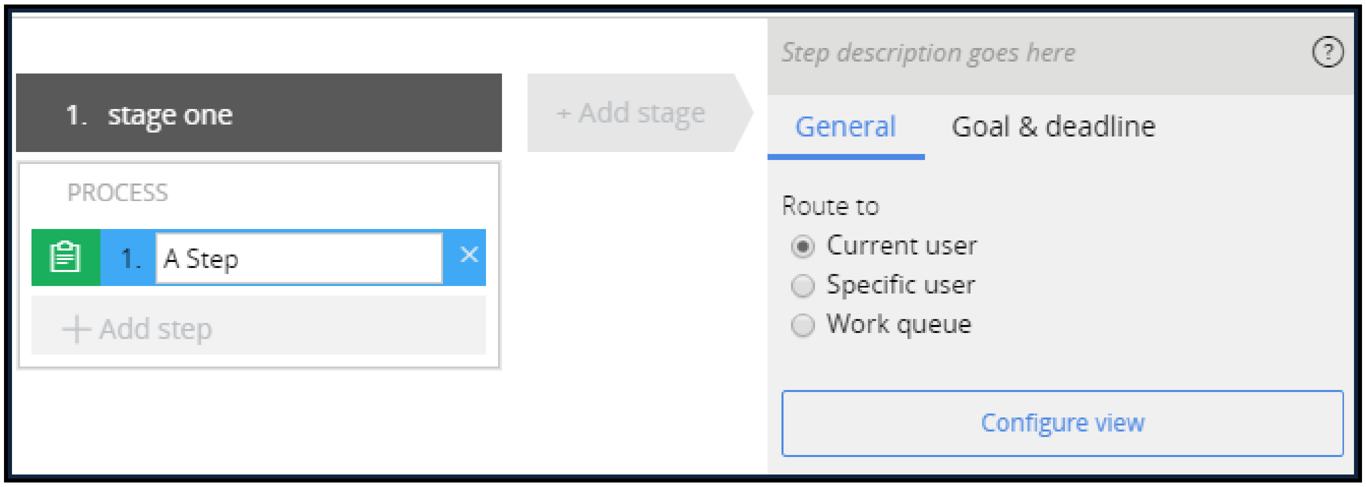
You configure routing as part of the step configuration. When you configure routing, you specify what user or group of users should complete the assignment. You configure routing using the *context properties panel* for the step.

The table below provides a list of who you can route assignments to.

Value	Description
Current user	The task is routed to the user who completed the previous task.
Specific user	The task is routed to a specific user— either a user name or reporting manager.
Work queue	The task is routed to a queue where any user with access to that queue can complete the task.

Follow these steps to configure routing:

1. From a case type, select the step you want to configure routing for.
2. Select who to route the task to.



Route to
- [✓ Current user]
- Specific user
- Work queue

3. Click **Done** to save your changes.

Enforcing service levels

Introduction to enforcing service levels

End users complete assignments and resolve cases to achieve performance milestones. These milestones are called service-level agreements (SLAs).

SLAs outline an expectation of service provided by a business to their customers. They stipulate the amount of time in which the business intends to respond to the issue. For example, end users must resolve customer complaints within 24 hours or complete cases in five business days.

Objectives

At the end of this lesson, you should be able to:

- Explain the purpose of goals and deadlines
- Explain how goals and deadlines can be used to improve case processing
- Explain the effect of urgency
- Describe the effect of an escalation action

Goals and deadlines

Case types model how and by whom work is completed, but equally important is how timely the work is completed. For example, if a customer submits a loan application, the customers should be able to expect a response withing a reasonable amount of time

A Service Level Agreement (SLA) helps ensure work completes within the expected time intervals.

You define service levels to define the expected resolution times for a step or case. An SLA contains three time interval milestones.

A **goal** milestone defines how long the assignment should take and is typically measured from when the step or case started.

A **deadline** milestone defines the longest amount of time the step or case may take before it is considered late. It is also measured from when the step or case was started.

A **past deadline** milestone defines when you may take further action if the step or case is too far past the deadline.

You define an **urgency** from between 10 and 100 for each milestone. The higher the value, the higher the urgency. Typically, the urgency increases as an assignment advances to the next milestone.

Note: You cannot configure the Past Deadline milestone in the Case Designer. If you have a requirement to use that time interval, document the details in the user story so a system architect can implement it.

KNOWLEDGE CHECK

ANSWER

You are designing an SLA for a mortgage request life cycle that includes a step in which a user collects the applicant's loan history. It is expected that the user completes this step within 24 hours when they receive the case. If the step is not completed within 48 hours, the case is considered late. What milestones and values do you use to support this requirement.

You enter a value of 24 hours in the goal milestone, and enter a value of 48 hours in the deadline milestone.

Adding service levels

Adding a service level to a case

To create a service level for an entire case:

1. In a case type, click **Life cycle**
2. Click **Settings**.
3. Click **Goal and Deadline** to configure a service level.
4. Select **Consider goal and deadline**.
5. Select when the timer for the goal starts.

Option	Description
This case	Starts the calculation when an instance of your case type is created
Parent case	Starts the calculation when the parent of your case type is created
Top level case	Starts the calculation when the top-level parent of your case type is created

6. Enter the goal.
7. Enter the deadline.
8. Click **Save**.

Adding a service level to a step

To add a service level to a step:

1. In a case type, select the step to add a service level to.
2. Click **Goal and Deadline** to configure a service level.
3. Enter the goal.
4. Enter an **Increase urgency by** value.
5. Choose an escalation action.
6. Enter the deadline.
7. Enter an **Increase urgency by** value.
8. Choose an escalation action.
9. Click **Save**.

Creating user views

Creating an Application User Interface

Introduction

Users perform tasks by entering information in forms. Application developers design forms so that users can enter the correct information to complete a specific task. This lesson helps you learn how to collect the necessary information when planning a new form, use a Pega configuration tool to map the information to your new form, and configure the fields..

Objectives

At the end of this lesson, you should be able to:

- Describe the role of user forms in completing tasks
- Remember the three questions to ask when planning the information you need on a user form
- Map the information to fields when creating a user form in the application
- Configure the fields to support your requirements
- Design a picklist field

Planning end-user forms

Enterprise applications typically require some human interaction. As a case goes through a process, end-users perform a variety of tasks along the way. To perform tasks, end-users enter or review information on end-user forms.

Not all end-users perform the same tasks. Forms are designed so that end-users can complete specific tasks. In order to complete the tasks, end-users enter information in fields on the forms. The system stores the values entered by end-users as data. The application uses this data to process a case. This data can be included on other forms so that a different set of end-users can perform their own specific tasks.

Forms designed for specific tasks

Consider a process for making loans. In this example, there are two steps and two forms.

The first step in the process requires customers to enter a loan application. The form contains fields for entering information such as the customer's name, the loan amount, and the loan type. After customers complete the form, the system sends the request to loan officers for review.

In the second step of the process, loan officers see a loan-officer form that displays the data collected from the application form. Loan officers can read the information but not update it. The loan-officer form also contains fields that allow officers to enter information such as loan insurance eligibility and the reason for approval.

Loan Application
Name
Address
Amount
Loan Type
select an item...
Requested Term
30 year
15 year

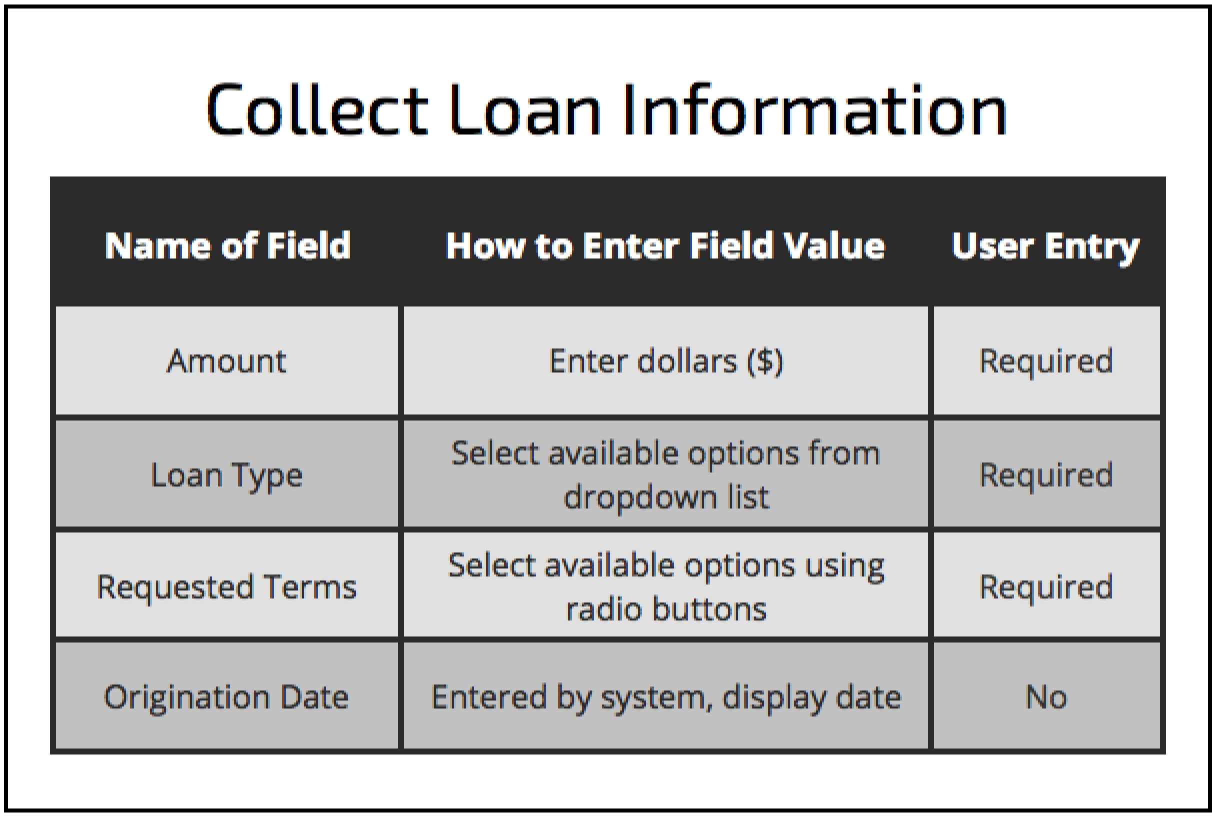
Identify the tasks a user will perform

As the business analyst or architect, you must determine what information end-users need to see or collect in order to perform their specific tasks. When you have defined the information end-users need, you create a form in which end-users enter the information.

Before you create a form, ask the following questions:

- What fields do end-users need to see?
- How will end-users enter values in those fields?
- Can end-users modify the field values or only read the values?

Record the values end-users enters in a specification. The following example shows the fields included in a loan application form.



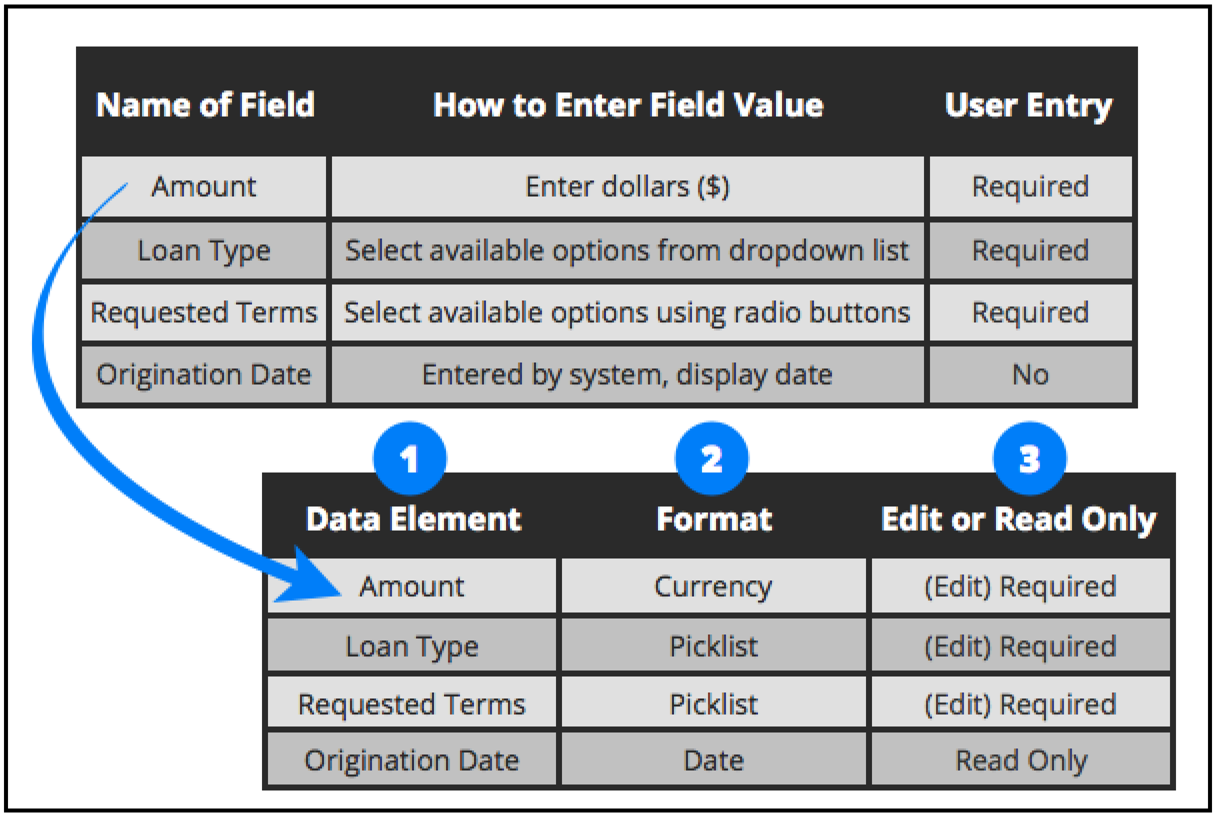
Collect Loan Information

Name of Field	How to Enter Field Value	User Entry
Amount	Enter dollars ($)	Required
Loan Type	Select available options from dropdown list	Required
Requested Terms	Select available options using radio buttons	Required
Origination Date	Entered by system, display date	No

Creating forms in your application

After you have your analysis, you are ready to create the form using the View Configuration tool. The tool enables you to define the data elements that the system uses for processing cases.

The View Configuration tool contains an array of rows — one for each UI field. Each row has three fields. The fields define the data element, format, and enable edit setting (required or optional).



Name of Field	How to Enter Field Value	User Entry
Amount	Enter dollars ($)	Required
Loan Type	Select available options from dropdown list	Required
Requested Terms	Select available options using radio buttons	Required
Origination Date	Entered by system, display date	No

1 Data Element	2 Format	3 Edit or Read Only
Amount	Currency	(Edit) Required
Loan Type	Picklist	(Edit) Required
Requested Terms	Picklist	(Edit) Required
Origination Date	Date	Read Only

Field 1: Which field do you want end-users to see in the form?

In the first field on a row, select the data element you want end-users to see in the UI field. Pega provides a large number of standard data elements to choose from such as Customer ID and Company. The list may also contain data elements that a system architect created for your application such as Loan Officer ID or Loan Office.

If the list does not contain the data element you need, type a name in the field. For example, assume your application does not have a data element for loan type. To create a loan type data element, enter the name Loan Type in the field. End-user sees the name as a label next to the new UI field.

When your form is complete, new data elements are saved to your application. You can reuse those data elements when you create new views.

Field 2: How do end-users enter the value in the field?

In the second field on a row, select a format. The format defines how end-users enter a value in the UI field. For example, if you want end-users to enter a date UI field, select the Date & time format. This format lets end-users select a date from a calendar icon. If you want to add a field that lets end-users select only one of two possible options, enter a Boolean format.

If you want end-users to select values from a list in the UI, select the Picklist format. A list enables you to define valid values the application uses for processing a case. Picklist formats have an extra field that lets you display the items either in a drop-down list or as radio buttons. This additional field also lets you enter the items you want to display in the list.

Selecting a Text format is incorrect for this field. This format allows end-users to enter free-form text in a box. End-users probably do not know which loan type names are valid. Entering incorrect names triggers error messages and creates an unsatisfactory user experience.

Field 3: Can end-users edit the field value?

In the third field on a row, select either Optional or Required if you want to allow end-users to enter a value in a field. Select Required if end-users must enter a value in order to complete the task and advance the case to the next step. Otherwise, select Optional so end-users do not have to enter a value to complete the task.

If you do not want end-users to enter or update the field value, select Read-only.

Configuring user views

After you add steps to the case life cycle design, you can configure user views for each of the steps.

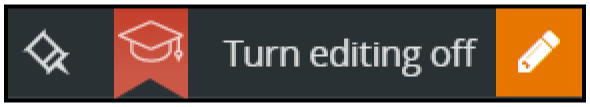
Before you configure a user view, remember to answer these questions:

- What fields do end-users need to see in the UI?
- How do end-users enter values in those fields?
- Can end-users modify the field values or are the fields read-only?

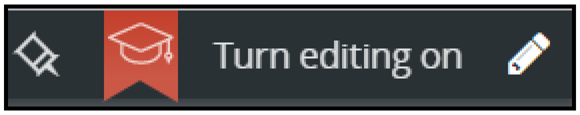
Configure a user view using Pega Express

To configure a user view using Pega Express:

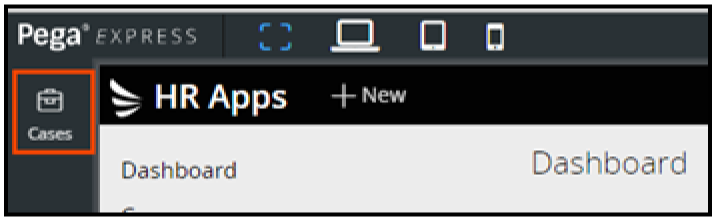
1. Confirm editing is enabled by looking at the upper right corner of the Pega Express dashboard. Look for the text **Turn editing off** as shown in the following image.



If editing is not enabled, click **Turn editing on**.



2. In the Navigation panel on the left, click **Cases** to view a list of current case types.

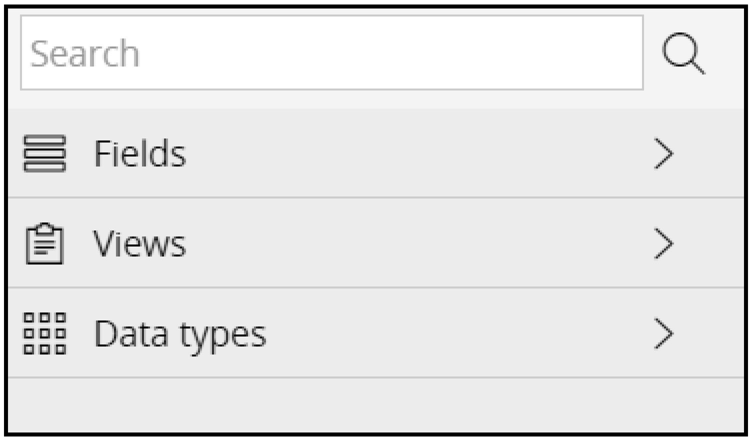


3. From the list of available case types, select the case type for which you want to configure a user view.
4. Select the step for which you want to configure a user view to display Contextual Properties panel for that step. The Contextual Properties panel displays to the right of the case life cycle.
5. In the contextual properties panel, click **Configure view**. The *Views* configuration page is displayed.

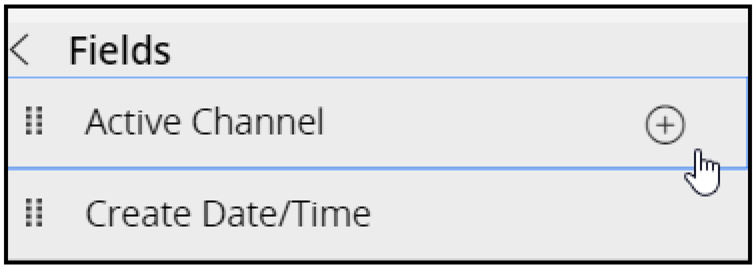
Add default fields to a user view

To view the default fields and select fields to add to the user view:

1. In the left panel, select **Fields** to view the default data elements.



2. If you require any of the default fields, select the row for the field, and then click the plus sign to the right of the field name.

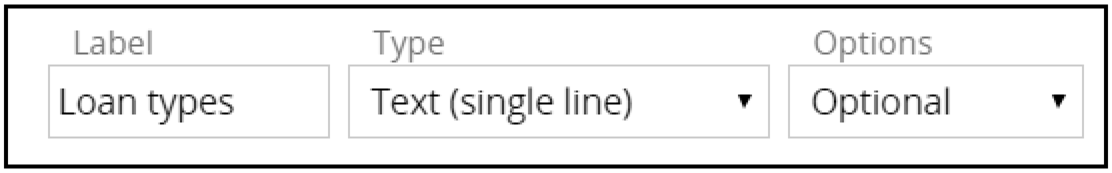


3. Repeat steps 1 and 2 to add more default fields.

Add new fields to the user view

To add new fields to the user view:

1. In the **Label** field, type a name for the new field. the following image shows a new Loan types field:

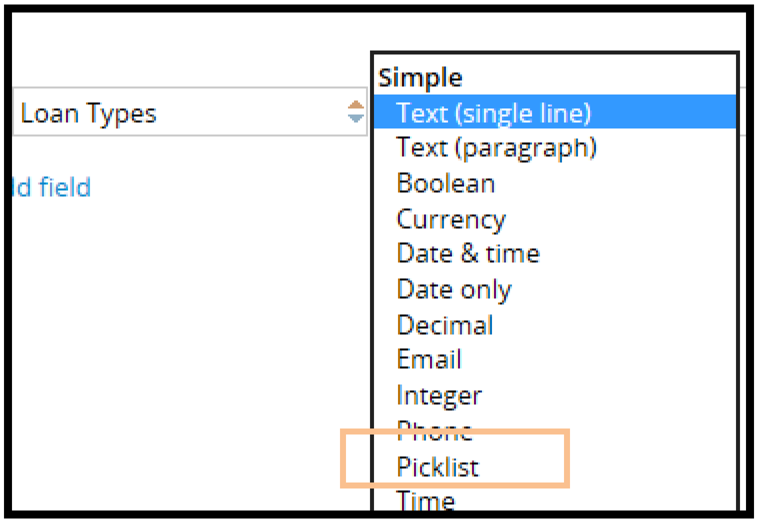


Label	Type	Options
Loan types	Text (single line)	Optional

Note: After you add the required data element and save your view, the system adds any new data elements to your application. You can then reuse those data elements when you create new views.

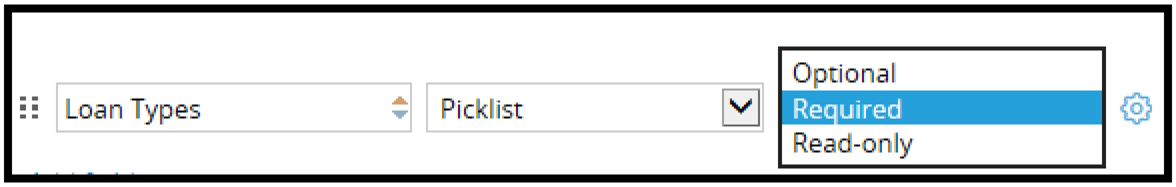
2. Use the **Tab** key to move to the second field on the row.
3. Select a data type for the data element. The data type defines how users enter a value in the UI field.

   For example, if you want the user to select a type of loan from a drop-down list in the Loan types UI field, select Picklist from the drop-down in the data type field as shown in the following image:



The Picklist data type has an additional field for you to choose the type of list (drop-down list or radio buttons) and the names of the items on the list. To learn how to choose the type of list and to configure item names, see the steps under **Designing a picklist**.

4. Use the **Tab** key to move to the third field, select either Optional or Required if you want to allow users to enter a value in a field. If you do not want users to enter or update the field value, select Read-only. In the following example, the developer is selecting Required to ensure the user selects a value from the Loan Types list.

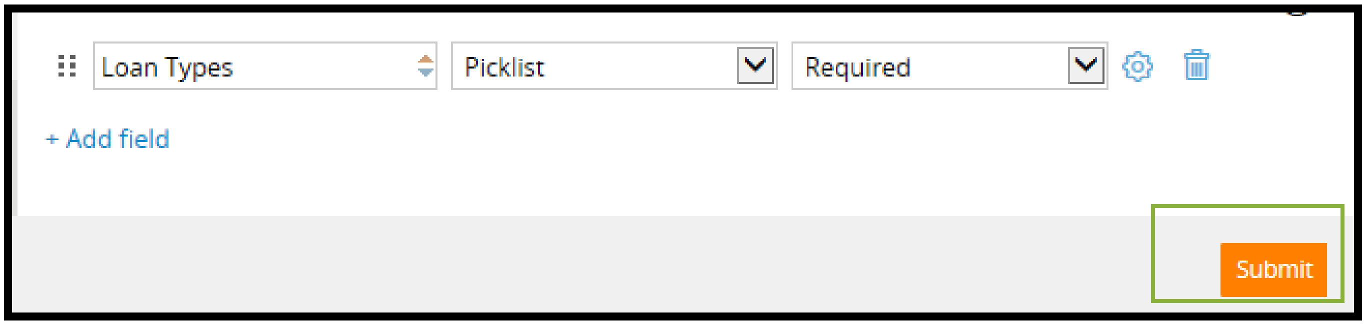


5. To add more fields, click **Add Field** beneath the bottom row.

Save and verify your work

To save your work and review the view:

1. On the bottom right corner of the View configuration page, click **Submit**. When you click **Submit**, the system saves your updates and creates the view that users see when they work on an assignment. The system also saves the data elements that you can reuse in the application.



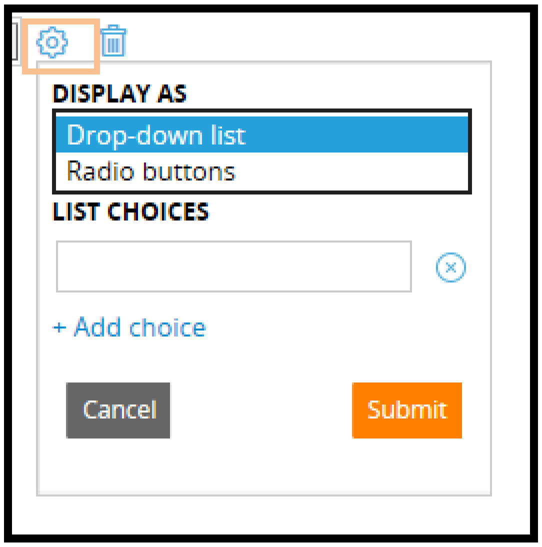
2. Click **Save** to save your changes to the case type.

3. In the upper right corner, click **Run** to run the application. The new fields in the standard Create view display.

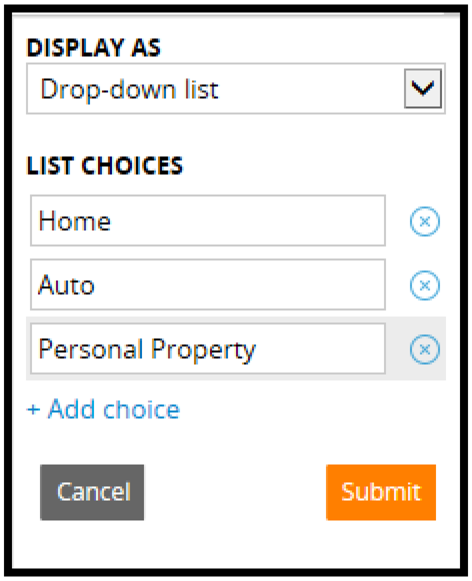
Designing a picklist

If you selected the picklist data type, you need to use an additional field to choose how to display the list and the names of the items you want to include on the list. To design the list, do the following:

1. From the end of the row containing the picklist data type, click the **Gear** icon.
2. In the **Display As** field, select one of the following:
   a. Drop-down list if you want to users to select an item from a drop-down list.
   b. Radio buttons if you want users to select an item by clicking a radio button.

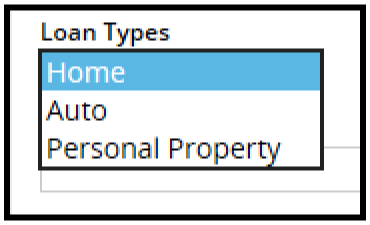


3. Under **List Choices**, enter the name of the first list item.
4. Click **Add choice** to add more fields for items in the list. The following example shows list choices for loan types.


DISPLAY AS: Drop-down list

LIST CHOICES
- Home
- Auto
- Personal Property

5. Click **Submit** in the dialog box to save your list. The items you entered in the List Choices column display in the **Loan Types** drop-down list in the user view.



CASE DESIGN USING DESIGNER STUDIO

Requirements management

Introduction

When creating an application, everyone must know what to build. Everyone should know what currently exists in the application and how the application relates to the requirements or specifications.

This lesson reviews how to build better applications by understanding the importance of requirements management. The lesson also reviews how business objectives, requirements, and specifications relate to one another. You will learn how DCO is used to ensure that your application continually meets business needs.

Objectives

At the end of this lesson, you should be able to:

- Describe the relationship between requirements and specifications in an application
- Describe how requirements management improves the effectiveness of application development
- Explain the role of business objectives in application design
- Describe the purpose of requirements
- Describe the purpose of specifications
- Explain how DCO helps to ensure that applications match business needs

Requirements management

Requirements management is the process of collecting, analyzing, refining, and prioritizing product requirements, and then planning for their delivery. Requirements management helps ensure the business application you build validates and meets the needs of all customers and stakeholders. Requirements management is a continuous process throughout a project.

When building an application, you follow a set of requirements that define what the application must be able to do. The success of an implementation depends on your ability to understand, track, and trace these requirements.

Requirements management helps keep the project team organized and provides visibility into every aspect of the project. In Pega, you use DCO throughout a project to keep your requirements up to date.

KNOWLEDGE CHECK

ANSWER How does requirements management improve your implementation process?

Having the ability to write, track, and trace a requirement allows for increased collaboration and accountability during the process.

Managing Requirements

Managing requirements, and other artifacts, for any project can be a daunting task - And it certainly is NOT for lack of trying. The challenges faced when trying to manage requirements isn't so much the requirements themselves.

It's the tracking and coordination – the management - of those requirements. And the other, inevitable, artifacts created to support and implement the requirements.

With any application development project, it takes many types of artifacts to produce the final product. For example, there is typically a product requirement document that defines the scope of what the application should be.

This document might be stored on a collaboration site for easy accessibility.

From that document, business requirements and use cases are usually written.

It is not uncommon to see these artifacts stored on a network drive somewhere so all project team members have access to them.

There will undoubtedly be functional and technical requirements.

Given the nature of these artifacts, they may end up in a versioning control system.

Finally, there will almost certainly be business processes – or, flow – diagrams and UI wireframes.

It would come as no surprise to see these artifacts stored locally on a process owner's local drive.

With so many important artifacts scattered across so many different locations,

under the control of so many different authorities, well….it's enough to make anyone scream.

Pega's Direct Capture of Objectives – or, D C O - can help bring order to this chaos.

Pega's DCO is a set of features that enable project team members to directly capture, organize and manage requirements

- and associate them with the specific parts of the application that are implementing them.

This visual representation and traceability of how requirements interact and depend on one another is very powerful. Designed to be an enabling technology,

Pega's DCO can be used collaboratively and productively by project teams that include members of the business analyst, developer, quality assurance and IT and end user communities of your organization.

Business objectives

To successfully run any business application development project, you must establish clear business objectives.

Business objectives are statements that describe the business value the application must provide, or the business needs the application must address. Business objectives may apply to the organization as a whole, lines of business, departments, employees, customers, and even marketing efforts. Business architects review the current state of a business process to identify inefficiencies. Business architects then create business objectives to fix the inefficiencies.

For example, a business determines that the existing process to manage purchase requests takes too long. You define the following business objective: *Processing time for purchase requests must be no longer than 3 business days.*

In another example, a business objective for an insurance claim application may state that the application must be able to *reduce inaccurate claims to less than 10% of all claims*.

Business objectives also help establish the scope of the application development project. Using clear, measurable objectives helps ensure that all stakeholders share a common understanding of the business application's abilities.

Application requirements

The term **requirement** has different meanings within different organizations. Gathering requirements is the most critical part of any business application development effort. When building the application nothing should be left to interpretation.

The IEEE Standard Glossary of Software Engineering Terminology defines a requirement as:

- A condition or capability needed by a user to solve a problem or achieve an objective;
- A condition or capability that must be met or possessed by a system to satisfy a contract, standard or specification.

In the simplest terms, Pega defines a requirement as an agreement between stakeholders on what a business application will do.

A requirement uses business language to describe what the application must do to meet your business needs.

Requirements can range from high-level abstract statements of services to more detailed functional specifications.

ABSTRACT
View Order History
REQUIREMENT
DETAILED
Modified data in a database should be updated for all users accessing it within two seconds
REQUIREMENT

Requirements can also provide benchmarks to test your application against.

Think of requirements as an inventory of events, conditions, or functions that must be implemented and tracked in a development project.

At Pega, requirements are categorized into one of five types:

Type	**Description**
Business rule	Identifies requirements usually associated with a specific use case or step in a process
Change Control	Identifies how to manage changes in the application
Enterprise Standard	Identifies requirements that apply across the enterprise, or are an industry standard that all applications must adhere to
Functional	Identifies a function that will be used in the application, such as calculations or data manipulation
Non-functional	Identifies performance metrics, such as screen-to-screen interaction times

Requirements 101

Knowing what a good requirement looks like is important. What seems like a good requirement to one person may not necessarily be understandable to another. Here you see a team creating requirements for browser support for their application.

Requirements should use business terms.

Using business terms allows all members of the implementation team to understand the requirement.

This is a good start. It uses business terms to define the requirement. The requirement does use business terms, but there are some things wrong with it.

First, requirements should be atomic. If a requirement needs to implement multiple elements separately, it is not atomic.

Our example is not atomic. The need for backwards compatibility should be in a separate requirement.

Requirements also should be clear and concise.

Write requirements so that they are clear to those who implement them. The use of specific and appropriate language can help avoid ambiguity in interpretation.

In the example, the word 'all' may seem clear, but it is not.

Requirements should also be verifiable. You must write requirements so they can be tested by inspection, analysis, or demonstration.

Besides 'all' not being clear, you could not likely test every version of every browser.

This is an example of a good requirement.

The requirement uses business terms.

The requirement is atomic.

The requirement is clear and concise.

The requirement is verifiable.

There is one last characteristic of good requirements. Requirements must be consistent. Consistent requirements do not conflict with other requirements. Make it a habit to inventory the existing requirements before adding new ones.

This is not an exhaustive list for writing good requirements. You should now be able to identify a good requirement.

Application specifications

Once you understand what you want to do, you need to turn your attention to how you are going to do it. Specifications define how you implement your application.

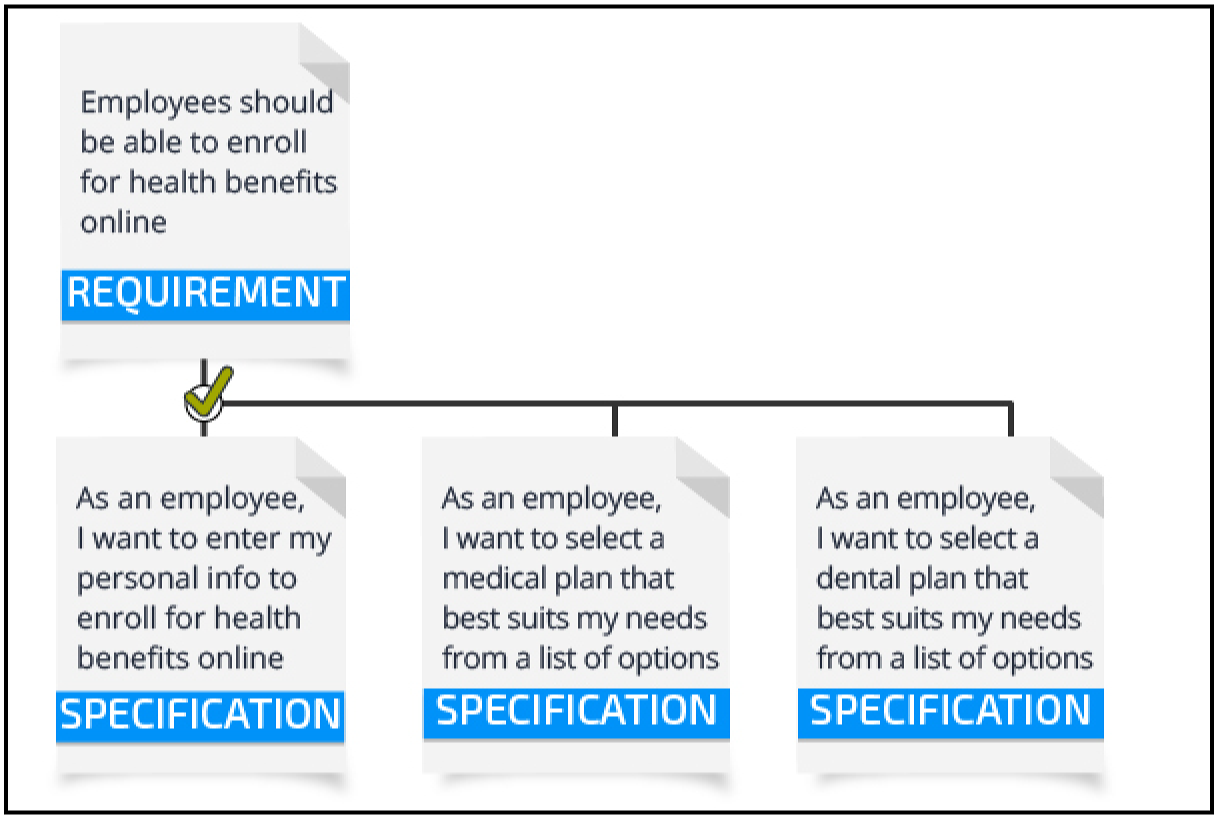
Specifications use business language to describe the steps needed to meet a requirement.

For example, you have a requirement that states employees should be able to enroll for healthcare benefits online.

The example defines what you need to build, but does not help you do it. Use specifications to define how to implement that requirement.

You first determine that to enroll for healthcare benefits online you need to capture an employee's personal information.

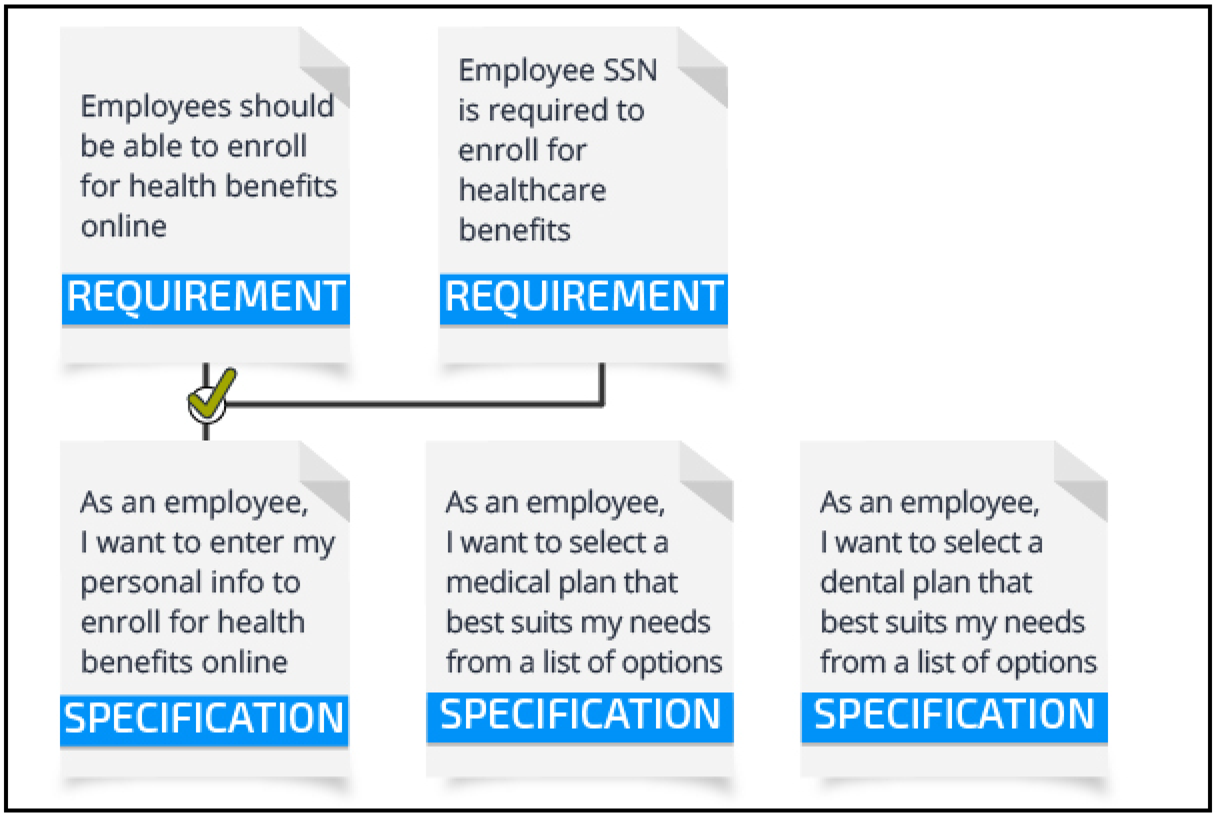
Next, you have employees select medical benefits from a list of available plans. Then, employees should enroll for dental benefits.


Employees should be able to enroll for health benefits online
REQUIREMENT
As an employee, I want to enter my personal info to enroll for health benefits online
SPECIFICATION
As an employee, I want to select a medical plan that best suits my needs from a list of options
SPECIFICATION
As an employee, I want to select a dental plan that best suits my needs from a list of options
SPECIFICATION

You continue documenting the steps needed to achieve the goal of the requirement.

If applicable, a requirement can be related to many specifications.

A single specification can also reference many requirements.


Employees should be able to enroll for health benefits online
REQUIREMENT
Employee SSN is required to enroll for healthcare benefits
REQUIREMENT
As an employee, I want to enter my personal info to enroll for health benefits online
SPECIFICATION
As an employee, I want to select a medical plan that best suits my needs from a list of options
SPECIFICATION
As an employee, I want to select a dental plan that best suits my needs from a list of options
SPECIFICATION

Specifications 101

Knowing what a good specification looks like is important. What seems like a good specification to one person is not necessarily understandable to another. For example, the requirement *An employee should be able to enroll in benefits online* has several specifications associated with it. Here you see a team creating a specification to collect employee information.

First, the system architect suggests a specification.

Like requirements, you write specifications in business terms. Using business terms allows "all" members of the implementation team to understand the specification. This specification is not in business terms and would not be clear to an end user or a business analyst.

Next, the end user suggests a specification.

Specifications need to be complete and not left open to interpretation. This example is not clear as to what information should be collected.

The team now comes up with another version of the specification.

The team is getting closer, but a specification should never involve a change of ownership until the step is complete. In this specification, the team suggests what should happen after the information is collected.

The team now decides to take out the part about manager review.

While close, this version of the specification is still missing something. A specification provides enough detail for an architect to implement the specification. A specification also needs enough detail so a tester can test the implementation.

The team now suggests yet another version of the specification.

The team has hit the mark! This is a good specification.

The specification uses business terms.

The specification is complete.

The specification does not change ownership.

The specification can be implemented and tested.

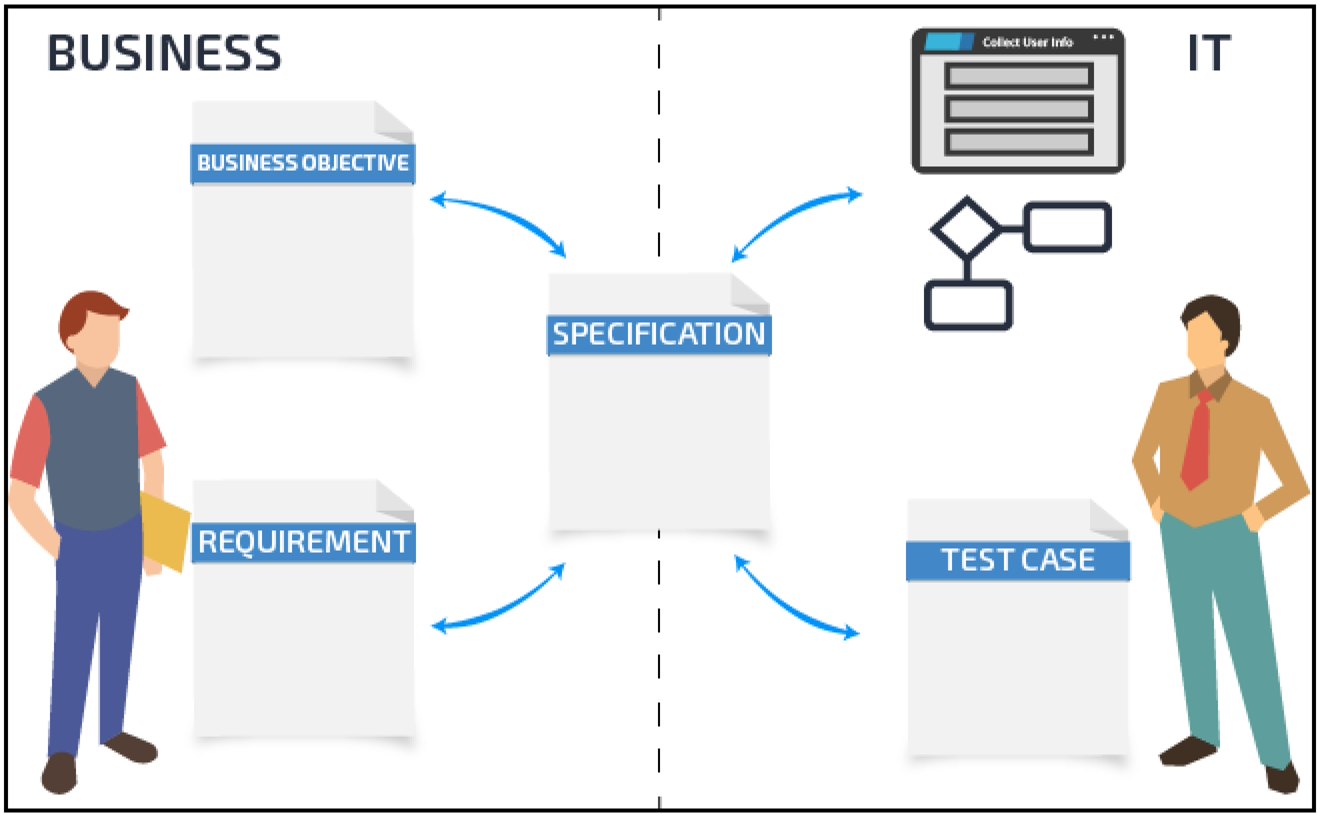
Creating good specifications makes your application easier to build. Good specifications also document what is in the application.

Relationship between Pega design artifacts

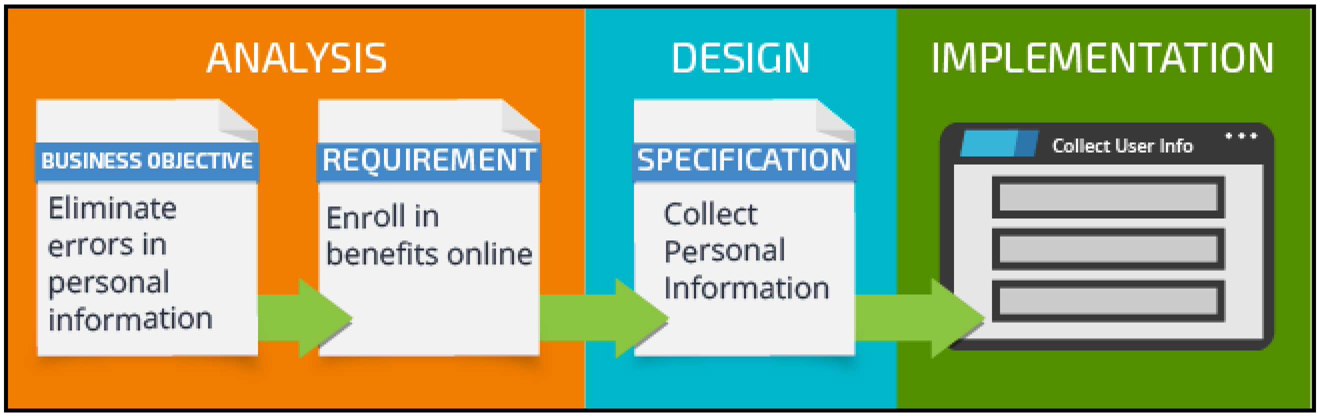
Your assignment is to work on the next version of an application. You need to see what already exists in the application and, more importantly, why it exists. To do this you need to examine how the specifications, requirements, business objectives, and implementation artifacts relate to one another.

Traceability is the ability to link specifications back to business objectives and requirements, and forward to implementation artifacts, and test cases.

In Pega, specifications are the center of traceability. With specifications at the center, it allows anyone to look backwards to see the requirements and business objectives, and forwards to see the artifacts that implement the specification. This gives a complete picture of what the business wants and IT builds.



In Pega, you link your business objectives, requirements, and specifications as you create them. To see how this works, consider the following example.


ANALYSIS
DESIGN
IMPLEMENTATION
BUSINESS OBJECTIVE
Eliminate errors in personal information
REQUIREMENT
Enroll in benefits online
SPECIFICATION
Collect Personal Information
Collect User Info

Business objectives are identified at the inception of the project. In the example the business objective identified was *Eliminate errors in personal information*.

A business architect elaborates on the *Enroll in benefits online* requirement and generates a set of specifications. At this time you link specifications to the requirements they link to. By doing this now you have an accurate picture that they relate to one another and you save time from having to do it later when there are more requirements and specifications to manage. When the specification is implemented, it is the system architects job to link the implementation to the specification. In the example that is mapping the UI screen to the *collect personal info* specification.

Linking specifications to business objectives and requirements

The Application Profile allows you to link specifications to requirements, business objectives, and the artifacts that implement the specification.

By tracking all of these artifacts in the Application Profile, you gain complete traceability in your application.

The table below displays the three tabs contained in the Application Profile.

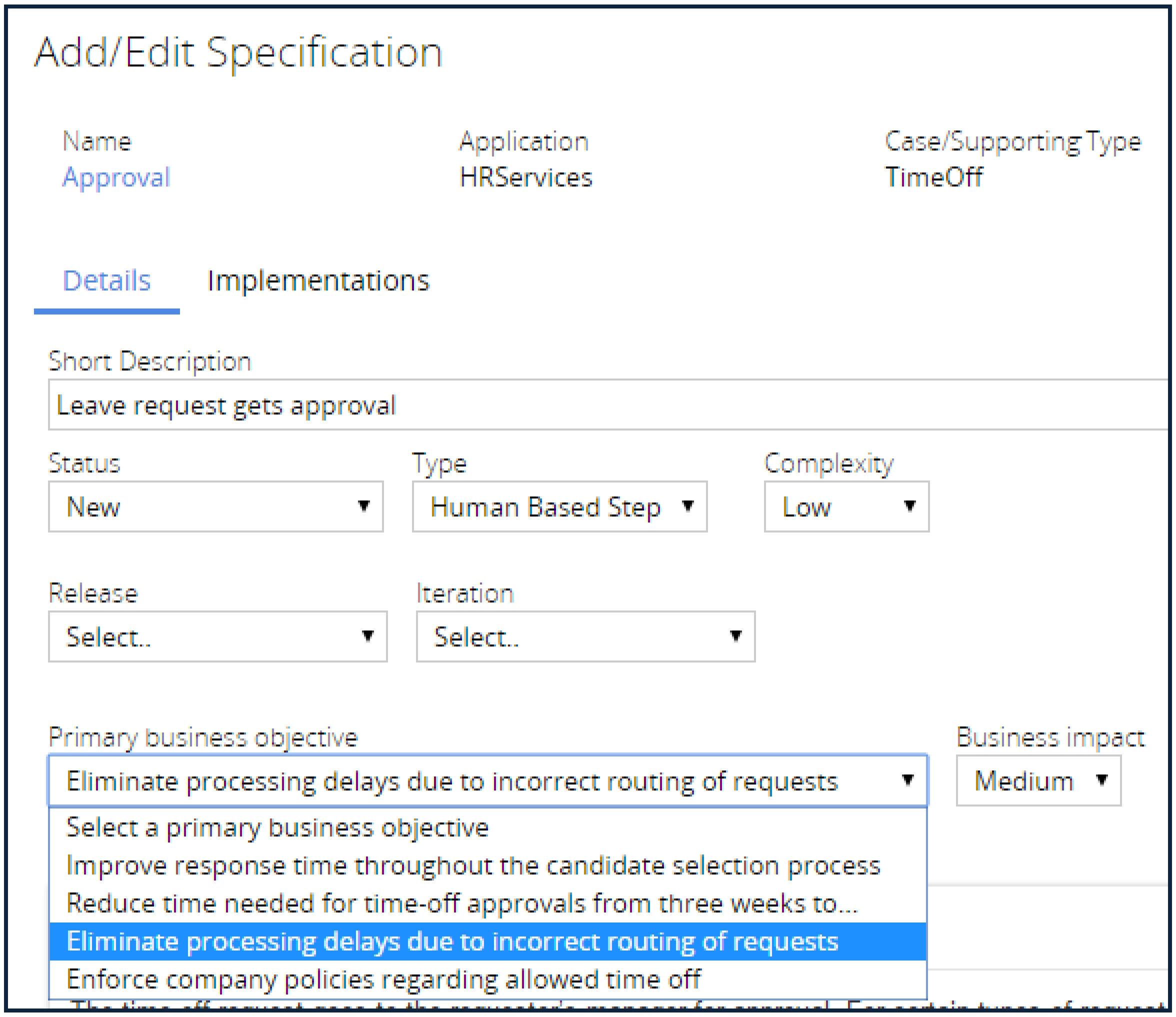
Tab	Description
Requirements	Review existing, create, or delete application requirements.
Specifications	Review existing, create, copy, or delete application specifications.
Analysis	Work with interactive charts to understand the distribution of specifications in your application across case type and status.

Linking a specification to a business objective

Follow these steps to link a specification to a business objective:

1. In Designer Studio, click **Designer Studio > Application > Profile > Specifications**.
2. Select a specification.

3. Select a primary business objective.



Add/Edit Specification

Name	Application	Case/Supporting Type
Approval	HRServices	TimeOff

Details | Implementations

Short Description: Leave request gets approval

Status: New
Type: Human Based Step
Complexity: Low

Release: Select..
Iteration: Select..

Primary business objective: Eliminate processing delays due to incorrect routing of requests
- Select a primary business objective
- Improve response time throughout the candidate selection process
- Reduce time needed for time-off approvals from three weeks to...
- Eliminate processing delays due to incorrect routing of requests
- Enforce company policies regarding allowed time off

Business impact: Medium

The time off request goes to the requestor's manager for approval. For certain types of request

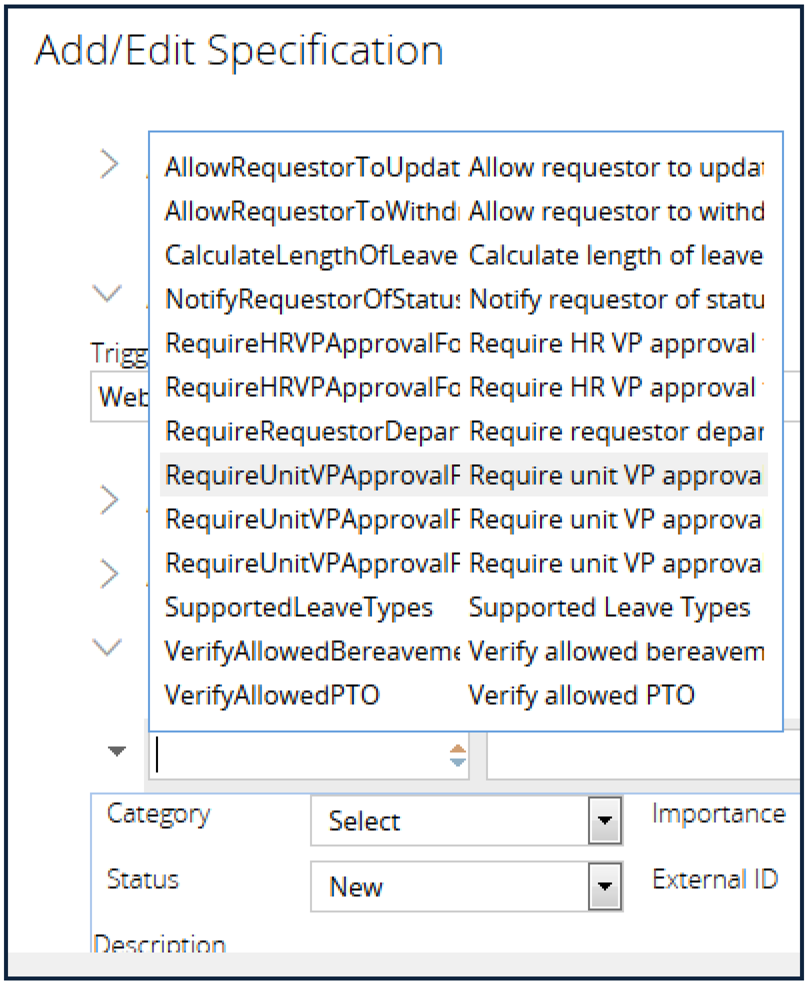
4. Click **Submit**.

Linking a requirement to a specification

Follow these steps to add a requirement to a specification:

1. In Designer Studio, click **Designer Studio** > **Application** > **Profile** > **Specifications**.
2. Select a specification.
3. Click **Advanced**.
4. Click **Requirements**.
5. Click the **plus** icon.

6. In the name field, select an existing requirement.



Add/Edit Specification

AllowRequestorToUpdat	Allow requestor to updat
AllowRequestorToWithdi	Allow requestor to withd
CalculateLengthOfLeave	Calculate length of leave
NotifyRequestorOfStatus	Notify requestor of statu
RequireHRVPApprovalFo	Require HR VP approval
RequireHRVPApprovalFo	Require HR VP approval
RequireRequestorDepar	Require requestor depar
RequireUnitVPApprovalF	Require unit VP approva
RequireUnitVPApprovalF	Require unit VP approva
RequireUnitVPApprovalF	Require unit VP approva
SupportedLeaveTypes	Supported Leave Types
VerifyAllowedBereaveme	Verify allowed bereavem
VerifyAllowedPTO	Verify allowed PTO

7. Click **Submit**.

Best practices for case design

Introduction to best practices for case design

There is no real right or wrong way to model the life cycle of a case, and the decision about which way is best usually comes down to opinion.

Establishing best practices helps ensure you are consistent in how you represent a specific event or action when translating a business process into a case type – including building the individual flows.

After this lesson, you should be able to:

- State the best practices for case design

Effective process design: collaborate, elaborate, iterate

Designing an individual process is a trial-and-error process. When case designers design a process, the result does not always align to the true needs of the business. To minimize the time spent designing incorrect processes, you must engage the business throughout the design process.

Collaborate with the business

The best way to ensure that you design what the business wants is to involve the business in the design process. You must engage business stakeholders at the beginning of the project, and keep them engaged throughout the project.

When you involve business in the design from the beginning, the design more accurately reflects the desired process. Also, by involving the business early in the design process, stakeholders can identify errors, mistakes, and misconceptions before they become costly to fix. A process playback during elaboration is a much more efficient use of project resources than fixing a broken process immediately before you release the application.

Elaborate your design

Case design on the Pega platform is a continuous process of elaboration at every level of the design. This elaboration is especially important when designing individual processes.

In the early stages of application development, you lack most of the rules that describe case behavior. Each process is only a collection of flow shapes arranged into a sequence of steps, and your UI is nonexistent. A lot of work remains to be done.

In Pega, the key to efficient process design is elaboration. Prototype the process and review the prototype with business stakeholders to identify errors or omissions in specifications.

Focus on the important components first: your data needs, and the processes in the primary path.

- Start with the most important aspects of process design, and make sure that your steps are named clearly and sequenced correctly. Add business context to the process through the use of instructions, audit notes, and work status. Use Pega's draft flow capabilities to play back the process with the business to review and correct the design.
- Once the process order is correct, further elaborate the design by determining your data needs and adding the user interface.
- Once the data and UI needs are well-established, continue the elaboration by focusing on the policies and reports that depend upon the data model. Correspondences that incorporate case data, service levels that enforce deadlines and escalate overdue tasks, decisions that evaluate case data, and reports that measure productivity should all be configured once the data model and UI are well-understood.

Typically, once case designers complete the primary processes, UI and data designers take over. This allows the case designers to start designing the less critical processes.

Iterate until the design is correct

Elaboration is a lot of work, and overwhelms case designers with details when they should instead focus on ensuring that the process is correct. Unfortunately, most of these details are not available at the beginning of elaboration. This leaves a development team with a quandary: start elaborating the design immediately and iterate the design as details are agreed upon, or wait until all of the details are known and elaborate the design once.

To avoid this quandary, take advantage of Pega's draft flow capability. With a draft flow, you bypass normal process validation to save and run a process without creating any of the rules that the process would call. Draft flow execution allows case designers to test a process even though the UI and decisions that normally guide case processing are not yet created. Instead, the application provides an alternative UI that allows the case designer to pick the outcome of a decision or assignment.

Thanks to the draft flow capability, you do not need to worry about creating the correct design right away. Rather, you can iterate your design throughout the elaboration process, starting with the most important aspects of the design and gradually incorporating rules as they are created, until the process design is complete. Refining a process through successive iterations allows stakeholders and case designers to identify issues, propose solutions, and test implementations early on, before the ramifications of a change ripple throughout the case design.

Remember that process modeling is an iterative process of discovery, construction, and assessment that yields the correct design. If you focus first on the design of your process, and then configure the shapes and subordinate rules, your processes are more understandable, and easier to maintain and update as business needs change.

View Transcript

When designing a business process, collaboration is key. Understanding how to define and represent each step is best accomplished when both the business stakeholders and the development team contribute to the process design.

Effective process design can only be accomplished when business stakeholders are engaged throughout the entire development cycle.

In the early stages of development, when the focus of the application build effort is on elaboration, business architects are usually the primary drivers of the design patterns.

Business stakeholders are key allies for business architects, helping to accurately capture business needs. Stakeholders can verify that the case and individual processes reflect the correct design, and that the steps are in the correct order and named clearly.

Once the design is correct, system architects become the primary drivers of the development cycle. System architects configure the behavior outlined in the specifications, and help to improve the design by identifying portions of the process that may be reusable and looking for manual steps that can be automated.

As system architect configure a process, they perform “playbacks” to review the partially implemented process with the business stakeholders, with the goal of discussion, consensus building, collaborative improvement and, ultimately, approval of the model.

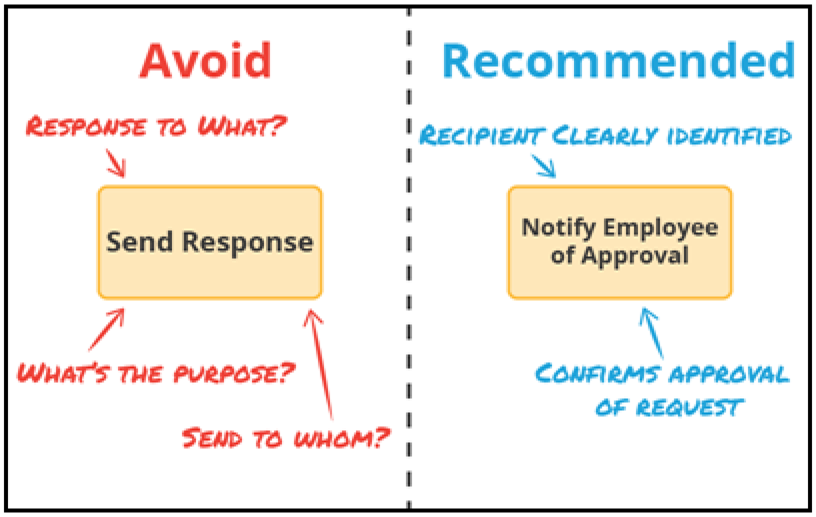
Designing intent-driven processes

When you design a process, ensure that all project team members and stakeholdersshare an understanding of the process. An **intent-driven process** fosters a common understanding between developers, stakeholders, and end users. With a common understanding, case designers collaborate efficiently with business stakeholders to review processes. Well-understood processes allow case workers to process cases with minimal application training.

When you create any process you first select the shape for the task. Next you ensure there is a single purpose for the shape and clearly label the shape and associated actions. intent-driven process, you add cues that guide business stakeholders and case workers through the process.

Label flow shapes and actions

To avoid confusion about a process, label flow shapes and actions to identify intent. Create labels that use terms familiar to the business. The label clarifies flow logic for project stakeholders during playbacks. The label also clarifies application documentation for case workers and other case designers.

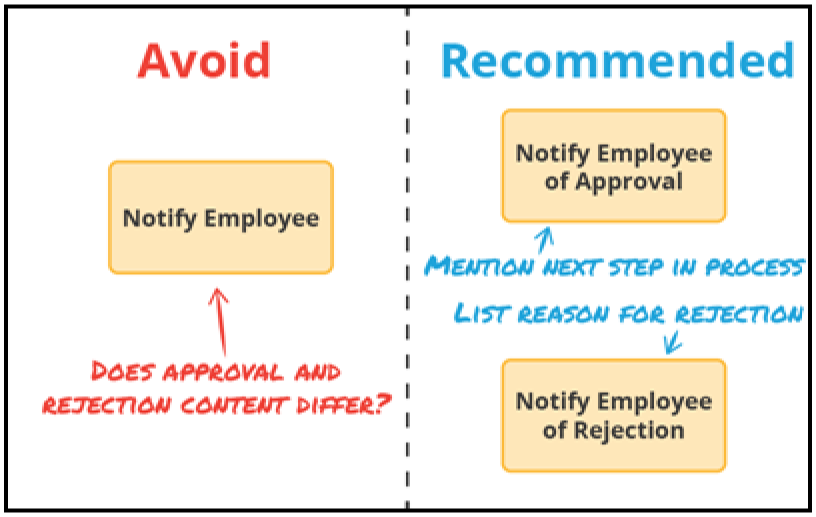

Avoid
Response to what?
Send Response
What's the purpose?
Send to whom?
Recommended
Recipient clearly identified
Notify Employee of Approval
Confirms approval of request

Ambiguous labels cause confusion and doubt about the behavior of a process. For example, do not label a shape with Send Response. Instead, use the label Notify Employee of Approval, which is easier to understand.

Assign a single purpose to each shape

When you design a process, use as few shapes as possible . If two shapes perform the same task, determine whether you can reduce the two shapes to one. Review the use case for the process. Ask yourself, "Does each shape need a unique specification?"If the answer is yes, then use two shapes to develop your process. Otherwise, remove one of the shapes from the process to eliminate unnecessary complexity.

For example, an application sends an email whenever an expense report is approved or rejected. The approval email identifies the next step in the process. The rejection email lists the reason for the rejection. Since the two notifications contain different content, the requirement indicates two different specifications. As a result, you must use two shapes; each shape serving a single purpose to approve or reject.

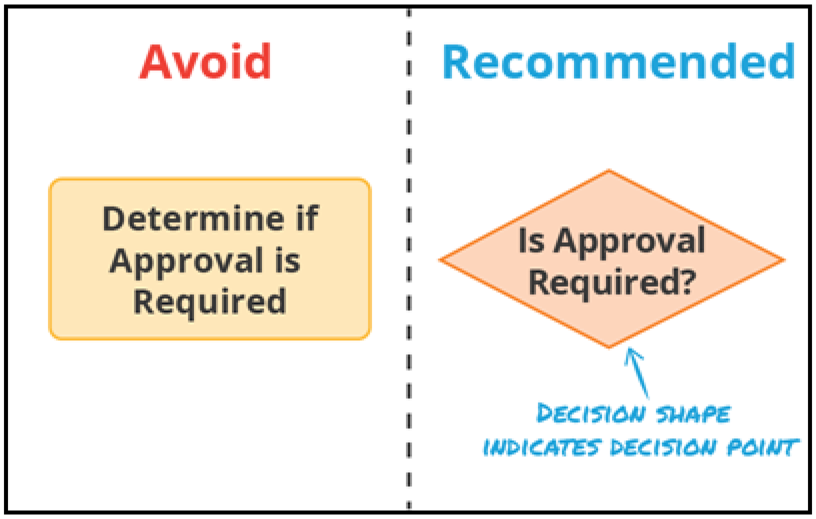

Avoid
Recommended
Notify Employee
Does approval and rejection content differ?
Notify Employee of Approval
Mention next step in process
List reason for rejection
Notify Employee of Rejection

Use the right shape for the task

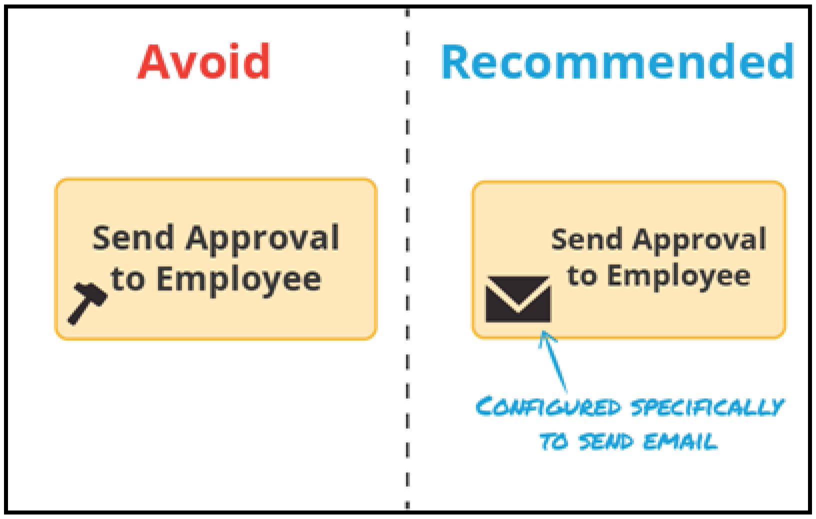
Each type of flow shape represents a specific type of event. For example:

- An assignment represents a task performed by a person. The person performing the selects one of the actions provided to complete the assignment.
- A utility represents an action performed by the application, with a single outcome.
- A decision represents a choice made by the application. Using provided logic, the application determines the appropriate result.

When you need to model an automated decision in a process, use a decision shape rather than a utility shape or an assignment shape.


Avoid
Recommended
Determine if Approval is Required
Is Approval Required?
Decision shape indicates decision point

When designing a process, use smart shapes rather than generic utility shapes whenever possible. Each smart shape represents a specific type of automated action. So if a process must send an email, use the Send Email smart shape.


Avoid
Recommended
Send Approval to Employee
Send Approval to Employee
CONFIGURED SPECIFICALLY TO SEND EMAIL

When you clearly identify the intent of each step in a process, you create a process that is easy for business stakeholders and case workers to understand.

Managing case life cycle exceptions

Introduction to managing case life cycle exceptions

Cases progress from one primary stage to the next as work is completed. However, under certain circumstances, a case may require work to be completed that is not part of the primary flow.

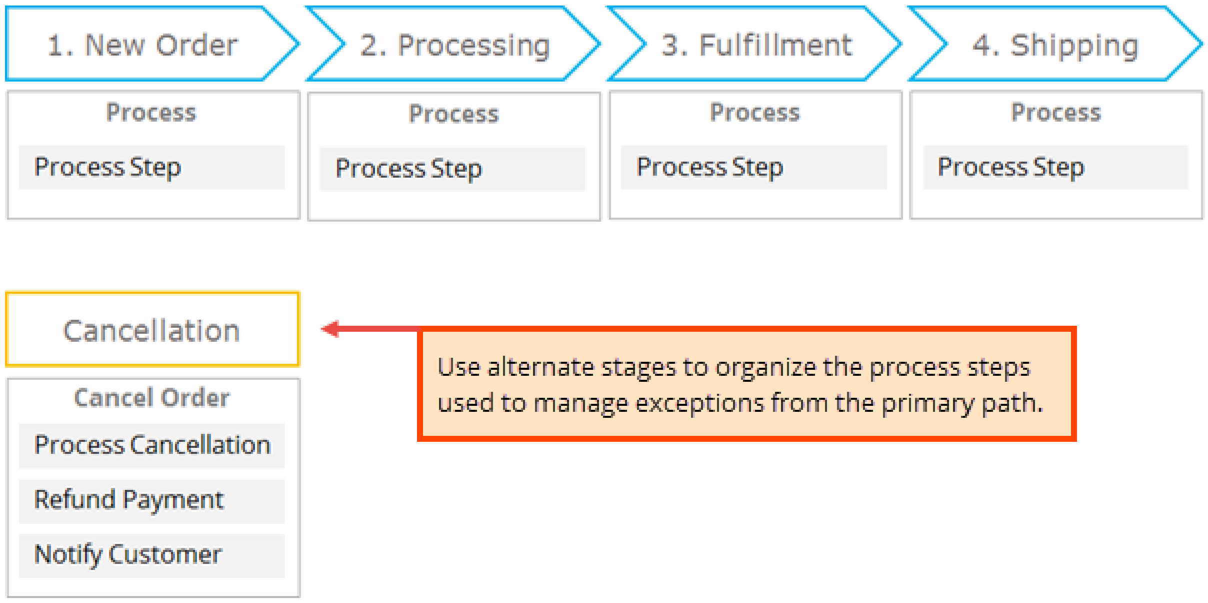
Alternate stages provide a way to model out-of-sequence events in the life cycle of a case. Using alternate stages, you can separate expected behavior from exceptions in life cycle of a case.

After this lesson, you should be able to:

- Explain the purpose of alternate stages
- Add alternate stages to a case type
- Explain the purpose of managing transitions from one stage to another
- Add a transition from one stage to another

Alternate stages

Cases usually progress in order from one primary stage to the next. In some situations, work does not always go according to plan. When that happens, use an alternate stage to describe the actions needed to resolve the situation. **Alternate stages** are used to organize process steps that are not part of the "normal course of events" but must be available under certain circumstances.



For example, when modeling the life cycle of an online ordering application, you must consider that orders can be canceled prior to being shipped. If an order is canceled, a number of tasks must be completed before the order is considered canceled. The first task is to process the order for cancellation, then the payment must be refunded and, finally the customer must be notified that the order was canceled.

Use alternate stages to organize the process steps used to manage exceptions from the primary path.

Guidelines for defining and naming alternate stages

To define alternate stages, consider the following guidelines.

Use names that are most meaningful and relevant to the business users. Use a noun, or noun phrase, to describe the context of the alternate stage. As much as possible, try to limit the stage name to no more than two words.

Consider limiting the number of alternate stages in any given case type to between three and five stages. If you find yourself needing more than five stages, consider combining one or more alternate stages, or using a separate case type.

Adding alternate stages to the case life cycle

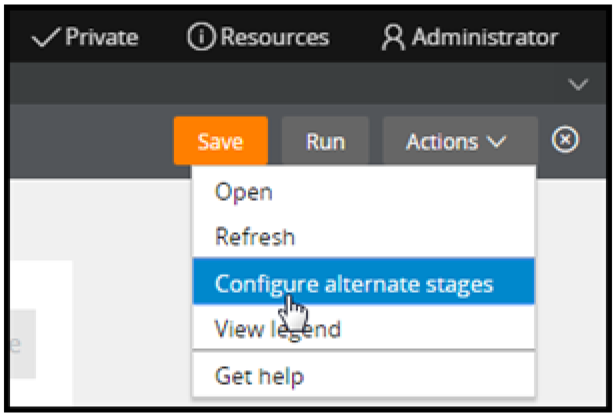
You use alternate stages to model out-of-sequence events in the life cycle of a case. By using alternate stages, you can separate exceptions from expected behavior in life cycle of a case.

Important: Although alternate stages are displayed in a sequence, alternate stages are not ordered. You must model specific behavior in each alternate stage, using a process, that defines how to transition out of the alternate stage.

Configuring a case type with alternate stages

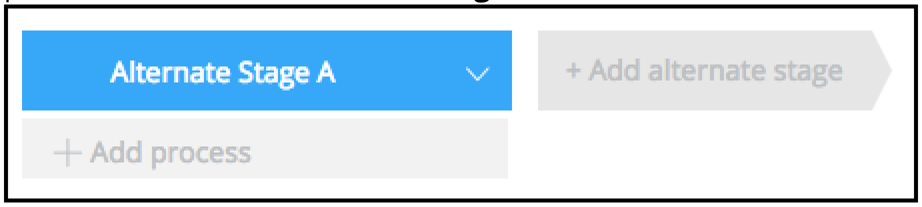
To configure a case life cycle with alternate stages:

1. In Designer Studio, open a case type.
2. In the upper right corner of Designer Studio, on the **Actions** menu, click **Configure alternate stages** to display alternate stages below the primary stages.



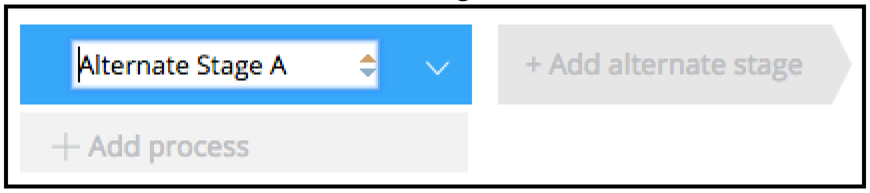
Editing alternate stage labels

When you configure alternate stages for the first time, a single alternate stage is added with a placeholder labeled **Alternate Stage A**.



To edit the text label of an alternate stage:

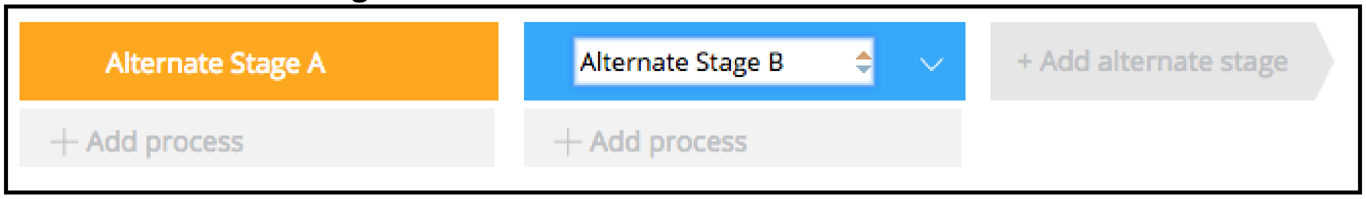
1. Click the label in the alternate stage, then enter a new name.



Adding additional alternate stages

To add additional alternate stages to a case life cycle:

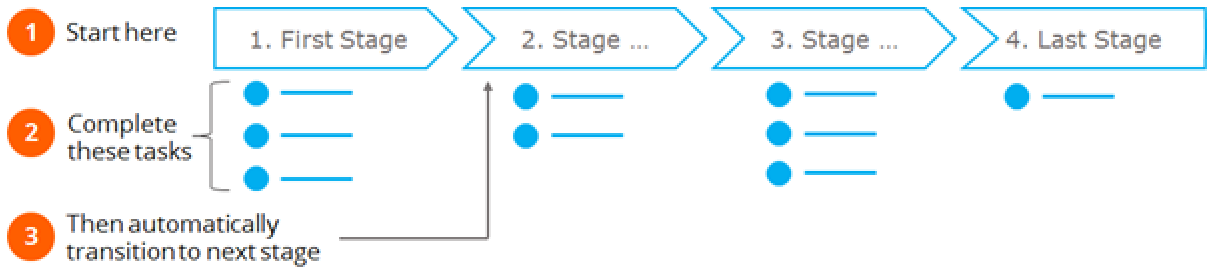
1. Click **+ Add alternate stage**.



2. In the text box, enter a unique name for the alternate stage.
3. Click **Save**.

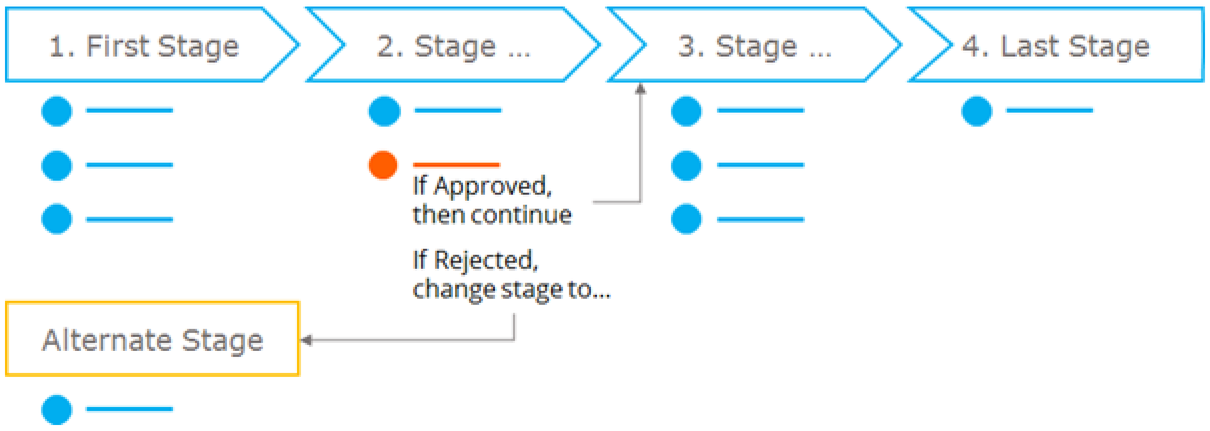
Stage transitions

Stage transitions allow you to further refine the run-time order of stages.

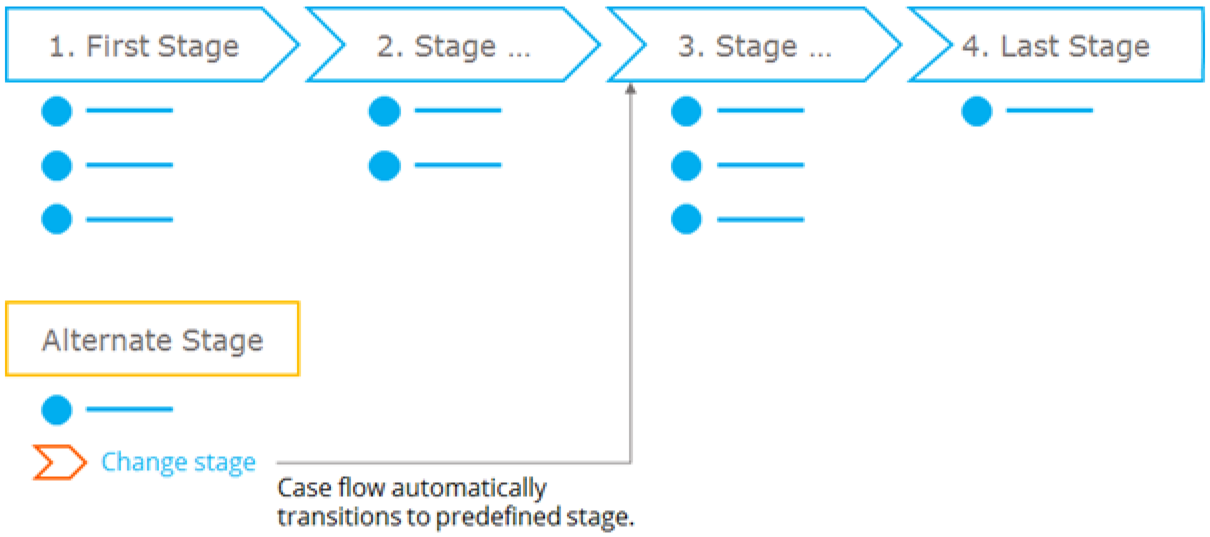


For primary stages, when all steps in a stage are completed, the default option is an automatic transition to the next primary stage.

To allow transitions to other stages before the completion of the current stage, you can add a controlled transition to a stage. Controlled transitions can be configured for any primary or alternate stage, and can occur either as an action in a step or as a specific step in a process.



You can configure a step to allow a user to select the stage to which the case transitions. This type of configuration is most useful for steps that require a Yes/No decision. For example, a case worker must review a request and can either approve or reject the request. If the request is approved, normal processing continues, and the case advances to the next step or stage in the primary path. If the request is rejected, the case advances to a predefined stage, which may or may not be in the primary path.



You can use a **Change Stage** process step to automatically transition the case flow to a specified stage. This type of configuration is most useful for automating transitions to and from alternate stages. For example, a rejected request is sent back to the originator to be updated. The process steps for updating the request are organized in an alternate stage. When the Change Stage step is encountered, the case flow automatically transitions to the stage defined in the Change Stage step.

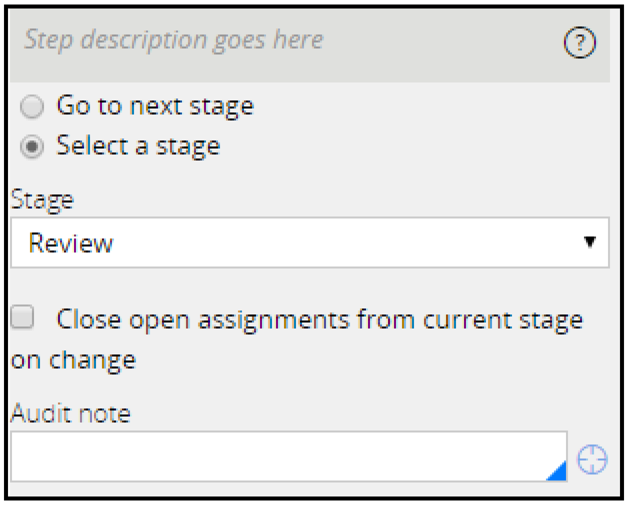
Controlling stage transitions

To allow transitions to alternate stages, or before the completion of a stage, you can add a controlled transition to a stage. Controlled transitions can be added as a process step using the Change Stage smart shape, or as an optional user action in a process step.

Use the Change Stage smart shape to control the stage to which a case transitions

Follow these steps to add a Change Stage smart shape.

1. In the process where you want to add the Change Stage smart shape, click **+ Add step**.
2. In the palette that is displayed, click **More**.
3. Click **Utilities** to display a list of available smart shapes.
4. Click **Change Stage**, and then click **Select** to add the smart shape to the process.
5. In the contextual property panel, select the **Select a stage** option.



Step description goes here

○ Go to next stage
● Select a stage

Stage
Review ▾

☐ Close open assignments from current stage on change

Audit note

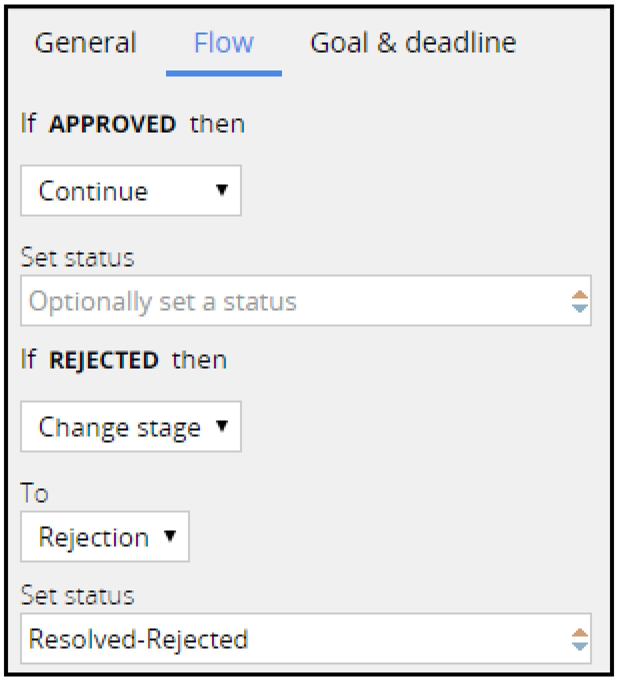
6. In the **Stage** drop-down list, indicate which stage the case transitions to when the Change Stage shape is executed.

Use the Approve/Reject step to control the stage to which a case transitions

Follow these steps to allow a user to select the stage to which a case transitions.

1. In the process where you want to add the Approve/Reject step, click **+ Add step**.
2. In the palette that is displayed, select the **Approve/Reject** step.
3. On the Flow tab of the contextual property panel, set the option for **If APPROVED then** or **If**

REJECTED then to **Change stage**, and then select the stage to which the case transitions when the Approve/Reject step is executed.



If **APPROVED** then

Continue

Set status

Optionally set a status

If **REJECTED** then

Change stage

To

Rejection

Set status

Resolved-Rejected

Adding optional business process events

Introduction to adding optional business process events

As a user works on a case, situations arise that may require the user to stray from the primary path to perform a task. These tasks are called user actions. In this lesson, you learn about the different types of user actions and how to configure those actions in a case.

Objectives

At the end of this lesson, you should be able to:

- Explain the role of user actions in a case
- Differentiate between local actions and optional processes
- Add user actions to a case

User actions

User actions supplement the tasks users can do as they work on a case. User actions allow users to leave the primary path of a case to complete another process. The key is that the user makes the determination to execute the user action; the user action is not automated.

For example, a customer gives you a new cell phone number while you are processing the car details of an Auto Loan. You can use the Update Contact Info user action to update the number. By using the user action, you do not need to move the Auto Loan case to an earlier step or stage.

When determining if you need to use a user action, consider the following questions:

- Should the user be allowed to update the information at any time during a case or stage?
- Do you need multiple steps to update the information?
- Can the information be updated in a single screen?

User actions can be made available for a stage or an entire case. By adding a user action to a case, the action is available to a user while the case is open. By adding a user action in a stage, the action is available only in the stage where the action is defined.

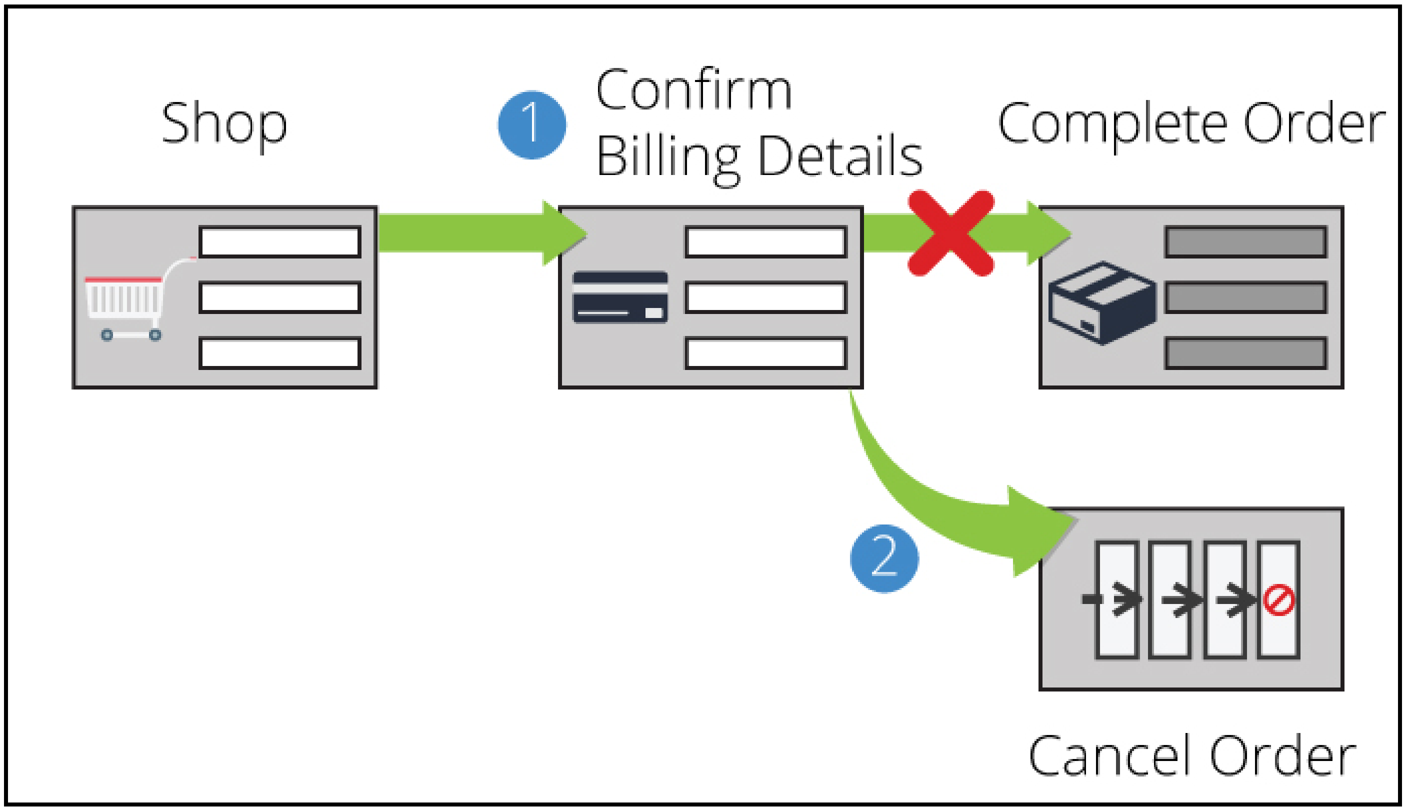
You can add two types of user actions: optional processes and local actions. You use a local action to display a single screen to a user, whereas you use an optional process to launch an entire process.

Optional process

You use an **optional process** when multiple steps are needed to update information. An optional process allows a user to run a new process from within the case. The only difference between an optional process and the other processes in a case is that the user determines when the optional process is executed.

An optional process allows a user to perform a series of tasks outside of the primary path of a case. After completion of the optional process, a user may or may not return to the primary path.

For example, in a commerce application you could have a *Cancel Order* optional process that allows the user to cancel an order as long as it has not been shipped.

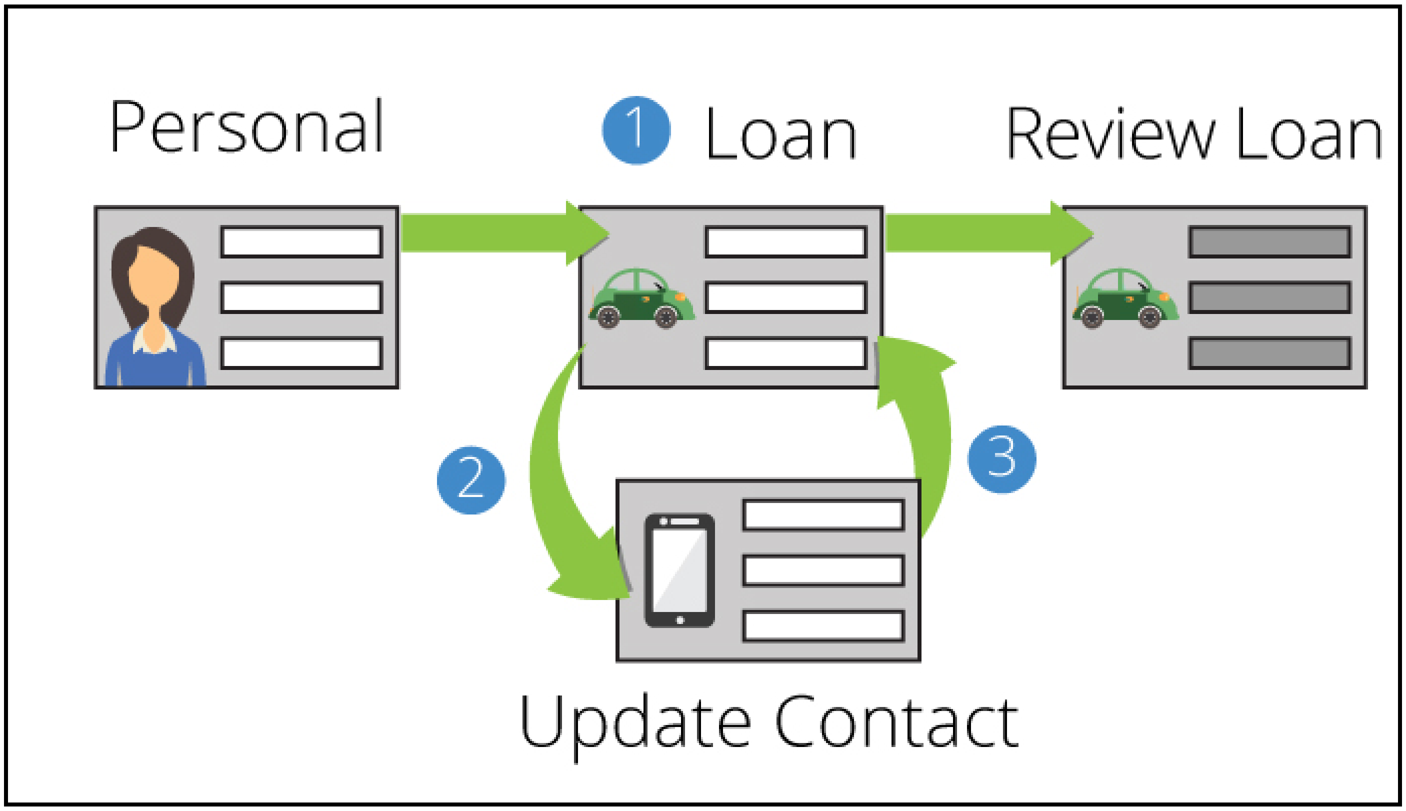


1. After shopping, a user starts the check-out process. While confirming the billing details, a user might decide to cancel the order. The user could launch a *Cancel Order* optional process.
2. The *Cancel Order* process executes. The configuration of the process determines if the case completes or it is transitioned back to continue the original flow.

Local actions

You configure a **local action** when the case information can be updated in a single screen. A local action allows the user to make a change but not interrupt the processing of the case. Think of a local action as a screen that is accessible to the user. A local action allows the user to perform a single task outside of the primary path of a case. After completion of the local action, the user returns to the primary path of the case.

The Update Contact Info example illustrates using a local action. You want to give the users the ability to change a customer's contact information at any time in the case, but you do not want the users to lose where they are in processing the case.



1. A user enters the personal information for a customer. While entering in the loan information, the customer asks to update a cell phone number. The user launches the **Update Contact Information** local action.
2. The user completes **Update Contact Information**.
3. After completing the local action, the user sees the loan information screen.

KNOWLEDGE CHECK

ANSWER What are the main differences between a local action and an optional process?

Local actions are single tasks and return to the primary path of a case, whereas an optional process is a series of steps that is not required to return to the primary path of a case.

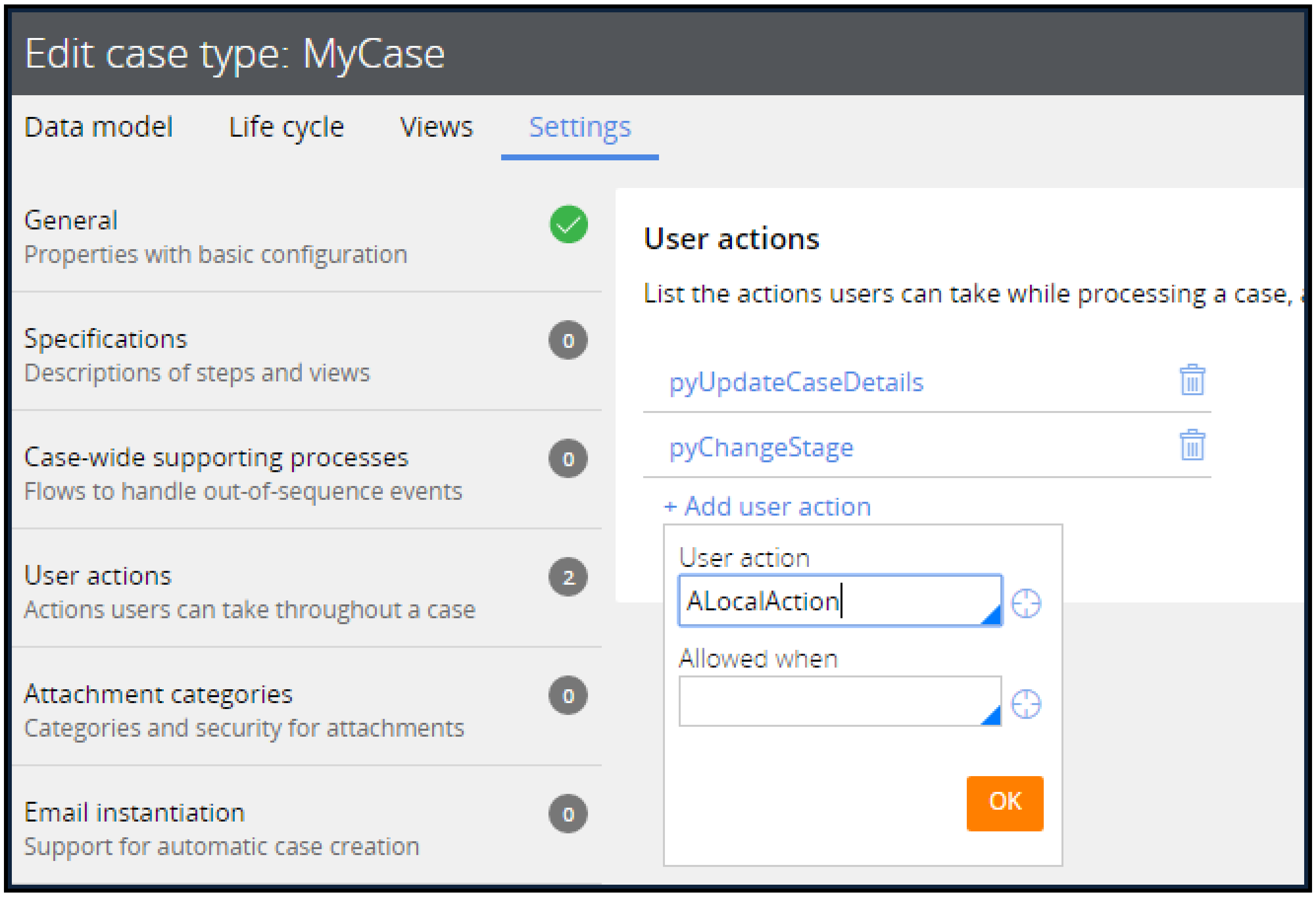
Adding user actions to the case life cycle

Adding a user action to a case

You can define user actions that are available to a user throughout the entire case. These user actions can be optional processes or local actions.

Follow these steps to add a local action or optional process to a case:

1. In the Case Designer, select a case type.
2. Click the **Settings** tab.
3. Select either **User Actions** to add a local action or **Case-wide supporting processes** to add an optional process.
4. Select the local action or the process to add to the case.


Edit case type: MyCase

General
Properties with basic configuration

Specifications
Descriptions of steps and views
0

Case-wide supporting processes
Flows to handle out-of-sequence events
0

User actions
Actions users can take throughout a case
2

Attachment categories
Categories and security for attachments
0

Email instantiation
Support for automatic case creation
0

User actions

List the actions users can take while processing a case,

pyUpdateCaseDetails

pyChangeStage

User action: ALocalAction

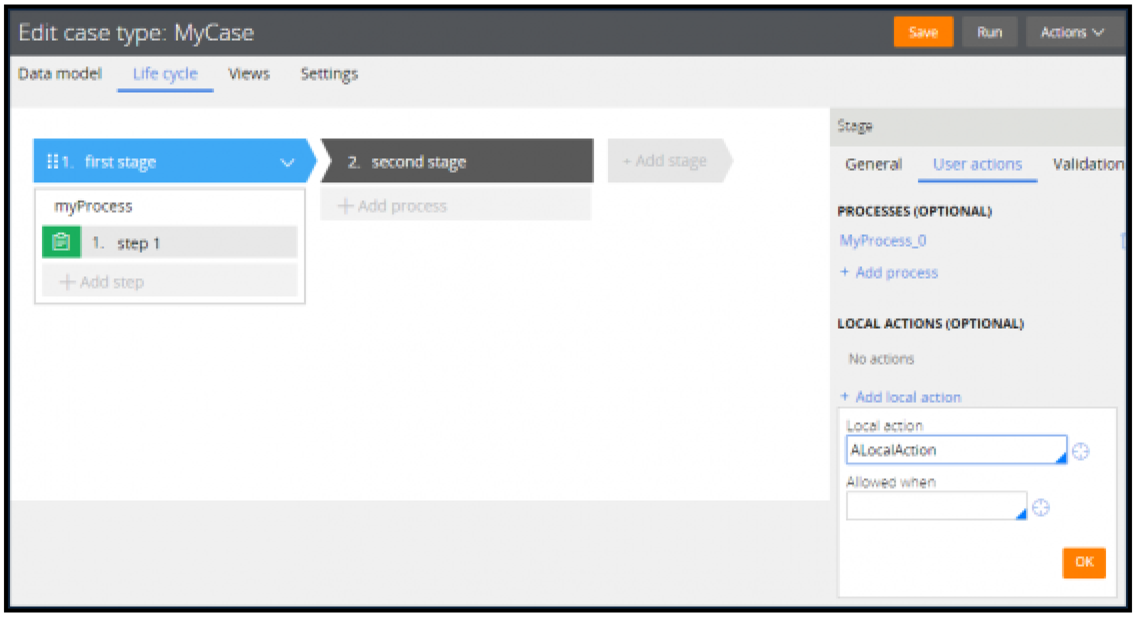
Allowed when:

Adding a user action to a stage

Stages can define both local actions and optional processes.

Follow these steps to add a local action or an optional process:

1. In the Case Designer, select a case.
2. Select a stage.
3. Click **User Actions**.
4. In the Processes (Optional) list, Select the optional process or local action to add to the stage.
5. Optionally, add an **Allowed when** condition to modify when the optional process or local action should be displayed.



Sending correspondence

Introduction to sending correspondence

Pega allows you to automate and create timely and clear communication with participants in a case. As a result, the right person receives the right information at the right time.

Objectives

At the end of this lesson, you should be able to:

- Explain how correspondence improves a process
- Add correspondence to a case type
- Send a correspondence while processing a case

Automating case communications

Common reasons for communicating with users

Organizations depend on timely communication to establish a shared understanding of transactions or assignments.

For example, consider a requirement for an auto claims application in which customers must be notified when their claims are successfully filed, or anytime the status of the claims changes.

Another common notification requirement is keeping case workers up-to-date. For example, you must notify case workers when they have a new claim to process. Also, you may want to notify them on the progress of the previous claim.

Finally, you may have a requirement to communicate with someone who is indirectly involved in the case, such as an external agency.

To achieve effective communication, answer three simple questions. First, who is the user that receives the communication? Second, how will the communication be sent? Third, when does the communication need to be sent?

Identifying users to communicate with

When sending a correspondence first determine 'who do I need to communicate with?' You can send a correspondence to a specific address, but what if that address is no longer valid? You would have to update the application anytime that address changed. To avoid this Pega uses a set of roles to use with correspondences.

Pega defines the following roles for a correspondence:

Role	Description
Owner	The person who created the case.
Customer	The person on whose behalf the case is transacted. This person may not process the case, but may want – or need – notification of any changes.
Interested	A person who tracks the progress of a case but does not process the case.

You are not limited to specifying a single role for a correspondence. For example, you may want to send a correspondence to the customer and all interested parties. To do this Pega uses something called a party. A party identifies the recipient of the communication and may contain one or more of these roles.

Identifying how to communicate with users

To generate correspondence, you need to know “how” you want to communicate with the recipient. In other words, what's the right channel a user should receive a correspondence.

Pega provides four correspondence types to communicate with users: email, text message, fax and regular mail. Each correspondence type provides unique functionality but share the same basic template.

Pega provides a rich text editor to create formatted correspondence. You create one or more correspondence templates for each type of correspondence.

Identifying when to communicate with users

The last question you need to answer is “when” do you communicate. Pega simplifies sending a correspondence by allowing you to simply add a step to your case. Then you just configure who to send it to and the content of the message.

Sending an email from a case

A common use case for sending a correspondence during a case is sending a confirmation email after the user has completed a series of steps. You probably experienced this many times while completing a purchase online.

You accomplish this in Pega by adding a Send Email step to your case and then configuring the step.

Adding a Send Email step

To add a Send Email step:

1. Click **Add Step**.
2. Click **More > Utilities**.
3. Select **Send Email**.
4. Click **Select**.

Configure the Send Email step

Email configuration has two parts: **Send to** and Message. To configure the Send Email Step:

1. Complete the **Send to** by specifying an email address or a party.
2. Enter a subject for the email.
3. Complete the body of the Message by specifying either a Correspondence template or using the rich text editor to create a message.
4. Click **Save**.

Guiding users through a business process

Introduction to guiding users through a business process

In this lesson, you learn how to add work statuses and instructions to a case. Adding work statuses and instructions keeps the user informed about a case. By using these options, you can help users complete their work more productively.

Objectives

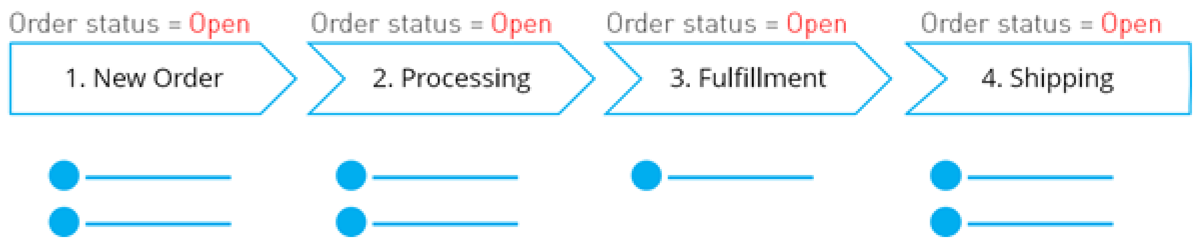
At the end of this lesson, you should be able to:

- Explain how work status adds context to a case
- Update the work status for a case
- Explain how instructions add context to a case
- Add instructions to assignments

Updating the case status

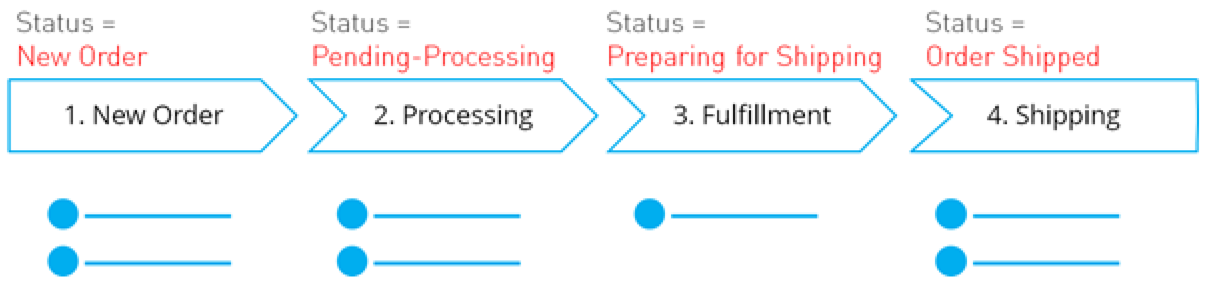
Case status

Consider what would happen if you placed an online order, and the status of that order could only be identified as *Open*.



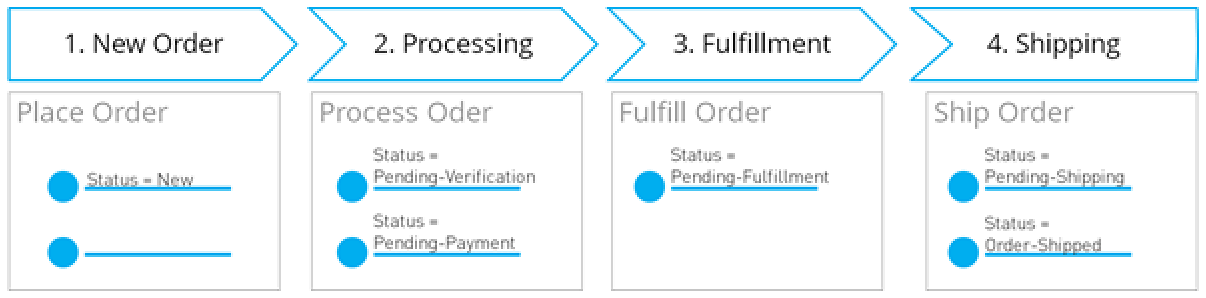
You would have difficulty determining the status of your online order.

Every case, whether it is an online order, a loan origination request, or an insurance claim, has a status. The **case status** is the primary indicator of the progress of a case towards resolution.



The case status is updated as the case moves through the case life cycle.

For example, a case status of *New* is assigned to each case when the case is created. As the case progresses through the case life cycle, the status of the case is updated at each step.



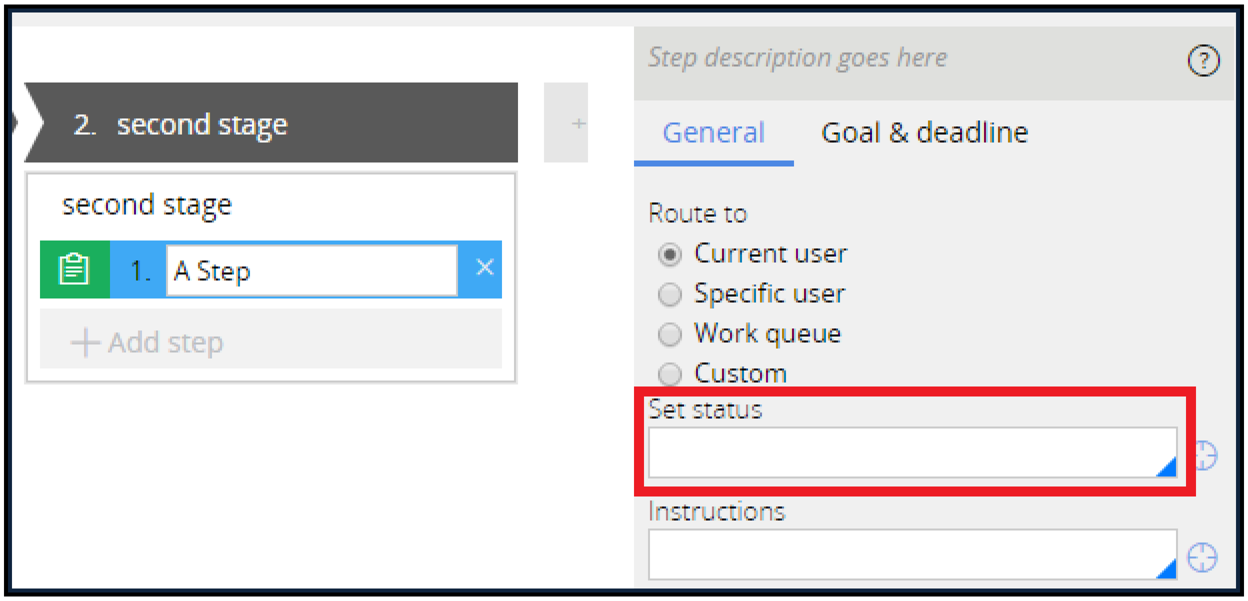
You can set the case status on each step in the case life cycle. When the case advances to a step, Pega automatically updates the status of the case to the value defined for that step.

Pega includes standard case status values, such as Open, Pending-Approval, and Resolved-Completed. You can also add custom status values.

Updating the status of a case

To update the status of a case:

1. In a case type, select a step where you want to change the status.
2. In the General settings, update the **Set Status** field.



Route to
- [✓ Current user]
- Specific user
- Work queue
- Custom

Set status

Instructions

Adding Instructions

Imagine you are a loan processing agent. Your company offers car loans, home mortgages, and personal loans. When it is time for you to work on your next case, you could see many cases with arbitrary IDs like B-5, I-23, N-33, G-57, or O-62. Which case do you choose? These case IDs by themselves give you no help in determining what work you need to perform in those cases. Some could take 5 minutes, others 1 hour. Pega solves this problem by allowing an architect to setup an instruction for each step in a process.

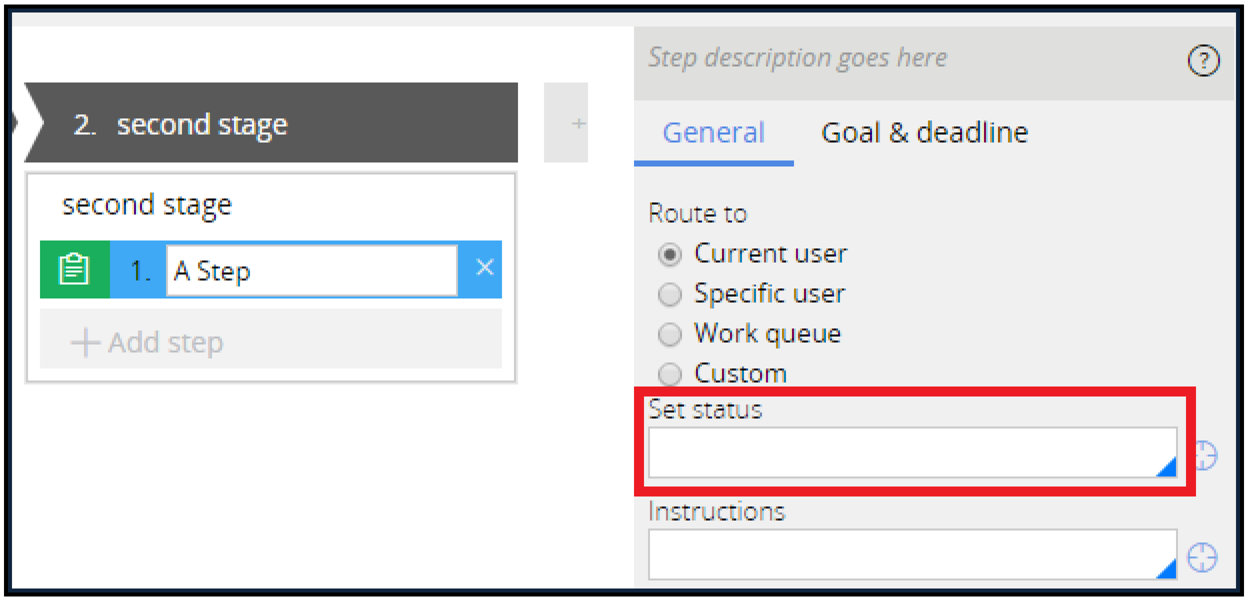
An instruction for a step identifies to an end user what should be accomplished in an assignment. End users see instructions in their worklist or when they open a case. For example, in a loan application there is a step for a manager to approve a loan. You would want to add instruction called Approve this item so it's clear what the work needs to be done on the case.

Adding an Instruction to a step

System architects create instructions based on the requirements defined by the business. Once created you add them to a step in a process.

To add an instruction to a step:

1. In a case type, select a step.
2. In the General settings, update the Instructions field.



Route to

- [✓ Current user]
- Specific user
- Work queue
- Custom

Set status

Instructions

Modeling complex process flows

Introduction to designing complex process flows

You use Pega Express and the Case Designer to define and configure processes used in a business transaction. You can then use the Process Modeler to add advanced features to the processes.

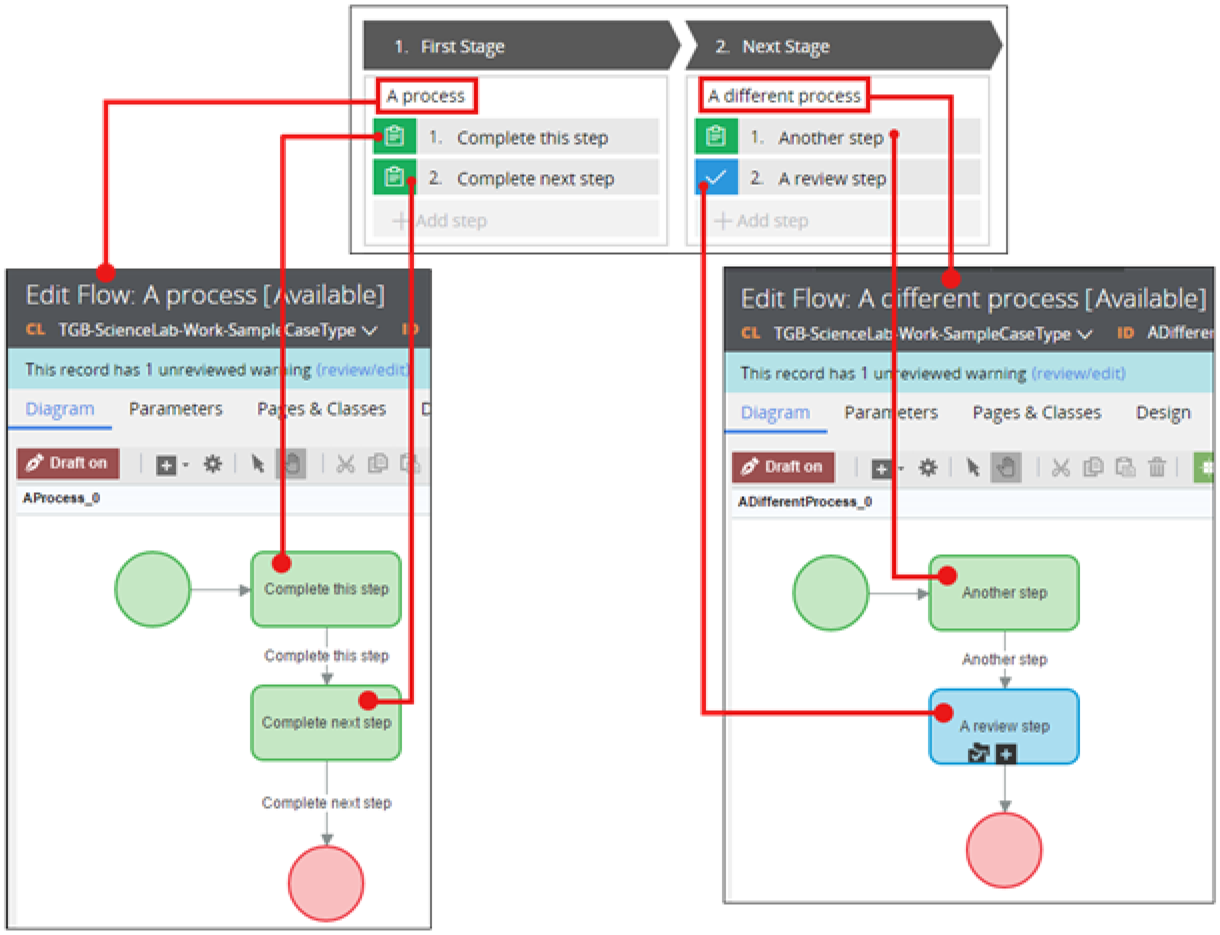
After this lesson, you should be able to:

- Explain how flow rules relate to processes in the case life cycle
- Choose the correct flow shape to accurately model a complex business process
- Design an intent-driven process
- Use the Process Modeler to add additional flow shapes and connectors to a flow
- Model a complex business process

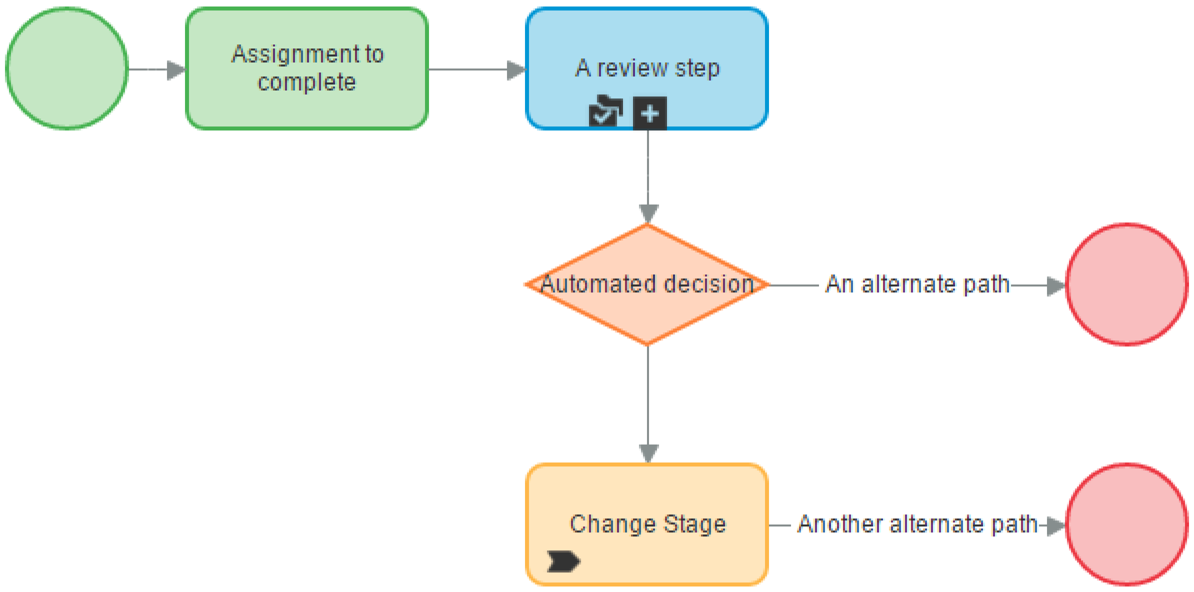
Flow rules

When you add a process to a case life cycle in Case Designer, Pega automatically creates a flow rule. A **flow rule** provides a visual method for modeling a process in your application using shapes and connectors to define a sequence of events.

Each process step in the case life cycle is represented by a flow shape. A **flow shape** represents a task that is accomplished as part of a business process. Flow shapes are differentiated by color, symbol, and name.



Use Case Designer to add standard processes used to define the case life cycle. You can then use the Process Modeler to add advanced features to the processes such as data-driven decisions, or parallel or iterative processing.



Each flow shape represents a specific processing action that you can configure to perform a specific action, such as an assignment an end user must complete, automated decisions used to determine the path a case takes, or other automated actions such as transitioning to a stage.

KNOWLEDGE CHECK

Flow rules represent a __________ in the case life cycle.

process

KNOWLEDGE CHECK

Flow shapes represent __________.

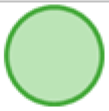
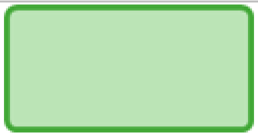
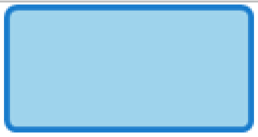
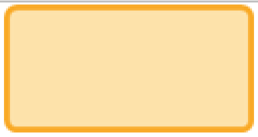
tasks to be completed as part of a business process

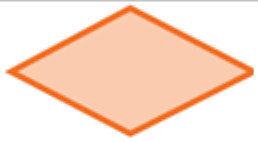
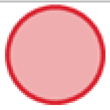
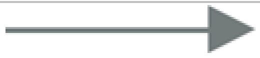
Flow shapes

You add shapes to a flow rule using the Process Modeler. You use the shapes to define a sequence of events in a flow that accurately models a business process in your application.

Standard shapes in a flow rule

Pega provides standard shapes that enable you to accurately model a business process using a flow rule.

Shape	Name	Use
	Start	The **Start** shape indicates the beginning of flow processing. Each flow rule must contain a single Start shape. A single Start shape is automatically added to every flow rule.
	Assignment	The **Assignment** shape creates a task in a work list or workbasket so a user can provide input to the case. Typically, the user either provides information or selects an outcome.
	Subprocess	The **Subprocess** shape indicates a reference to another flow rule from the current flow rule. Portions of a process can be divided into a smaller process to enable reuse in other flow rules.
	Utility	The **Utility** shape indicates an automated system action. Pega executes automated system actions, without requiring human intervention. Examples of automated system actions include changing the stage of the current case, sending an email, or creating a new case.

Shape	Name	Use
	Decision	The **Decision** shape indicates an automated step used to determine the path a case takes. A Decision shape evaluates an expression or calls a decision rule to determine which step is next in the flow progression.
	End	The **End** shape indicates the end of flow processing. Each flow rule may include one or more End shapes to represent the potential end points of the process.
	Connector	Each flow shape connects to other flow shapes through the use of a connector. The **Connector** is used to define the sequence of flow execution. The flow execution begins with the Start shape and proceeds from one shape to the next in the order the shapes are connected to each other.

KNOWLEDGE CHECK

ANSWER Which standard flow shape creates a task in a work list or workbasket?

Assignment

KNOWLEDGE CHECK

ANSWER Which standard flow shape represents an automated system action?

Utility

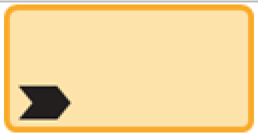
KNOWLEDGE CHECK

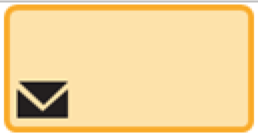
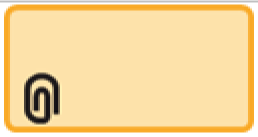
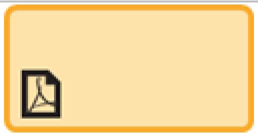
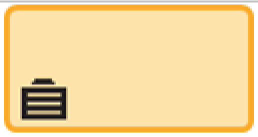
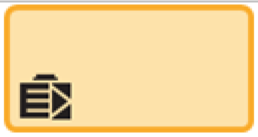
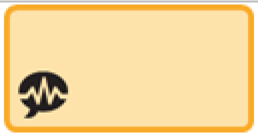
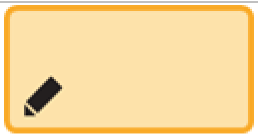
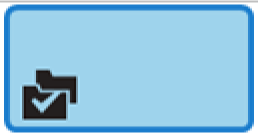
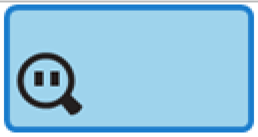
ANSWER How many Start shapes can a flow rule contain?

One

Smart Shapes

Pega provides smart shapes to help speed up development. **Smart shapes** are preconfigured shapes that perform a specific task, such as sending an email, attaching a file to a case, or changing to a different case stage.

Shape	Name	Use
	Change Stage	Transitions the case to a different stage in the case life cycle

Shape	Name	Use
	Send Email	Sends an email to one or more work parties
	Attach Content	Attaches a file, URL, or note to a case
	Create PDF	Creates a PDF file from a specified section and attaches it to the case
	Create Case (s)	Creates a top-level case or one or more child cases
	Persist Case	Converts a temporary case to a permanent object in the database
	Post to Pulse	Creates a message that is sent to the Pulse social stream
	Update a case	Updates the case or all child cases and descendants
	Approval	Routes a case to one or more reviewers, based on a user name, reporting structure, or authority matrix
	Duplicate Search	Returns a list of cases that match the search criteria that are defined in the case type

KNOWLEDGE CHECK

ANSWER

Smart shapes are ________________.

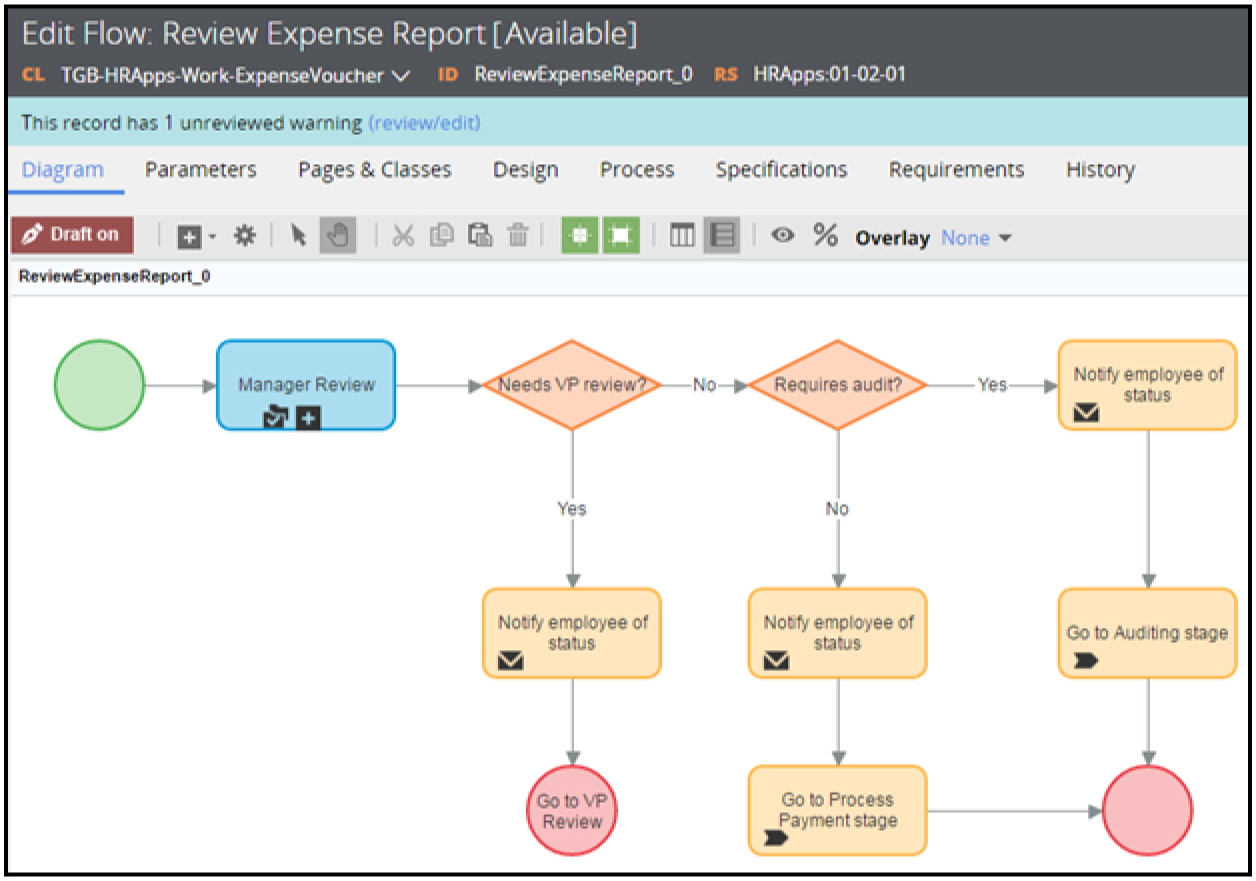
predefined shapes configured to perform a specific task

Adding shapes to a flow rule

You use the Process Modeler to add shapes and connectors to the diagram to indicate the sequence of events in the process flow. When Pega executes the process flow, processing begins with the Start shape and follows the connectors from shape to shape until reaching an End shape. If a shape has one or more connectors, the process branches based on either user selection or the result of an automated decision.

The Diagram tab of a flow rule displays the process in graphical form.

The following example shows a complex process where the sequence of events is determined by one or more decisions.


Edit Flow: Review Expense Report [Available]

CL TGB-HRApps-Work-ExpenseVoucher ID ReviewExpenseReport_0 RS HRApps:01-02-01

This record has 1 unreviewed warning (review/edit)

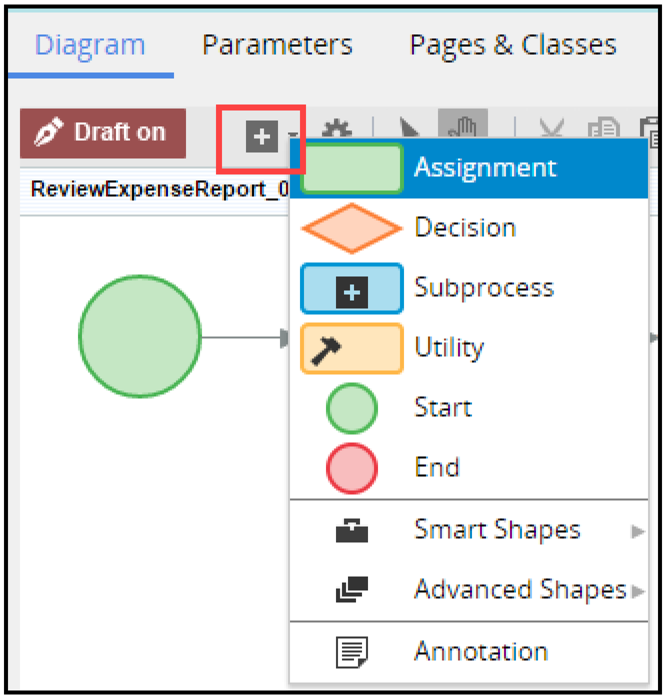
Each unique decision — *Needs VP review?* and *Requires audit?* — is represented by a single shape with connectors indicating multiple paths for the decisions and associated steps.

Add a shape to a process flow

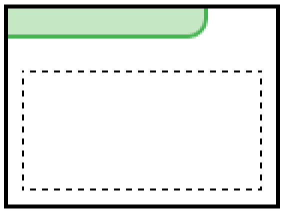
To add a shape to a flow rule, open the flow rule, and then add a shape using one of the following methods.

Add a shape using the Flow Shapes menu

1. On the **Diagram** tab of the flow rule, click the **Flow Shapes** menu.

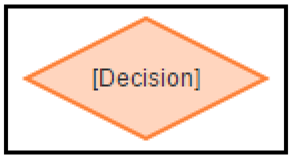


2. Click and drag a shape onto the diagram. A dashed rectangle follows the mouse cursor to indicate where the flow shape is added.



3. Release the mouse button.

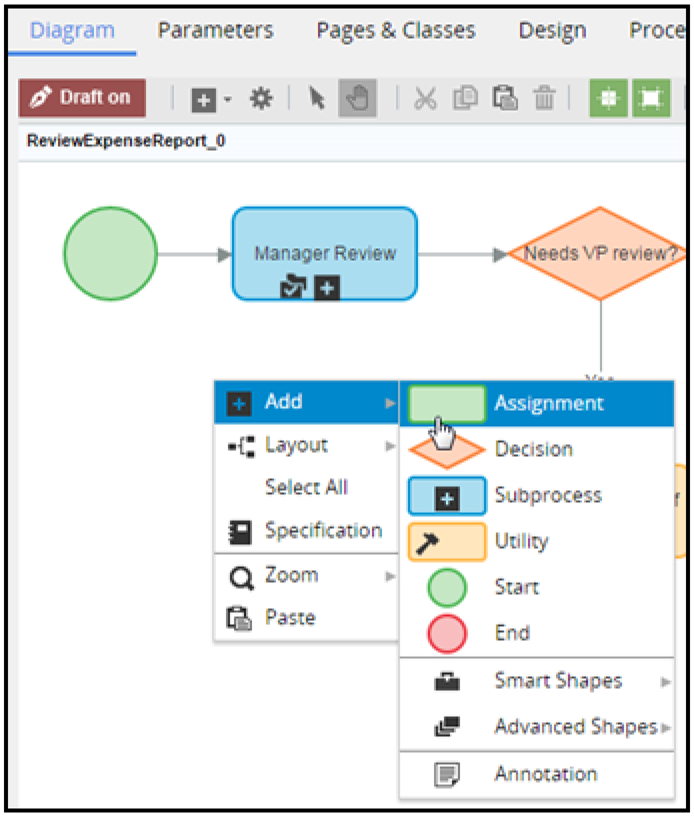
   The flow shape is added to the process, with a generic name that identifies the purpose of the flow shape.



Note: To quickly add a shape to the diagram, click the shape once. After the shape is added to the diagram, drag the shape to the desired position on the diagram.

Add a shape using the sub-context menu

1. On the **Diagram** tab of the flow rule, right-click the flow diagram where you want to add the shape.

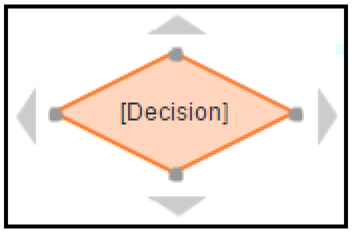


2. Click the appropriate shape to add it to the diagram.

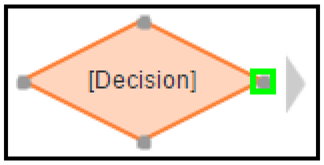
Connect flow shapes with a connector

Connectors indicate the order of steps in a process flow.

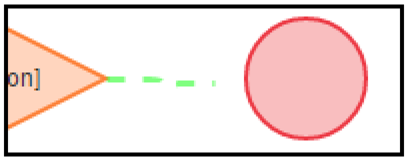
1. Position the cursor over the flow shape from which you want to connect. A set of connector points is displayed on the border of the flow shape.



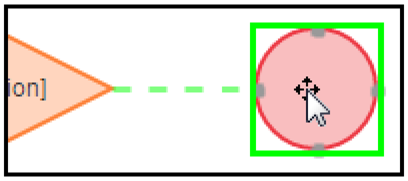
2. Click the connector point from which you want to start the connector. The connector point is highlighted with a green square.



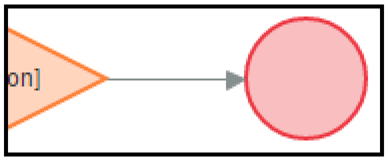
3. Click the mouse button and drag the cursor to draw the connector. A green dashed line indicates the path of the connector.



4. Position the cursor over the shape on which you want to end the connector. The connector points are displayed on the border of the flow shape.



5. Release the mouse button to connect the connector to the shape.



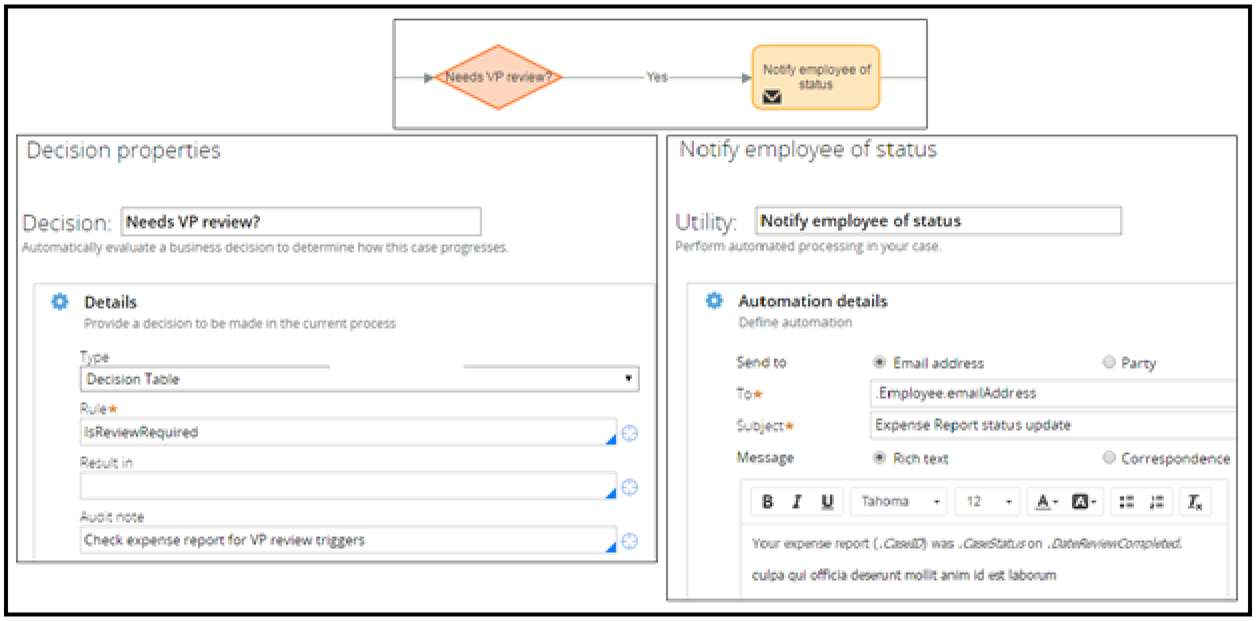
Configuring flow shapes and connectors

To configure the behavior of a flow shape or connector, open the Properties dialog for the shape.

In the Properties dialog you can associate a flow shape or connector with a specific rule and add additional processing instructions. For example, you can add a likelihood to a connector to identify the probability of the process following the connector during case processing or add an Audit note.

To configure a flow shape or connector using the Properties dialog:

1. Double-click the flow shape or connector to open the Properties dialog .



Decision properties

Decision: Needs VP review?

Automatically evaluate a business decision to determine how this case progresses.

Details

Provide a decision to be made in the current process

Type	Decision Table
Rule*	IsReviewRequired
Result in	
Audit note	Check expense report for VP review triggers

Notify employee of status

Utility: Notify employee of status

Perform automated processing in your case.

Automation details

Define automation

Send to	● Email address	○ Party
To*	.Employee.emailAddress	
Subject*	Expense Report status update	
Message	● Rich text	○ Correspondence

Your expense report (*.CaseID*) was *.CaseStatus* on *.DateReviewCompleted.*

culpa qui officia deserunt mollit anim id est laborum

2. Complete the Properties dialog.

Note: Fields marked with an asterisk (*) are required fields.

3. Click **Submit** to close the Properties dialog.
4. Click **Save** to commit your configuration changes.

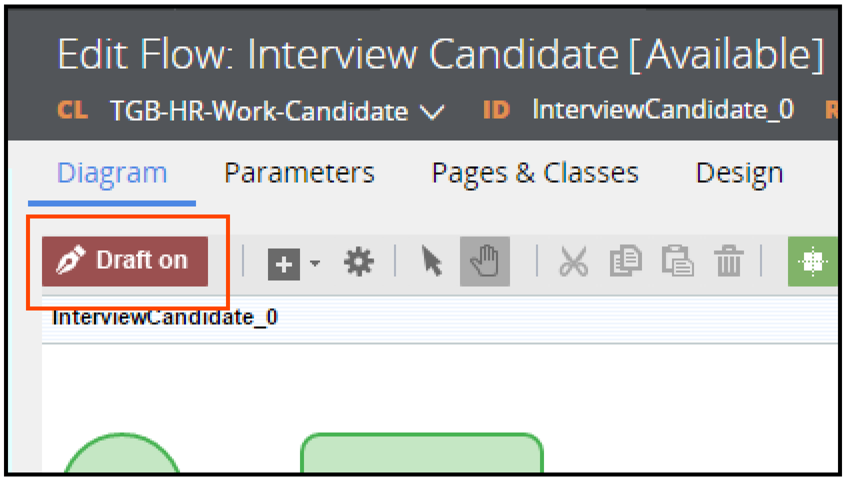
Draft mode

When you create a flow rule, Pega defaults the flow rule to draft mode.

Draft mode enables you to add flow shapes and connectors that reference other rules, even when those other rules do not yet exist. Using draft mode, you can run a process even if the rules that would otherwise be required, such as user interfaces and automated decisions, are missing.

Note: Flows with draft mode enabled will not run in a production environment and will display a Guardrail warning. To resolve the warning, turn off draft mode before releasing the application.

To disable draft mode for a flow rule, click **Draft on**on the toolbar. When draft mode is disabled, the label of the button updates to Draft off.



KNOWLEDGE CHECK

Draft mode enables you to _______________.

Save, and run a flow rule even if other referenced rules do not yet exist.

REPORT PLANNING AND DESIGN

Process visibility through business reporting

Introduction to process visibility through business reporting

You need ways of understanding how business processes are functioning — where the bottlenecks are, where there are opportunities to improve response time, and what emerging trends need attention. A report that asks the correct questions, and therefore provides relevant information rather than an unsorted heap of data, can show you what's going on now, what has been going on over a period of time, or how what is going on matches or differs from what was planned.

After this lesson, you should be able to:

- Explain the difference between business data and process data used in reports
- Describe the different types of reports available for Pega applications
- Use the Report Browser to customize an existing report

Business reports

Business applications often target performance gains in time spent and process efficiency as a method of reducing the cost of performing work. But poor work quality may indicate a poorly designed application, rather than poor effort from end users. You need ways of understanding how complex processes are functioning — where the bottlenecks are, where there are opportunities to improve response time, and what emerging trends need attention.

A business report that asks the correct questions, and therefore provides us with relevant information rather than an unsorted heap of data, can show us what's going on now, what has been going on over a period of time, or how what is going on matches or differs from what was planned.

Business reports and process reports

There are two types of metrics associated with report data, business metrics and process metrics.

Business metrics represent the data you define for an application. For example, business metrics are the number of orders for a certain item, or how many orders of a certain type get canceled.

Process metrics are defined and tracked by Pega. For example, process metrics include how long it takes to complete an assignment, how often a path is followed in a flow, or how often Service Level Agreements (or SLAs) are violated.

Business reports

Organizations can design business reports that describe and measure what the organization's work is. Organizations can use these business metrics to make informed decisions about improving its business performance. This data that provides the metrics is collected from outside the application and is stored in a database. The system retrieves the information when users generate a report.

The following table gives examples of business report information and how the information can be used in business decisions.

What is being measured?	What is the business decision?
What was the average profit margin for all automobile sales last year?	The average margin was below the target percentage. The sales department decides to train its sales staff on promoting cars and options that have the highest margins.
How many auto loans are made in a month as compared to personal loans for the same period?	The number of personal loans is significantly lower than the number of auto loans. The goal is to have the numbers approximately equal. The marketing department increases marketing resources for personal loans.
How many orders for office desks were shipped each week for the past month and how many are now left in inventory?	The number of orders shows an upward trend. As a result, inventory levels are unacceptably low. The purchasing department decides to restock more desks on a weekly basis.

Process reports

Process reports track statistics on how work is performed in Pega applications. Unlike business metrics, process data is automatically defined and generated within the application. Having this information enables business analysts and business managers to discover issues that may affect processing performance.

The following tables gives examples of process report information and how the information can be used in process design decisions.

What is being measured?	What is the process design decision?
Which open loan application cases have exceeded the standard three-day service level deadline?	Most of the cases were for loan amounts greater than $300,000. Loan amounts that exceed this amount must go through an additional review step, which accounted for the delay. The department manager decides to increase the service level deadline for loans exceeding $300,000 from 3 to 4 days.
What is the average duration and by assignment type and action?	This report might help identify which user actions take the longest to complete, and which are used more or less often than expected.

KNOWLEDGE CHECK

ANSWER

A sales manager is required to run weekly performance reports on how long it takes for a car to be prepared for customer delivery after the sale has been signed. Which type of report metrics does this report apply to?

Business metrics.

About the Report Browser

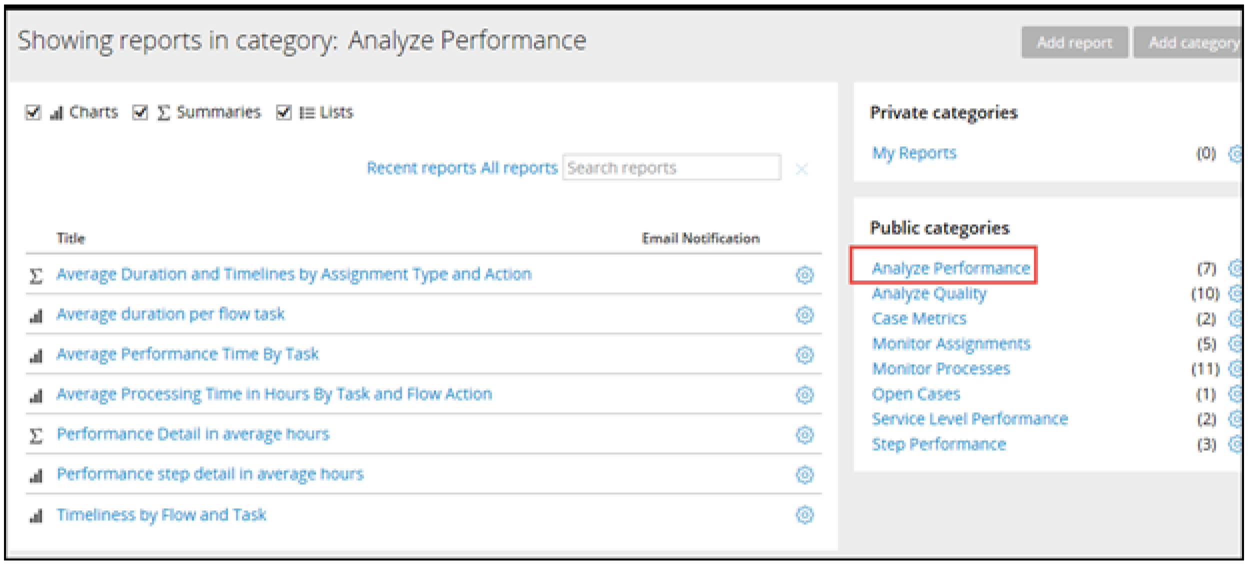
Work managers use the Report Browser to search for, organize, schedule, and run reports.

Use the Report Browser to review the library of available reports. Reports are grouped into public and private categories.

- The public category group include hundreds of standard process reports provided by Pega. Public category reports are available to all managers in an application.
- The private category group includes reports that are created and saved for individual work managers. Managers can share their reports with other managers by putting them into the public categories group.

The Report Browser organizes the report categories into Private categories and Public categories lists. The lists are displayed on the right side of the Report Browser.

When you select a category from a category list, the available reports within the report category are displayed in a list on the left side of the Report Browser. The following screenshot shows the list of standard Analyze Performance reports when the category is selected in the Public categories list.



Title	Email Notification
Average Duration and Timelines by Assignment Type and Action	
Average duration per flow task	
Average Performance Time By Task	
Average Processing Time in Hours By Task and Flow Action	
Performance Detail in average hours	
Performance step detail in average hours	
Timeliness by Flow and Task	

Private categories

My Reports	(0)

Public categories

Analyze Performance	(7)
Analyze Quality	(10)
Case Metrics	(2)
Monitor Assignments	(5)
Monitor Processes	(11)
Open Cases	(1)
Service Level Performance	(2)
Step Performance	(3)

Standard Pega reports

Standard Pega process reports are grouped into eight categories.

Report category	Information the reports provide
Analyze Performance	Resolved cases in an application at the level of each individual step, or actions, within a business process. The reports analyze the completed work to determine whether business processes are efficient and effective. For example, there is a report that tracks processing time in hours by task and action.
Analyze Quality	Resolved cases in an application. Similar to reports that analyze

Report category	Information the reports provide
	performance, quality reports analyze completed work to determine if business processes are efficient and effective. For example, there is a quality report that measures the average elapsed time per status.
Case Metrics	The number of cases created each day for the last seven days and the time per stage for resolved cases.
Monitor Assignments	Assignments for open (or unresolved) cases in an application. The reports tracks the work based on the user to whom the case is assigned. For example, there is a report that measures time lines by task.
Monitor Processes	Assignments for open (or unresolved) cases. The reports focus on the work and not individual users. For example, there is a report that measures throughput in the past week by work type.
Open Cases	Case-level SLA status for open cases. This report focuses on the timeliness of a case from the time a case is created to the time the case is resolved.
Service Level Performance	Assignment SLA status grouped by assignment or by operator.
Step Performance	Assignment-level SLA status grouped by assignment, and an historical view of the time it took an operator to complete a step.

Viewing reports in lists, summary layouts, or charts

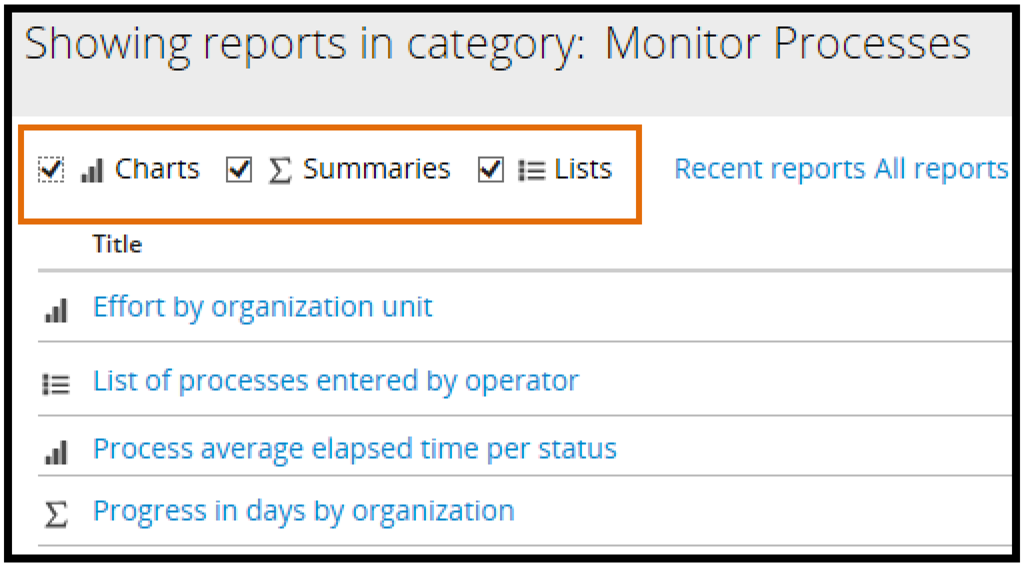
When you select a report in a category, the report opens in the Report Viewer. The data can be displayed in a list, in a summary format, or in a chart.

- List reports display a list of cases whose data is organized into columns and rows.

- Summary reports aggregate case data into categories by use of a summarizing function, such as counting the number of results in a particular column or averaging the values in the column.

  Add charts to summary reports to help end users visualize report data, and allow users to “drill” down into the report data for greater insight.
- The checkboxes at the top of the report list

As shown in the following screenshot, checkboxes at the top of the report list lets you filter the list based on how the reports are presented


Showing reports in category: Monitor Processes

[✓] Charts [✓] Summaries [✓] Lists   Recent reports All reports

Title
Effort by organization unit
List of processes entered by operator
Process average elapsed time per status
Progress in days by organization

Working with the Report Browser

Use the Report Browser to review, run, and edit the library of available reports.

Running reports from the Report Browser

In the Case Manager portal, the Report Browser provides report shortcuts to the reports that are available for you to run. Report shortcuts are organized into categories.

1. Open the Report Browser by clicking Reports in the left navigation pane.
2. To run a report immediately, click the title of the report shortcut. The results display in the Report Viewer.

   Note: The Report Viewer provides options for working with the results, such as formatting, filtering, saving, printing, and exporting the report.

After a report runs, the results display in the Report Viewer. The Report Viewer shows the title of the report, and the date and time when the results were generated. The Report Viewer also provides options for working with the results.

From the Report Viewer, you can complete the following tasks:

- View the results of a report and the filters applied to generate the results.
- Expand and collapse all group headings.
- Search for text within the report by using the search field. Click the search icon repeatedly to move from instance to instance of the search string in the report.
- Drill down to view detailed information about a row or cell by clicking on the row or cell of a summarized report.
- Interact with the data displayed in a chart.
- Sort results by the values in a column by clicking the column heading.
- Filter which rows of data are included in a report.
- Initiate actions from the Actions menu. Actions might include editing the report in the Report Editor, printing the report, and exporting the report.

  Note: For more information about the available actions, refer to the Actions menu options Help topic.

Scheduling reports in the Case Manager portal

You can schedule reports to run at a time, interval, and frequency that you define. When you schedule a report, you are subscribed to that report by default. You receive a copy of the report by email each time the report runs.

1. In the Report Browser, click the gear icon for the report shortcut and select **Schedule** to open the *Schedule Reporting Task* form.

2. On the Schedule Reporting Task form, specify the task definition and description.

   Note: To learn more about how to schedule reports, refer to the Scheduling reports Help topic.

Working with the Report Editor

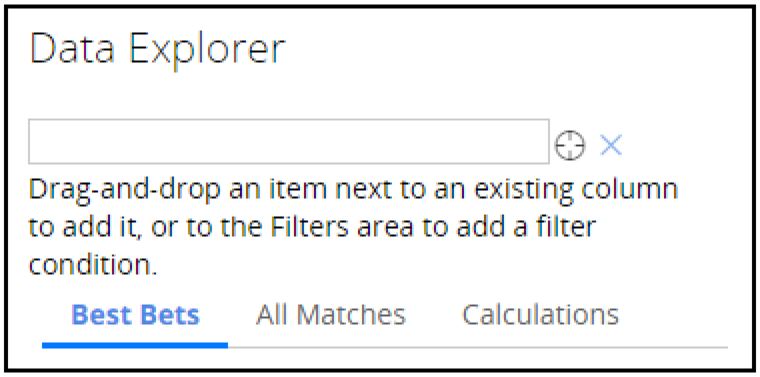
The Report Editor displays a report and provides options for editing it. The Report Editor also displays the name of the report, the date and time when the report ran, and whether simulated or actual data is being used.

Using the Data Explorer

The Data Explorer panel on the left of the screen provides a quick way to find a property or calculation to include as a column in the report, or to use in defining a filter condition.

Enter a value in the search box of the Data Explorer and click the magnifying glass to limit the display in the current tab to only properties whose name or label match the search string you entered. Click the **X** to clear the search box and display all properties.

The Data Explorer includes three tabs:


Data Explorer

Drag-and-drop an item next to an existing column to add it, or to the Filters area to add a filter condition.

Best Bets

The Best Bets tab displays the properties that you are most likely to use in your report, organized in a tree structure. Expand any subfolders (representing page lists and other embedded properties) to see more properties.

All Matches

The All Matches tab displays all the properties that are available for use in the report, organized in a tree structure. Navigate the tree and add a property to populate a column in the report.

Calculations

The Calculations tab allows you to select an SQL function and identify one or more properties for it to work with. The result of the calculation can populate a column in the report.

Using the Actions menu

The Actions menu provides additional options such as:

Report Details

Edit the description or change the category of the report.

Summarize

Displays the Summarize form. The form displays all columns in the report, and you can specify sorting information and a summarization function for each column.

List

Converts a summarized report to list report.

APPLICATION DESIGN

The role of the System Architect

Introduction to the Role of the System Architect

Enterprise application development is a team effort. A successful project requires that each member of the team know and perform their duties.

In this lesson, you learn about the system architect role, and how this role relates to other roles during application development. You also learn the types of tasks that system architects perform on projects. Finally, you learn about the different types of system architects, and how they work together on an application.

After this lesson, you should be able to:

- Explain the role of the system architect during application development.
- Identify system architect tasks during application development.

The role of the system architect

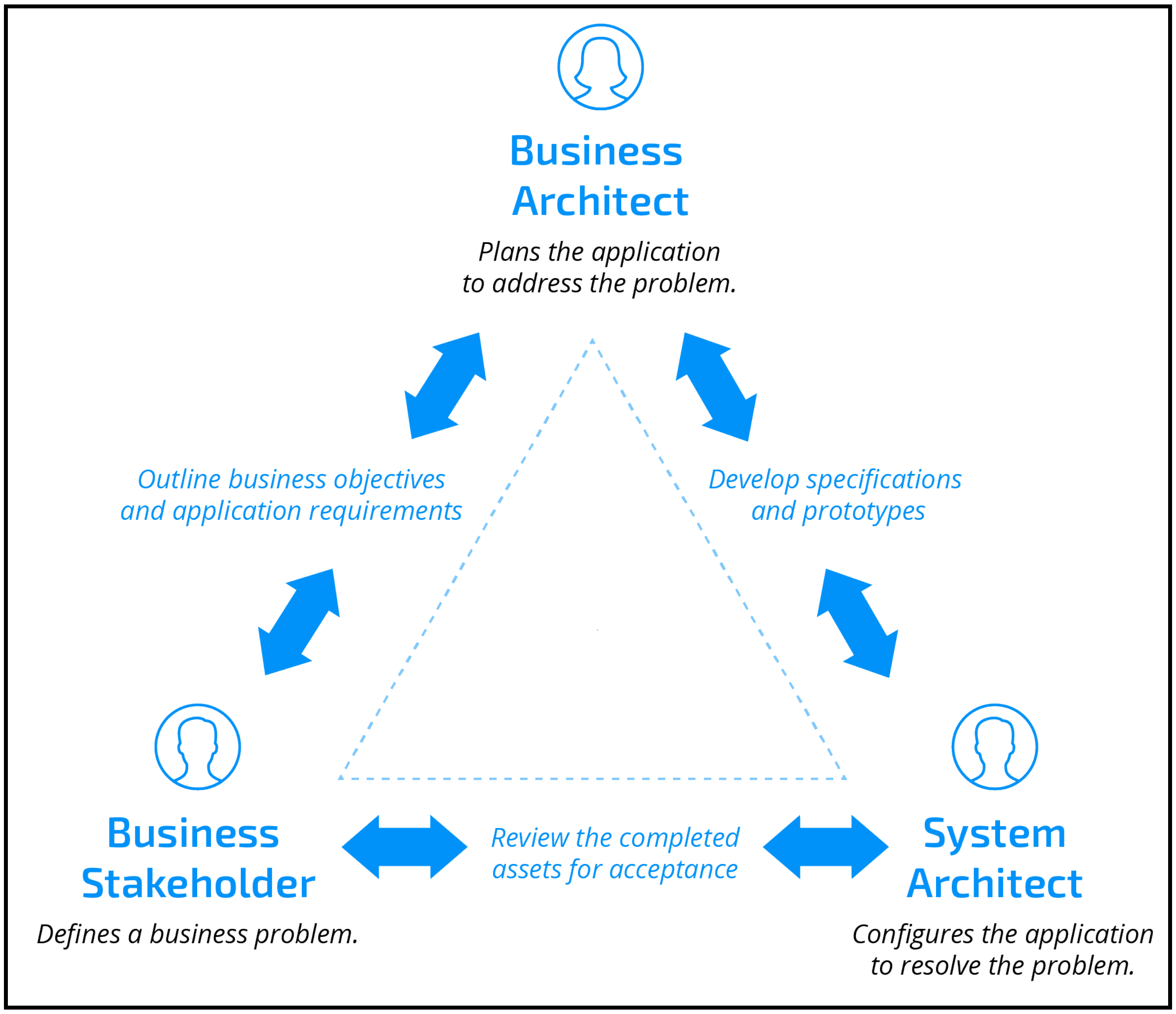
A successful Pega application requires collaboration between three parties — business stakeholders, business architects, and system architects — to solve business problems.

- Business stakeholders define a business problem.
- Business architects plan the application to address the problem.
- System architects configure the application to resolve the problem.

To start, business architects and business stakeholders outline business objectives and application requirements. The goal is to describe what the application must do to address the business problem.

Next, business architects and system architects plan application behavior with specifications. These specifications describe how the application manages and automates work. System architects often prototype application features to help refine the specifications. These prototypes help align the application with the business needs.

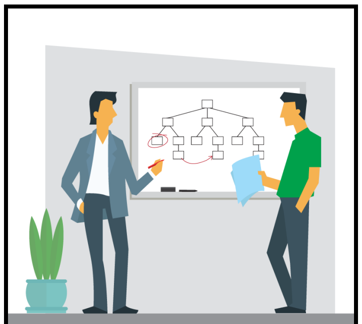
Finally, system architects provide the technical skills needed to complete the application. System architects configure application assets such as UI forms, automated decisions, and correspondence. System architects then review the application with business stakeholders for approval.


Business Architect
Plans the application to address the problem.
Outline business objectives and application requirements
Develop specifications and prototypes
Business Stakeholder
Defines a business problem.
Review the completed assets for acceptance
System Architect
Configures the application to resolve the problem.

Types of system architects

Most Pega projects staff the system architect role with three levels of system architects. These three levels of system architects work together on application design and configuration.

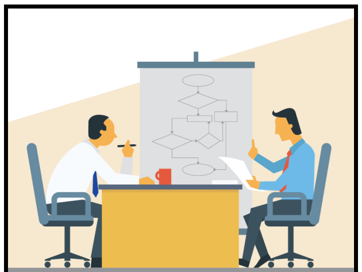
Lead System Architects


Lead System Architect (LSA)

Lead System Architects (LSAs) are the most experienced system architects. LSAs have the following responsibilities on a Pega project:

- Direct the technical effort on a project.
- Work with business architects (BAs) to design an application architecture.
- Design the architecture to reuse application assets as much as possible.
- Meet quality goals, including application performance.

Senior System Architects


Senior System Architect (SSA)

Senior System Architects (SSAs) supervise development on the application. SSAs have the following responsibilities on a Pega project:

- Focus on a particular process or UI form, and supervise the development of that process or form.
- Add technical details to specifications, translating application requirements into guardrail-compliant feature designs.
- Identify opportunities to reuse existing assets within the application design.

System Architects

System Architect (CSA)

Finally, System Architects (SAs) perform much of the development work. SAs have the following responsibilities on a Pega project:

- Configure and unit test individual application elements such as correspondence and automated decisions.
- Help draft processes and user interfaces during DCO sessions.

The building blocks of a Pega application

Introduction to the Building Blocks of a Pega Application

The key to efficient application development is to only develop the assets you need to develop. You can reuse existing assets to limit development of new assets. Pega delivers an impressive array of application assets. Reusing application assets is more efficient than creating their equivalents.

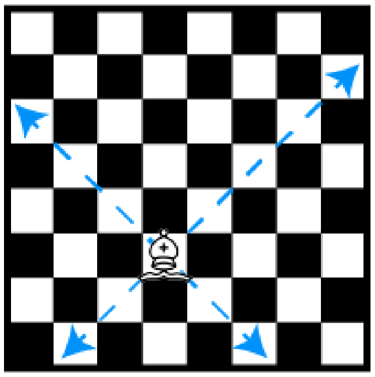
When an application requires new assets, you only need to create key features and functions once. Pega's inheritance structure lets you reuse the new resources wherever they are needed in your application. Eliminating redundant assets simplifies maintaining and extending the application.

In this lesson, you learn how Pega manages application assets and how you can reuse assets through application design and Pega's principle of inheritance.

After this lesson, you should be able to:

- Describe the relationship between an application and rules.
- Differentiate between a rule and a rule type
- Explain the principles of rule inheritance and scope
- Differentiate between pattern inheritance and directed inheritance
- View class inheritance

Rules and rule types

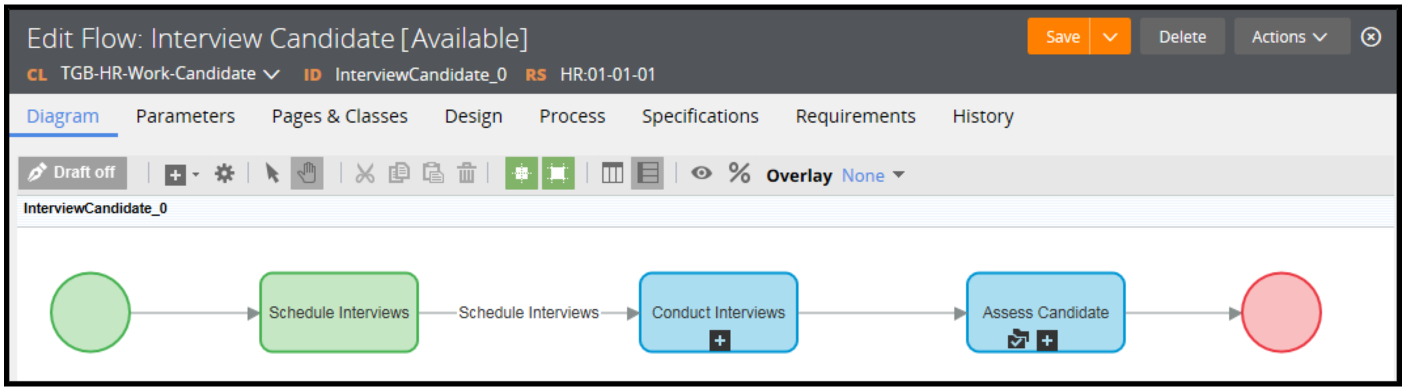


When you play a game of chess, you and your opponent agree to follow a specific set of instructions. These instructions govern game play, such as how each piece moves on the game board. These basic instructions are the rules of chess.

When you model a case type in a Pega application, you configure the application with instructions to create, process, and resolve a case. These instructions are **rules**. Rules describe the behavior of individual cases. The Pega platform uses the rules you create to generate application code.

Pega provides wizards that create and modify many of the rules in an application for you. For example, the Case Designer automatically creates rules to describe cases, processes, and UI forms. Much of the work of designing an application can be completed by using these wizards, although you may need to access a rule directly.

The following screenshot shows an example of a flow rule. A flow rule is used to describe a process. Pega Express and the Case Designer automatically create a flow rule whenever you add a process to a case life cycle.



Each rule is an instance of a rule type. Pega provides many rule types, with each type tailored to a specific type of case behavior. For example, Pega provides one type of rule to describe a process flow, and another type of rule to describe an automated email notification.

You create an individual rule from one of the rule types provided by Pega. Each rule you create describes specific aspects of case behavior, such as a submission form or an audit process. The use of individual rules makes your application modular. By describing case behavior with modular, task-focused rules, you can combine and reuse rules as needed. In this manner, rules are analogous to classes in Java or other object-oriented programming languages. For example, you create a rule to describe the content of an email to send to a customer regarding the status of a change of address. Your application automatically sends this email after the customer enters the old and new address. By creating the message as a separate rule, rather than embedding the message in the case life cycle, you can update the content of the email without impacting the rest of the business process.

This modularity provides three significant benefits:

1. **Versioning** — System architects create a new version of a rule whenever case behavior needs to change. Pega maintains a history of changes to a rule, allowing system architects to review the change history and undo changes if needed. Since each rule describes a specific case behavior, the rest of the case is unaffected. For example, a system architect updates a UI form with instructions and removes a critical field. You can review the history of the form and revert back to the version before the changes were made, without changing other rules in the application.
2. **Delegation** — System architects delegate rules to business users to allow business users to update case behavior as business conditions change. The business user updates the delegated rule, while other parts of the application remain unchanged. For example, expense reports that total USD25 or less are approved automatically. You create a rule to test whether an expense report totals USD25 or less and delegate the rule to the Accounting department. The Accounting department can then update the rule to increase the threshold for automatic approval increases to USD50, without submitting a change request for the application.
3. **Reuse** — System architects reuse rules whenever an application needs to incorporate existing case behavior. Otherwise, you must reconfigure the behavior every time the behavior is needed. For example, you create a UI form to collect policyholder information for auto insurance claims. You can then reuse this UI form for property insurance claims and marine insurance claims.

KNOWLEDGE CHECK

ANSWER What is the purpose of a rule in a Pega application?

A rule is an instruction for describing a specific case behavior, such as a process or automated decision.

Rules and rulesets

To package rules for distribution as part of an application, you collect rules into a group called a **ruleset**. If a rule is similar to a song, a ruleset is similar to an entire album. Just as you can copy the album to share with a friend and allow your friend to listen to your favorite song, you can share a ruleset between applications to allow several applications to use the same rules.

Ruleset versioning

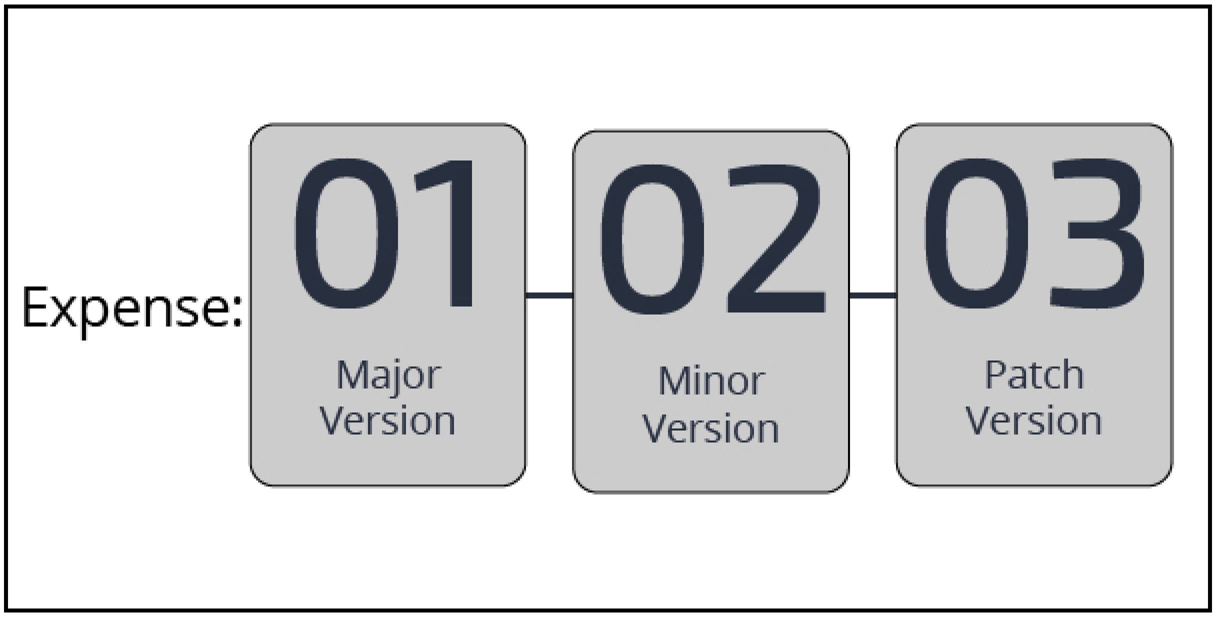
System architects collect individual rules into a subset of a ruleset, called a **ruleset version**. To update the contents of the ruleset, you create a new ruleset version. Ruleset versioning allows system architects to easily update applications by providing access to an entire set of rules at once.

You identify each ruleset by its name and version number. For example, an application to process expense reports includes a ruleset named Expense. You refer to the ruleset as Expense:01-02-03, where Expense is the name of the ruleset and 01-02-03 is the version number.

The version number is divided into three segments: a major version, a minor version, and a patch version.

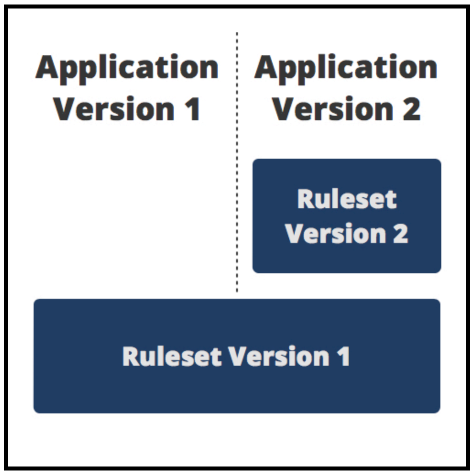
- The major version represents a substantial release of an application. A major version change encompasses extensive changes to application functionality. For example, the Accounting department uses an application to manage expense reports. If Accounting wants to extend the application to track employee time off for payroll accounting, you create a new major version of the ruleset.
- The minor version represents an interim release or enhancements to a major release. For example, you need to update an expense reporting application to automatically audit travel reimbursements. You create a new minor version of the ruleset.
- The patch version typically consists of fixes to address bugs in an application. For example, you notice that a field in the current version of an application is labeled incorrectly. You create a new minor version to correct the field label.

Each segment is a two-digit number starting at 01 and increasing to 99. Ruleset version numbering starts at 01-01-01, and increments upward.



Each application consists of a sequence of rulesets, called a **ruleset stack**. The ruleset stack determines the order in which Pega looks through rulesets to find the rule being used. Each entry in the ruleset stack represents all the versions of the specified ruleset, starting with the listed version and working down to the lowest minor and patch version for the specified major version.

Each version of an application contains a unique ruleset stack. This allows an updated application to reference new ruleset versions that contain updates and new features.



Bob is a system architect working on the first version of an application to manage expense reports. Bob creates rules for the first version of the application, such as processes, UIs, and notifications. Bob collects these rules into the first version of the Expense ruleset, Expense:01-01-01.

Months later, Tanya receives an enhancement request to update a UI in the application to collect extra information from employees due to a policy change. This update enhances the rules created earlier by Bob. Tanya creates rules to model this new behavior in a second version of the ruleset, Expense:01-02-01. She then uses the Expense:01-02-01 ruleset in the updated expense reporting application.

Employees who use the first version of the application view the UI that Bob created. Only employees who use the updated application view the UI that Tanya created. This allows users to use the first version of the application while the second version is in development.

KNOWLEDGE CHECK

ANSWER A ruleset version is identified with a string of three numbers. What do these three numbers indicate?

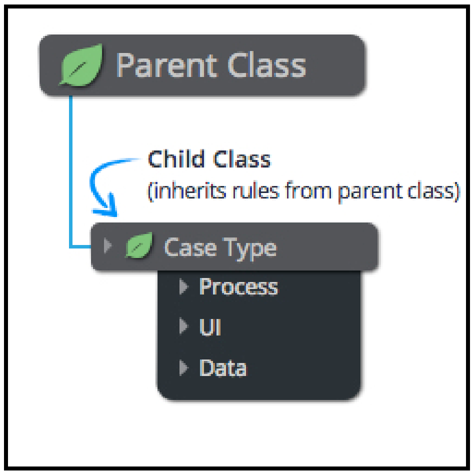
The three numbers used to identify a ruleset version indicate the major version, minor version, and patch version of the ruleset.

Classes and class hierarchy

One strength of the Pega platform is the reuse of rules between case types and applications. System architects often reuse rules — from single data elements to entire processes — in applications. The reuse of rules improves application quality and reduces development time. Organizations that adopted the Pega 7 Platform reduced development costs by 75 percent and time-to-market by 50 percent, launching new business applications up to 90 days earlier ([1]).

Within an application, Pega groups rules according to their capacity for reuse. Each grouping is a **class**. Each application consists of three types of classes.

- **Work class** — The work class contains the rules that describe how to process a case or cases, such as processes, data elements, and user interfaces.
- **Integration class** — The integration class contains the rules that describe how the application interacts with other systems, such as a customer database or a third-party web server.
- **Data class** — The data class contains the rules that describe the data objects used in the application, such as a customer or collection of order items.

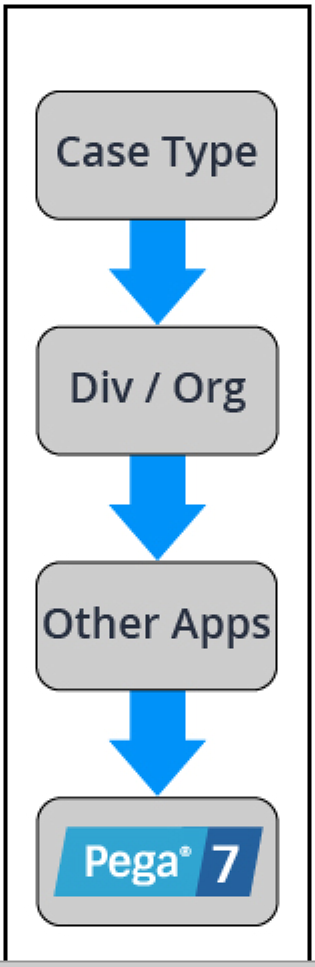


A class can also contain other classes. A class that contains another class is called a **parent class**, while a class that is contained by another class is called a **child class**. A child class can reuse, or **inherit**, any of the rules defined for its parent class.

The work class contains a child class for each case type in your application. Each child class contains all of the rules unique to a case type, such as an auto insurance claim. The data class contains a child class for each data object.

The classes that comprise an application are called a **class hierarchy**. The class hierarchy determines how system architects can reuse rules within the application. The class hierarchy consists of several groups of classes:

- Classes that describe a specific case type, such as expense reports or auto insurance claims
- Classes that collect common rules and data elements. These classes allow the reuse of rules at the division and organization level, such as an approval process shared across the entire IT department.
- Classes from other applications, such as industry-specific Pega applications. So you can create a generic application for policy administration to use as a base for customizing versions for different countries.
- Base classes provided by the Pega platform. These classes contain rules that provide basic functionality for processing cases. For example, the Pega platform provides data elements that record who created a case and the time needed to complete an assignment.



Any rule available to an application through the class hierarchy is considered in scope. Rules that an application can not access through the class hierarchy are considered out of scope.

Pega names each class to identify the position of the class within the class hierarchy. Consider the class TGB-HR-Work. Each level of the class hierarchy is separated by a hyphen (-). So *TGB-HR-Work* is a child of the class *TGB-HR*, which is a child of the class *TGB*.

KNOWLEDGE CHECK

ANSWER

What is the purpose of a class in a Pega application?

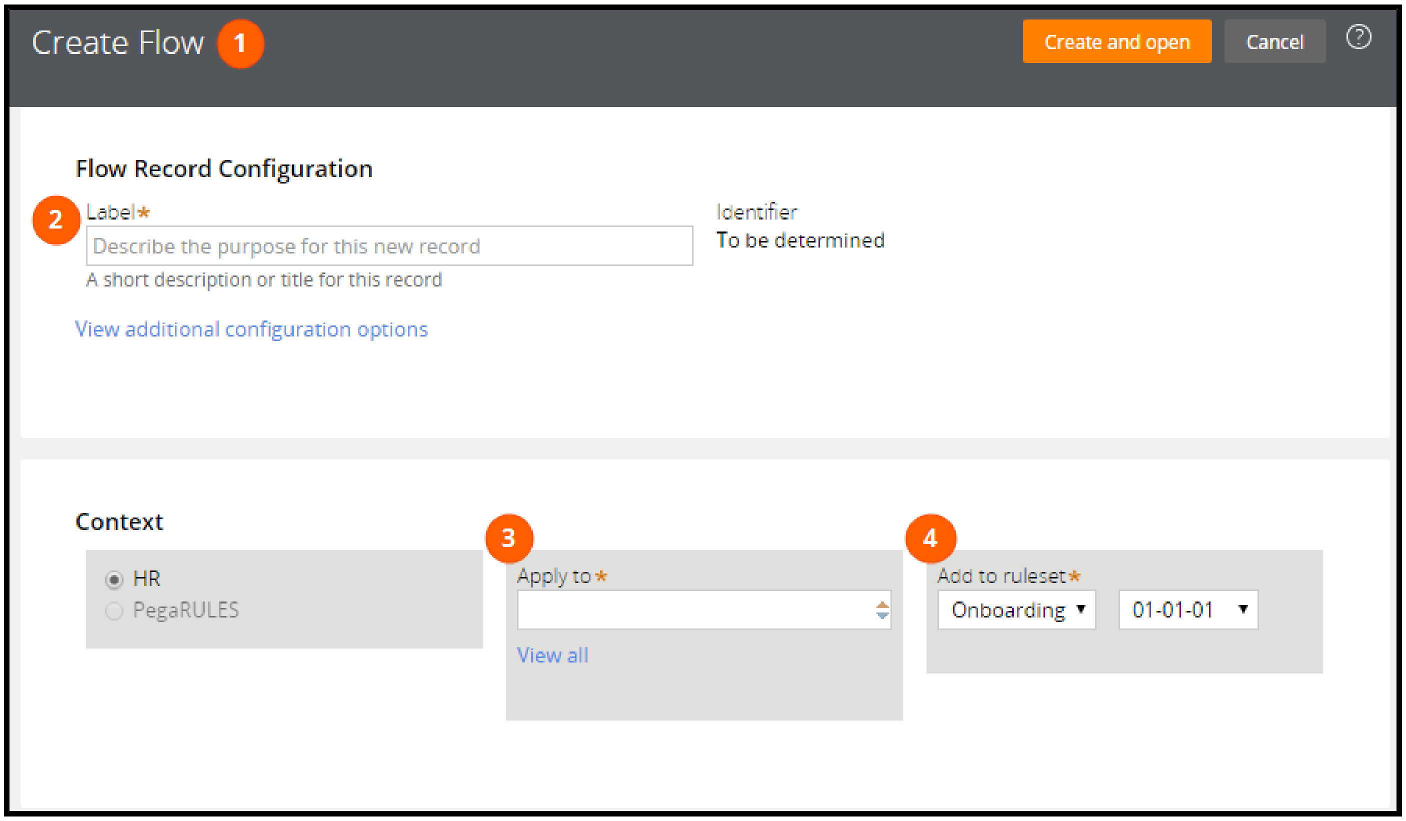
A class organizes rules within an application. The position of a class within the class hierarchy determines the reusability of the rules in that class.

[1] Forrester Consulting. (2015). *The Total Economic Impact™ Of The Pega 7 Platform*. Retreived from https://www.pega.com/forrester-tei

How to create a rule

When you create a rule, Pega provides you with the New Record form. The New Record form allows you to create either a rule or a data instance.

When you create a rule, the New Record form prompts you to provide four pieces of information: rule type, identifier, class, and ruleset. This information is used to identify the rule uniquely within your application.



Flow Record Configuration

Label* Describe the purpose for this new record
A short description or title for this record

Identifier
To be determined

Context

● HR
○ PegaRULES

Apply to*

Add to ruleset* Onboarding 01-01-01

1. The **rule type** determines the type of behavior modeled by the rule. Each rule type models a specific type of behavior, such as automated decisions, UI design, or data storage. For example, to model a process, you use a specific type of rule called a flow rule. You determine the rule type when you open the New Record form.
2. The **identifier** identifies the purpose of the rule. For example, to model the process for approving insurance claims, you use a identifier such as ClaimsApproval. This identifier allows you to differentiate the approval process from a submission process. Pega automatically determines the identifier from your entry in the **Label** field.
3. The **class** identifies the scope of the rule. You specify the class of a rule in the **Apply to** field. The class you select determines how extensively you can use the rule — within one case type, or across case types.
4. The **ruleset** is the container for the rule. The ruleset identifies, stores, and manages the set of rules that define an application or a major portion of an application.

The combination of rule type, name, class, and ruleset allows Pega to uniquely identify each rule. This combination allows an application to call the correct rule during case processing, through a process called **rule resolution**. With rule resolution, Pega determines the appropriate rule to run when an application calls a rule.

You can access the New Record form several ways. Based on your choice, Pega provides default values in some or all of the fields on the form. You can change these values before you create the rule.

How to access the New Record form	Default values provided
From the **+Create** menu, select the rule category, then the rule type.	Rule type, ruleset
In the Application Explorer, select the class in which you want to create the rule, then select the rule category, then select the rule type.	Rule type, apply to class, ruleset
In a field on a form, enter the name of the rule to create, then click the **Target** icon.	Rule type, identifier, apply to class, ruleset

After you complete the New Record form, click **Create and open** to configure the rule behavior.

How to update a rule

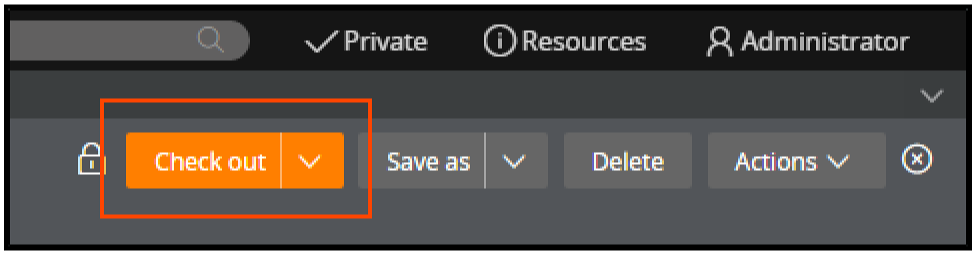
System architects often secure rulesets to prevent unauthorized or unintended changes to rules. When you edit the rules in a secured ruleset, you either check out the rule or perform a private edit.

Rule check out and check in

The check-out feature is used to manage changes to rules when multiple developers work on an application. This feature allows a system architect to update a rule while preventing updates by other system architects. Rule check out creates a copy of a rule in a ruleset that is only visible to you, called a **personal ruleset**. After you update the rule and test the changes, you check in the rule. This updates the application ruleset with a new version of the rule.

Checking out a rule

On the rule form header, click **Check out** to check out the rule.



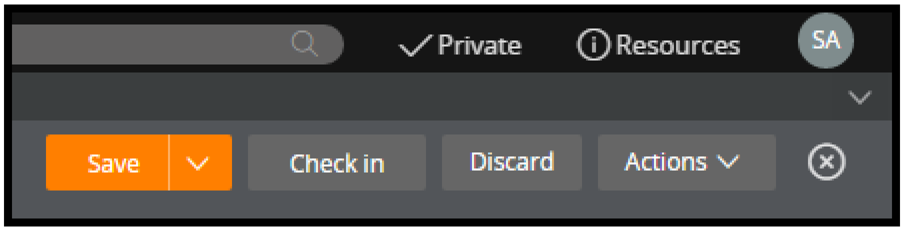
Checking out a rule creates a copy of the original rule in your personal ruleset and prevents other system architects from checking the rule out until you check in your changes.

The personal ruleset occupies the top spot in the ruleset stack. The rules in your personal ruleset override rules in the rest of the application. This allows you to test your changes to the rule without affecting other system architects.

In the rule header, click **Private** to view a list of the rules you have checked out.

Checking in a rule

When you check out a rule, the rule header updates with three new buttons: Save, Check in, and Discard.



When you finish editing the rule, click **Save** to save your changes to the checked out rule. This commits the updated rule to your personal ruleset. After you save the rule, you can test your changes.

After you test the rule and confirm that your configuration works as expected, click **Check in** to replace the original rule with the version in your personal ruleset. Unless approval is required, your changes immediately affect application behavior.

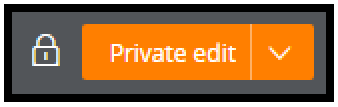
You are not required to check in your changes immediately. You can log off and return to a checked out rule later or click **Discard** to remove the rule from your personal ruleset.

Select **Private > Bulk actions** to check in several records at the same time.

Private edit

A **private edit** provides a nonexclusive check out of a rule. This allows other system architects to edit a rule at the same time. Private edits are useful for quick debugging without interrupting development by other team members.

As a best practice, older versions of a ruleset are locked to prevent changes. For rules in a locked ruleset, a lock icon is displayed on the rule form. To update a rule in a locked ruleset version, save the rule to an unlocked ruleset version, then check out the rule if necessary.



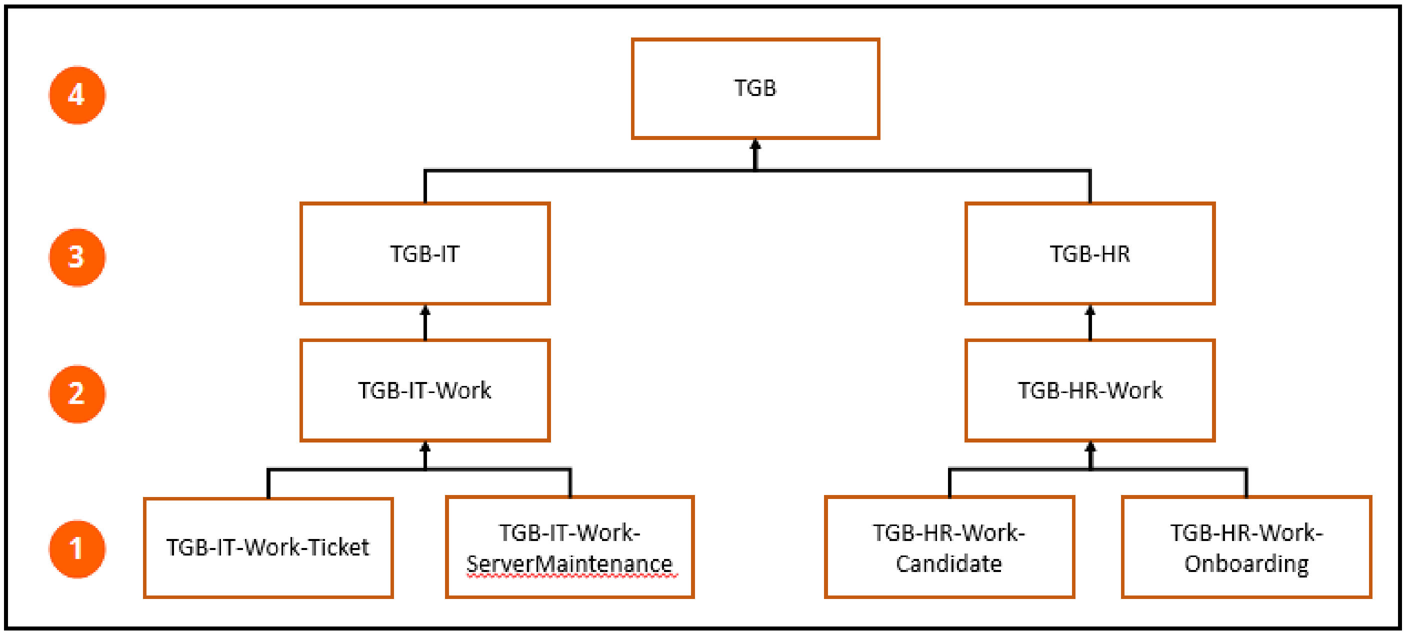
How to reuse rules through inheritance

Inheritance allows your application to reuse rules that have already been created for other cases or applications. By reusing a rule through inheritance, rather than creating an identical copy of the rule, you reduce development and testing time without sacrificing application quality.

Pega provides two methods for inheriting rules: pattern inheritance and directed inheritance.

Pattern Inheritance

Pattern inheritance describes the business relationship between classes. Pattern inheritance allows your application to share rules with other applications throughout an organization. The following image demonstrates a basic pattern inheritance hierarchy.



1. Rules for a specific type of case are stored at the lowest level of the hierarchy. Rules at this level only affect a single type of case, such as IT service tickets or onboarding requests.
2. The next level is the **class group**. The class group contains all of the case types in an application. In the previous image, TGB-IT-Work contains all of the case types for the IT department, while TGB-HR-Work contains all of the case types for the human resources (HR) department. Rules at this level affect all the case types in the class group.
3. Above the class group is the **division layer**. The division layer contains the work, data, and integration classes for the division.
4. Above the division layer is the **organization layer**. The organization layer contains all of the classes for applications across an entire business or other organization. The organization layer often contains data and integration classes that can be applied across the entire organization

For example, an organization creates an application to manage IT requests. In this application, you use a data element to record the due date for the request. The concept of a due date is not unique to IT requests. Other business processes also use due dates, such as expense reports. You create the data element for the due date in the organization layer, so the application to track expense reports can reuse this data element.

KNOWLEDGE CHECK

ANSWER What type of relationship is described by pattern inheritance?

Pattern inheritance describes the business relationship between classes. Pattern inheritance indicates the reusability of rules throughout an organization, such as whether a rule is usable by a single case type, an entire department, or even an entire organization.

Directed inheritance

Directed inheritance describes the functional relationship between classes. Directed inheritance allows your application to reuse rules from classes in other applications and standard rules provided with the Pega platform. For example, a class that describes automobile insurance policies can inherit from a class that describes a generic insurance policy, and even the generic case type defined by the Work-Cover class provided by the Pega platform. For example, directed inheritance allows you to reuse the standard data element to record the case ID, provided as part of the Pega platform, in your application.

KNOWLEDGE CHECK

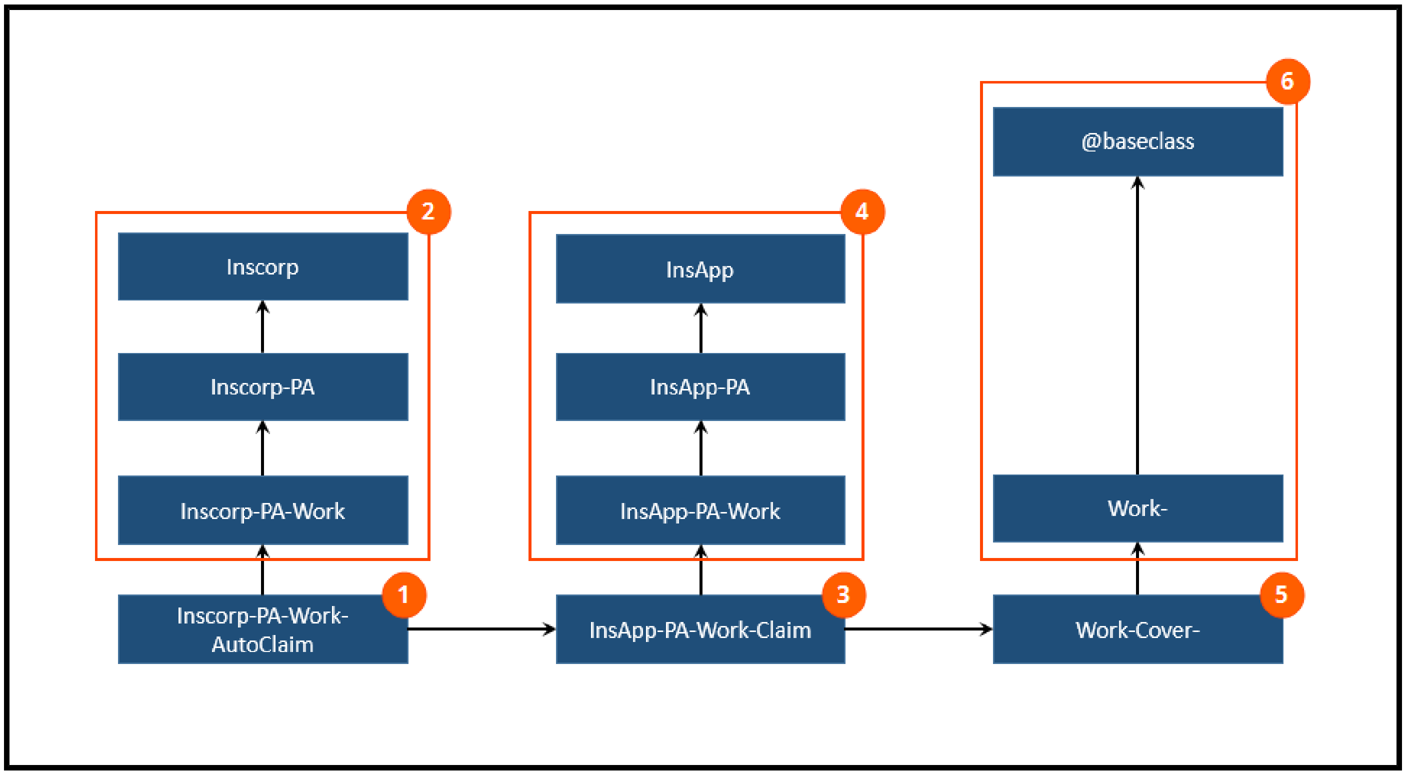
ANSWER How does directed inheritance differ from pattern inheritance?

Pattern inheritance allows you to reuse rules within a single application. Directed inheritance allows you to reuse rules in other applications, including standard rules provided as part of the Pega platform.

Reusing rules through inheritance

When attempting to reuse rules through inheritance, Pega first searches through the parent classes indicated by pattern inheritance. If unsuccessful, Pega then searches the parent class indicated by directed inheritance as the basis for another pattern inheritance search. This process repeats until Pega reaches the last class in the class hierarchy, called the ultimate base class or *@baseclass*. If the rule cannot be found after searching *@baseclass*, Pega returns an error.

Consider the following example in which an auto insurance claim case references the data element that stores the case ID. This data element belongs to the ultimate base class, *@baseclass*. The application containing the auto insurance claim is built on a generic policy administration application. That generic application is built upon the Pega platform.



1. An auto claim case, described by the class *Inscorp-PA-Work-AutoClaim*, references the case ID data element.
2. The data element is not found in the class *Inscorp-PA-Work-AutoClaim*, so Pega searches through the parent classes using pattern inheritance.
3. The data element is not found though pattern inheritance, so Pega searches the parent class specified by directed inheritance, *InsApp-PA-Work-Claim*. This class belongs to the generic policy administration application.
4. The data element is not found in the class *InsApp-PA-Work-Claim*, so Pega searches its parent classes using pattern inheritance.
5. The data element is not found though pattern inheritance, so Pega searches the parent class specified by directed inheritance, *Work-Cover-*. This class belongs to the Pega platform.
6. The data element is not found in the class *Work-Cover-*, so Pega searches its parent classes using pattern inheritance, finally locating the data element in *@baseclass*.

KNOWLEDGE CHECK

ANSWER

From which class does *@baseclass* inherit?

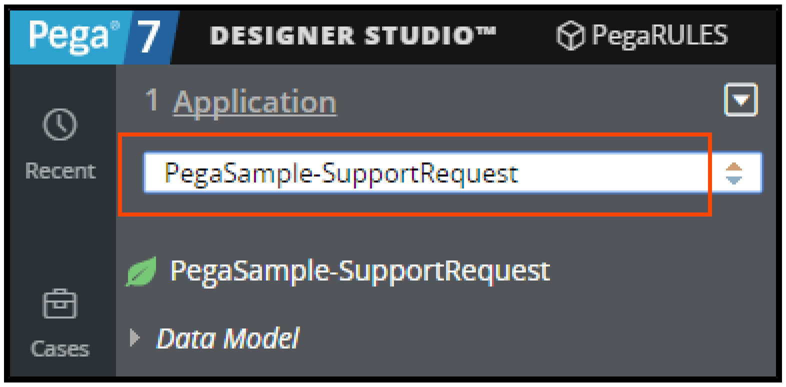
None. In a Pega application, *@baseclass* is the ultimate base class. All other classes inherit from *@baseclass*.

Reviewing class inheritance

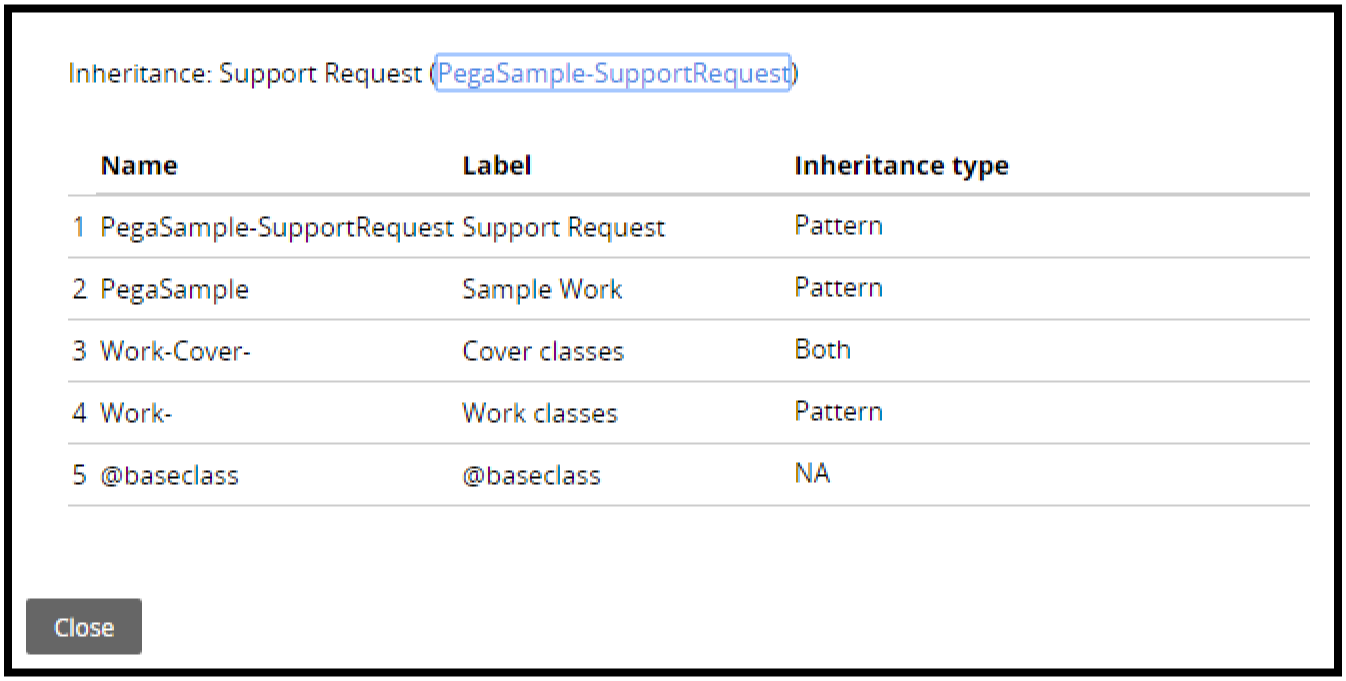
Before you create rules, review the inheritance tree for your application. This allows you to determine the appropriate class to use when creating rules, and allows you to review the rules already available in the application.

To review inheritance for a class:

1. Open the Application Explorer.
2. If necessary, use the application scoping control to enter or select the name of the class. In the following example, PegaSample-SupportRequest is entered into the control to display the contents of the class PegaSample-SupportRequest and its child classes.



3. Right-click the class to review, and select **Inheritance**. The Inheritance Viewer for the selected class opens in a pop-up window. The following example shows the Inheritance Viewer for the PegaSample-SupportRequest class.



Inheritance: Support Request (PegaSample-SupportRequest)

	Name	Label	Inheritance type
1	PegaSample-SupportRequest	Support Request	Pattern
2	PegaSample	Sample Work	Pattern
3	Work-Cover-	Cover classes	Both
4	Work-	Work classes	Pattern
5	@baseclass	@baseclass	NA

4. Review the classes listed as parents for the selected class. The Inheritance Viewer lists each parent

class in hierarchy order, and the inheritance method that provides access to the class.

5. Optional: Click a class to open the class rule. The **History** tab of the class rule provides documentation about the class, such as the purpose and recommended usage of the class.
6. If necessary, click **Close** to close the Inheritance Viewer.

Accessing Applications

Introduction to accessing applications

In Pega, application developers use the integrated developer environment to configure their application. This environment, known as the Designer Studio, provides tools to manage and analyze the application configuration.

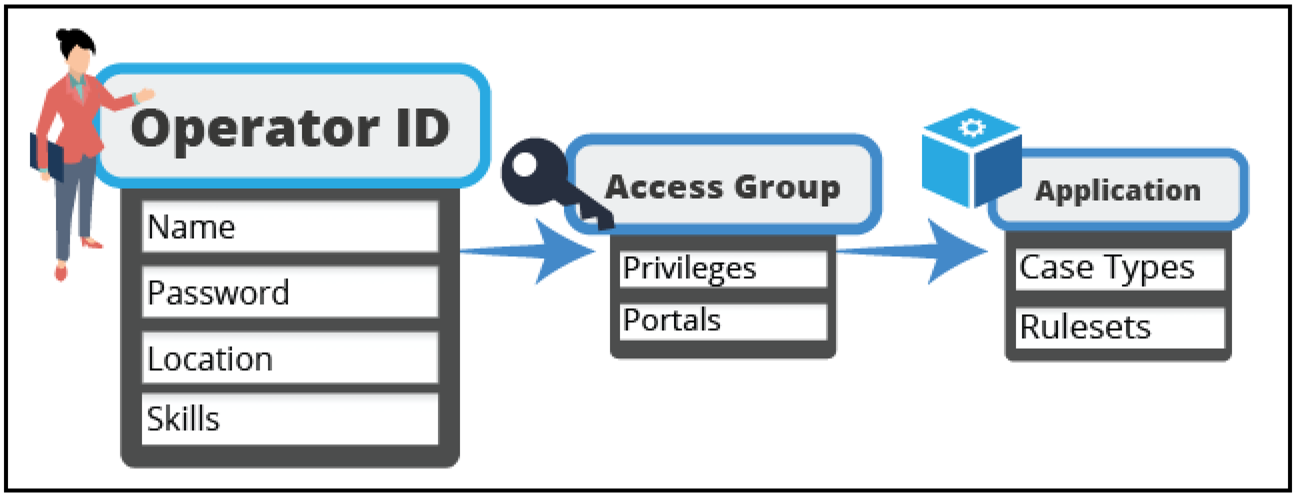
To ensure that only application developers access the Designer Studio, Pega provides a system to manage user privileges for an application.

After this lesson, you should be able to:

- Explain the relationship between a user and an application.
- Switch between applications.

How to manage user access to an application

When users log on to Pega — either through the Designer Studio or an end-user portal — Pega provides the user with access to an application. Pega manages user access through a combination of three items of information: an operator, an access group, and an application. To ensure that users access the correct application, you configure the operator to reference the correct access group.



In Pega, each user is represented by an **operator record**. The operator record contains information such as the operator's name, position, organizational hierarchy, and location.

Each operator is a member of an **access group**. The access group record describes a set of privileges. These privileges are available to users who belong to the access group. Any operator can belong to a number of access groups. However, only one access group is active at any time. The access group also indicates the portal through which the user interacts with Pega.

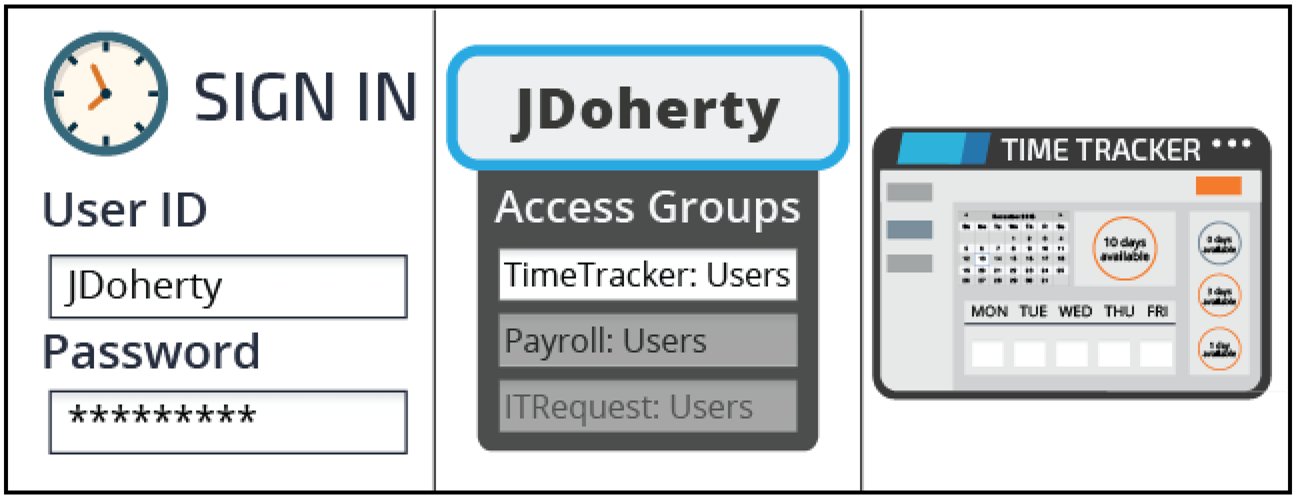
Each access group references a specific application. The privileges available to members of the access group apply only to the referenced application. So, a human resources (HR) analyst may have developer access to an HR application, and user access to an IT application.

How Pega determines the application a user opens

When users log on to a Pega server, they enter their ID and password. The ID corresponds to a unique operator record, and the password authenticates the users.

The operator record lists a set of access groups. The users belong to each access group listed on their operator record, but only one access group is active at any time.

Finally, Pega uses the active access group to determine the application to run.



Allowing a user to access an application

To allow a user to access an application, you add the appropriate access group to the user's operator ID record. Each access group has a unique name that references both the application name and the access level. The following access groups are created automatically for each Pega application.

Access Group	Usage
[application]:Administrators	Developer access to the application. Configured to open the application in Designer Studio.
[application]:Authors	Developer access to the application. Configured to open the application in Pega Express.
[application]:Managers	Manager access to the application. Configured to open the application in the Case Manager portal.
[application]:Users	User access to the application. Configured to open the application in the Case Worker portal.

For example, a Pega application named TimeTracker manages time-off requests for users. To gain developer access to the application , you add the **TimeTracker:Administrators** access group your operator ID record. To open the application when you log on, select the radio button to the left of the access group on the operator ID record.

Assessing Guardrail compliance

Introduction to assessing guardrail compliance

When you develop an application in Pega, Pega monitors your application for configurations that lead to maintenance or performance issues. By addressing these issues, you improve application quality and prevent performance issues for users.

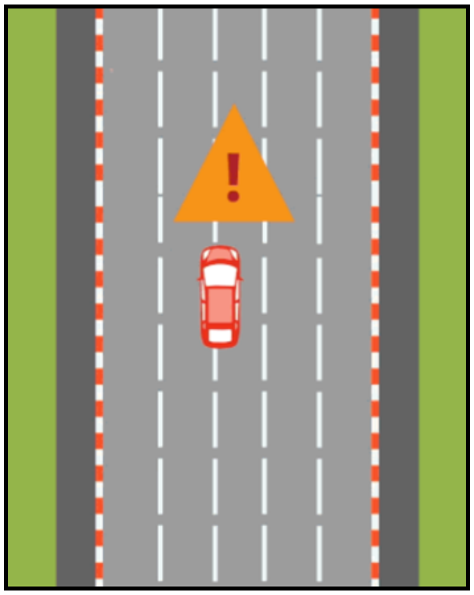
After this lesson, you should be able to:

- Navigate to the Application Guardrails landing page.
- Review guardrail compliance for an application.
- Address application design warnings.
- Justify application designs that violate guardrails.

Compliance Score

When you develop an application, you want to ensure that the application meets established standards for quality and performance. Users quickly adapt to applications that meet established standards, resulting in faster adoption and fewer user errors. The key to releasing quality applications is to identify potential issues during development. By doing this, you can correct these issues before they affect users.

To help you develop high-quality applications, Pega monitors your application for compliance with application design best practices. These best practices, or guardrails, guide you to design applications that avoid potential maintenance and performance issues. When a rule in your application violates one or more guardrails, Pega notifies you with a rule warning. This warning prompts you to review the deviation and allows you to update your application prior to release. Similar to the lane-departure warning system in a vehicle, these warnings alert you to dangerous or risky behavior before they cause a serious problem.



To help you develop high-quality applications, Pega continuously monitors the rules in your application for compliance with established best practices. If a rule violates a best practice, Pega applies a warning to the rule. This warning indicates the severity and type of error that may result, and often describes how to address the violation. Each warning indicates a particular issue with the configuration of the rule, and each rule may indicate multiple warnings.

To quickly assess the overall quality of an application, Pega provides a **compliance score**. Pega assesses the rule warnings for an application to measure overall compliance with Pega Platform best practices. The compliance score measures the number of rules with severe or moderate warnings in an application, compared to the number of rules with no warnings or caution-level warnings. Use the compliance score to quickly assess the quantity and severity of rule warnings in your application.

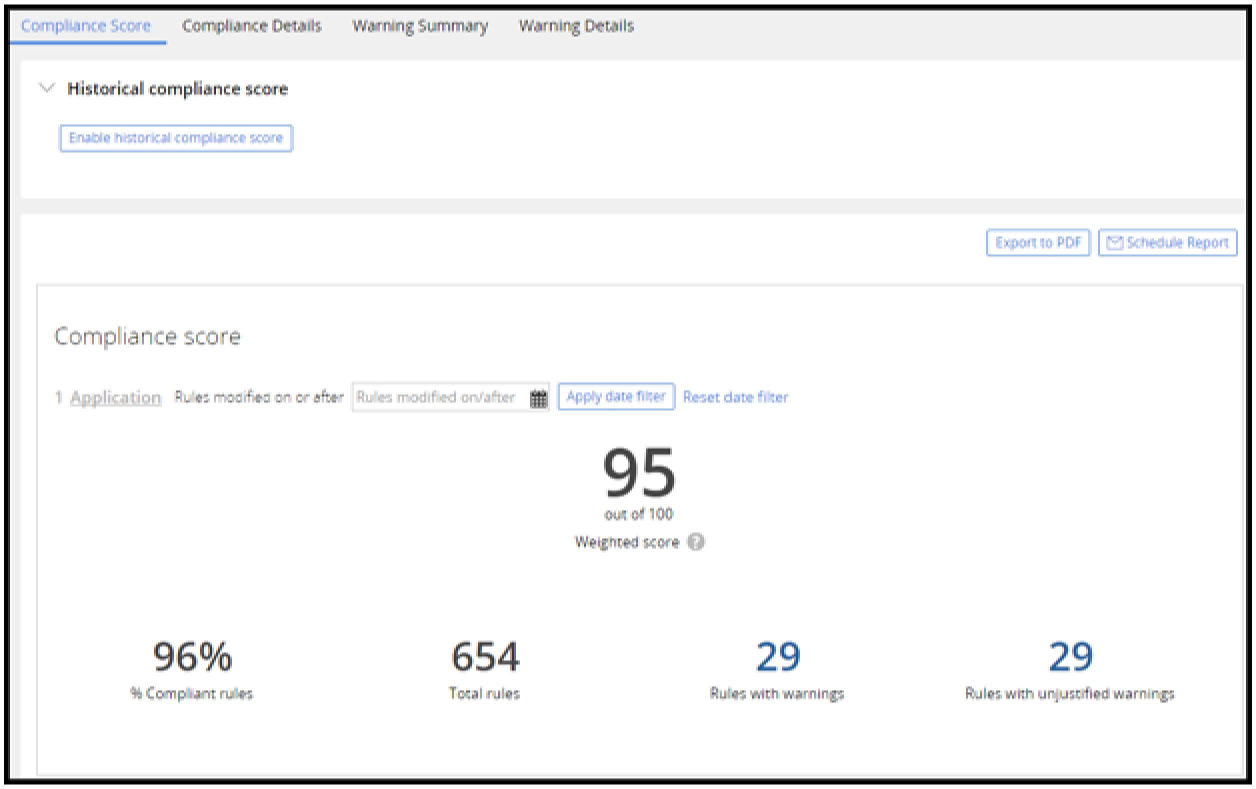
KNOWLEDGE CHECK

ANSWER What is the purpose of the compliance score for an application?

The compliance score provides a quantifiable assessment of application quality by indicating the amount and severity of rule warnings in an application.

How to assess guardrail compliance

During application development, the Application Guardrails landing page provides a single point for assessing application quality. To open the Application Guardrails landing page, open the Designer Studio menu and select **Application > Guardrails**.



The best tool for assessing overall compliance with guardrails is the application's compliance score. The compliance score indicates the impact of complex or custom code on application maintenance and performance. Pega assesses the rules in your application and calculates the compliance score on a scale of 0-100, where 100 is the best possible score.

- A score of 90 or greater indicates your application is in good standing.
- A score of 80-89 indicates your application needs review for improvement.
- A score below 80 indicates that your application requires immediate action.

To generate the compliance score, Pega assesses the rule warnings for an application to measure overall compliance with Pega Platform best practices. The compliance score measures the number of rules with severe or moderate warnings in an application, then compares this result to the number of rules with caution level or no warnings. The more rules with severe or moderate warnings in your application, the lower the compliance score.

The Guardrails landing page also categorizes the rules in your application that include warnings. The **Warning Summary** tab presents two bar charts that report the number and severity of rule warnings in your application, organized by rule type. Use the information on this tab to determine which parts of your application generate the greatest number of rules with warnings.

KNOWLEDGE CHECK

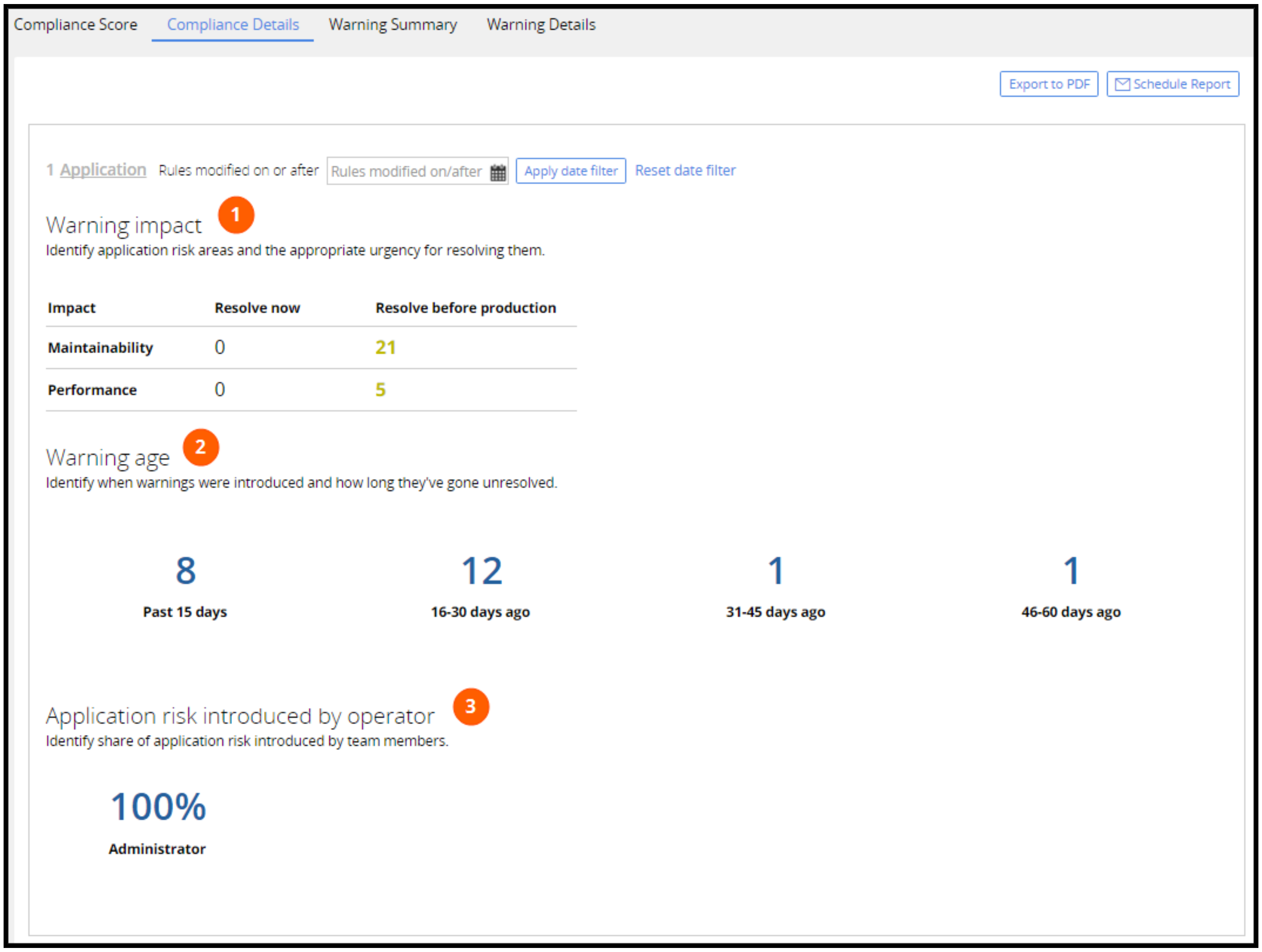
ANSWER The compliance score for an application is 85. Is the application ready to be deployed to users?
A compliance score of 85 indicates that the application requires review before deployment to users. Warnings should be reviewed and addressed to raise the compliance score to 90 or greater.

How to address guardrail violations

Address guardrail violations to improve the quality of your application. Each time you address a guardrail violation, you improve the compliance score for your application and eliminate issues that may impact end users. To improve the compliance score for your application, you resolve rule warnings by correcting the indicated configuration issue.

To address guardrail warnings, you start on the **Compliance Details** tab of the Guardrails landing page. The Compliance Details tab provides three options to analyze the risk areas in your application:

1. **Warning impact**: The severe and moderate configuration issues to address before releasing the application
2. **Warning age**: The age of the warnings in your application
3. **Application risk introduced by operator**: The developer(s) responsible for introducing the behavior generating the warnings



Warning impact

Identify application risk areas and the appropriate urgency for resolving them.

Impact	Resolve now	Resolve before production
Maintainability	0	21
Performance	0	5

Warning age

Identify when warnings were introduced and how long they've gone unresolved.

8	12	1	1
Past 15 days	16-30 days ago	31-45 days ago	46-60 days ago

Application risk introduced by operator

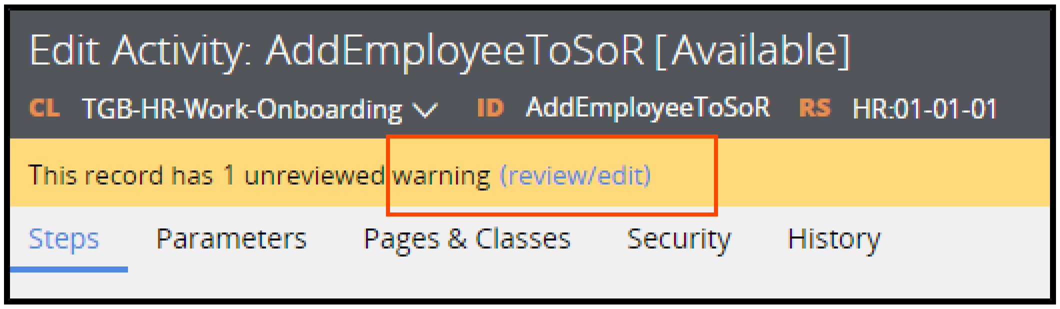
Identify share of application risk introduced by team members.

100%

Administrator

The Warning impact section lists the number of rules with severe (Resolve now) or moderate (Resolve before production) warnings. When preparing to release an application, focus on resolving these issues first. The list is organized by warning type, and highlights the warnings with the greatest impact on the compliance score. The preceding example shows an application with 26 rule warnings that should be addressed before releasing the application — 21 warnings for maintainability issues, and five warnings for performance issues. You can click each number to view the rules that violate guardrails.

When you click a rule warning in one of these lists, you open the rule that contains the warning. Pega displays the rule warning at the top of the rule form, as seen in the following example.



Edit Activity: AddEmployeeToSoR [Available]

CL TGB-HR-Work-Onboarding ID AddEmployeeToSoR RS HR:01-01-01

This record has 1 unreviewed warning (review/edit)

To address a guardrail violation, you either resolve or justify the rule warning.

To resolve a rule warning, you eliminate the cause of the guardrail violation. When you eliminate the cause of a guardrail violation and save the rule, the warning is removed from the rule form and the compliance score improves. For example, Pega displays a warning on any flow rule with draft mode enabled. Once you disable draft mode and save the flow, the warning is removed from the rule, and the compliance score improves.

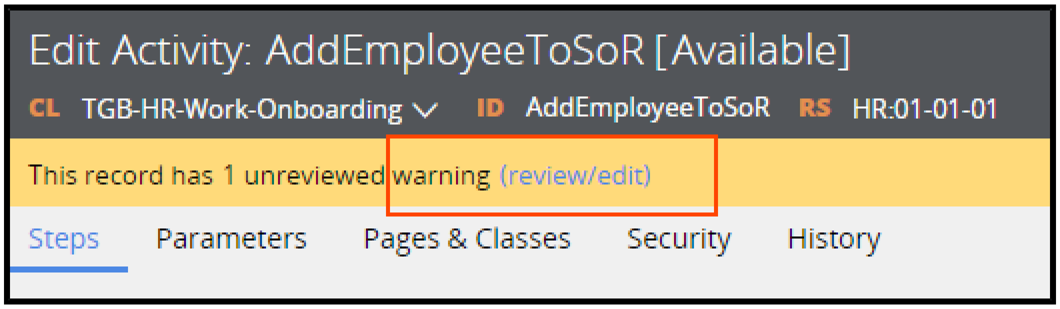
Not all rule warnings can be resolved. A requirement may force a design approach that results in a warning on a rule. For example, your application may require that you output data to a system of record, such as an external database. To output data to an external database, you use a specific type of rule called an activity. But since activity rules are difficult to maintain, Pega applies a rule warning whenever you use an activity in an application. In this case, you must use the activity to output data to the system of record, so you cannot resolve the warning.

If you cannot resolve a rule warning, you **justify** the configured behavior instead. When you add a justification to a rule warning, you acknowledge the guardrail violation and explain the limitation of your application design. Justifying a warning provides a reduced improvement to the compliance score compared to resolving the warning.

Justifying rule warnings

Justifying rule warnings documents the required application behavior that does not adhere to Pega guardrails. Justifying rule warnings improves the compliance score for your application and prepares the application for release.

1. Open the rule containing the warning. The presence of a rule warning is indicated in the rule header.

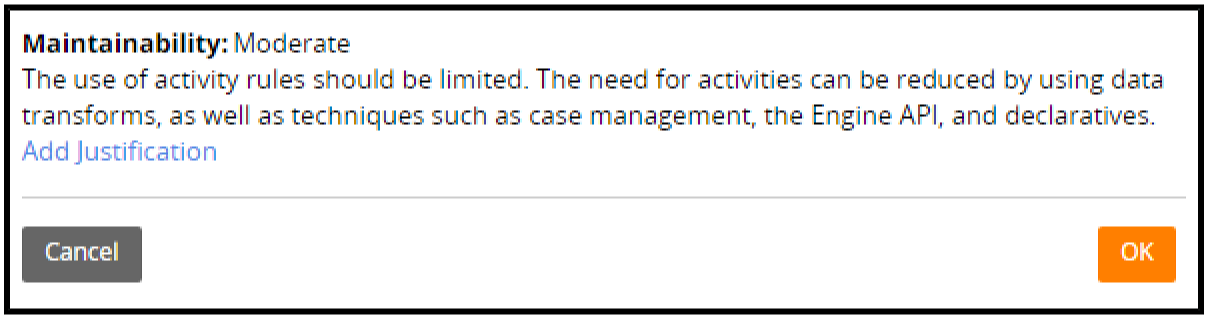


Edit Activity: AddEmployeeToSoR [Available]

CL TGB-HR-Work-Onboarding ID AddEmployeeToSoR RS HR:01-01-01

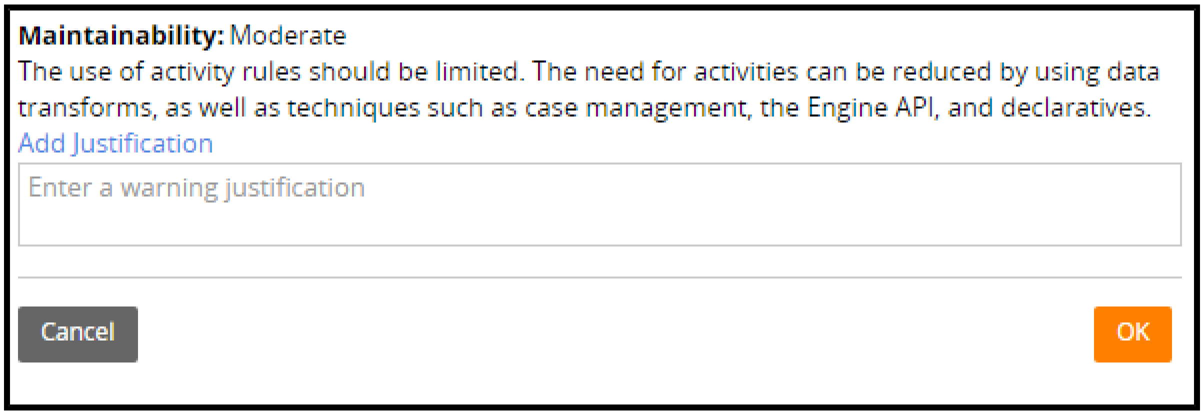
This record has 1 unreviewed warning (review/edit)

2. Click **review/edit** to review the warning. A pop-up displays the reason for the rule warning.



Maintainability: Moderate
The use of activity rules should be limited. The need for activities can be reduced by using data transforms, as well as techniques such as case management, the Engine API, and declaratives.
Add Justification

3. In the pop-up, click **Add justification**. A text field displays in the pop-up.



Maintainability: Moderate
The use of activity rules should be limited. The need for activities can be reduced by using data transforms, as well as techniques such as case management, the Engine API, and declaratives.
Add Justification
Enter a warning justification

4. In the text field, enter the justification for keeping the current configuration. For example, the configuration is necessary to satisfy a requirement, and you cannot satisfy the requirement in a guardrail-compliant manner.

5. Click **OK** to close the pop-up.

6. Click **Save** to record your justification on the rule. The compliance score updates to reflect the justification you entered for the rule.

CASE DESIGN

Creating cases and child cases

Introduction to Creating Cases and Child Cases

When you represent a business process in Pega, you create a template for processing work. This template, called a case type, is used to create individual instances of work, called cases.

Some business processes are too complicated to model with a single case type. To address this situation in Pega, you create more than one case type. Each case type you create represents a part of the business process, so you establish relationships between case types to reflect dependencies in the business process.

After this lesson, you should be able to:

- Explain the relationship between a case type and a case.
- Explain the relationship between a parent case and a child case.
- Add a case type to an application.
- Create additional cases during case processing.

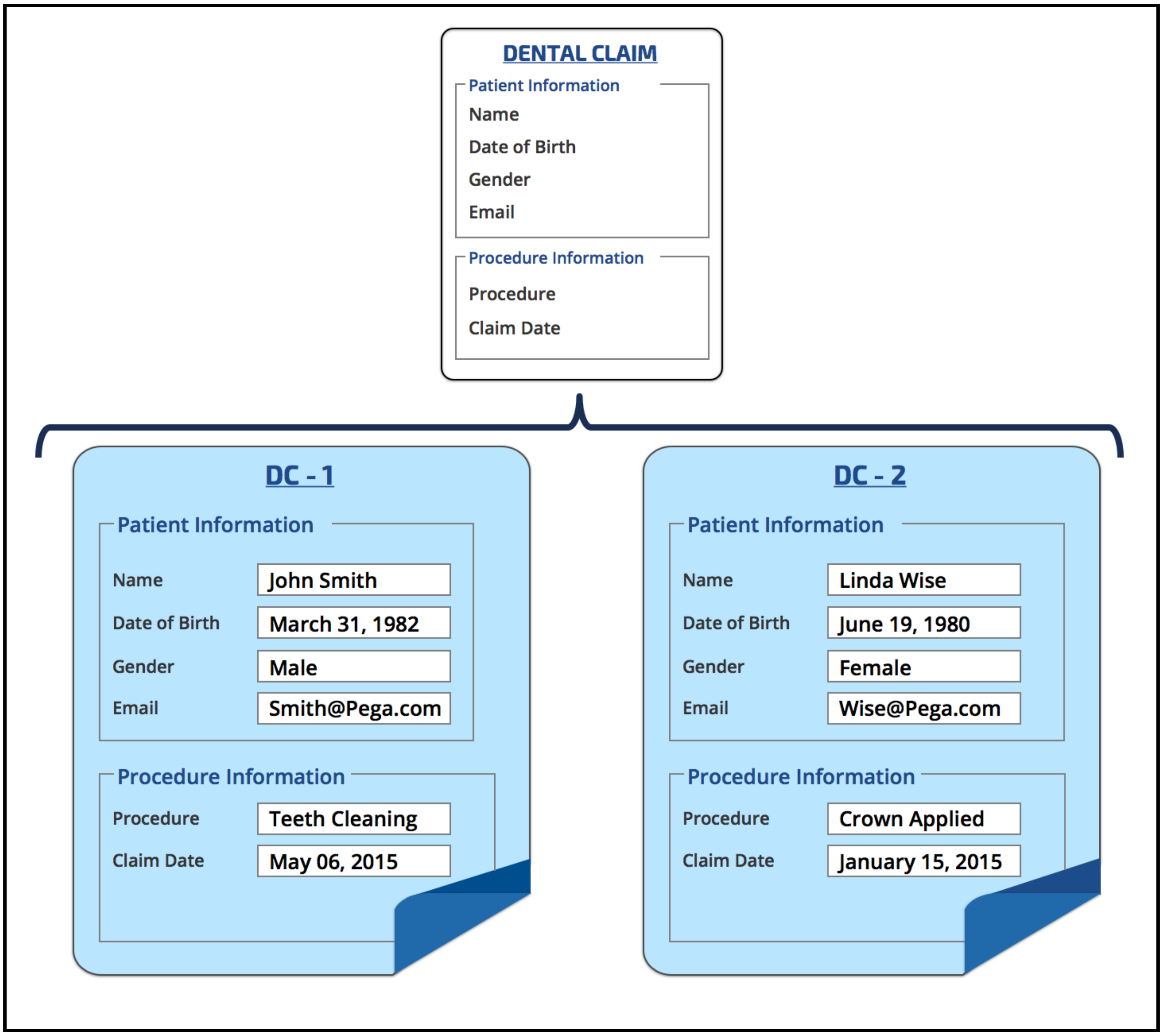
Case type and case

A case type is an abstract model of a business transaction, while a case is a specific instance of the transaction. You can think of a case type as a template for creating and processing cases. When a new transaction starts, a new case is created based on the case type definition.

In a Pega application, you model repeatable business transactions with case types. Each case type captures the life cycle of a specific type of transaction, from creation to resolution. A case type defines data structures, processes, tasks, and user interfaces required for processing the transaction.

For example, a Dental Claim case type models the filing and processing of patient claims of dental procedures. The case type contains data models for holding patient information and dental procedure information. The case type defines processes for reviewing and approving or rejecting claims. The case type also provides user forms for attaching medical documents.

Each time a patient files a dental claim, a new case is created. There can be a case for John Smith for teeth cleaning performed on May 3, and another case for Linda Wise for a new crown applied on January 15. Each case moves through the processes such as review and approval as defined in the case type.



Each case can hold different data and progress through the case life cycle on a different path. One claim case can go through the approval process quickly since it is on a common procedure. Another case might require review since it is on a rare procedure and the claim amount exceeds a certain limit.

Case type relationships

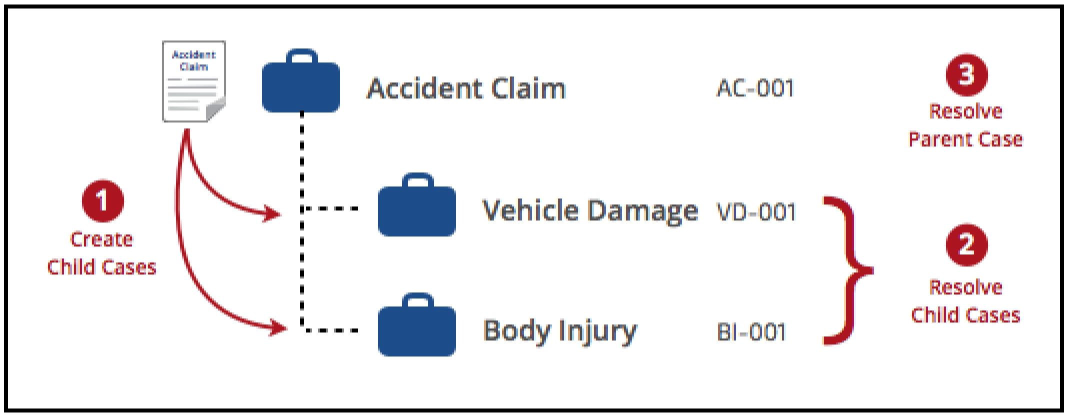
A business transaction can be complicated and involve multiple cases. For example, consider the new hire process. During the interview process, the human resources (HR) department opens a Candidate case for each job applicant. The applicant may be interviewed. If the interview is successful, the applicant receives a job offer. When the candidate accepts the job offer, then HR considers the candidate hired and is now an employee. The Candidate case is completed and creates a Onboarding case to prepare for the new employee's start date. In this example, the Onboarding case is independent of the Candidate case.

Sometimes, the created case is closely related to the original case. In the previous example, part of the onboarding process is to enroll the new employee in a benefits plan. You create a Benefits Enrollment case type because it is a separate business transaction. The outcome of the transaction is an employee's benefits plan — a business transaction that is distinct from the onboarding transaction. However, before an Onboarding case can be approved the Benefits Plan case must be resolved. In this example, the Onboarding case type and the Benefits Enrollment case type are of a parent-child relationship. The system associates the Benefits Enrollment information with the Onboarding case. This allows you to join this associated information when reporting or auditing Onboarding cases. For example, you can create a report for a set of employees showing the medical, dental, and vision plans each employee has.

In a Pega application, you can model this parent-child relationship with a case type hierarchy that contains a top-level case type and child case type.

- **Top-level** — A case type that does not have any parent case type, but can cover, or become a parent of, other case types.
- **Child**— A case type that is covered by a parent case type. When you configure a case type as a child case, Pega maintains a relationship between the parent and child cases. Child case types represent work that must be completed to resolve the parent.

For example, an Auto Insurance application has a top-level case type Accident Claim. The Accident Claim includes two child case types — Vehicle Damage and Bodily Injury. For any Accident Claim case, both of its child cases — vehicle damage and bodily injury — must be addressed before the Accident Claim can be closed. In addition, reports can associate a parent case with any or all of its child cases.



A parent case creating multiple child cases allows for work to be processed in parallel. Each child case can be handled by different parties with different expertise. Under the cover of an Accident Claim case, the Vehicle Damage child case can be handled between a customer service representative, an adjustor, and a repair shop. Meanwhile, the Bodily Injury child case can be handled by a medical claim specialist and certain medical providers.

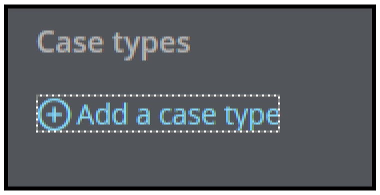
Implementing a business process in a separate case type also allows you to reuse the case type as needed. For example, claims for both automobile and property insurance may involve a bodily injury claim. By implementing bodily injury claims as a separate case type, you can use the bodily injury case type with both automobile and property claims.

Adding a top-level case type in an application

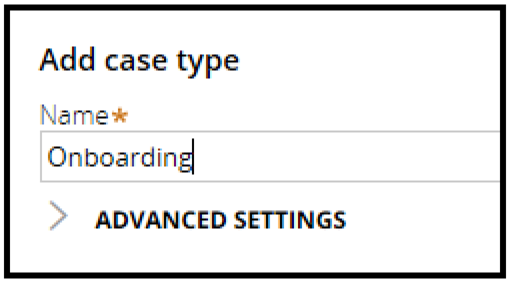
You can add a top-level case type to your application in the Case Designer.

Follow these steps to add a top-level case type to your application:

1. In the navigation panel, click **Cases** to view the list of current case types in your application.
2. Click **+ Add a case type**.

   The Add case type dialog opens.
3. In the **Name** field, enter a name for the case type.

   
   Add case type

   Name*

   Onboarding

4. Optional: Expand the **Advanced Settings** section to configure the rule resolution for the case type. Accepting default settings should suffice in most cases.
5. Click **Submit**.

Adding a child case type in an application

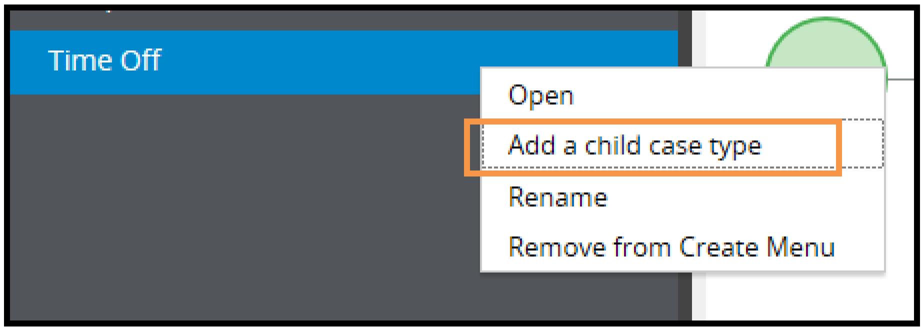
You can define a parent-child relationship by either reusing an existing case type or adding a new case type.

Adding a new child case type to your application

Add a new case type as a child case when no existing case types meet your business requirements.

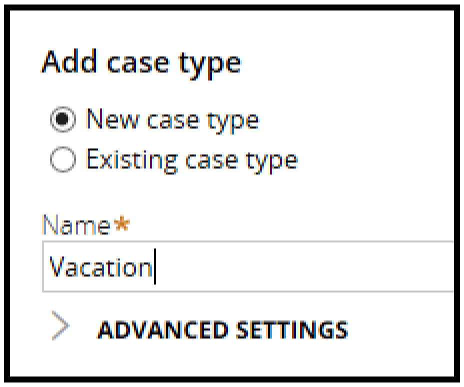
Follow these steps to add a new child case:

1. Open the Case Type Explorer.
2. Hover over a parent case type name and click the **options** menu.
3. Select **Add a child case type**.



The add case type dialog opens.

4. Click **New case type**.
5. In the **Name** field, enter a name for the child case type.


Add case type

(●) New case type
( ) Existing case type

Name*
Vacation

6. Optional: Expand the **Advanced Settings** section to configure the rule resolution for the case type.
7. Click **Submit**.

Adding an existing child case type to your application

Review existing case types and case-type dependencies before reusing a case type for a child case, since demoting top-level case types can introduce unexpected complexity.

Follow these steps to use an existing case type for a child case:

1. Open the Case Type Explorer.
2. Hover over a parent case type name and click the **options** menu.
3. Select **Add a child case type**.
4. Click **Existing case type**.
5. Select a case type from the list.
6. Click **Submit**.

Creating a case during case processing

Create new cases during case processing to begin a new business process or a portion of the existing business process.

Creating another case during the case life cycle

Follow these steps to add and configure a process step to create a case during case processing:

1. In the **Life cycle** tab of the case designer, identify the process where you want to add a step for creating a case.
2. Click **+ Add step**. A pop-up opens to select the type of step to add.
3. Select **More > Utilities > Create Case(s)**.
4. Click this new step to configure the Create Case shape.
5. Indicate how to create the new case: as a top-level case, as a child case, or as multiple child cases.

Creating a top-level case

Create a top-level case when you want the new case to be independent of the current case. Processing on the current case can finish while the new case is still open.

a. Click **Create a case**.

b. In the **Case type** list, select the case type to create. Specify the name of the case type, or select **[Other]** to specify the case type using a parameter or property.

c. In the **Starting process** list, select a flow that creates the case.

d. Optional: In the **Property to store ID of case** field, enter the name of a single-value property that you can reference from the current case to open the new case.

Creating a child case

Create a child case when you want the current case to be dependent upon the new case. Processing on the current case cannot be completed until the child case is resolved.

a. Click **Create a child case**.

b. In the **Case type** list, select a case type that is a child of your current case type.

   If you do not know which case type to select at design time, you can select **[Other]** from the list to create a case based on the run-time value of a parameter or property.

c. In the **Starting process** list, select a flow that creates the child case.

Creating multiple child cases

Create multiple child cases when you want to create a child for each item in a list. Processing on the current case cannot be completed until each of the child cases are resolved.

a. Click **Create multiple child cases**.

b. Enter a page list property in the **For each item in list** field.

At run time, a child case is created for each entry that is found in the page list.

c. In the **Case type** list, select a case type that is a child of your current case type.

   If you do not know which case type to select at design time, you can select **[Other]** from the list to create a case based on the run-time value of a parameter or property.

   If no case type is selected, the class of the page list that is provided in the **For each item in list** field is used to create the child cases.

d. In the **Starting process** list, select a flow that creates each child case.

e. Optional: Enter a page name in the **Source page parameter name** field that you can reference in a data transform to copy information to each child case.

6. In the **Data transform** field, enter a data transform that sets initial property values for the case.

   If you are creating more than one child case, you can select the **Copy page data to new child case** check box instead.

7. Optional: In the **Audit note** field, press the **down arrow** key and select the name of a field value that is added to the history, or audit trail, of the case when the Change Stage shape is processed.

8. Click **Save**.

DATA MODEL DESIGN

Data elements in Pega applications

Introduction to Data Elements in Pega Applications

Pega 7 applications allow users to create, process, and resolve cases. The applications collect data that is important to the case. Based on the data collected, decisions on how to best process and resolve the case are made.

For example, if you want to create a case to process a change of address for a customer, you need data. The data includes the identity of the customer and the new address.

The fundamental unit of the Pega data model — the element that stores the data — is called a property.

After this lesson, you should be able to:

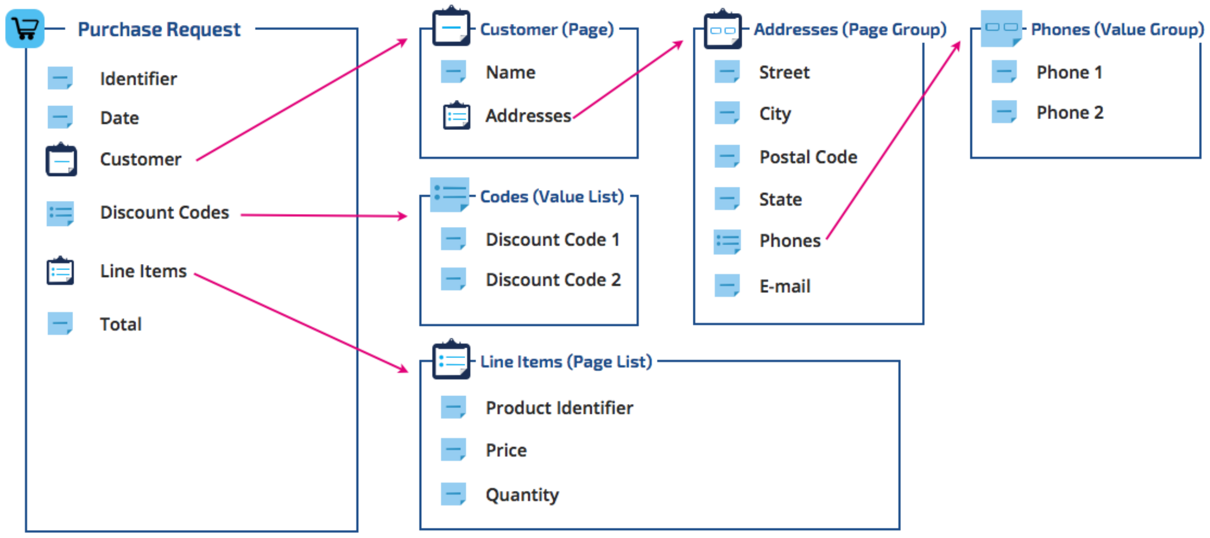
- Explain the relationship between data and a case.
- Describe the role of a data object.
- Explain how data elements relate to an object.
- Describe the relationship between properties and data objects and data elements.
- Define the components of a data model.

Data elements in Pega applications

Data in Pega applications

All applications collect data to use for case processing. Decisions on how to best process and resolve cases are made based on the data collected. If you want to create a purchase request case, the data includes, for example, the customer and line items.

A case type's **data model** defines the data structure for the case. The following table provides the data model for a purchase request.



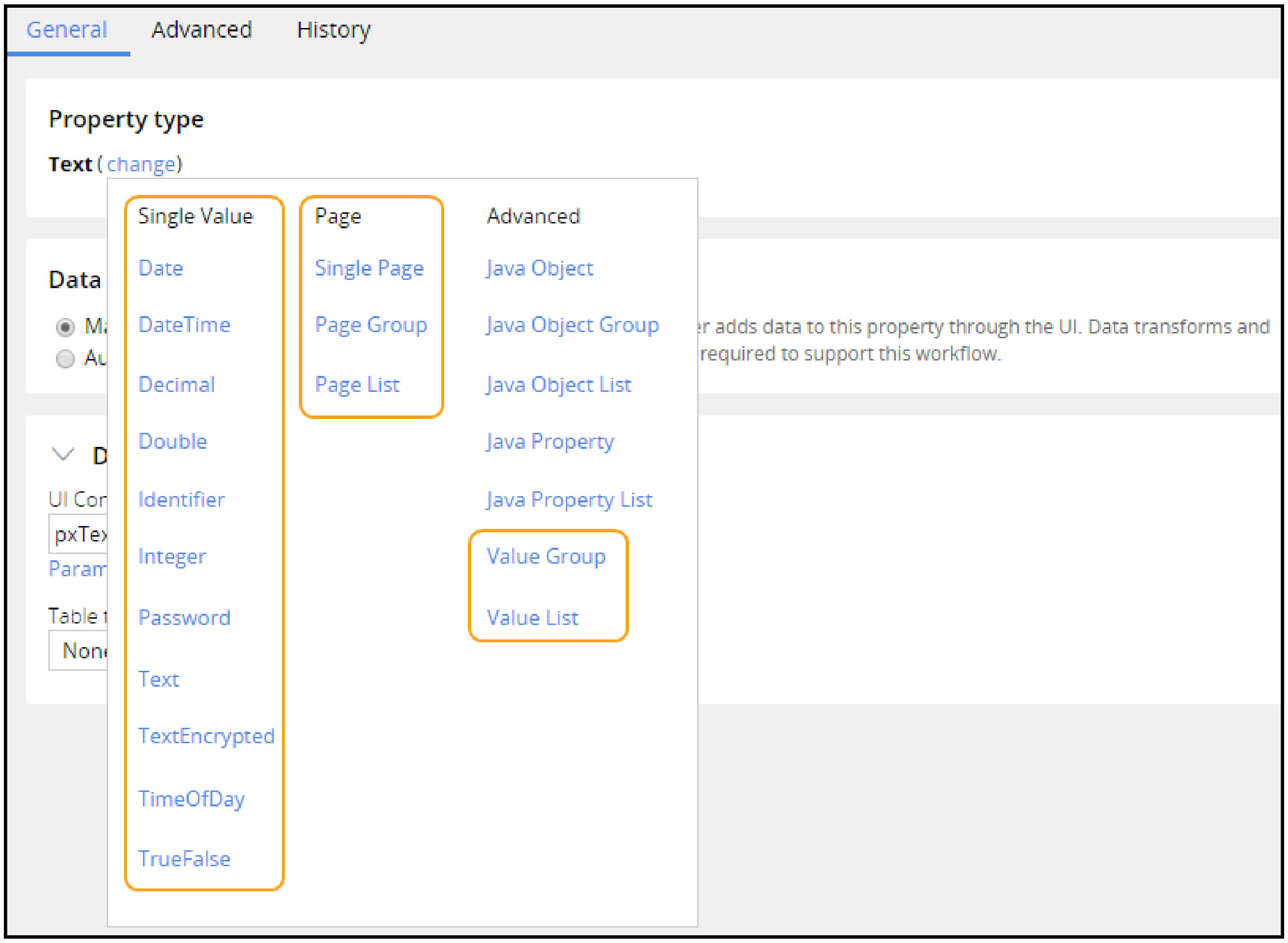
A data model is built from single value elements or collections of related single value elements. A collection of related elements is called a **data object**.

A purchase request case has a unique identifier, date, customer, list of line items, and total. The unique identifier, date, and total are single value data elements. The customer and line item elements consist of more than one related property. Therefore, the customer and line item elements are data objects.

There is a one-to-one relationship between purchase request and customer, and a one-to-many relationship to line items. The customer has a name and lists of phone numbers, addresses, and discounts. A name is a single value element and there is a one-to-many relationship to the phone number, address, and discount code elements.

Properties

In Pega 7, data elements are called **properties** or **fields**. Property and field are different names for the same thing. Properties can be either single value with no intended correlation with any other value, or a collection of related values. This distinction is explained by the mode of a property. System architects typically work with two types of property modes: value modes and page modes. **Value modes** describe a single piece of information such as a total. **Page modes** describe a data object such as a customer. The screenshot highlights the value and page mode property types.



Value mode properties

Use value mode for properties with no correlation to other properties. For example, the identifier and date in the purchase request are value mode properties. There are three value mode properties available: single value, value list, and value group.

- A property of mode **single value** - also known as a single value property - stores text, numbers, dates, Boolean values, and amounts.
- A **value list** acts as a container for an ordered list of text values. The discount codes property is an example of a value list. Each code is a single piece of information, but a clear relationship exists between the codes.

- A **value group** acts as a container for an unordered list of text values. The customer's phone numbers are defined as a value group identifying the contextual meaning of each number: home, work, or mobile.

When you create a value property, you can assign it to one of 10 different Property types. This identifies the type of information the property stores. By assigning a type to a property, you ensure that users provide valid information. For example, users provide a number for an age, and a date for a date of birth.

The table below provides a list of property types and the information each type stores.

Property type	Stores	Example
Text	Any text	Steve
Identifier	Text strings that do not contain double quotation marks (“”), tabs, carriage returns, or line breaks	XYZ
Password	Encrypted graphical characters	Password
Encrypted text	Similar to the password type, but can be decrypted for display	Password
Date	Calendar date in the format YYYYMMDD	20131202
TimeOfDay	Local time in the format HHMMSS	052709
DateTime	UTC (Coordinated Universal Time) value normalized to Greenwich Mean Time (GMT)	20131202T052709
Integer	Positive and negative whole numbers, and zero	4
Decimal	Non-whole numbers	23,55
TrueFalse	Boolean value	True

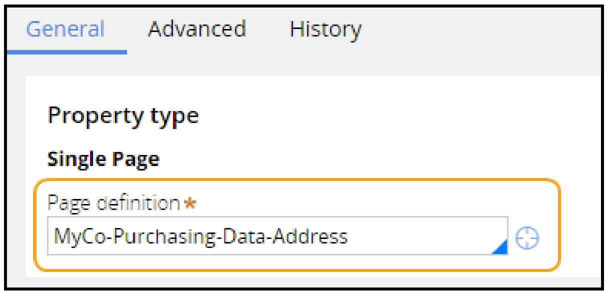
Page mode properties

If you need to establish a contextual relationship between single value properties, you can use one of the three **page-mode properties**: pages, page lists, and page groups.

Page mode properties are organized similar to value mode properties.

- A **page** is a single entity. The customer is an example of a page property.
- A **page list** is a numerically ordered list. The line items that make up the purchase request is an example of a page list.
- A **page group** is a semantically ordered list. The address property is an example of a page group.

The page mode properties require you to specify a definition, or a data type, that defines the structure of the page property.



Property type

Single Page

Page definition*

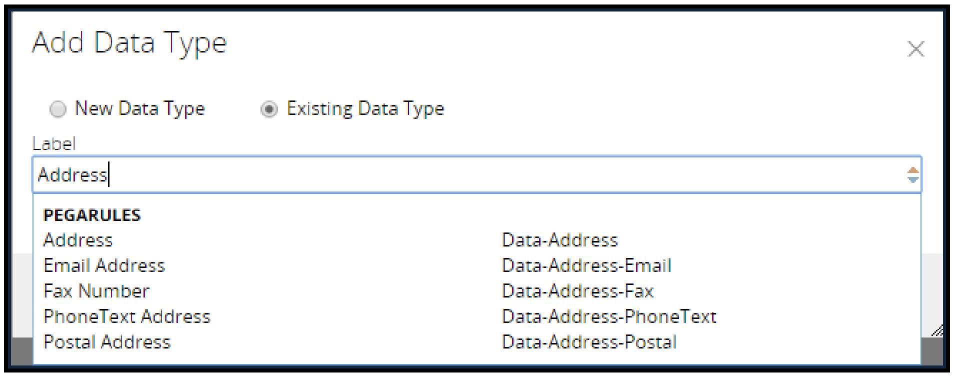
MyCo-Purchasing-Data-Address

How to manage properties

Pega provides several tools that help manage properties. The tools provide easy-to-use interfaces that add, update, and remove classes and properties. This section looks at the Data Explorer and the Data Model tab as well as the property rule form.

The Data Explorer

Use the Data Explorer to add or remove data types. Always check if a suitable data type is available before creating a new one. Pega comes with many standard classes you can use directly in your application. Select an existing data type and specify the data type you want to use.


Add Data Type

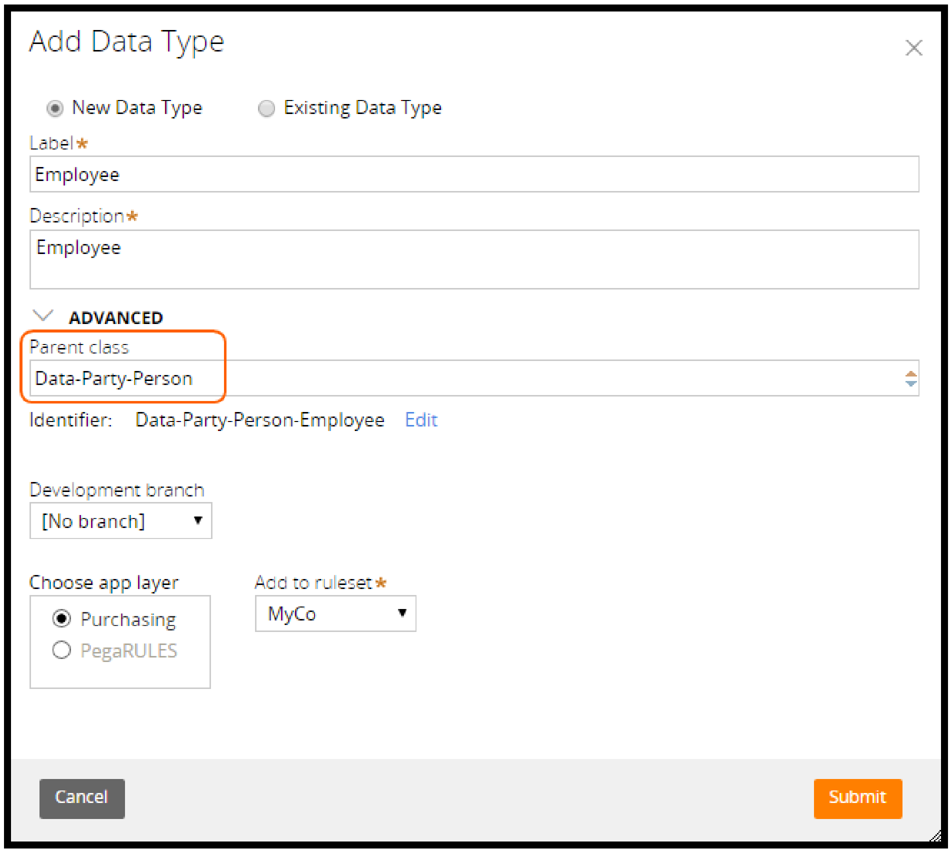
New Data Type [✓ Existing Data Type]

Label: Address

PEGARULES

Address	Data-Address
Email Address	Data-Address-Email
Fax Number	Data-Address-Fax
PhoneText Address	Data-Address-PhoneText
Postal Address	Data-Address-Postal

You can extend an existing data type if it only partly meets your needs. For example, you might want to create an employee data type based on the Party-Person data type. Select a new data type and specify the data type you want to extend as the parent in the advanced settings. You can use all properties defined in the parent in addition to the ones you create in your new data type.


Add Data Type

(● New Data Type) ○ Existing Data Type

Label* Employee

Description* Employee

ADVANCED

Parent class: Data-Party-Person

Identifier: Data-Party-Person-Employee Edit

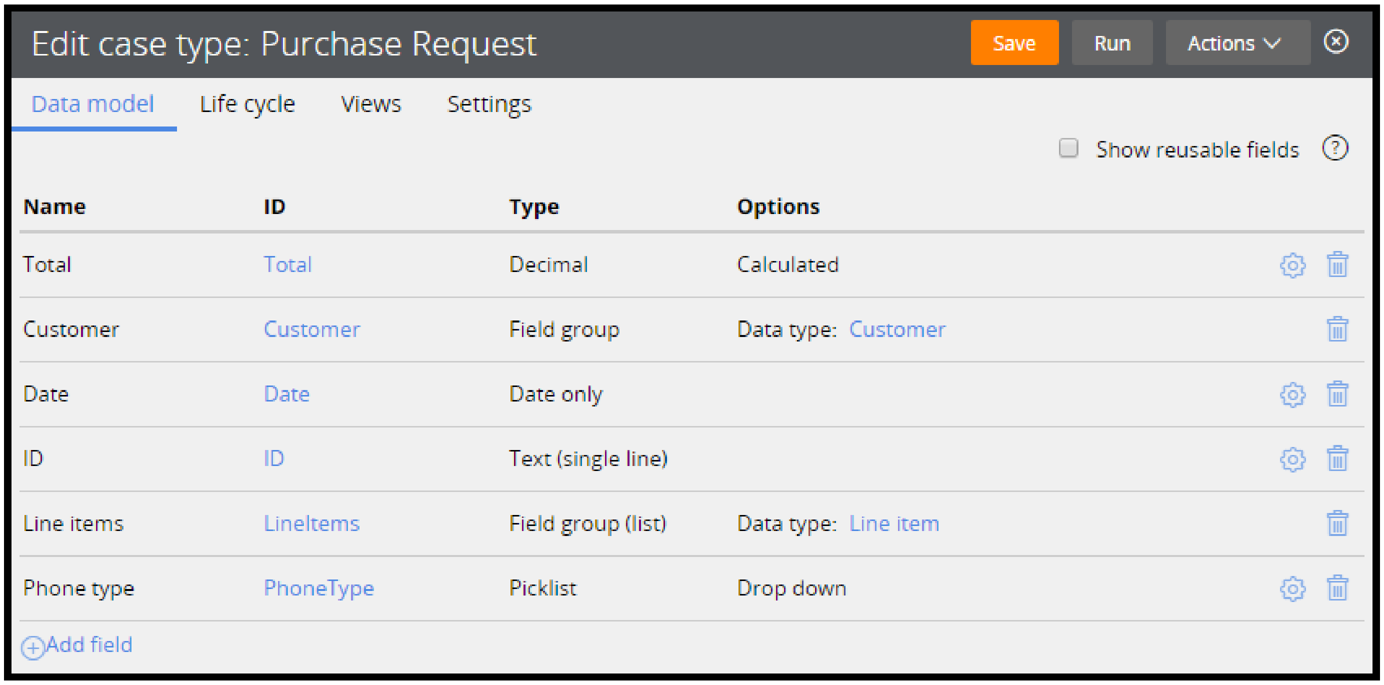
Development branch: [No branch]

Choose app layer: [✓ Purchasing] PegaRULES

Add to ruleset* MyCo

The Data model tab

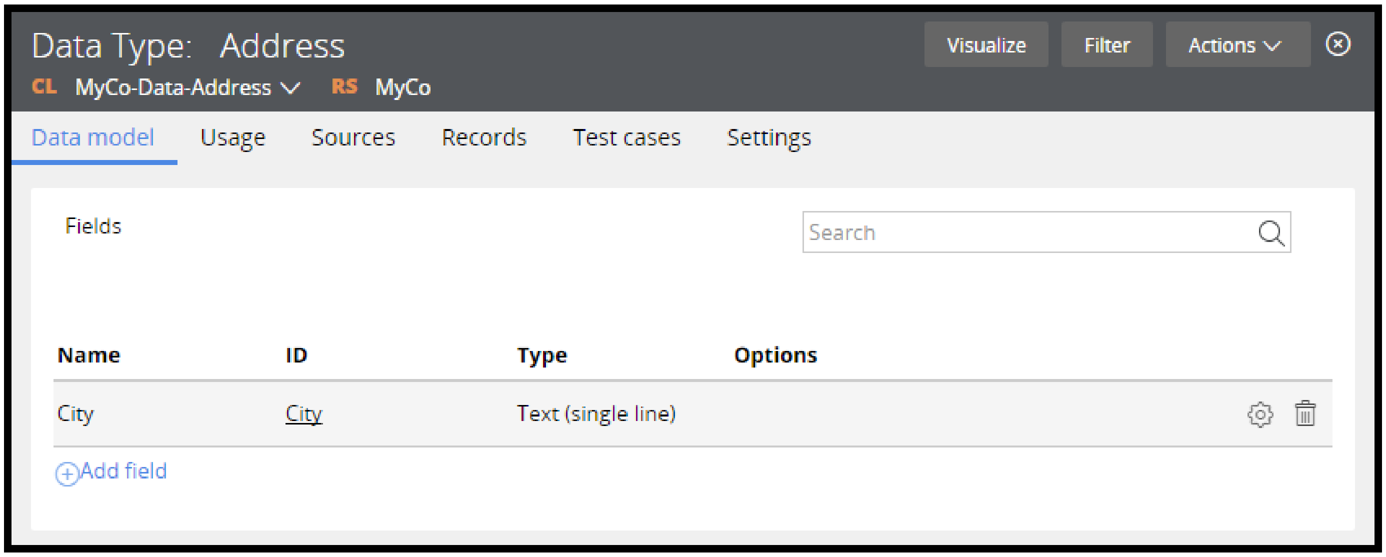
You can use the Data model tab in the Case Designer to add or remove properties from your case type. Properties are called fields in the Data model tab. Select **Show reusable fields** to display all fields inherited and available in the case type.



Edit case type: Purchase Request

Name	ID	Type	Options
Total	Total	Decimal	Calculated
Customer	Customer	Field group	Data type: Customer
Date	Date	Date only	
ID	ID	Text (single line)	
Line items	LineItems	Field group (list)	Data type: Line item
Phone type	PhoneType	Picklist	Drop down

The Data model tab for a data type looks very similar.

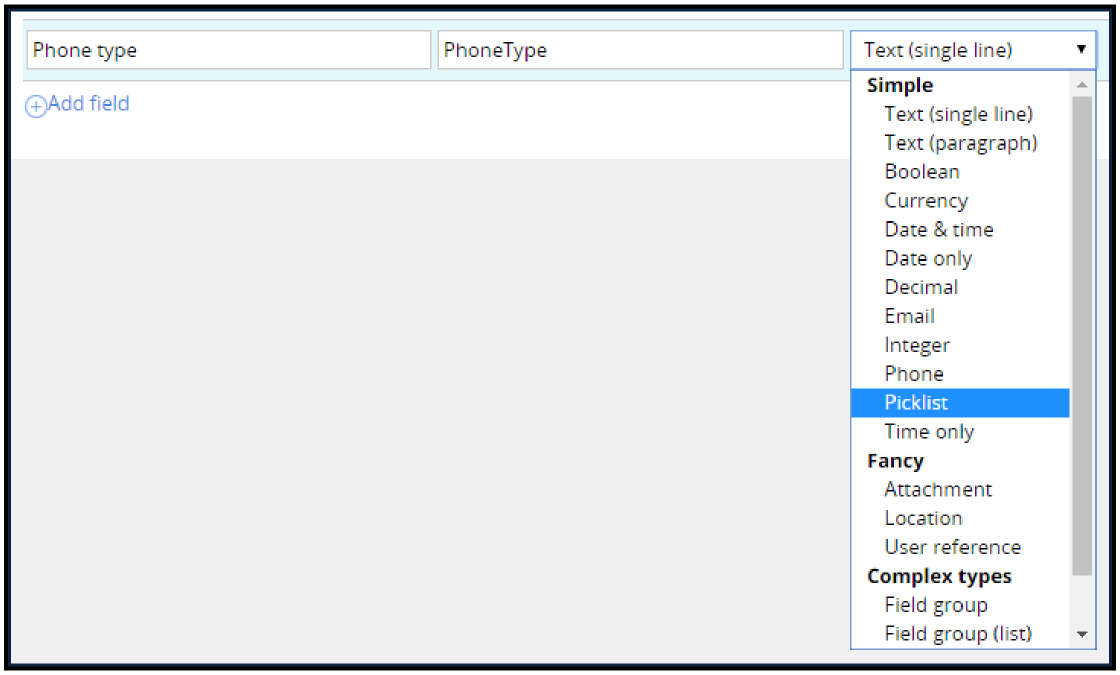


Data Type: Address
CL MyCo-Data-Address RS MyCo

Name	ID	Type	Options
City	City	Text (single line)	

Selecting the field type

When creating a new field, you need to specify a type. The options in the list pair the field with a control in the user interface. The type options are divided into three categories: simple, fancy, and complex.



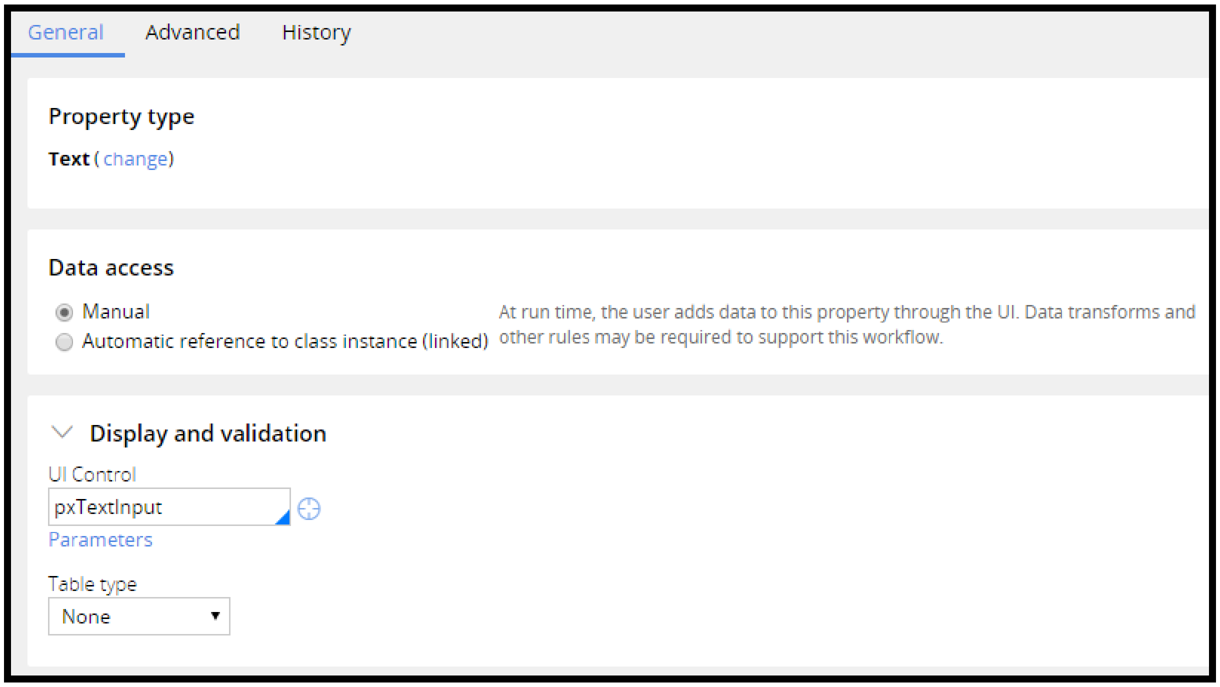
The simple types are similar to the property types defined on the property itself. Use a picklist if you need to display a static list of options to the user. For example, if you want to capture a phone number, you might want to specify a list of types, such as home, work, and mobile.

The fancy types allow you to provide the capability to upload an attachment, show a location on a map, or reference a user on the system.

Use the complex types to define page and page list properties. A field group is a page and a field group (list) is a page list.

The Property rule form

The property rule form contains the property definition. Because a property definition is a rule, it shares the benefits of versioning, inheritance, and access control that the Pega 7 Platform provides to all rules.



Property type

Text (change)

Data access

● Manual
○ Automatic reference to class instance (linked)

At run time, the user adds data to this property through the UI. Data transforms and other rules may be required to support this workflow.

Display and validation

UI Control: pxTextInput

Parameters

Table type: None

The property has one of 11 types.

Note: The property type cannot be changed after the property has been saved.

Use the Data Access section to configure automatic data access and persistence settings. Use Manual if you are explicitly setting the value (for example, in a user interface). Other options depends on the property type selected and are not covered in this lesson.

The Display and validation section allows you to define how the property should appear on the screen by specifying a UI control. You also have the option to specify a table with valid values for the property.

Pega comes with a set of standard property rules. The standard properties have names that start with px, py, or pz. You cannot create new properties starting with px, py, or pz.

The table below provides a list of the prefixes for standard rules.

Prefix	Meaning
px	Identifies special properties — your application can read but not write to these properties.
py	You can use these properties in your application.
pz	Supports internal system processing — the meaning of values may change with new product releases. Your application can read but not write to these properties.

How to reference a property

You have learned about two property modes: value and page. Value mode properties store single strings of data such as text, numbers, or dates. Page mode properties act as a container for value mode properties. You refer to a property in Pega 7 by prefixing the property name with a period (or dot, ".").

- To refer to a single value property named OrderDate, type .OrderDate.
- To refer to an entry in a value group property, such as the mobile phone number, type .Phone (Mobile), where Mobile is the group subscript.
- To refer to the first entry in a value list property, such as one of the discount codes, type .DiscountCode(1), where 1 is the list index.

Page mode properties are similar.

- To refer to a page that contains customer information, type .Customer.
- To refer to an entry in a page group property, such as the work address, type .Address(Work).
- To refer to the third page of a page list that contains purchase request line items, type .LineItems(3).

To refer to a specific property on the page, use the name of the page as a prefix for the property name. By doing this, you establish an important piece of information about that property — its context. The context of a page — by itself or as part of a page list or page group — acts as a container for the properties it contains. If you want the city in the work address, specify .Address(Work).City.

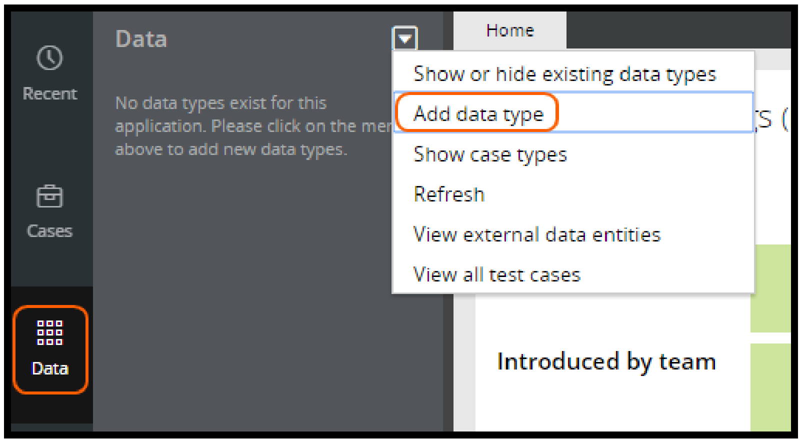
Defining properties

You can use the Data Explorer to create a data type and the Data Model tab to manage properties. You will also look at how to create a static list of data entry options.

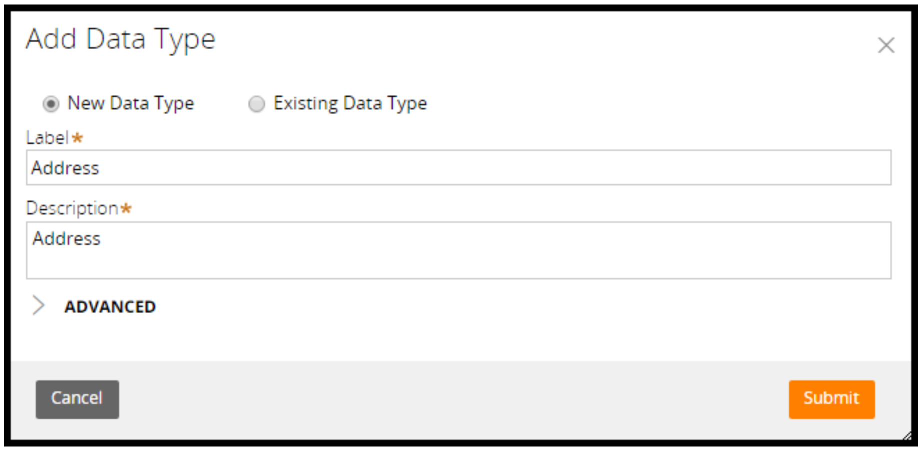
Creating a data type

Follow these steps to create data types from the Data Explorer:

1. Select **Data** in the left pane to open the Data Explorer.
2. Select **Add data type**.


Data
Recent
Cases
Data
No data types exist for this application. Please click on the men above to add new data types.
Home
Show or hide existing data types
Add data type
Show case types
Refresh
View external data entities
View all test cases
Introduced by team

3. Select **New Data Type** if you want to create a new data type, or select **Existing Data Type** to include an existing one.
4. Provide a label and description.



Add Data Type

[✓ New Data Type] Existing Data Type

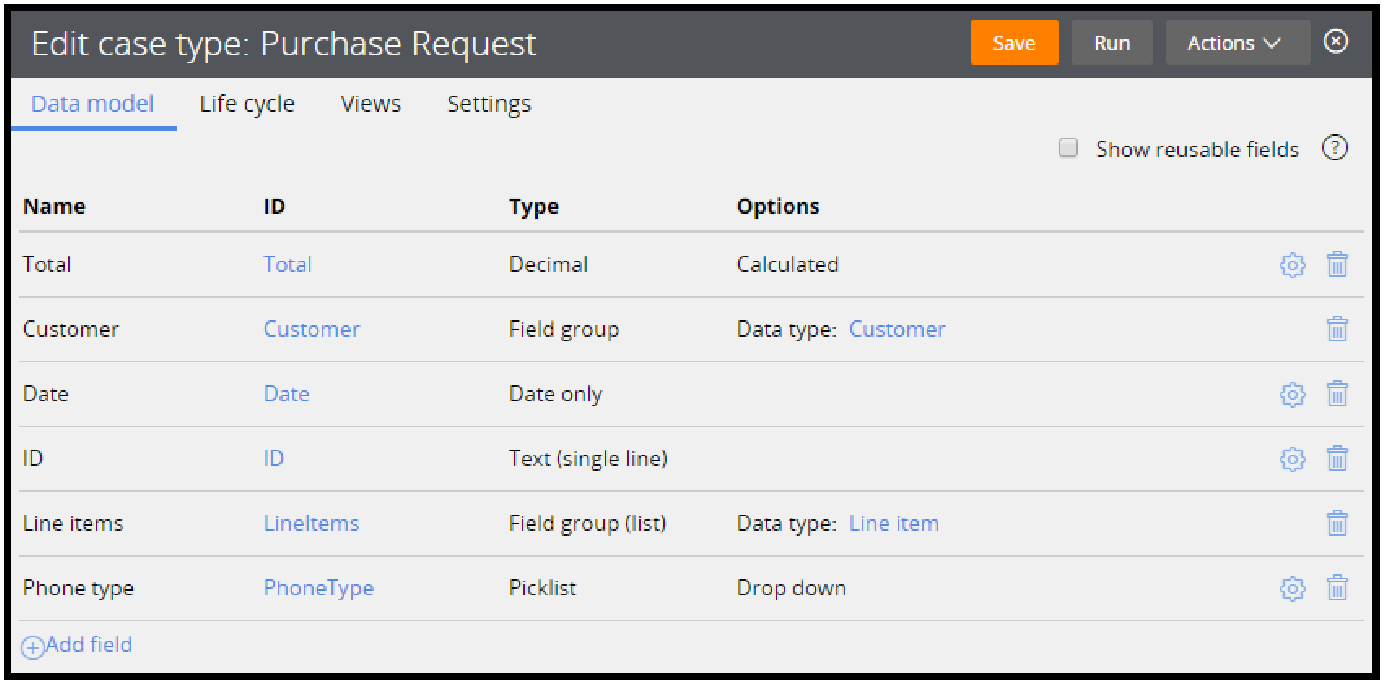
Label* Address

Description* Address

5. Click **Submit** to create the data type.

Managing properties in a case or data types

Use the Data Model tab in the Case Designer to add or remove properties from your case type. Properties are called fields in the Data model tab.



Edit case type: Purchase Request

Save | Run | Actions

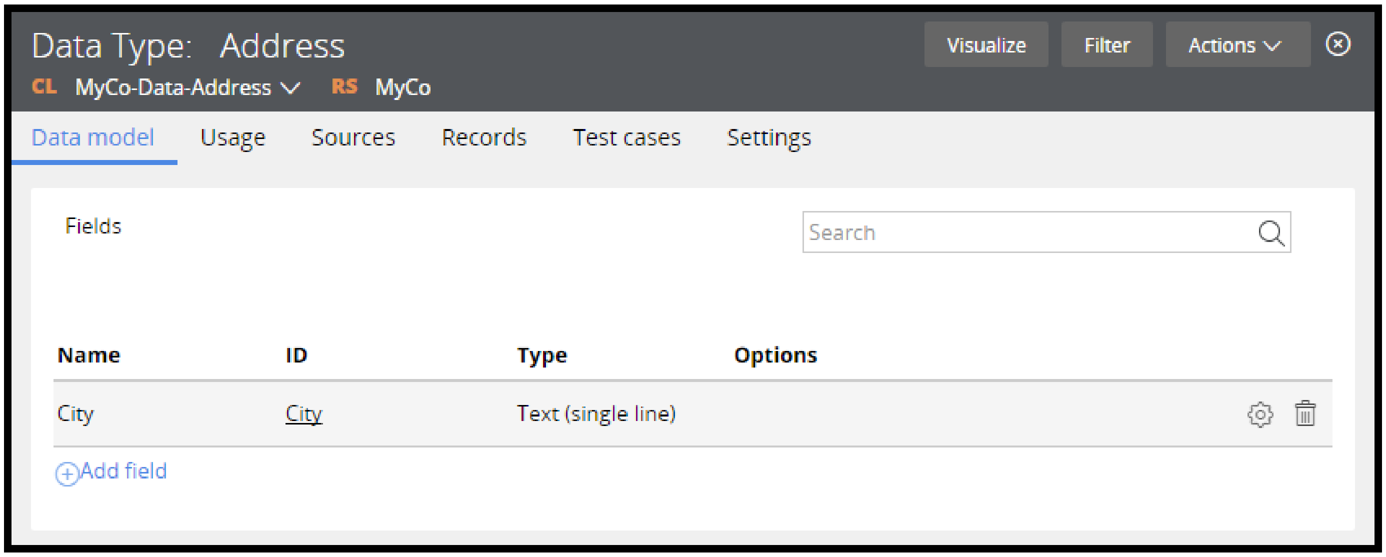
Data model | Life cycle | Views | Settings

Show reusable fields

Name	ID	Type	Options
Total	Total	Decimal	Calculated
Customer	Customer	Field group	Data type: Customer
Date	Date	Date only	
ID	ID	Text (single line)	
Line items	LineItems	Field group (list)	Data type: Line item
Phone type	PhoneType	Picklist	Drop down

Add field

The Data Model tab for a data type looks very similar.



Data Type: Address

CL MyCo-Data-Address RS MyCo

Visualize | Filter | Actions

Data model | Usage | Sources | Records | Test cases | Settings

Fields

Search

Name	ID	Type	Options
City	City	Text (single line)	

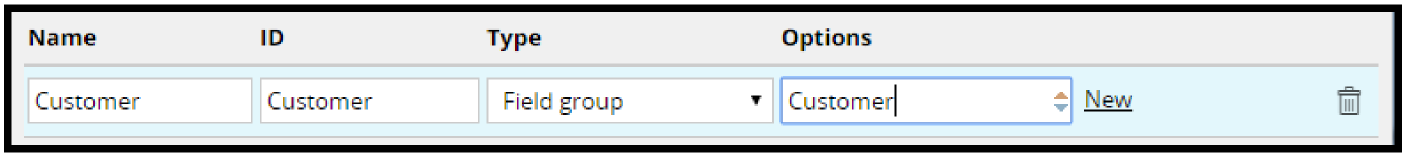
Add field

Adding a field to a case or data type

Follow these steps to add a field:

1. Open the **Data model** tab in the Cases Explorer.
2. Click the **Add field** link.
3. Specify a name for the field (property).

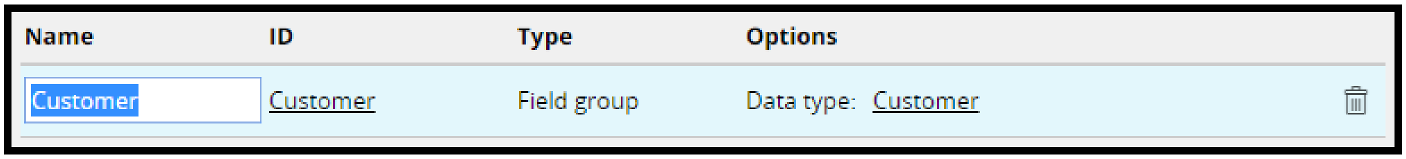
4. The ID field is automatically populated by the system. You can choose to edit the ID field.
5. Select the type.
6. If you select a field group or field group list, you need to provide a data type in the options field.



Name	ID	Type	Options
Customer	Customer	Field group	Customer New

Updating a field in the case or data type

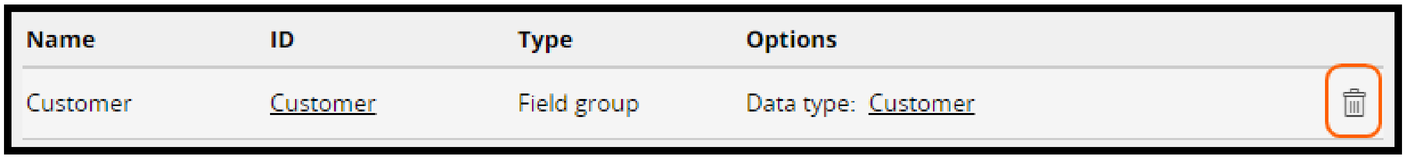
Click the row to update the name of a field.



Name	ID	Type	Options
Customer	Customer	Field group	Data type: Customer

Remove a field from the case or data type

Click the **trash can** icon to remove a field.



Name	ID	Type	Options
Customer	Customer	Field group	Data type: Customer

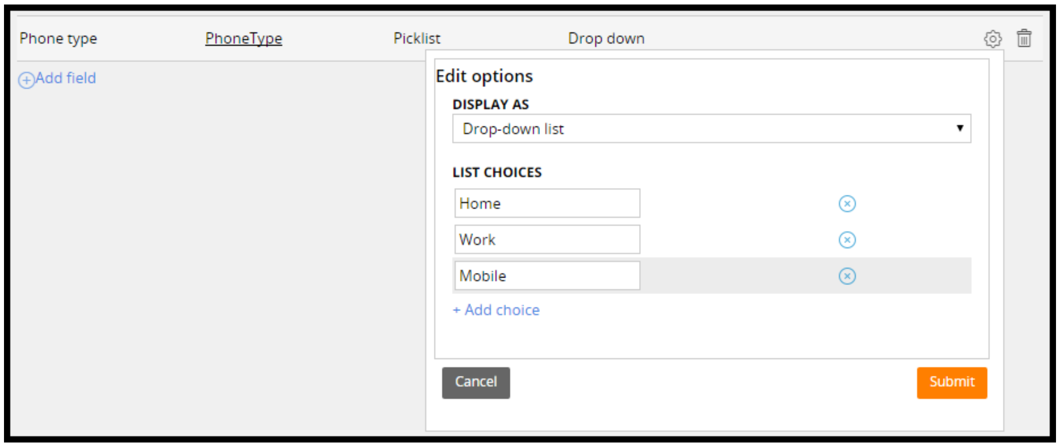
Property rules are automatically added, removed, or updated as you use the Data Model tab.

Define a static list of data entry options

You can define a static list of data entry options for a field. For example, to capture a phone number, you can specify a list of types such as home, work, and mobile.

Follow these steps to create a picklist:

1. Open the **Data model** tab in the Cases Explorer.
2. Click **Add field** and enter a name and ID.
3. Select the Picklist type.
4. Click the **gear** icon to configure the picklist.


Phone type PhoneType Picklist Drop down

Edit options

DISPLAY AS: Drop-down list

LIST CHOICES

- Home
- Work
- Mobile

5. Select whether you want the list to display as a drop-down or as radio buttons.
6. Enter the list options.
7. The property is now ready to be used in the application.

Setting property values automatically

Introduction to Setting Property Values Automatically

As you process a case, you may need to copy or manipulate data. For example, you collect an individual’s first name and last name, but want to combine them into a full name. In other situations, you might want to set default values for fields, or add two numbers together.

After this lesson, you should be able to:

- Explain the use of data transforms in an application.
- Identify situations in which to set property values automatically.
- Set initial property values using the pyDefault data transform.
- Explain how data transform superclassing works.

Data transforms

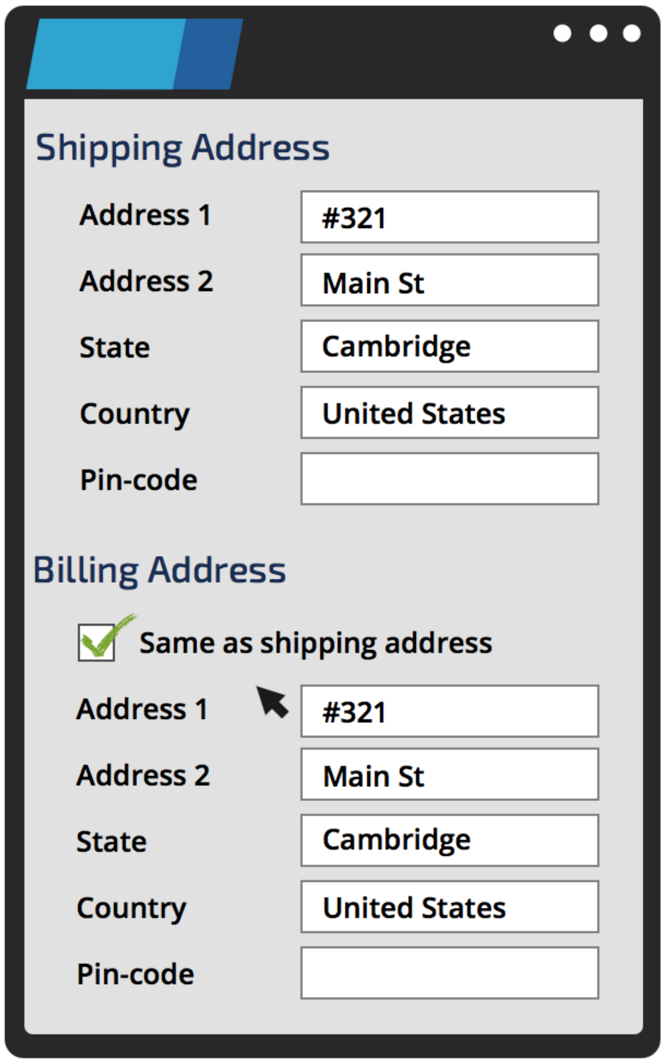
When you create and process a case, you need data. You collect, process, act upon, and present that data back to the user. Sometimes, you need to copy data from one place to another. Other times, your data is not in the form you require, so you need to find a way to manipulate that data into an acceptable form.

In a purchasing application, for example, items are added to a cart and the checkout process begins. The customer provides a shipping address and credit card information, and is prompted to provide a billing address.

The shipping address might be the customer's home address — the billing address and shipping address are likely to be the same. Reusing rather than having to reenter the shipping address is helpful and more efficient. Similarly, you might collect an individual’s first name and last name, but need to combine the two into a full name for credit card processing.

One option for copying and manipulating data is the **data transform**. The purpose of a data transform is self-explanatory: it transforms data in the application. This example uses a data transform to copy the shipping address to another page — in this case, the billing address — and to copy the first and last name properties into a single property full name.

You can use data transforms in several ways. For example, you can call a data transform from a flow action rule or from a connector. Also, you can use a special data transform rule — pyDefault — to initialize property values when creating a case. Data transforms can be used to iterate over page lists or page groups, and copy entire pages at a time.



Shipping Address

Address 1	#321
Address 2	Main St
State	Cambridge
Country	United States
Pin-code	

Billing Address

✓ Same as shipping address

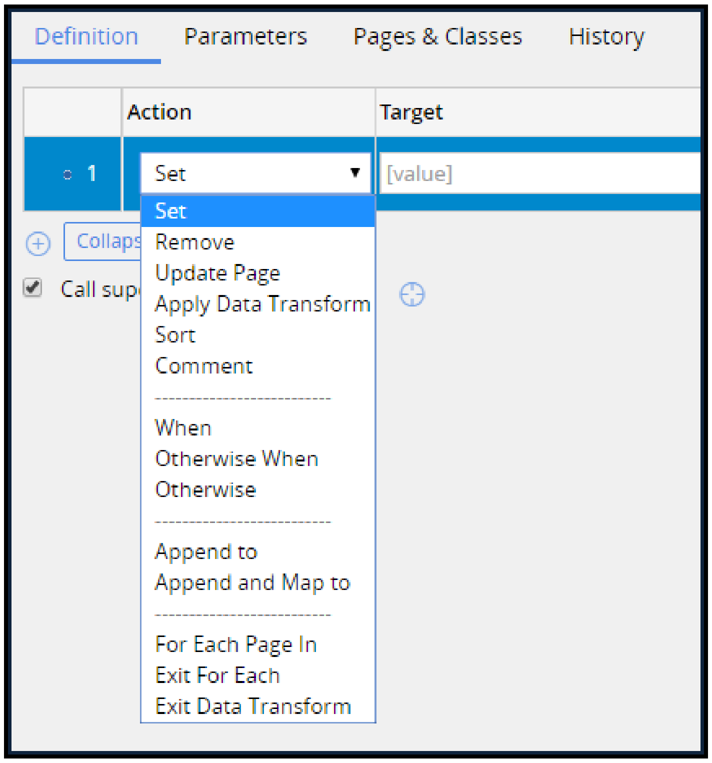
Address 1	#321
Address 2	Main St
State	Cambridge
Country	United States
Pin-code	

How to set values with data transforms

Use a data transform to define how to take source data values — data that is in one format and class — and turn those values into data of another target format and class. In general, data transformation involves mapping data from a source to a target as well as performing transformations on that data required to achieve the intended mapped results.

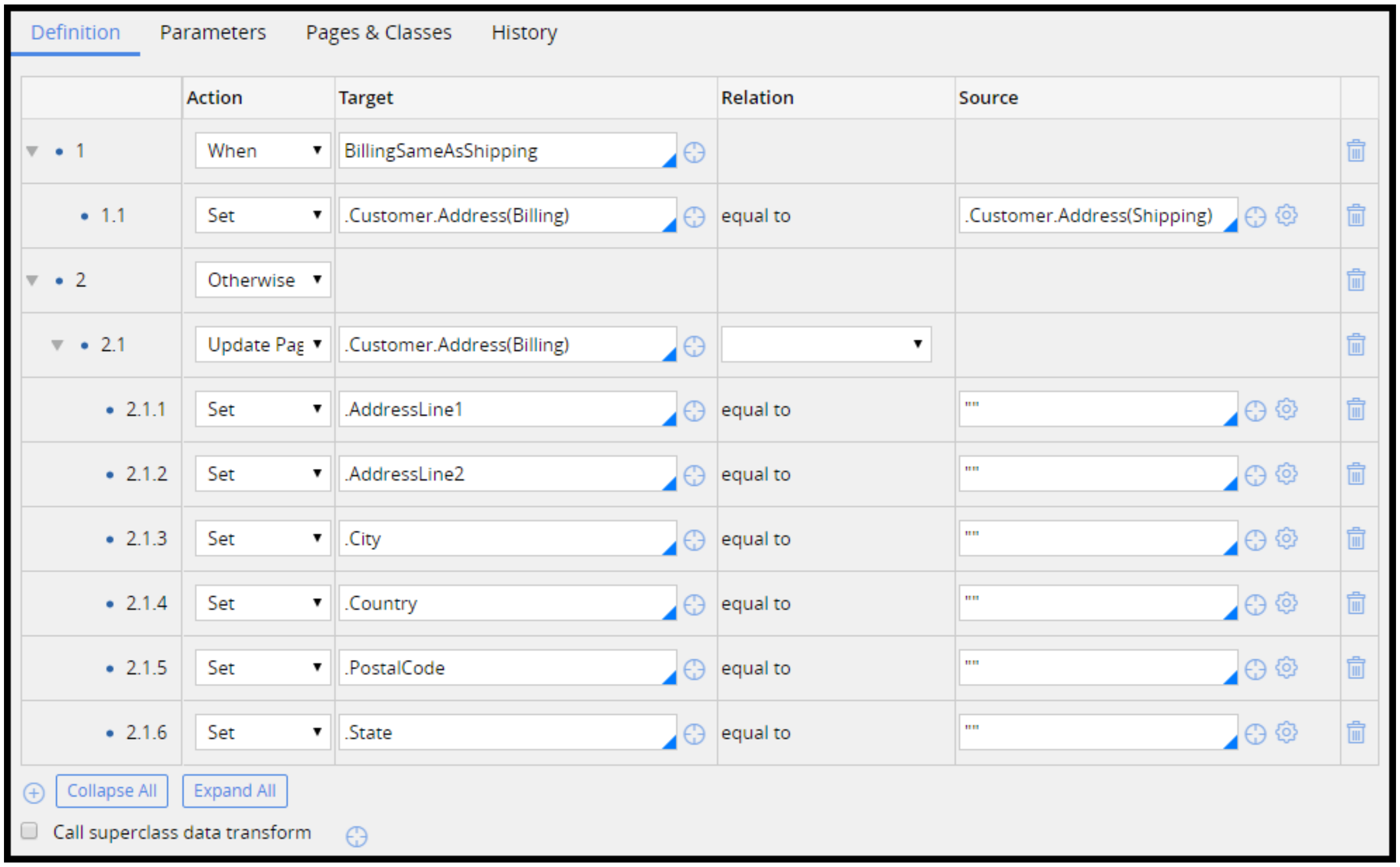
The first thing you do when configuring a data transform is specify an action. Actions are the individual operations that are specified in each row on the Definition tab of a data transform. The system invokes the actions at run time. Most actions do some kind of data manipulation. Other actions perform conditional processing and iterate through page lists and page groups. Consult the Developer Help in your system for additional details on actions.

Select the appropriate action for what you want to do.



Next, enter the **Target**, **Relation** and **Source**. Depending on the action selected, the target field has a different meaning. For the Set and Update Page actions, the Target field identifies a property or page reference, and the Source column provides an expression that results in a value or values. For the when action, a when condition needs to be specified.

In the data transform below, the first step checks if the billing address is the same as the shipping address. If the two addresses are the same, the shipping address is copied to the billing address. Otherwise, the billing address is set to empty values.



	Action	Target	Relation	Source
1	When	BillingSameAsShipping		
1.1	Set	.Customer.Address(Billing)	equal to	.Customer.Address(Shipping)
2	Otherwise			
2.1	Update Pag	.Customer.Address(Billing)		
2.1.1	Set	.AddressLine1	equal to	""
2.1.2	Set	.AddressLine2	equal to	""
2.1.3	Set	.City	equal to	""
2.1.4	Set	.Country	equal to	""
2.1.5	Set	.PostalCode	equal to	""
2.1.6	Set	.State	equal to	""

☐ Call superclass data transform

If you want to refer to a property on a specific page, use the name of the page as a prefix for the property name. For example, the shipping address on the customer page becomes .Customer.Address (Shipping). Also, the type of the third asset on an asset list becomes .Asset(3).Type.

The most important thing to remember when using a data transform is to establish the context correctly when reading and writing property values.

The pyDefault data transform

Often when you create a case, you want to set default values for some properties. For example, in an insurance claims application, you might want to set the date of loss to today's date. In other situations, you might want to use data from the operator record — such as the organizational structure — to initiate properties with default values.

The Pega 7 Platform invokes a data transform called pyDefault whenever a new case is created. The pyDefault data transform allows you to set properties as the case is created. For example, you can use the pyDefault data transform to default the date of loss to today's date in a claim case. The pyDefault has no specific characteristics and the name is not reserved. You can create data transforms called pyDefault in any class or ruleset.

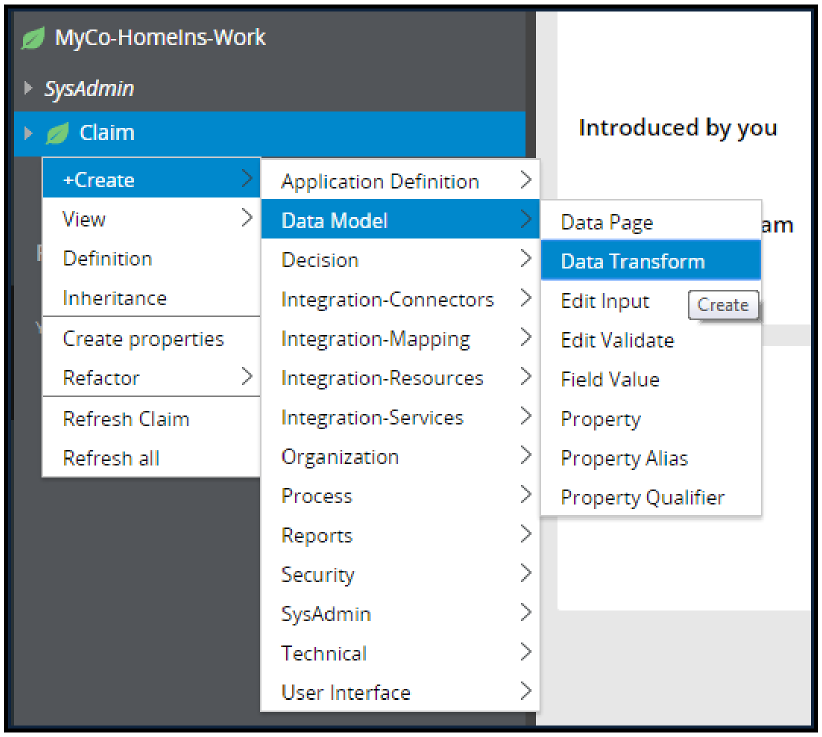
The Pega 7 Platform comes with standard pyDefault data transforms in the work classes that case types inherit from. If you do not create a pyDefault for your case, the standard pyDefault in the inheritance path is invoked.

Setting property values using the pyDefault data transform

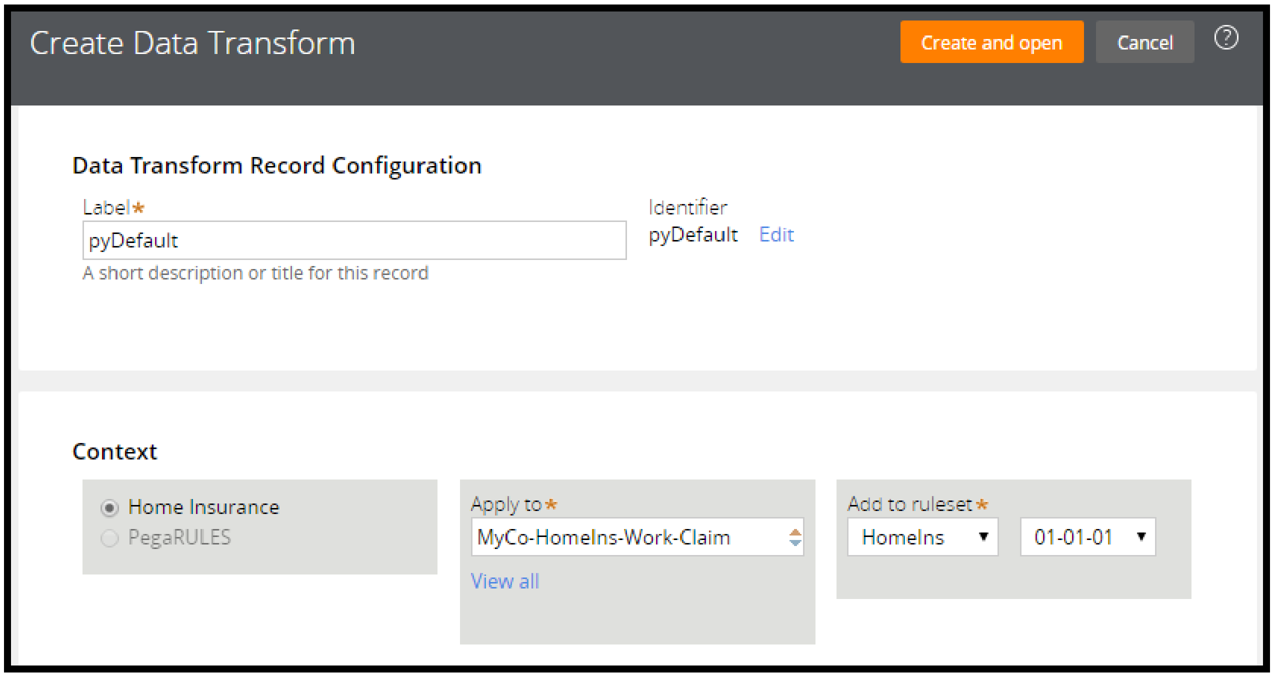
Create a pyDefault data transform in your case type class to set properties when the case is created. This example shows how to add an item to the items of loss list and set the date of loss to today's date in an insurance claim.

Create a pyDefault data transform

1. In the Application Explorer, right-click the case type and select **Create > Data Model > Data Transform** in the case type class.



2. Enter pyDefault as the **Label**, and then select **Context**.


Create Data Transform

Data Transform Record Configuration

Label* pyDefault
A short description or title for this record

Identifier pyDefault Edit

Context

(●) Home Insurance
( ) PegaRULES

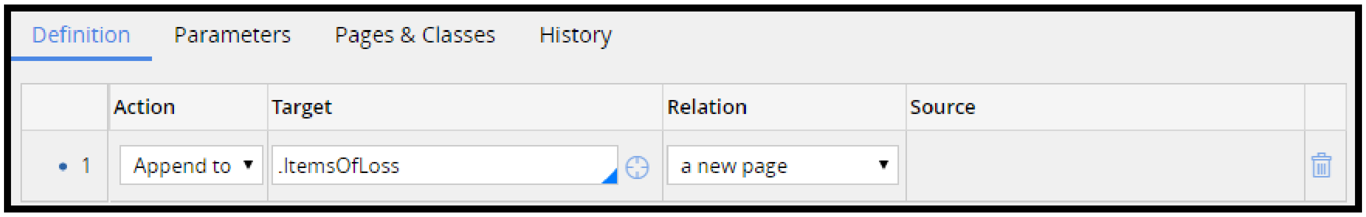
Apply to* MyCo-HomeIns-Work-Claim
View all

Add to ruleset* HomeIns 01-01-01

3. Click **Create and open** to create the data transform.

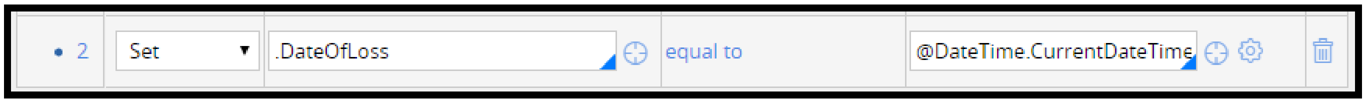
Set Property values with the pyDefault data transform

1. Add an item to the list using the **Append to** action.



	Action	Target	Relation	Source
• 1	Append to	.ItemsOfLoss	a new page	

2. Use the **Set** action to set the date of loss. Use the **gear** icon and select the **CurrentDateTime** function to set to today's date.



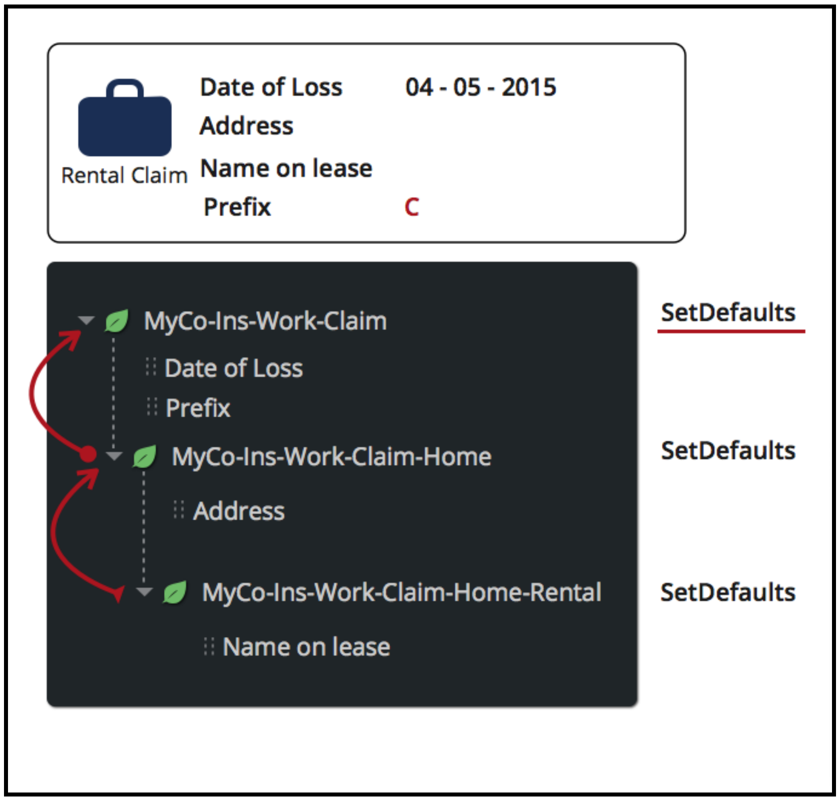
• 2	Set	.DateOfLoss	equal to	@DateTime.CurrentDateTime

Data transforms and superclassing

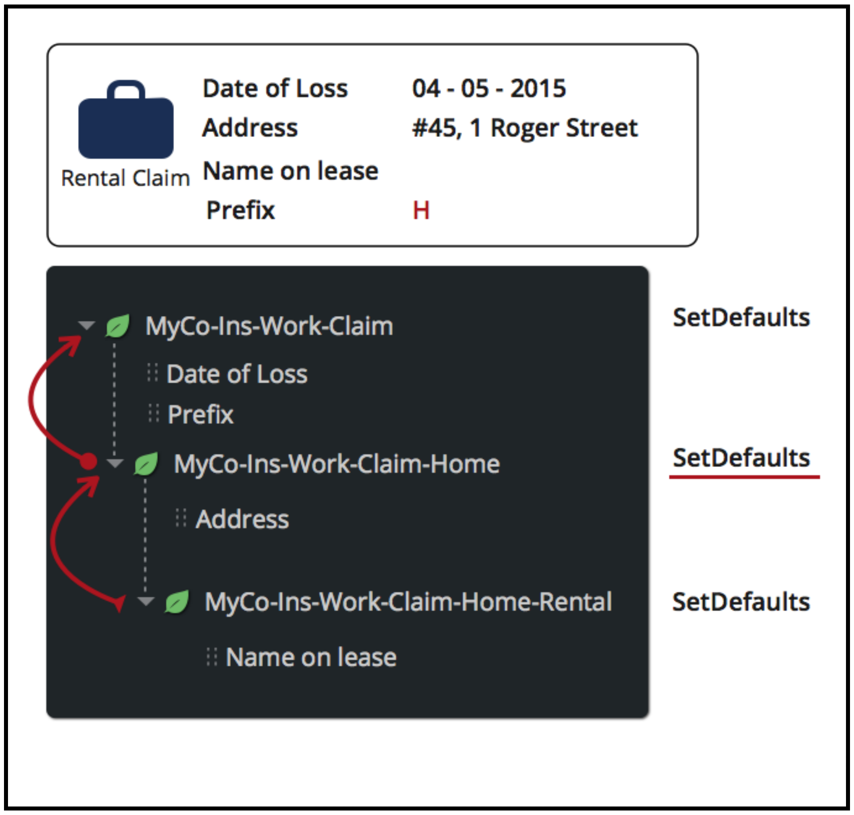
You can combine several data transforms using the superclass feature to set values at multiple levels of the class hierarchy. For example, you can have a class Claim with a subclass Home. The subclass Home in turn has a subclass Rental with data transforms at each level that sets default values. You can set up your data transforms so that common default values are set in the claim class and specific values are set in the subclasses. Taking advantage of this feature improves the maintainability of data transforms.

Here the date of loss and prefix are in the Claim class, address is set in the Home class, and name on the lease in the Rental class.

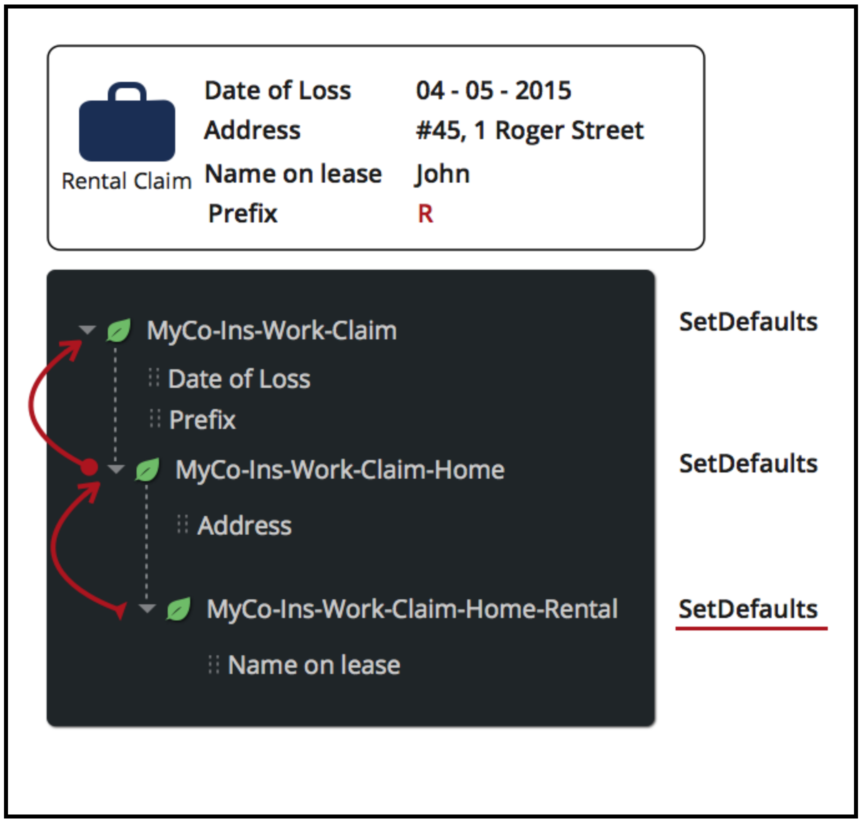
The system first identifies parents until the highest parent is reached. In this case, the highest parent is the Claim class. The system locates the data transforms with the same name in the parent class and invokes it.



The system then goes down to the second highest parent and locates the data transforms with the same name and invokes it. Note that the prefix is overwritten.



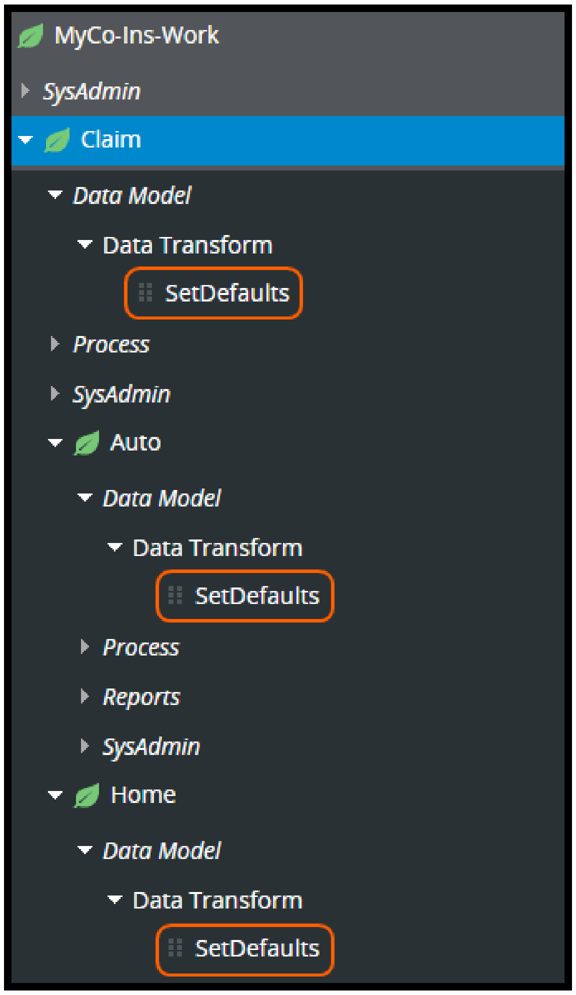
Finally, the data transform in the Rental class is invoked. Note that the prefix is overwritten again.



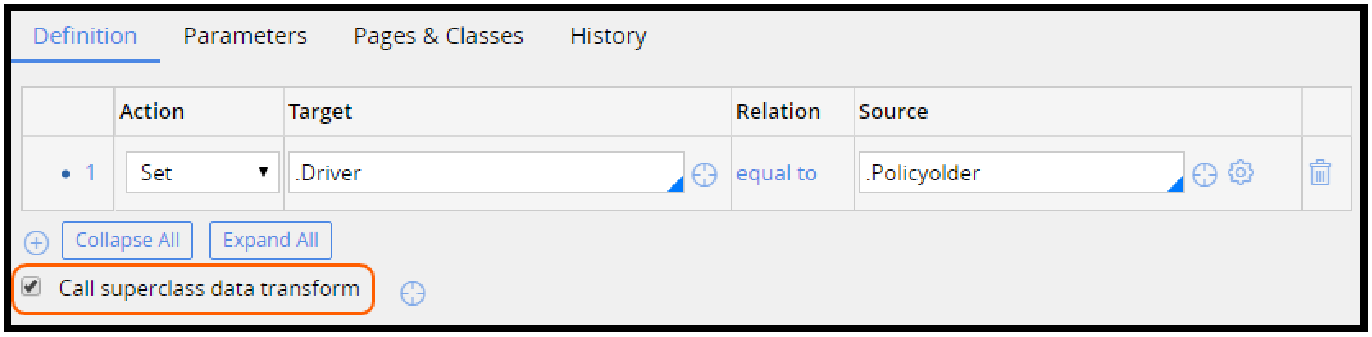
How to configure superclassing for data transforms

You can use the superclass feature in a data transform to set values at different levels in the class hierarchy. For example, if you have a claim class with two subclasses, home and auto, you can set values common to the subclasses in the superclass and specific values in the subclasses.

Use the superclass feature by creating a data transform with the same name at each level and selecting the **Call super data transform** options. In this example, the parent claim has a data transform called SetDefaults. This data transform sets values common to the subclasses. The subclasses have a data transform with the same name. This data transform sets any values specific to the subclasses. If properties are specified in both the super and child classes, the data transform in the subclass overwrites anything already set by the data transform in the superclass.



You must select the **Call superclass data transform** option to cause the system to invoke data transforms with the same name in any of its parent classes before the data transform get invoked itself.



Definition | Parameters | Pages & Classes | History

	Action	Target	Relation	Source
• 1	Set	.Driver	equal to	.Policyolder

☑ Call superclass data transform

Setting property values declaratively

Introduction to Setting property values declaratively

When a user enters a value in a form, related values can also change. When you buy an item like a laptop online, you enter the quantity you want to purchase. The system displays the amount of your order automatically. Declarative processing allows you to easily configure your application so that the system can automatically update property values such as an order amount.

After this lesson, you should be able to:

- Describe the declarative processing model.
- Describe the procedural processing model.
- Calculate a property value with a declare expression.
- Explain how forward chaining works to update property values.
- Explain how backward chaining works to update property values.

Declarative processing

When a user enters a property value in a form, related values on the same form or on other forms can change as a result. The application must make the changes automatically so that users see the most current information. For example, assume you are purchasing a laptop online. On the form, you enter "1" as the quantity. A total order amount field displays the price of one laptop. However, when you change the quantity to "2", the total order amount automatically doubles.

Declarative processing allows you to configure your application so that the system automatically updates property values such as a total order amount. Declarative processing identifies and maintains computational relationships among properties. When input values change, the declarative process automatically updates related property values. In the previous example, a declarative process maintains a relationship between the total order amount property value with the quantity and price property values. When a user orders laptops, the system multiplies the price of one laptop times the quantity of laptops to calculate the total order amount.

The primary benefit of declarative processing is that updates occur only when triggered in the application. You use declarative rules to define the **trigger event**. The system monitors the application to determine when a trigger event occurs. Using the previous example, the system is always monitoring changes to the item quantity property. When the value changes, the system triggers a computation to update the order total.

The following video describes how declarative processing works and provides an example.

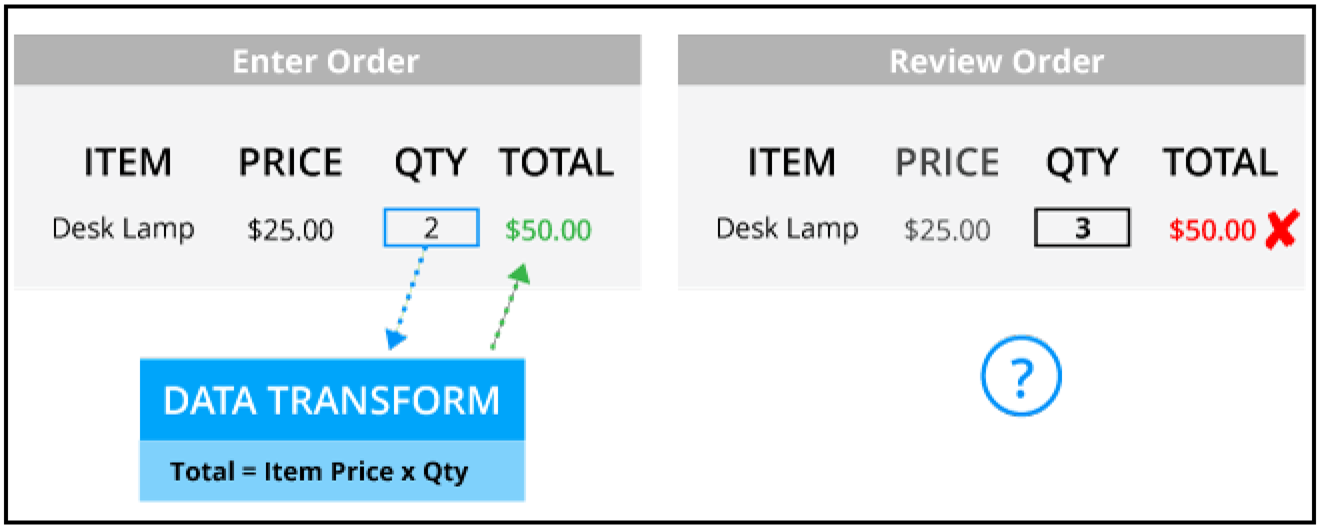
Procedural processing

A single declarative expression can monitor trigger events no matter where that expression is used in the application. Declarative processing rules do not depend upon other rules, such as data transforms, activities, or user interface (UI) rules, to perform updates.

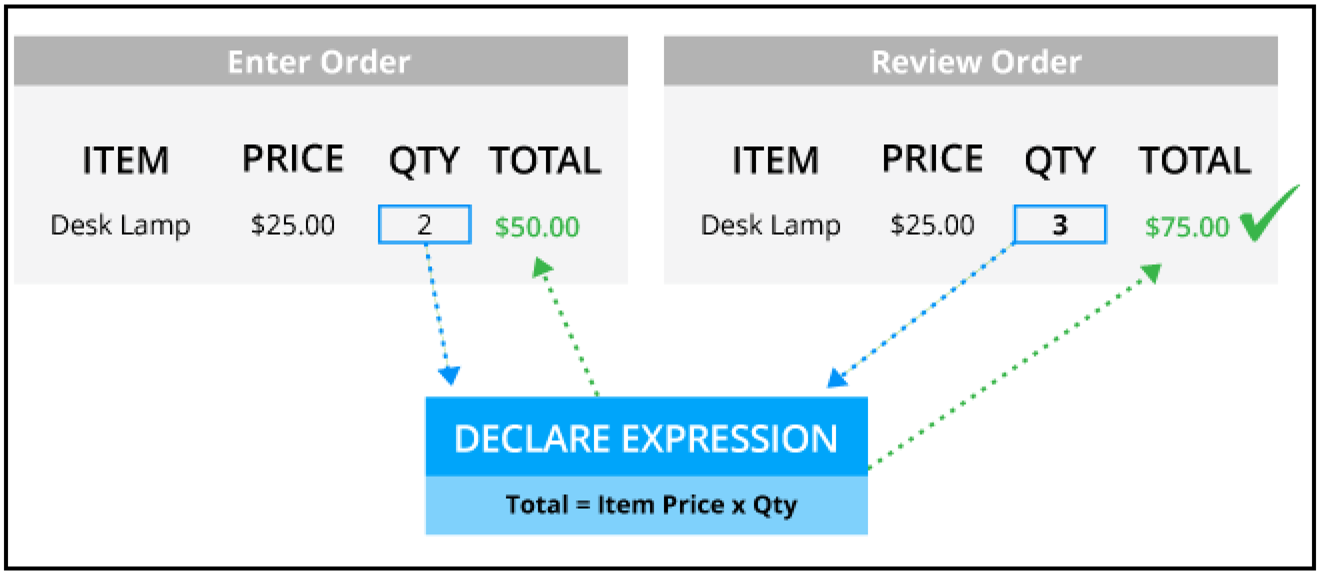
Procedural processing depends upon rules, such as data transforms, activities, or user interface (UI) rules, to instruct the application when to look for a trigger event. For instance, to trigger updates to the order total, you add a data transform to a flow action. When a user enters values, nothing changes until the user submits the form. The updates are not automatic. The submit process triggers the data transform to perform the update. In order to make the changes visible to users as they enter values, you must configure sections to use the data transform to refresh the fields.

Procedural processing maintenance

Procedural processing is more difficult to configure and maintain than declarative processing. For example, assume you have designed an Enter Order form that uses a data transform to calculate the total order amount based on the item price and order quantity. Then, you add a Review Order form to your application. This form reuses the fields for calculating the total order amount. If you do not add the same data transform to the Review Order form, the Total order amount is not updated when the user changes the order quantity from 2 to 3.



When you use a declare expression, the system only monitors changes to the source property values. In the following example, when a user updates the quantity from 2 to 3 in the Review Order form, the declare expression recalculates the total.

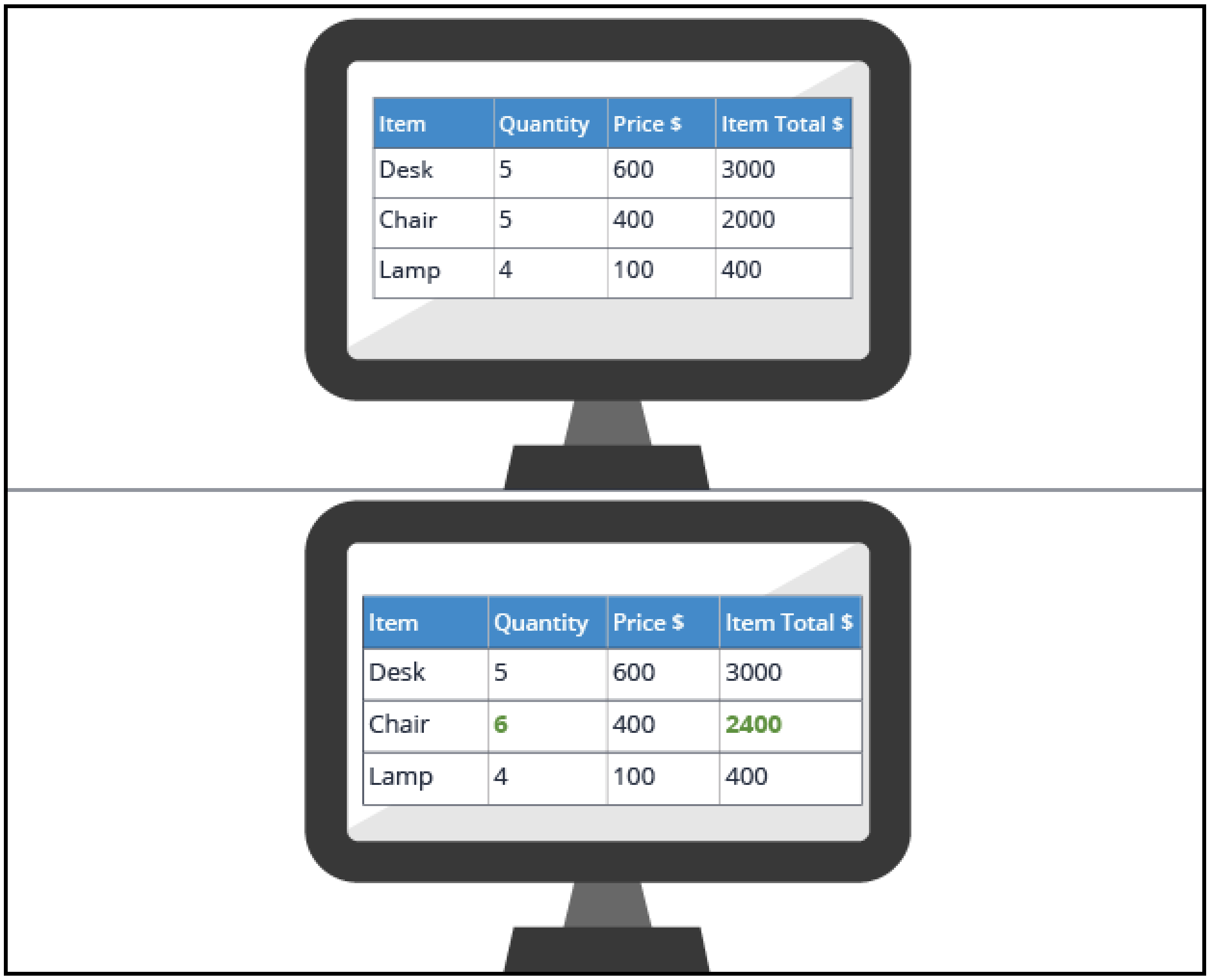


Pega provides many types of standard declarative rules that support declarative processing. For more information, see the PDN article Declaratives, Decisions, and Validation.

Declare expressions

You most often use declare expressions to calculate and make immediate updates to property values on user forms. Declare expressions contain an **expression** and a**target property**. The expression calculates and updates the target property value. The expression uses **source property** values in its calculation. Referencing a source property used in the expression initiates the calculation. The calculation then updates the target property value.

For example, assume an office furniture purchase order form includes fields for three target properties items: chairs, desks, and lamps. Each item has fields in which users select an item and enter the quantity. A declare expression uses the two source properties, item cost, and quantity to calculate the target property — item total. The expression multiplies the item cost times the number of quantity to calculate the item total. When the user changes the quantity, the expression recalculates the item total.



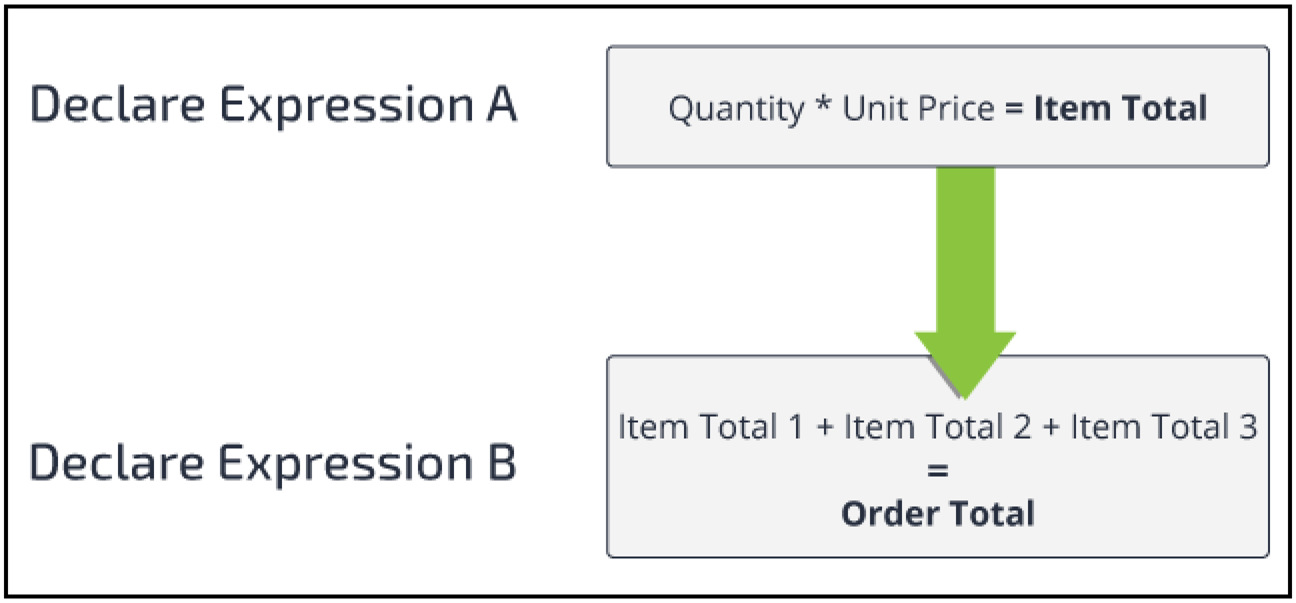
Item	Quantity	Price $	Item Total $
Desk	5	600	3000
Chair	5	400	2000
Lamp	4	100	400

Item	Quantity	Price $	Item Total $
Desk	5	600	3000
Chair	6	400	2400
Lamp	4	100	400

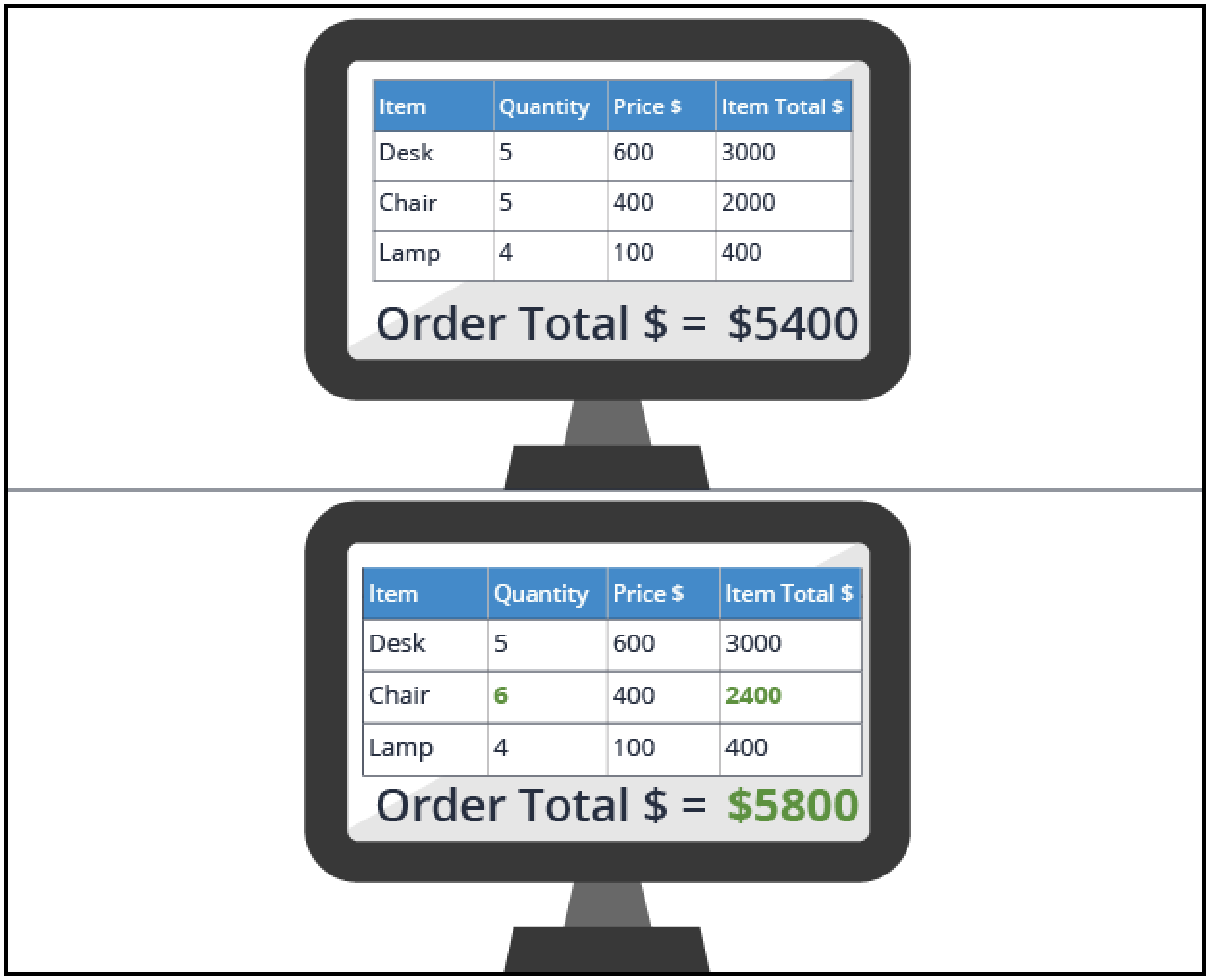
Declarative networks

You can use a sequence of interdependent declare expressions in a **declarative network**. A declare expression in a network can use a target property from another declare expression as a source property.

For example, assume you added an Order Total field to your purchase order form. This field uses a declare expression to calculate its target property — order total. The source property in this expression uses the item total target value.


Declare Expression A
Quantity * Unit Price = **Item Total**
Declare Expression B
Item Total 1 + Item Total 2 + Item Total 3 = **Order Total**

When a user updates a Quantity value, the system updates the Order Total value.



Item	Quantity	Price $	Item Total $
Desk	5	600	3000
Chair	5	400	2000
Lamp	4	100	400

Order Total $ = $5400

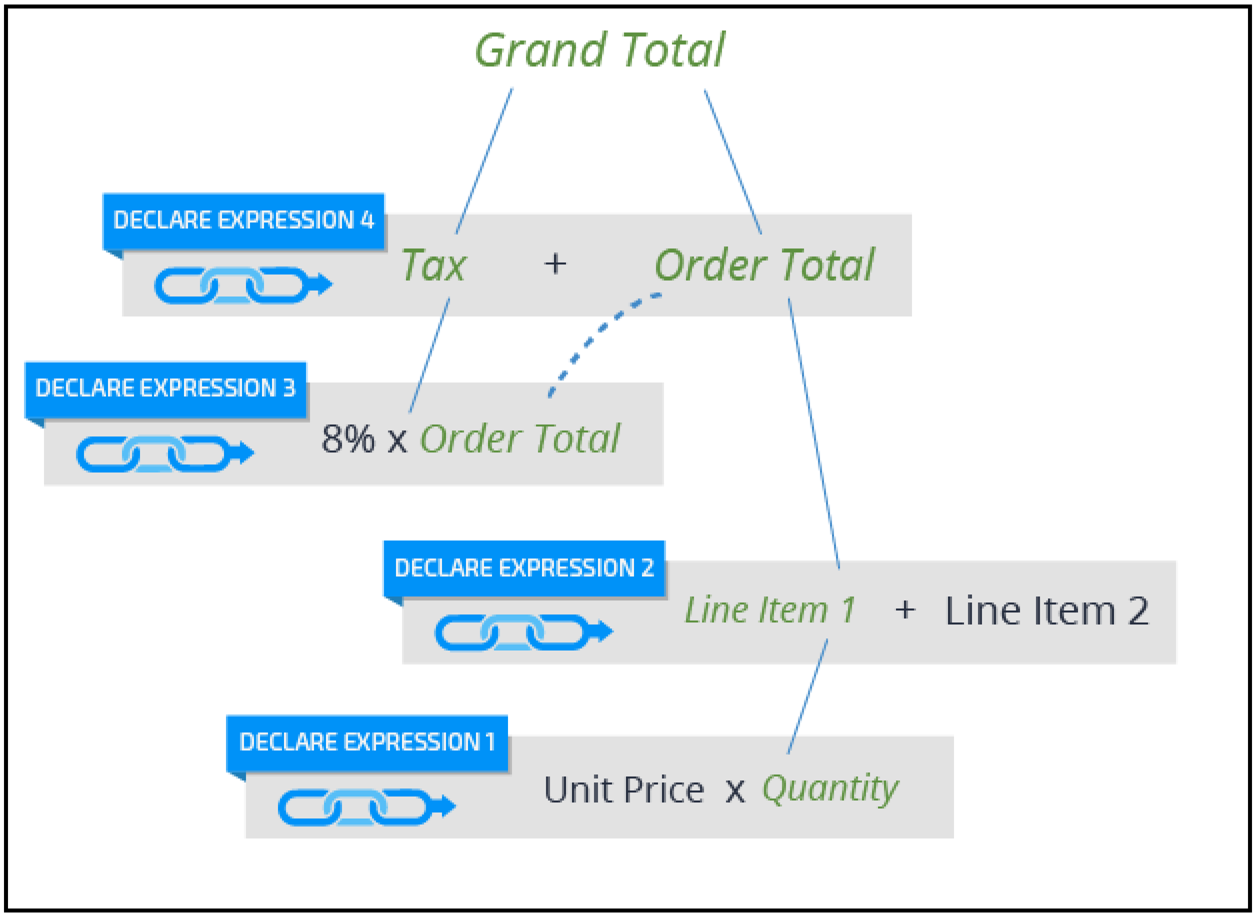
Item	Quantity	Price $	Item Total $
Desk	5	600	3000
Chair	6	400	2400
Lamp	4	100	400

Order Total $ = $5800

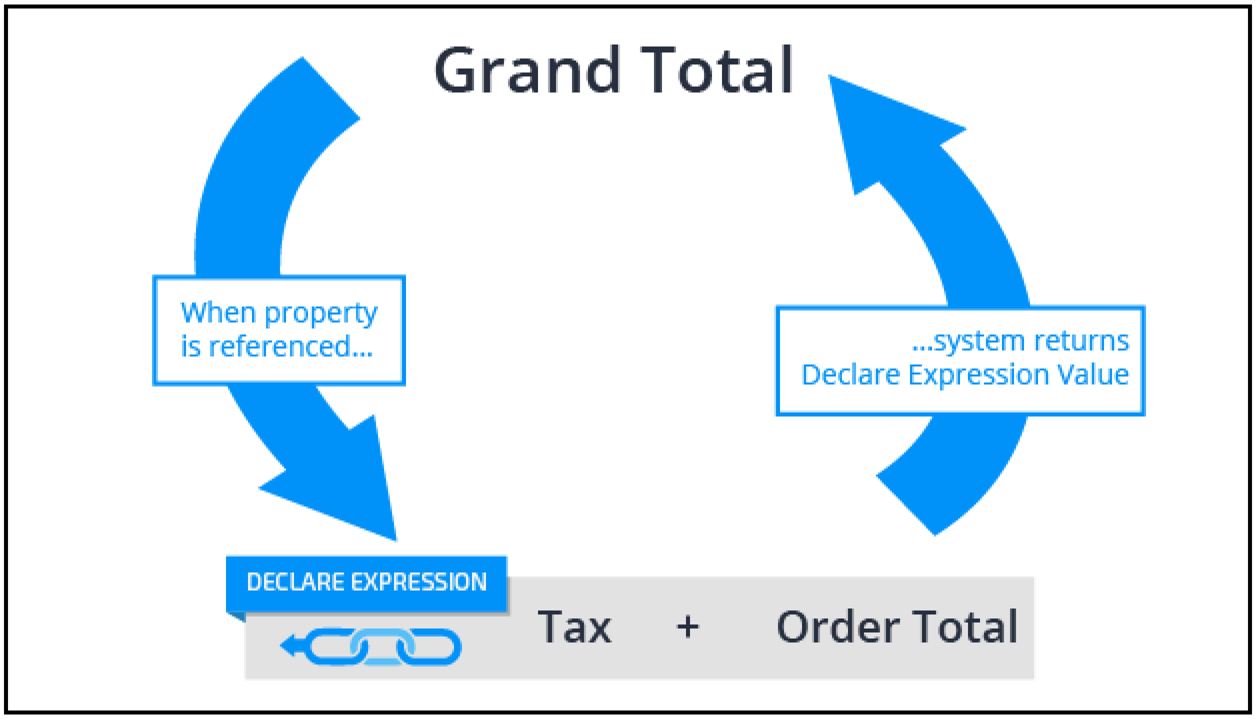
Pega provides the Declarative Network Analysis tool to display a list of declarative networks in your application. Access this tool by selecting the **Designer Studio > Process & Rules > Business Rules** menu. For more information about using the Declarative Analysis Network tool, see the help topic Business Rules landing page.

Forward and backward chaining

Forward chaining in a declare expression updates the target property value when a source property value changes. When you display a shopping cart where users add items to it and the cart should reflect the total based on the changes immediately, choose forward chaining. By default, declare expressions use forward chaining. Declarative networks are commonly designed with declare expressions that are configured for forward chaining.



Backward chaining in a declare expression means that a target property value is not automatically updated when other declare expressions in a network update their target values. An expression using backward chaining only updates its target property when the application references the property by name. A form, a decision table, or a data transform can reference the property. When the property is referenced, the expression goes back in the network to reference the source property or properties the expression needs to update its target.


Grand Total
When property is referenced...
...system returns Declare Expression Value
DECLARE EXPRESSION
Tax
+
Order Total

Chaining and performance

To optimize the chaining modes, consider where the source property is referenced and how the target property is referenced. Forward chaining can slow system performance if an expression uses many source properties that change frequently. For example, assume you are calculating the value of a home insurance quote based on more than 20 property values such as location, tax assessment, appraisal value, and land area. These values are collected in a large number of forms. When you use forward chaining, the home insurance quote declare expression recalculates the value every time users enter or change any of these 20 values. The impact to performance might affect response time when the user enters values or submits forms.

If you are only going to display the insurance quote after you collect all the values, use backward chaining for the home insurance quote expression. When you display the home insurance quote on the form, the expression performs the update only once. If you use forward chaining, the system performs the calculation even if the user does not see or need the value.

For more information about declare expressions and chaining, see the help topic More about Declare Expression rules.

How to set a property value with a declare expression

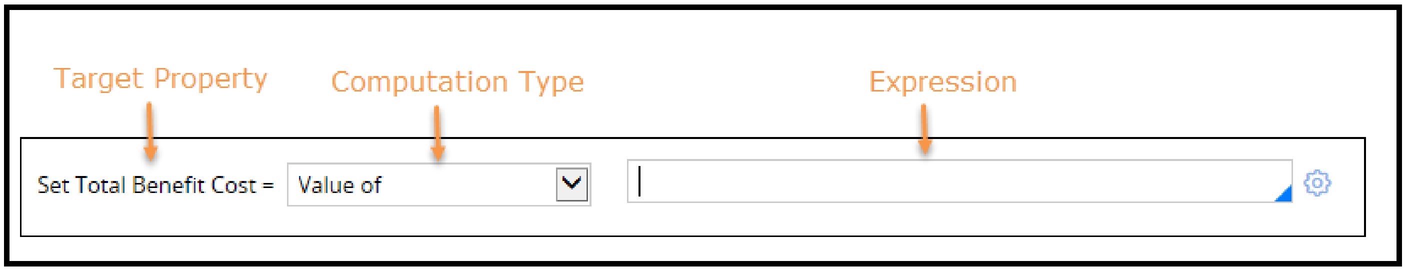
Creating a declare expression involves three major steps. You first define the target property — the value that is updated when a declare expression calculation is performed. Then, you define the expression that calculates the target property value. Finally, you configure the declare expression to use either forward or backward chaining.

Create the declare expression and target property

When you create a declare expression rule, you enter the target property as a key part. The available properties are defined by the Apply to key part. The easiest way to create a declare expression is to select the target property in the Application Explorer. Right-click and select the **Define Expression** option.

Configure the expression

After you have created the declare expression, on the **Expression** tab, configure an expression in a row as shown in the following example. The row consists of three fields.



The target property is the one you specified when you created the declare expression. In the previous example, the target property is Total Benefit Cost.

The drop-down allows you to select the type of computation for the expression. The default is Value of as shown in the previous example. This means that the source for the expression is one or more property values. You can select other options. The choices available depend upon the target property type. Options for numeric types include a summing or greater than/less than operators. You can also use the result of a decision tree, decision table, or a map value to provide a value.

You enter an expression in the form of a function and its inputs. You can also use the gear icon on the right side of the field if you want to build your expression using Pega standard functions such as CompareDates or getLocalizedValue.

Specify the chaining method

On the Change Tracking tab, configure the declare expression to calculate the target value using either forward or backward chaining. Use the **Whenever Inputs Change** option for forward chaining. There are three backward chaining options. For example, select **Whenever used** if you want the declare expression target value to be updated whenever the property is referenced in a form.

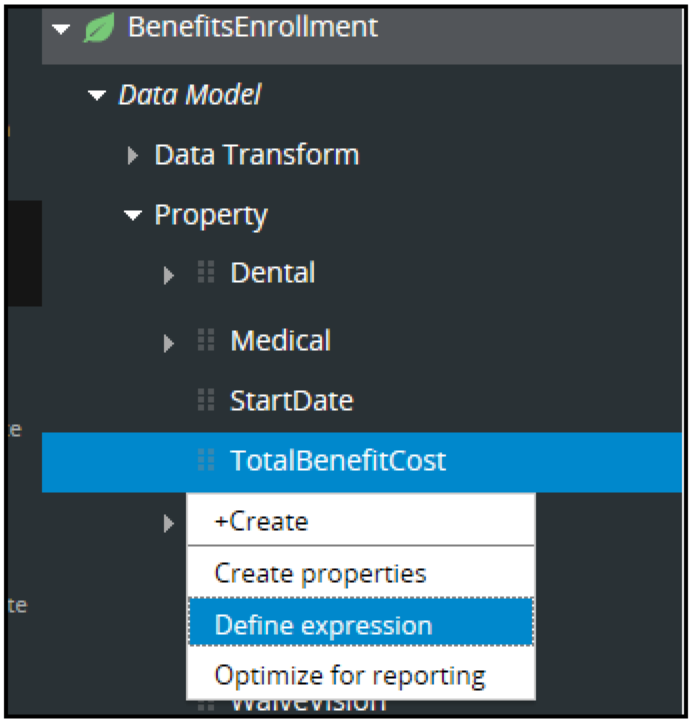
Setting a property value with a declare expression

Defining a property value with a declare expression has three steps:

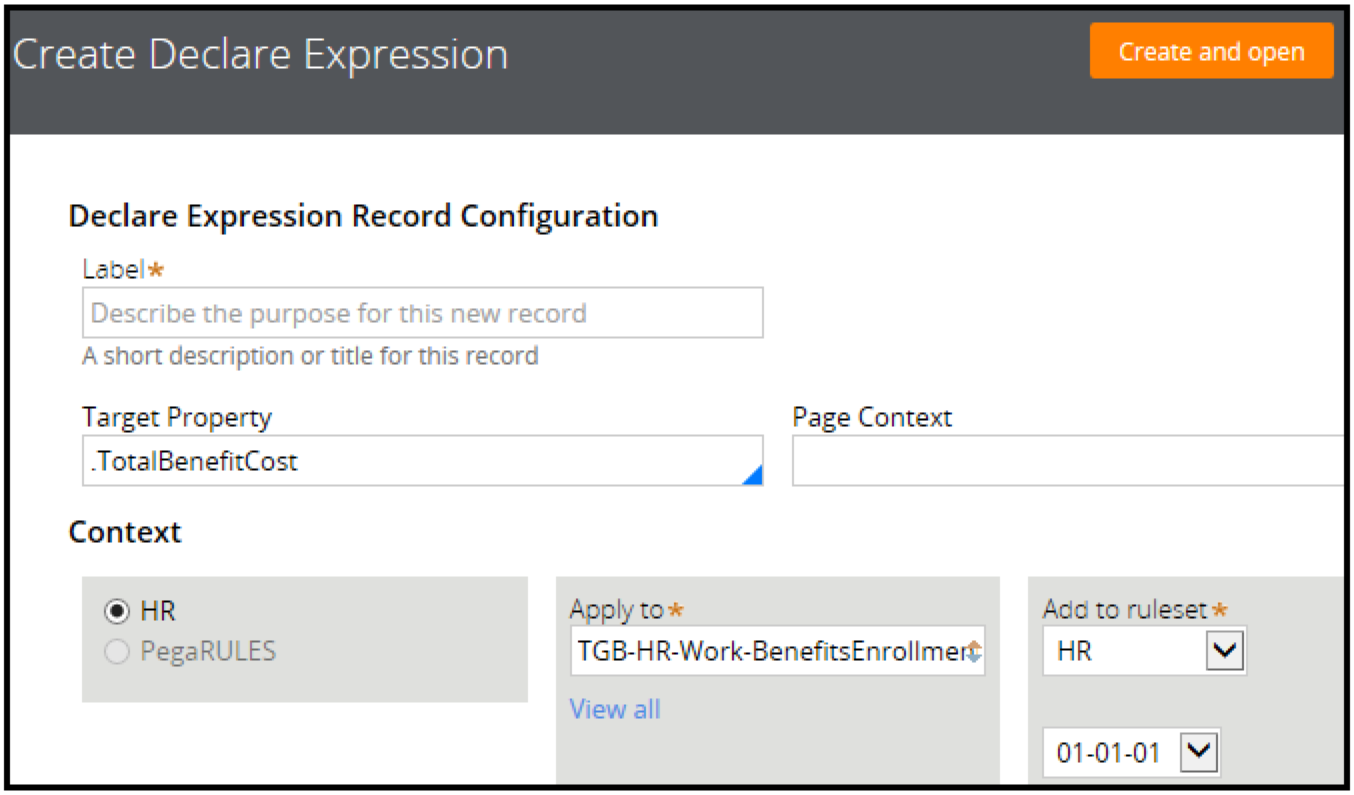
1. Identify the target property when you create the declare expression.
2. Identify the source properties when you define the expression.
3. Specify the chaining direction.

Identify the target property and create the declare expression

1. In the Application Explorer, select the class that contains the property you want to use as the target.
2. Select **Data Model > Property** to display the properties in the class.
3. Select the target property, right-click, and select **Define expression**.



The Create Declare Expression form is displayed.


Create Declare Expression

Declare Expression Record Configuration

Label*

Describe the purpose for this new record

A short description or title for this record

Target Property: .TotalBenefitCost

Page Context:

Context

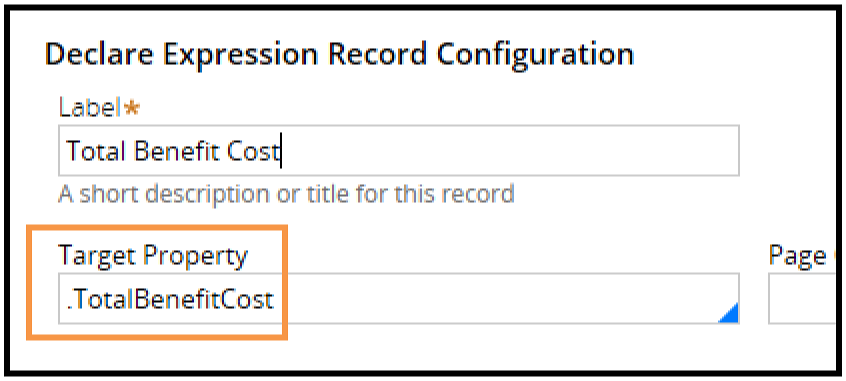
(● HR) PegaRULES

Apply to*: TGB-HR-Work-BenefitsEnrollmer

Add to ruleset*: HR

01-01-01

4. In the **Label** field, enter the name for the declare expression. Note that the property you selected in the Application Explorer is the target property. As a best practice, use the target property name to label the declare expression. The label is used to name the expression on the rule form.


Declare Expression Record Configuration

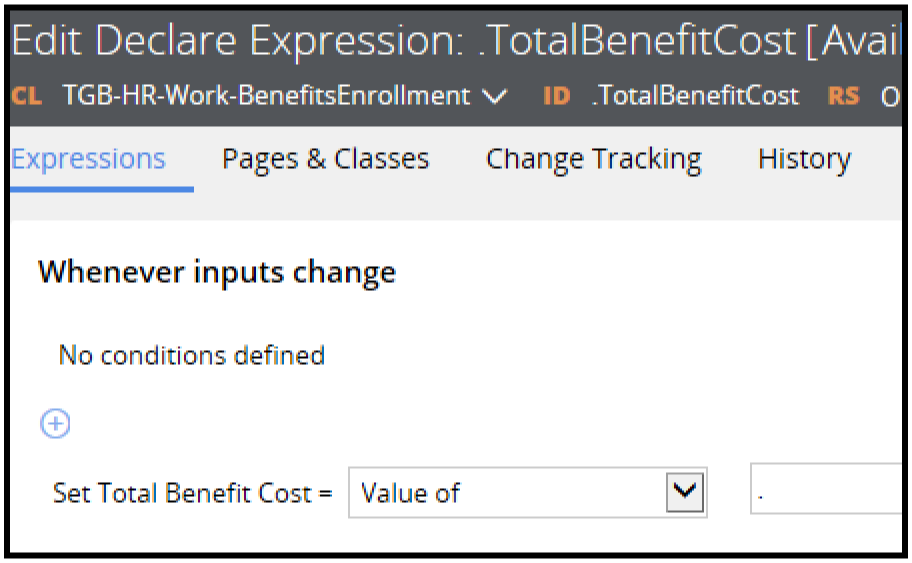
Label*: Total Benefit Cost

A short description or title for this record

Target Property: .TotalBenefitCost

Page

5. Click **Create and open**. The Declare Expression form opens.



Edit Declare Expression: .TotalBenefitCost [Avail

CL TGB-HR-Work-BenefitsEnrollment ID .TotalBenefitCost RS O

Whenever inputs change

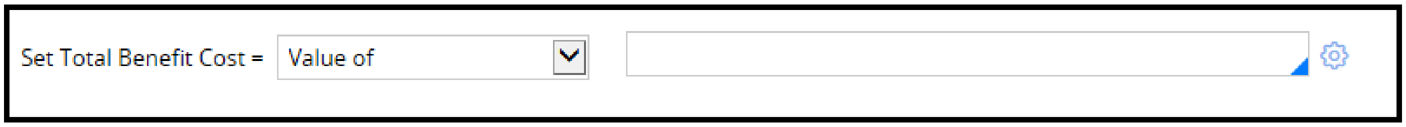
No conditions defined

Set Total Benefit Cost = Value of .

Note: The label **Whenever inputs change** indicates that the expression is configured for forward chaining. This is the default setting.

Define the expression

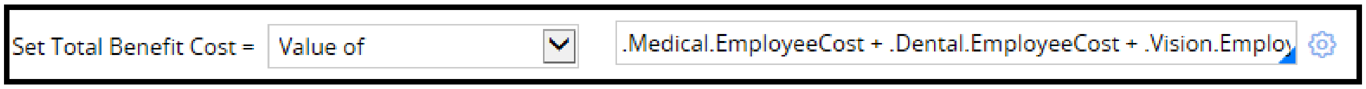
On the Expression tab, define the expression. Each expression is defined in three fields in a row. The target property is displayed at the beginning of the row.



Set Total Benefit Cost = Value of

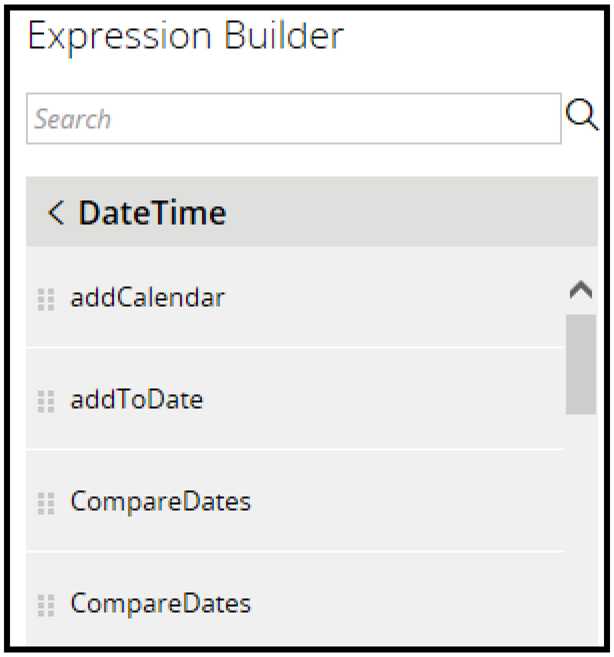
1. From the drop-down field, define the expression. select a type of computation. Select the **Value of**if you want to use property values as the source of the computation. For numerical values, you can use computations such as sum, minimum, maximum, or average. You can also use the result of a decision table, decision tree, or map value.

2. In the following example, the target property value — total benefit cost — equals the summed value of the three source properties — employee medical, dental, and vision costs.



Set Total Benefit Cost = Value of .Medical.EmployeeCost + .Dental.EmployeeCost + .Vision.Employ

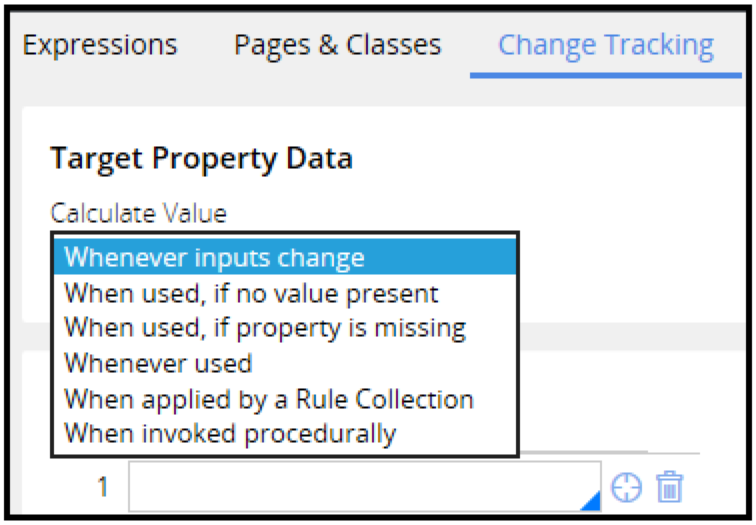
3. Optional: On the right side of the expression field, select the **gear** icon if you want to use functions from the selection list.



For more information about expressions you can use in declare expressions, see the help topic [Declare Expressions form Completing the Expression tab](#).

Specify forward or backward chaining

1. On the form, select the **Change Tracking** tab to specify whether you want to use forward chaining or backward chaining. By default, the declare expression uses forward chaining.
2. Select the **Calculate Value**drop-down to display the chaining options.


Expressions
Pages & Classes
Change Tracking
Target Property Data
Calculate Value
Whenever inputs change
When used, if no value present
When used, if property is missing
Whenever used
When applied by a Rule Collection
When invoked procedurally
1

3. Do either of the following:
   - Select **Whenever inputs change** to use forward chaining. The target property is computed when one of the expression source property values change.

- Select one of the backward chaining options, listed in the following table:

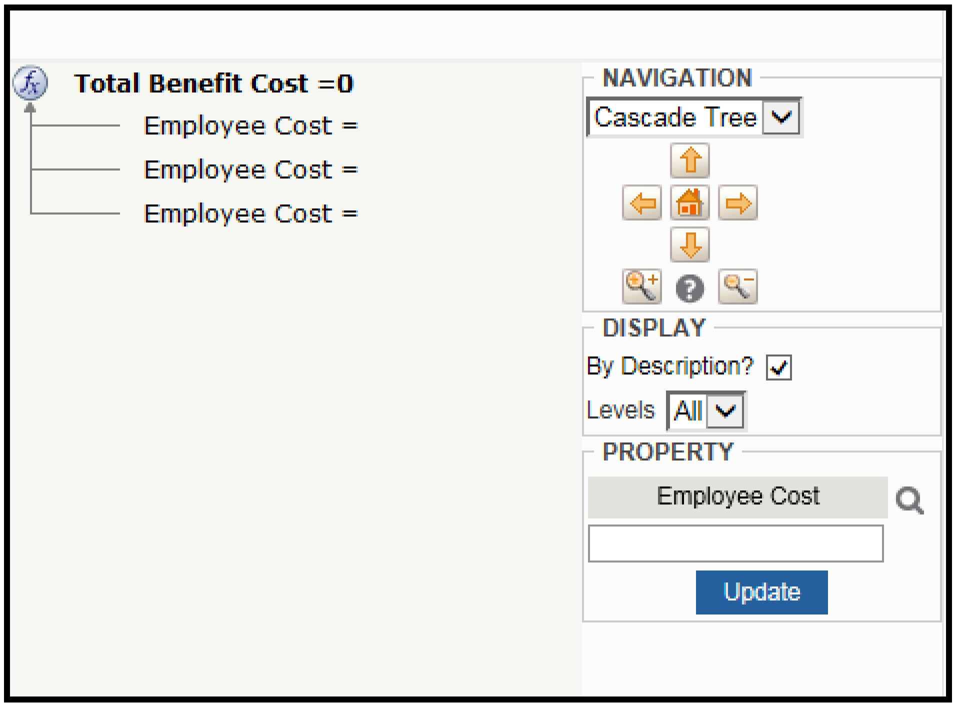
Option	Description
When used, if property is missing	Compute when the target property is not present on the clipboard.
When used, if no value is present	Compute when the value is null, blank, zero, or does not yet appear on the page. Later requests for the target property do not cause the declare expression to run.
Whenever used	Compute even when the property has a value.

For more information about the options in the Change Tracking tab, see the help topic Declare Expression form Completing the Change Tracking tab.

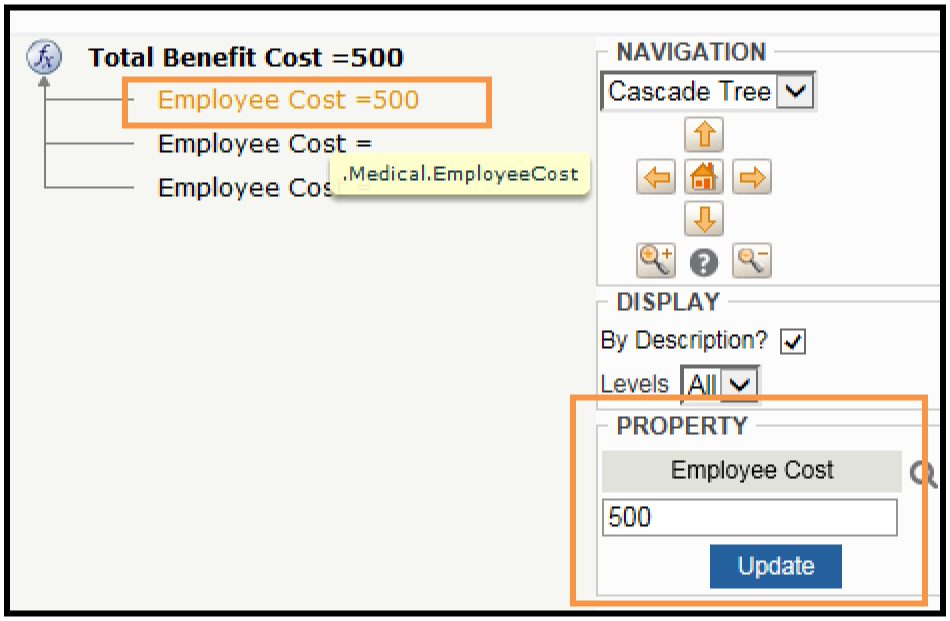
4. Click **Save**.

Test your declare expression

1. On the rule form header, select **Actions > Run** to test your declare expression. The test page is displayed.



2. On the declare expression tree, select a source property.
3. In the Property area on the bottom right of the page, enter a value and click **Update**. The value appears in the tree on the left.


Total Benefit Cost =500

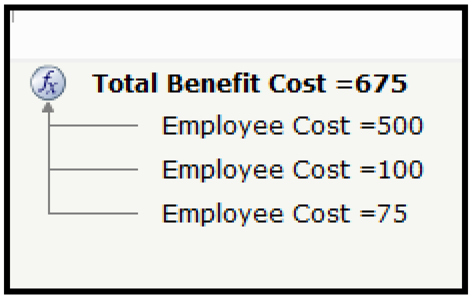
Employee Cost =500

Employee Cost =

Employee Cost

.Medical.EmployeeCost

4. Enter values for all of the source properties. The target value (Total Benefit Cost) is the sum of the source values.


Total Benefit Cost =675

Employee Cost =500

Employee Cost =100

Employee Cost =75

If the calculations perform as expected, you have successfully configured your declare expression.

Passing data to another case

Introduction to Passing Data to Another Case

Data propagation maps run-time values of properties in your parent case type to properties in child or spin-off case types. For example, you can propagate the urgency of an accident claim case to ensure that subcases or spin-off cases gets the same urgency. By sharing information among case types, you can make data-driven decisions.

After this lesson, you should be able to:

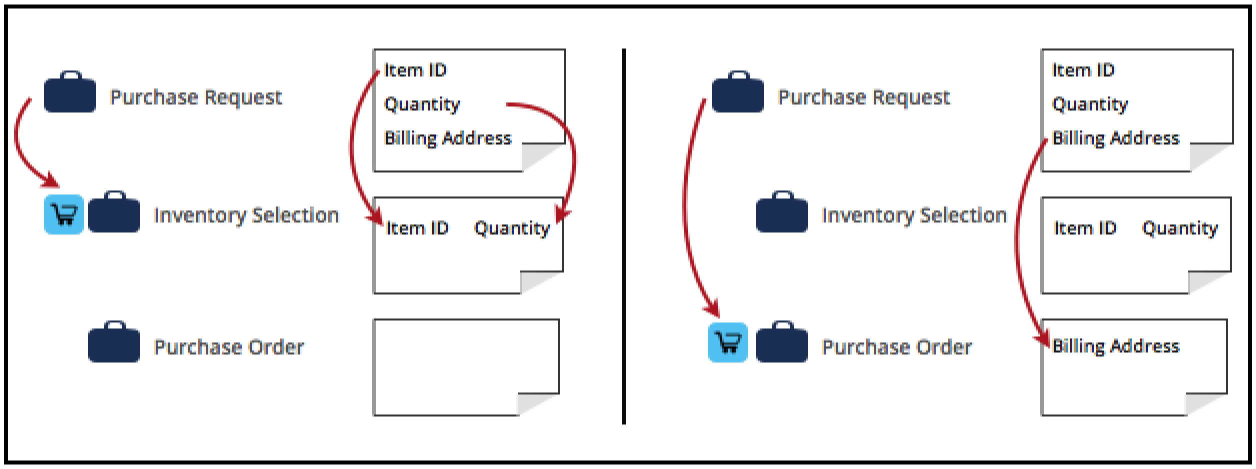
- Identify the role of data propagation in creating cases.
- List the options for passing data from one case to another.
- Copy data from a parent case to a subcase or spin-off case.

Data propagation

The role of data propagation

Data propagation is the mechanism of copying data within the case hierarchy. By sharing data among cases, you save time and provide relevant information to caseworkers.

Data propagation ensures that the appropriate information is propagated to a subcase. For example, a purchase request case may initiate an inventory selection subcase when units in stock must be confirmed. In the purchase request case, each line item in the purchase request contains a product identifier and a quantity. The inventory selection subcase then uses the product identifier and quantity to verify that the units are in stock.



Data propagation is not limited to subcases. Data can also be propagated when creating spin-off cases. For example, a purchase request case might spin off a supplier case if a new supplier is provided in the purchase request.

Data propagation happens on case creation. When you propagate a property from a parent case to a child case or spin-off case, and the property value later changes on the parent case, the property on the child case does not get updated.

For example, the product identifier is set to 0211 and the quantity is set to 3 in a purchase request. These values are propagated to the inventory selection subcase on creation. If the quantity in the purchase request later changes to 4, the value is not automatically propagated to the inventory selection subcase. The quantity in the inventory case will remain set to 3. You need to handle the subsequent synching of data between the cases manually.

Data propagation options

You can specify the properties to propagate from a parent to a subcase in the case explorer in two ways. You can define properties directly one by one or specify a data transform. If you need some conditional logic to determine what to propagate, use a data transform. For example, use a data transform to loop through a list and only propagate items that were selected.

You can create subcases and spin-off cases using the create cases utility in a stage step. The create cases utility allows you to propagate properties using a data transform.

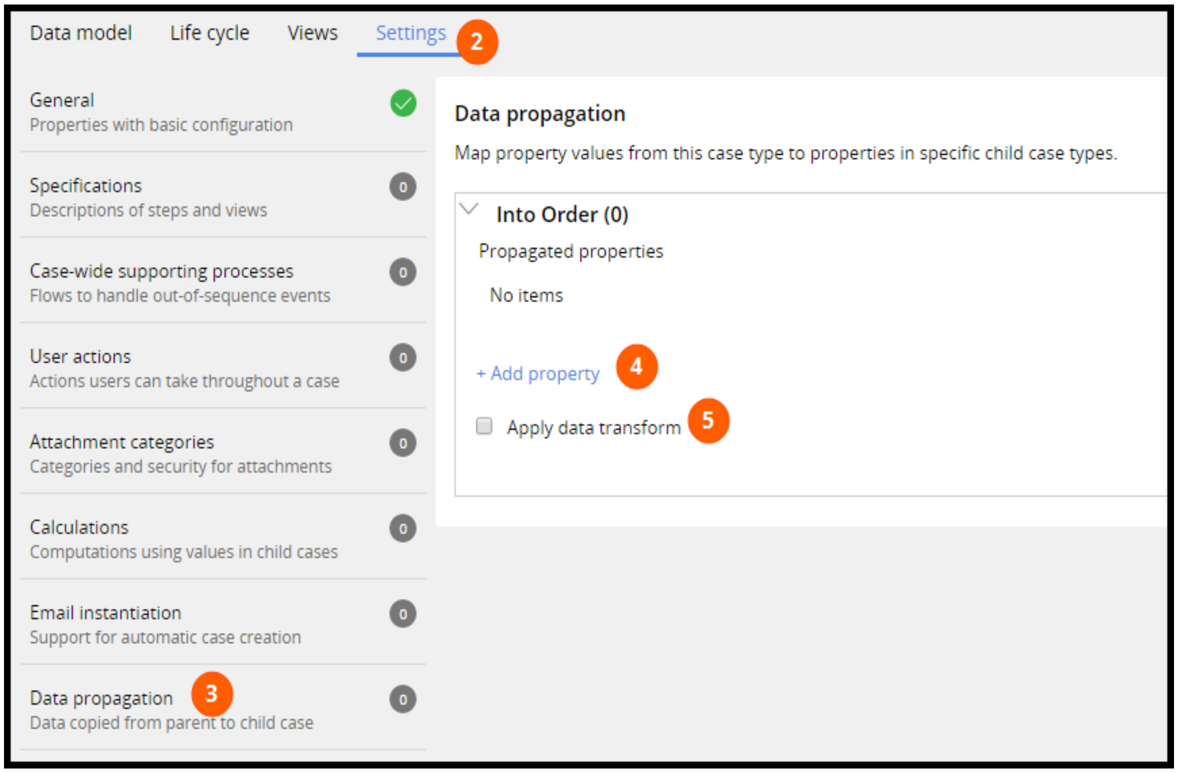
Propagating data to another case

Define data propagation in the Case Explorer

Review how data can be propagated from a purchase request case to a purchase order subcase. In this scenario, the parent case is the purchase request case.

Follow these steps to configure data propagation on the parent case:

1. Select the parent case in the Case Explorer.
2. Click the **Settings** tab.
3. Click **Data propagation**.
4. Click the **Add Property** link to specify the properties you want to propagate into the purchase order.
5. Select **Apply data transform** if you specify a data transform with the data propagation settings.

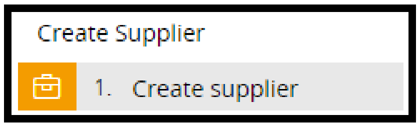


Data model
Life cycle
Views
Settings
2
General
Properties with basic configuration
Specifications
Descriptions of steps and views
0
Case-wide supporting processes
Flows to handle out-of-sequence events
0
User actions
Actions users can take throughout a case
0
Attachment categories
Categories and security for attachments
0
Calculations
Computations using values in child cases
0
Email instantiation
Support for automatic case creation
0
Data propagation
3
Data copied from parent to child case
0
Data propagation
Map property values from this case type to properties in specific child case types.
Into Order (0)
Propagated properties
No items
+ Add property
4
Apply data transform
5

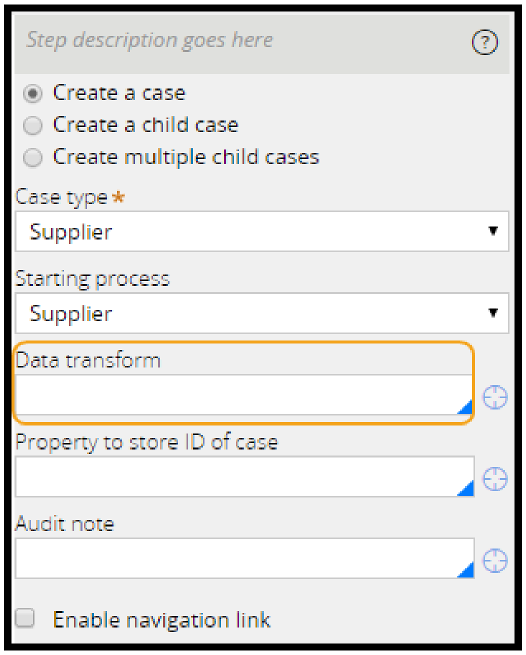
Configure data propagation for the create case utility

You can use the create cases step utility to create subcases and spin-off cases. In this scenario, a supplier case is spun off. Follow these steps to configure data propagation for the create cases utility:

1. Select the parent case in the Case Explorer.
2. Select the step with create cases utility.



3. Specify a data transform for the create cases utility in the settings.


Step description goes here

(●) Create a case
( ) Create a child case
( ) Create multiple child cases

Case type * Supplier

Starting process Supplier

Data transform

Property to store ID of case

Audit note

☐ Enable navigation link

You can specify a data transform regardless of if you want to create a spin-off, child, or multiple child cases.

Reviewing application data

Introduction to Reviewing Application Data

Applications generate large amounts of data: data about cases, data from outside sources, and data about users. While developing applications, you may need to review the data generated by your application. This information, if incorrect, can cause errors that lead to undesired results for cases.

To verify that cases are processed correctly, you may need to review the data generated by your application. To view data in memory, you use a tool called the Clipboard tool. The Clipboard tool allows you to review the information currently in memory to determine whether rule behavior is configured correctly.

After this lesson, you should be able to:

- Explain how data is stored in memory for use in Pega applications.
- Describe the relationship between pyWorkPage and case data.
- Explain how the Clipboard tool organizes data in memory.
- Use the Clipboard tool to review case data in memory.
- Use the Clipboard tool to set values for case data.

Data storage in memory

Cases are collections of data. To process and resolve a case, Pega applications capture, manipulate and present data throughout a business process. While processing a case, this data remains in memory for use by one or more users.

Each **data element** in a Pega application is a pairing of two pieces of information: the name of the data element, and the value assigned to the data element. For example, when you use a data element to capture the date of birth of an person, the data element name is date of birth, and the corresponding value is a date such as July 20, 1969.

Name of the data element	Value assigned to the data element
First Name:	Neil
Date of Birth:	July 20, 1969
Occupation:	Astronaut

Each data element is stored in memory on a page. A **page** is a structure for organizing data elements in an application. Some pages are created by the system to track user or session data. Other pages are created by system architects to describe a data object, such as a hotel reservation or a customer.

CUSTOMER
First Name: Neil
Date of Birth: July 20, 1969
Occupation: Astronaut

During case processing, each page remains in memory in a structure known as the **clipboard**. The clipboard is the portion of memory on the server reserved by Pega for the data generated by applications. The clipboard consists of all of the pages used to track the name-value pairs that represent case and session data. The clipboard receives its name because pages can be added to or removed from memory as needed to track case or session data. So, when a value is assigned to a data element, the data element and its value are said to be on the clipboard.

As you run a process, Pega sends information to the clipboard, adding or removing pages and properties from memory. Your application uses this information to populate fields on UI forms, perform calculations, and evaluate decisions.

KNOWLEDGE CHECK

ANSWER How is information, such as the color of a vehicle, stored in memory for use in a Pega application?
Information such as the color of a vehicle is associated with a data element. The data element (property and value) is stored on the clipboard in a structure called a page.

pyWorkPage

When you debug case behavior, you often need to view the case data that is in memory. By viewing this data, you can determine whether your application is functioning as expected. If your application functions in an unexpected way, viewing the data on the clipboard can help you identify the cause of the issue. For example, if a declare expression returns an unexpected result, you can review the contents of the clipboard to determine if one of the input properties has been set with an unexpected value.

All the data generated as you create and process a case is stored on **pyWorkPage**, which is a specific page on the clipboard. For example, data such as the date the case was created or the ID number for a case is stored on pyWorkPage. Data that describes a data type is stored on an **embedded page** within pyWorkPage. For example, if a case uses a data type named Customer, then Customer is considered an embedded page within pyWorkPage. All the properties that describe the Customer data type — such as first name — are written to the embedded page, rather than pyWorkPage.

Each page on the clipboard is an instance of a specific class, including pyWorkPage. When you refer to data on pyWorkPage, you may need to specify the class of the page. If you omit the class information, Pega cannot obtain property values from the correct page. Pega does not know whether the properties are valid or not, and the rule that references the properties does not function correctly. To ensure that the report obtains the correct information whenever you reference pyWorkPage, you need to specify the class of pyWorkPage.

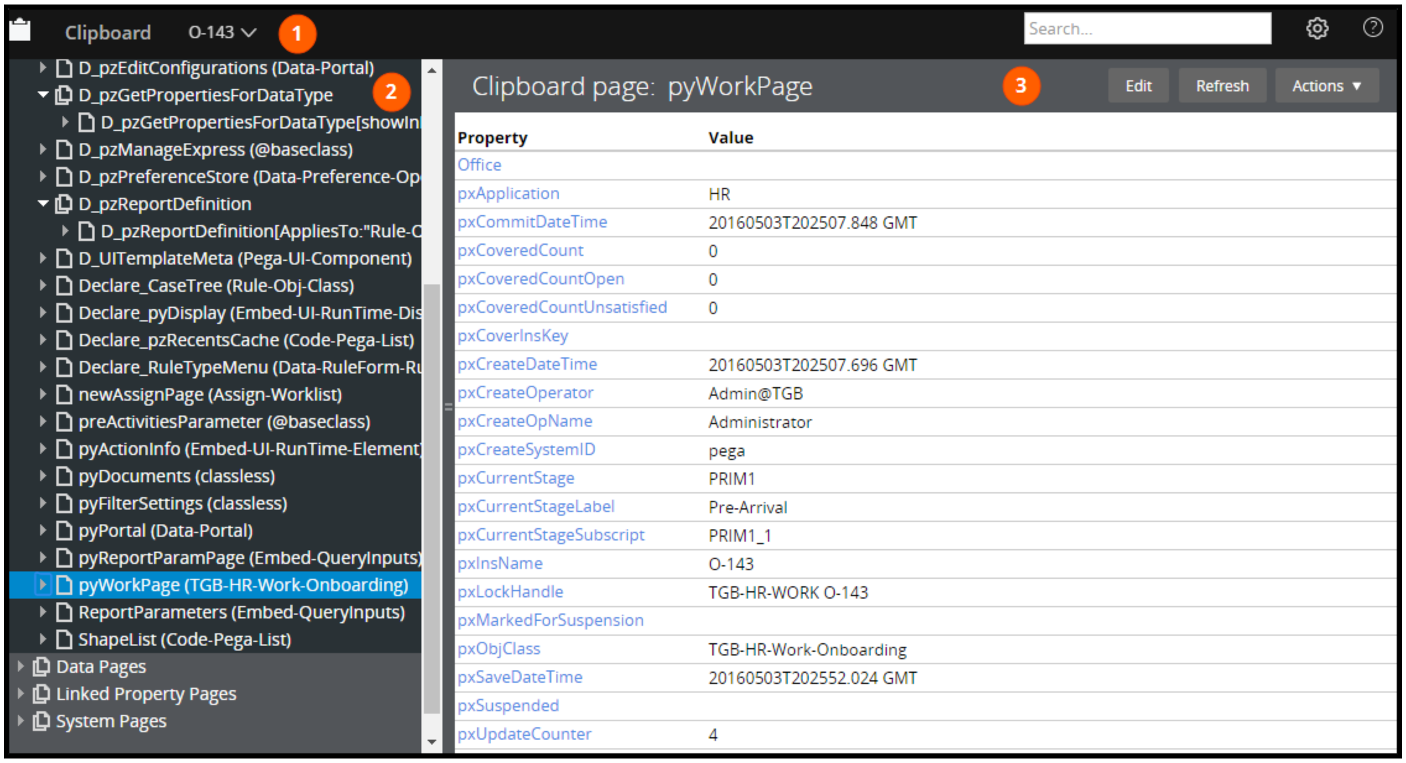
For example, consider an application to process automobile insurance quotes. To price the quote, you need to know the accident history of the driver. Each accident record is an instance of a specific data type. You create a report to return the accident history for a driver, and use a filter to return only accidents for the driver requesting the quote. If the report filter uses the *UserName* property from pyWorkPage, you must tell the report the class for pyWorkPage. This allows Pega to reference the *UserName* property and the report filter functions as intended. Otherwise, Pega assumes that *UserName* is part of the data type, rather than the case, and the filter does not work correctly.

When you open a child case, the clipboard also contains the page **pyWorkCover**. pyWorkCover contains the case data for the parent case. This allows you to copy data between the parent case and the child case.

How to view clipboard data

As users process a case, Pega sends information to the clipboard, adding or removing pages and properties from memory. Your application uses this information to populate fields on UI forms, perform calculations, and evaluate decisions. Pega also uses clipboard information to track the progress of a case through its life cycle, and to record information about the current operator.

If your application behaves unexpectedly, viewing the data on the clipboard can help you identify the cause of the issue. To view data that is in memory, you use the **Clipboard tool**. The Clipboard tool organizes and presents all the pages on the clipboard. When you select a page, the Clipboard tool lists all the properties on each page and the value of those properties. To open the Clipboard tool, click the Clipboard icon on the Developer toolbar in Designer Studio.



Clipboard
O-143
1
2
3
Search...
Edit
Refresh
Actions
D_pzEditConfigurations (Data-Portal)
D_pzGetPropertiesForDataType
D_pzGetPropertiesForDataType[showIn
D_pzManageExpress (@baseclass)
D_pzPreferenceStore (Data-Preference-Op
D_pzReportDefinition
D_pzReportDefinition[AppliesTo:"Rule-C
D_UITemplateMeta (Pega-UI-Component)
Declare_CaseTree (Rule-Obj-Class)
Declare_pyDisplay (Embed-UI-RunTime-Dis
Declare_pzRecentsCache (Code-Pega-List)
Declare_RuleTypeMenu (Data-RuleForm-Ru
newAssignPage (Assign-Worklist)
preActivitiesParameter (@baseclass)
pyActionInfo (Embed-UI-RunTime-Element)
pyDocuments (classless)
pyFilterSettings (classless)
pyPortal (Data-Portal)
pyReportParamPage (Embed-QueryInputs)
pyWorkPage (TGB-HR-Work-Onboarding)
ReportParameters (Embed-QueryInputs)
ShapeList (Code-Pega-List)
Data Pages
Linked Property Pages
System Pages

Clipboard page: pyWorkPage

Property	Value
Office	
pxApplication	HR
pxCommitDateTime	20160503T202507.848 GMT
pxCoveredCount	0
pxCoveredCountOpen	0
pxCoveredCountUnsatisfied	0
pxCoverInsKey	
pxCreateDateTime	20160503T202507.696 GMT
pxCreateOperator	Admin@TGB
pxCreateOpName	Administrator
pxCreateSystemID	pega
pxCurrentStage	PRIM1
pxCurrentStageLabel	Pre-Arrival
pxCurrentStageSubscript	PRIM1_1
pxInsName	O-143
pxLockHandle	TGB-HR-WORK O-143
pxMarkedForSuspension	
pxObjClass	TGB-HR-Work-Onboarding
pxSaveDateTime	20160503T202552.024 GMT
pxSuspended	
pxUpdateCounter	4

The Clipboard tool is organized into three parts: the header, the left pane, and the right pane.

1. The header allows you to select the **thread** to view. Each thread corresponds to a unique action currently managed by Pega. The clipboard contains one thread dedicated to the Designer Studio environment. Other threads are dedicated to open rule forms. Pega assigns each open case a unique thread. By assigning each case or action its own thread, Pega ensures that the data for one case or action does not affect data for another case or action.

2. The left pane lists each page defined on the clipboard for the selected thread. For each page, the Clipboard tool identifies the name and class of the page. If a page contains embedded pages, an expand arrow is displayed to the left of the page name. To view the embedded pages, click the expand arrow.

   Pages on the clipboard are organized into four categories:

- The **User Pages** category contains pages created due to user action, either directly or indirectly. User pages contain data related to work being performed in the selected thread. While a user processes a case, all the pages used to store data about the case are listed in the User Pages category. Likewise, when a system architect configures or tests a rule, all the pages that store data used by the rule are listed in this category. For example, the data you enter onto a form is stored on the user page pyWorkPage.
- The **Data Pages** category contains read-only data pages defined by data page rules. Data pages are persistent pages in memory, used to cache data. This data is often sourced from outside the application, such as from a third-party or a system of record. For example, your application converts currency from one type to another, such as converting US dollars to Euros. The conversion rates, which are determined by the currency markets, are cached to a data page for use by one or more users of the application.
- The **Linked Property Pages** category contains read-only pages created by linked properties, which contain information from data objects referenced by a linked property. Linked properties are advanced data constructs, typically created and configured by Senior System Architects (SSAs) or Lead System Architects (LSAs).
- The **System Pages** category contains pages that describe the current user session, such as the active user and the active application. For example, while a user is logged in to Pega, Pega maintains a clipboard page containing information about the user, such as their current time zone.

3. The right pane lists all of the properties defined on the selected page, and their values. In the right pane, you view data in memory. You can also update property values and even add new properties to the page to represent data not captured in your application. This allows you to test application features that rely on data that has not been added to the case type, such as decisions and UI forms. For example, in an expense report case you want to branch a flow based on the project type. The application currently lacks a field in the UI to allow the user to select the project type. In this situation, you can use the clipboard to set a value for the property and verify that the flow branches properly.

When you view data with the Clipboard tool, you see a snapshot of the contents in memory. As you navigate your process, refresh pages in the Clipboard tool to ensure that the Clipboard tool always displays current property values and page contents.

KNOWLEDGE CHECK

ANSWER While testing case behavior for an online shopping application, you want to confirm that the application properly generates a list of the customer's previous orders when querying the company's order management system. In which category of clipboard pages would you expect to find the page that contains this list?

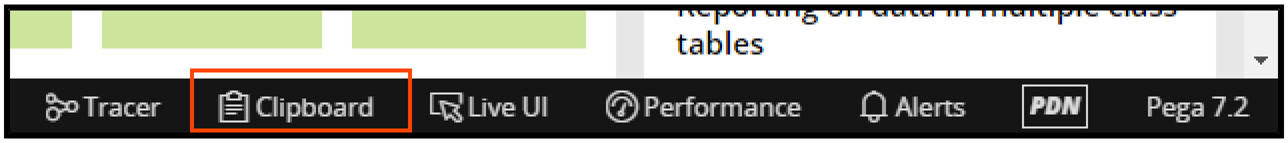
This page should be located in the Data Pages category.

Viewing clipboard data

Use the clipboard to view data in memory and determine whether rules are generating or updating case data as expected.

To view case data on the clipboard:

1. Create a case and note the case ID.
2. On the Developer toolbar, click the **Clipboard** button to open the Clipboard tool. By default, when you open the Clipboard tool, the tool displays the thread that corresponds to the active tab in Designer Studio.



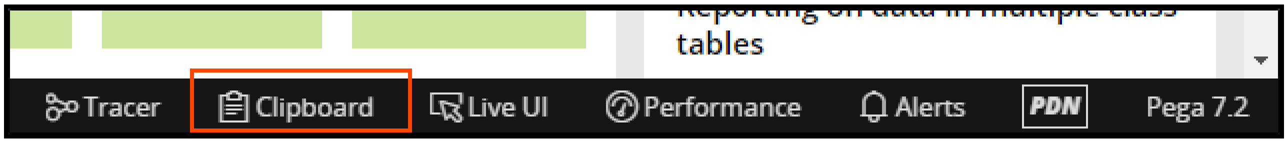
3. Optional: In the Clipboard tool, in the header use the thread selector to select the thread that corresponds to the case you want to review.
4. In the left pane, select the page to view. For example, to view data from a form used in the case, select **pyWorkPage**. To view the contents of a node-level data page, expand **Data Pages > Node** and select the data page to view.
5. In the right pane, locate the property name in the alphabetical list and confirm the value.

Setting property values using the Clipboard tool

Set data values on the clipboard to test rules that rely on case data to function correctly.

To set a property value using the Clipboard tool:

1. Create a case and note the case ID.
2. On the Developer toolbar, click the **Clipboard** button to open the Clipboard tool. The Clipboard tool defaults to displays the thread that corresponds to the active tab in Designer Studio.



3. Optional: In the Clipboard tool, in the header use the thread selector to select the thread that corresponds to the case you want to review.
4. In the left pane, select the page that contains the data element you want to update. The page opens in the right pane, displaying the value for each property defined on the page.
5. In the toolbar, click **Edit**. The page contents update to display fields for editable properties.
6. Locate the property that you want to update, and enter an updated value in the field.
7. Optional: to set the value of a property that has not been defined on the page, position the cursor in the upper-left corner of the right pane and click **Add**. The Add properties dialog opens, allowing you to enter the name and value of the property.
8. Click **Save** to update the clipboard with your changes.
9. In Designer Studio, return to the case that corresponds to the thread you edited in the Clipboard tool.
10. From the **Actions** menu, select Refresh to reload data from the Clipboard. The contents of the active form update to reflect the updated values you provided in the Clipboard tool.

PROCESS DESIGN

Activities

Introduction to Activities

The Pega platform provides rules to model almost any type of application behavior, from defining data values to automating decisions to connecting with other systems. One such rule is an activity, used to describe the logic for an automated procedure. Application developers new to the Pega platform often write new activities, rather than embracing more suitable and easier to maintain alternatives, such as reusing Pega defined activities.

In this lesson, you learn about activity rules, and how activities automate certain types of system actions. You also learn how to use the contents of Pega's activity library and how to avoid writing custom activities in Pega applications.

After this lesson, you should be able to:

- Identify the uses for activities in applications.
- Explain the activity execution model.
- Use API activities in an application.
- Identify options for minimizing the use of activity rules in applications.

Activities

During case processing, an application often needs to perform automated procedures. For example, saving a case record might require multiple operations to be carried out in sequence as one single step. The sequence can include updating case property values in memory, making a database connection, writing the case record to the database, handling errors, and writing messages to a log file. In Pega, you implement this kind of procedural logic in the form of activities. This form is similar to conventional programming language.

Activities are the primary processing rules in Pega. An **Activity** is an automated procedure, structured as a series of steps that execute in sequence. Each step can call a method, transfer control to another activity, or execute custom Java code.

Activities are often used to implement complicated logic. Some typical use cases for using activities are:

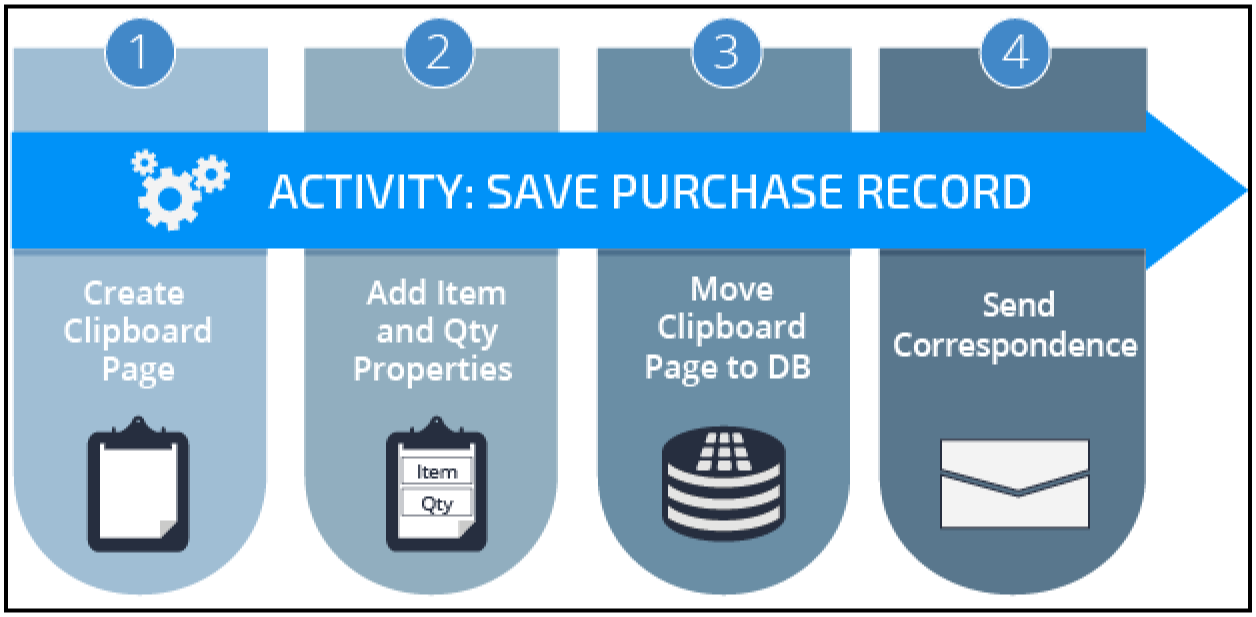
- Case processing related functions — To perform case-related functions such as creating a case instance, routing the case, or updating the work status
- Integration — To send requests to external systems or to receive requests from external systems
- Ancillary functions — To perform functions such as writing to a log file or to the history database

A **Method**is an operation that can be performed by one step of an activity. A method contains a predefined set of basic operations that perform computations, update properties and other aspects of the clipboard, or move data between memory and the database.

Some common methods are:

- **Property-Set**— Set the value of one or more properties.
- **Page-New** — Create a page.
- **Page-Remove** — Delete a page from the clipboard.
- **Apply-Data-Transform** — Update property values based on a data transform.
- **Call** — Call another activity.

As an activity executes, it can create or remove clipboard pages, create and update properties on these pages, save data to the database, and interact with end users by sending and receiving HTML documents and forms.


1
2
3
4
ACTIVITY: SAVE PURCHASE RECORD
Create Clipboard Page
Add Item and Qty Properties
Item
Qty
Move Clipboard Page to DB
Send Correspondence

Activity execution

By default, an activity executes its steps in sequence. You can control the flow of execution by repeating a group of steps, defining a set of preconditions for a step, or jumping to a later step.

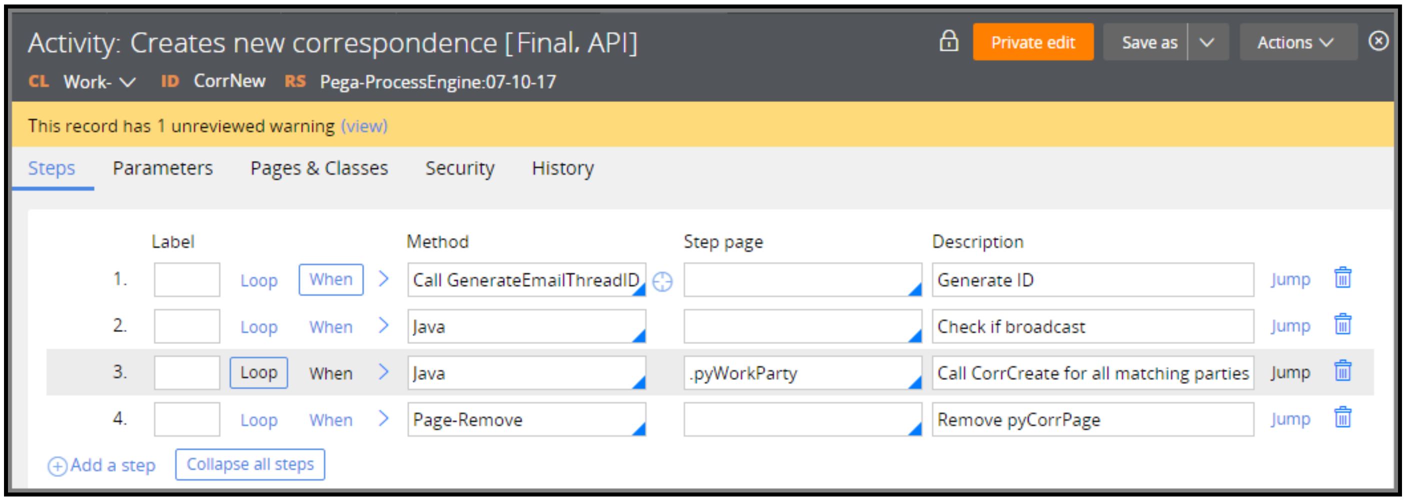
Each step of an activity contains multiple parts. There are nonaction items such as Label, Description, and Step Page. There are also action items such as Loop, When, Method, and Jump to indicate an action or condition for an action.

Use the following options to control how the activity processes a step at run time:

- The **Label** provides an identifier for the step that can be referenced from other steps.
- The **Loop** allows you to set up an iteration through the elements of a Value List, Page List, Value Group, or Page Group, and performs the provided activity method on each value or each embedded page.
- The **When** allows you to define a precondition that controls whether the step is executed or skipped.
- The **Method** indicates which method or activity the step will execute.
- The **Step Page** identifies a page to be used as the context for referenced properties within the step.

- The **Description** is text that explains to other developers the action that the step is performing.
- The **Jump** condition or post-condition is similar to the When precondition. The Jump condition defines how this step transitions, or jumps, to a later step.

The following image shows the standard activity CorrNew as an example. The activity executes four steps to send emails.


Activity: Creates new correspondence [Final, API]

CL Work- ID CorrNew RS Pega-ProcessEngine:07-10-17

This record has 1 unreviewed warning (view)

	Label			Method	Step page	Description	
1.		Loop	When	Call GenerateEmailThreadID		Generate ID	Jump
2.		Loop	When	Java		Check if broadcast	Jump
3.		Loop	When	Java	.pyWorkParty	Call CorrCreate for all matching parties	Jump
4.		Loop	When	Page-Remove		Remove pyCorrPage	Jump

- Step 1 executes another activity with the Call method.
- Step 2 executes some Java code to check if the email is a broadcast.
- Step 3 loops through a page group and creates emails for matching targets.
- Step 4 executes the Page-Remove method to clean up clipboard pages.

Page context of activity execution

During execution, an activity can access data from three data pages: Primary page, Step page, and Parameter page.

A **Primary page** is a clipboard page which has the same class as the Applies To class of the activity and is designated when the activity is called. The Primary page provides data context for the whole activity.

Each step in an activity may have a designated **Step page**. This step page provides data context during the execution of this step. If a step page is not specified for a step, the primary page become the step page.

A **Parameter page** contains parameter names and values, as listed in the parameters tab. An activity can access incoming parameters and update outgoing parameters, as a way of communicating information with callers.

Activity parameters

Activities may have parameters that can be accepted as inputs, used to convey results, or serve as both inputs and outputs. Not every activity has a parameter.

Parameters allow the execution of an activity to vary at run time, which promotes reuse. Think of an activity with a parameter as an instruction with a missing object, such as "Go to the market to buy _____ ____________." For a trip to the market, you can specify any object, such as strawberries, flour, or eggs. The activity is the general instruction — in this example, "Go to the market to buy". The parameter is the object that you provide to complete the instruction. You complete the instruction by telling someone to go to the market to buy eggs, or strawberries, or flour.

For example, Pega provides a parameterized activity for updating the work status of a case, named *UpdateStatus*. Whenever you apply a work status to a case, Pega runs this activity to update the work status of the case. This activity accepts the work status as a parameter. Without the use of a parameter, Pega would need an activity for each work status value. If you define a new work status, you would need to create a copy of the activity to set that work status. By using the parameter, Pega only needs one activity to set any work status value, even a custom work status value you define for your application.

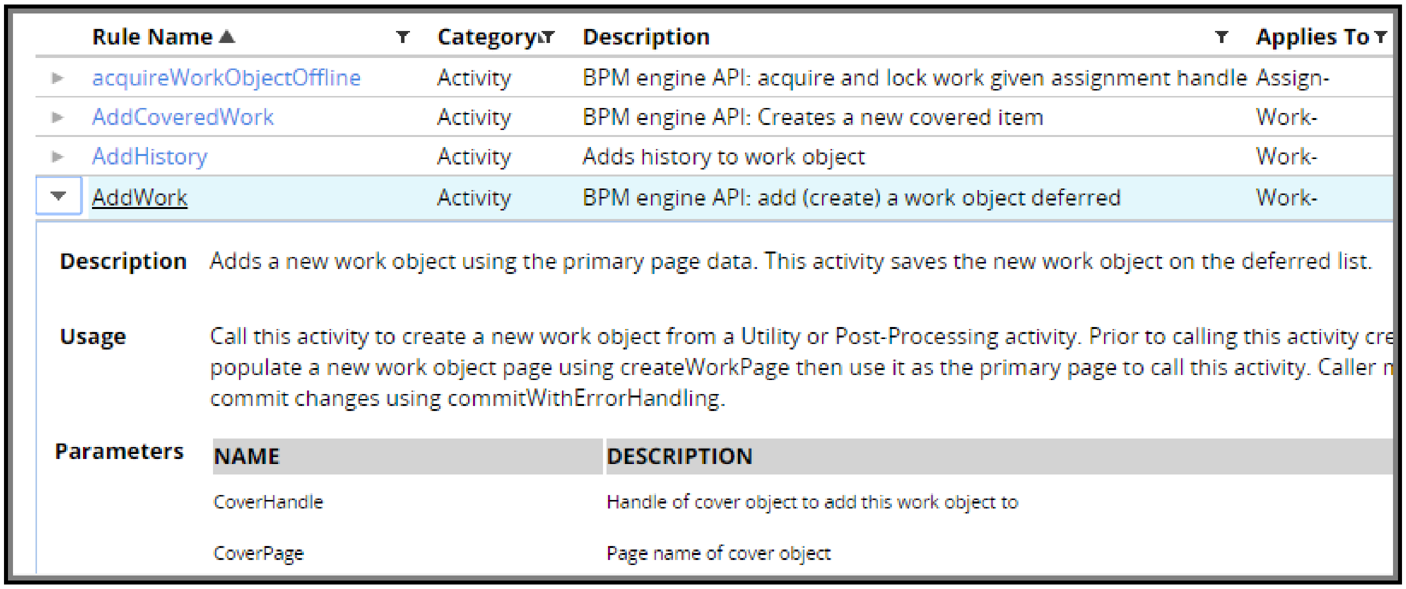
Parameters to an activity are listed in the **Parameters** tab of the Activity form. If a parameter is an input to the activity, you can access the value received from the calling activity. If a parameter is an output, or result, of the activity, you can set values for it using the *Property-Set* method.

API activities

Pega provides many predefined activities that perform standard functions. These standard activities are called API activities. Prior to creating any new activities, explore the API activities to see if one exists that can meet your requirements. Some of the commonly used API activities are **CorrNew** for creating and sending emails, **AddWork** for creating a new case instance, and **UpdateStatus** for setting the status of a case instance.

The Process API is a group of such standard activities that you can use to start and advance work flows without implementing user forms.

To see a list of the Process API rules, select **Designer Studio > Process & Rules > Processes > APIs**. Expand each row to learn more about each activity.



Rule Name	Category	Description	Applies To
acquireWorkObjectOffline	Activity	BPM engine API: acquire and lock work given assignment handle	Assign-
AddCoveredWork	Activity	BPM engine API: Creates a new covered item	Work-
AddHistory	Activity	Adds history to work object	Work-
AddWork	Activity	BPM engine API: add (create) a work object deferred	Work-

Description Adds a new work object using the primary page data. This activity saves the new work object on the deferred list.

Usage Call this activity to create a new work object from a Utility or Post-Processing activity. Prior to calling this activity cre populate a new work object page using createWorkPage then use it as the primary page to call this activity. Caller n commit changes using commitWithErrorHandling.

Parameters

NAME	DESCRIPTION
CoverHandle	Handle of cover object to add this work object to
CoverPage	Page name of cover object

You can invoke API activities from a flow shape or a flow action. To invoke an API activity from a custom activity rule, use the Call method from an activity step.

Activities best practice

While activities can appear to be an easy and flexible way to automate the work process, they can become very complex to analyze, execute, debug, and maintain.

As a best practice, consider alternatives such as data transforms and linked property references before creating activity rules. The alternatives are easier to understand and maintain.

Consider the following approaches before you write new activities:

- Look for alternatives. Refrain from writing new activities simply because it is easy to relate activities with programming languages that you know from experience.
- For data manipulation, use Data Transforms instead of activities.
- For data calculation, use Declare Expressions instead of activities.
- For queries from an external database, use Report Definitions instead of activities.
- Use activities only when none of the standard or out-of-the-box activities are available and appropriate for your requirement.

Refer to this PDN article — Nine tactics to reduce your need for custom activities — for extended discussion on this topic.

Configuring a work party

Introduction to Configuring a Work Party

Pega applications allow system architects to describe the people and organizations interested in a case by their role, such as Customer or Manager. Identifying a party to the case by their role allows you to describe their participation in a business process, such as receiving correspondence. For example, you can notify the Customer party of the status of their insurance claim throughout the claims adjustment process.

In this lesson, you learn how Pega describes the participants or interested parties for a case, and how you can design cases to interact with these parties throughout a business process.

After this lesson, you should be able to:

- Explain how work parties are used in an application
- List the standard work party classes available in Pega applications
- Configure a work party for a case type
- Populate a work party with case data

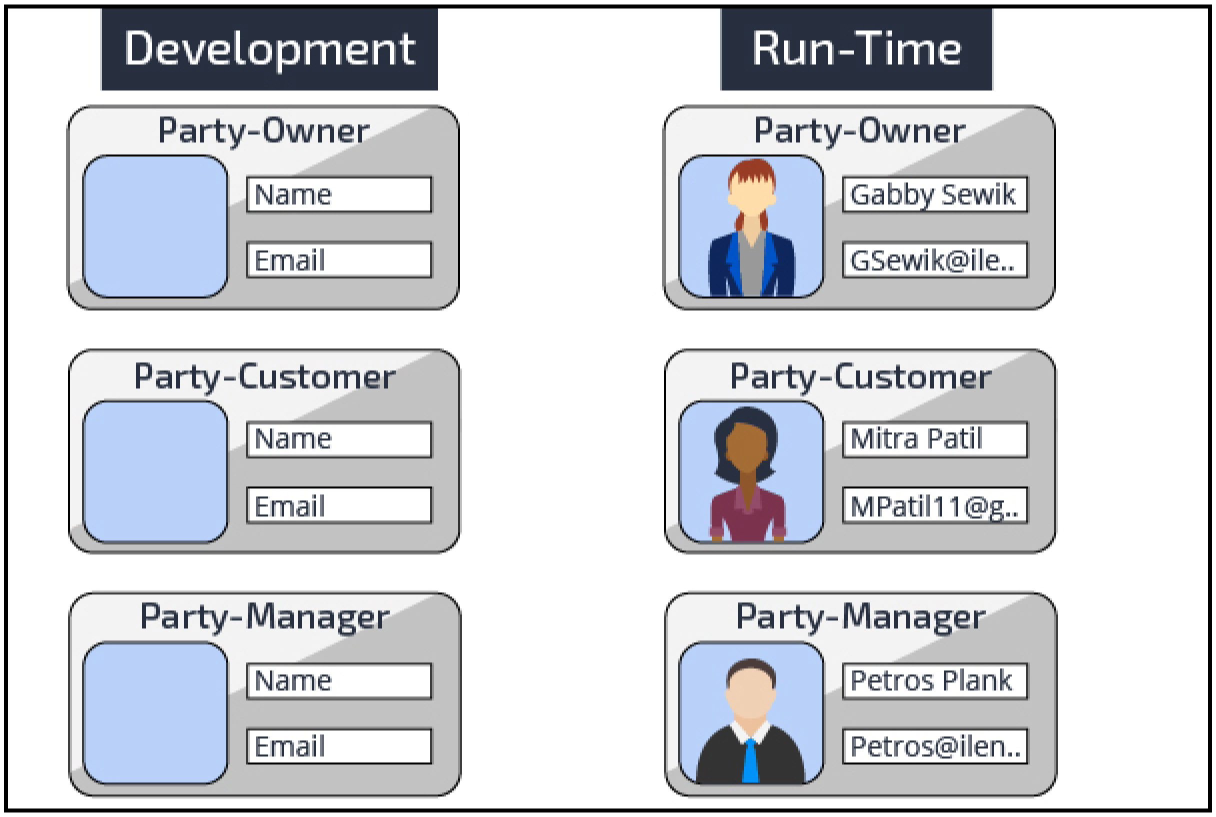
Work parties

Accomplishing work requires participants with different roles, such as managers, case workers, and customers. For example, an automobile insurance claim management process includes roles such as: the customer service representative (CSR) who creates the claim; the customer, on whose behalf the claim is filed; and the claims adjustor who reviews the claim.

In Pega, you create a **work party** to describe each role. Work parties allow you to refer to a case participant by role, without knowing any identifying information. For this reason, applications commonly use a work party as the recipient of correspondence. Also, work parties are sometimes used to assign work.

Imagine your automobile insurance claims management process requires that an email be sent to every customer who submits a claim. You would need the email address for every person submitting a claim in your application. There is no practical way to get this information prior to the claim. Instead, you model correspondence by role during development, and provide the identifying information, in this case, email address, for each case. So, you configure your claims management process to send email to the "Customer" party to confirm receipt of the insurance claim. For each case, the application populates the work party with information about the customer submitting the claim, and sends the email to the correct customer.

In the following image, you see how the work party is defined by a system architect during development, but the information about the party is provided only when a user processes a case.



KNOWLEDGE CHECK

ANSWER How is a work party used in an application?

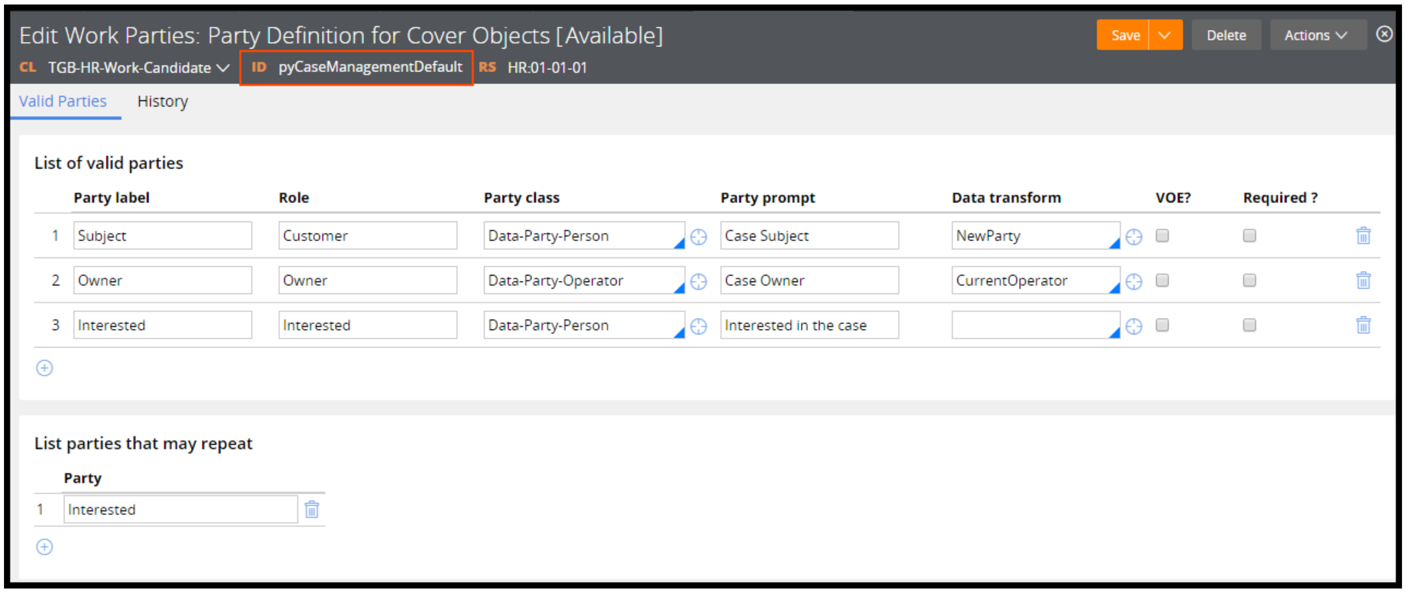
A work party represents a case participant. A work party allows you to refer to the participant by their role, and is often used to send correspondence during case processing.

How to add a work party to a case

Adding a work party to a case requires you to perform two actions: define the party using a Work Parties rule, and populate the party when performing a case. Pega creates a Work Parties rule named *pyCaseManagementDefault* for each case type. When you need to add or modify a work party for a case type, use the copy of *pyCaseManagementDefault* created in the class of the case type.

Define the work party using a work parties rule

First, you add the work party to the *pyCaseManagementDefault* work parties rule for the case type. Each row on the work parties rule lists a party available for a case type and describes the relationship of the party to a case. For example, a customer is the subject of a dispute resolution case, while a customer service representative (CSR) is the owner of the case and responsible for resolving the dispute.


Edit Work Parties: Party Definition for Cover Objects [Available]
CL TGB-HR-Work-Candidate
ID pyCaseManagementDefault
RS HR:01-01-01
Save
Delete
Actions
Valid Parties
History

List of valid parties

	Party label	Role	Party class	Party prompt	Data transform	VOE?	Required ?
1	Subject	Customer	Data-Party-Person	Case Subject	NewParty	☐	☐
2	Owner	Owner	Data-Party-Operator	Case Owner	CurrentOperator	☐	☐
3	Interested	Interested	Data-Party-Person	Interested in the case		☐	☐

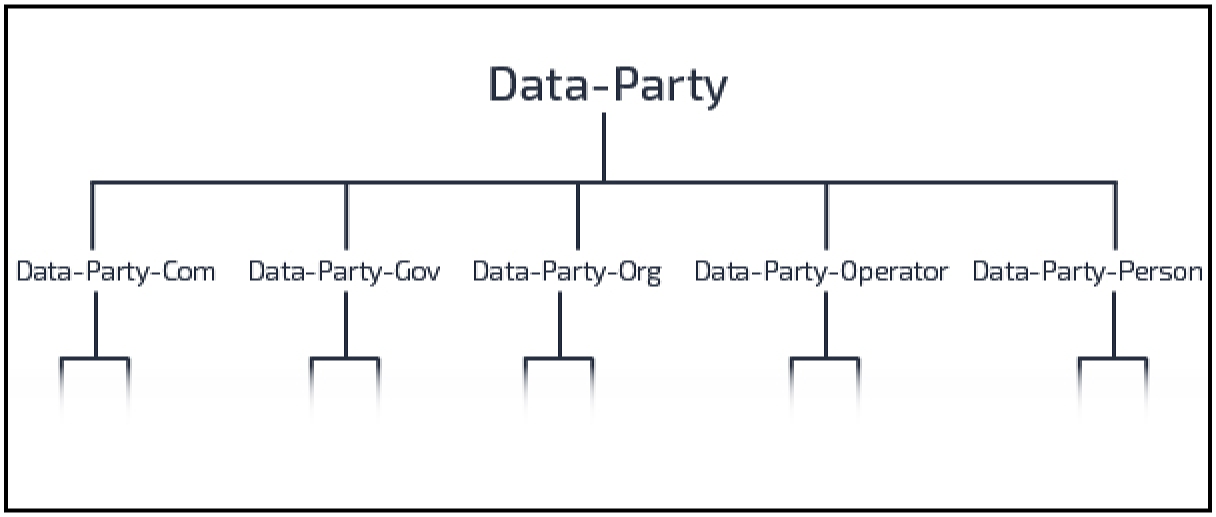
List parties that may repeat

	Party
1	Interested

When you create a work party, consider the following information:

- What type of case participant do you need to model? Is the participant an individual or an organization? Does the party work on the case, or just receive status updates?

  In Pega, work parties are derived from the *Data-Party* class. *Data-Party* contains rules to describe a party, such as properties to store identifying information. Pega also provides five child classes that build upon the rules in *Data-Party*. These child classes represent specific types of persons and organizations. The party class for a work party describes the person or entity participating in the case and determines how the participant interacts with other participants in the business process.

  - *Data-Party-Com* models a business that has a web domain ending in ".com", such as a corporation.
  - *Data-Party-Gov* models a government agency, such as the Department of Revenue.
  - *Data-Party-Org* models a non-profit organization that has a web domain ending in ".org", such as a charity.
  - *Data-Party-Operator* models a case participant with a Pega login and represents a case participant, such as a case worker or case manager.

    - *Data-Party-Person* models a case participant who lacks a Pega login, such as a customer. Typically this class is used to describe a work party that receives correspondence about a case, but who does not perform any actions on the case.
- What information do you need to know about the party? For example, do you only need the name and email address, or do you also need to know their marital status?
- How will you obtain the information to populate the party details? Do you expect the user to enter this information? Can you copy the information from session information or case data?
- Is the information available when the case is created? Or does the information become available during case processing?
- Are multiple instances of the party associated with the case? For example, does the case involve one customer, or several customers?

Populate the work party with case data

Next, configure your process to populate the work party with participant-specific information. Pega provides several options for populating work party information during case processing.

- If your cases already include the data needed to populate the party, use the *addWorkObjectParty* API activity. For example, a case creates several child cases to manage IT tasks prior to an employee's first day at a company. Each child case includes the HR representative assigned to the parent case as a work party. You use the *addWorkObjectParty* API activity to create an HR Partner work party using information already included in the case. The activity parameters allow you to specify the information needed to create the work party. Add this activity to the **Action** tab of a flow action rule, or call the activity by adding a utility shape to the appropriate process.
- If you want to allow case workers to add party information during case processing, use the *AddParty* flow action. Add this flow action to an assignment or stage as a local action. Users select the action during case processing to add information for a work party. For example, a customer wants to add a lawyer to a dispute resolution case. The CSR selects the flow action to add the lawyer as an interested party.
- If you want case workers to provide party information when creating a case, select the **VOE?** option for the party on the Work Parties rule form. When you enable this option, Pega automatically presents users with a form to enter the information needed to create the work party. For example, a CSR must enter customer information to open a new savings account for the customer. The initial form in the new account case allows CSRs to create a work party using information provided by the customer.

Configuring a work party for a case type

In Pega 7 applications, work parties are defined using the work parties rule *pyCaseManagementDefault*. To use a work party in a case, add the work party to the *pyCaseManagementDefault* rule in the class for the case type.

Defining a work party for a case type

Create a work party to describe the role of an individual or organization in a business process.

1. In the Application Explorer, expand the appropriate case type, then select **Process > Work Parties > pyCaseManagementDefault**.
2. Click the Add a party icon to add a work party to the rule.
3. In the **Party label** field, enter a unique name that indicates your work party's relationship to a case, such as Lawyer or Design Manager. The party label identifies the party role, and appears on the case to identify the work party. The party label must be a unique value. The **Owner** party label is reserved to describe the operator who creates the case.

   Once you exit the **Party Label** field, the **Role** field populates with the party label, with any spaces and special characters omitted. The role identifies the party on the clipboard. Each work party is a page within the WorkParty page group, and the role is used as the page index. For example, a work party with the role Customer is identified on the clipboard as **WorkParty(Customer)**.
4. Select an option from the **Party class** list to identify the class used to describe the party. The party class must be either the *Data-Party* class or one of its descendants.
5. Optional: Enter a label in the **Party Prompt** field that displays on user forms for a case. The Party Prompt field allows you to enter an optional descriptor for the party. You specify a party prompt to differentiate the party from another party with the same party label. For example, you can enter Real Estate to distinguish Real Estate Lawyers from Family Lawyers. On the work item entry form, a user sees Lawyer — Real Estate.
6. Optional: Press the Down Arrow key in the **Data transform** field and select the name of a data transform that runs when users add the work party to a case. The data transform defines initial values for the work party. You can use a data transform to copy information such as a first name or email address to the work party. Specify a data transform if you plan to create the work party upon creating a case. The data transform must be in the party class or a parent class.
7. Optional: Select the **VOE?** (Visible on Entry) check box to prompt users to add this work party every time a case is created. Selecting this check box adds required fields to the work item entry form, to collect information about the party.
8. Optional: Select the **Required?** check box to indicate that this work party must be present in every new case.
9. Optional: Add the party role to the **Party** list in the **List parties that may repeat** section to allow more than one instance of your work party to participate in a case.
10. Click **OK**.
11. Click **Save**.

Tip: You can also define a work party from the Case Designer. To do so, open a case type in Case Designer, navigate to the **Settings** tab, and click **Parties**.

Configuring a service level agreement

Introduction to Configuring Service Levels

When modeling service level agreements (SLAs), goals and deadlines are not always sufficient. Some SLAs include behavior for work that has passed its deadline and is considered late. And some assignments or cases are more urgent than others.

In these situations, you configure a service level rule. Service level rules allow you to model behavior for work that is past its deadline or to customize the urgency of work.

After this lesson, you should be able to:

- Explain how a service level rule relates to goals and deadlines.
- Describe the role of the Passed Deadline interval in service level processing.
- Explain how assignment urgency is calculated.
- Configure a service level rule to enforce processing expectations.

Service level agreement rules

Organizations often establish service level agreements to enforce obligations on the timely performance of work. These obligations range from informal promises of response times to negotiated contracts. A service level agreement establishes a deadline for when the specified work must be completed. Service level agreements may also establish a more aggressive goal to reflect [something]. For example, an IT help desk may have a requirement to respond to service requests within 24 hours, with a goal of 8 hours.

In Pega, you use a **service level agreement rule** to represent the performance expectations for an assignment, process, or entire case. When you establish a goal and deadline in Pega Express or the Case Designer, Pega creates a service level agreement rule for you. For processes, stages, and case types, goal and deadline intervals are often sufficient. In these situations, you can configure a service level from the Case Designer.

Service level agreements for assignments are often more complex. A service level agreement for an assignment may dictate actions to perform after a deadline passes. For example, a company establishes a deadline to respond to a customer inquiry in 48 hours. For any inquiry open after 48 hours, the company notifies a customer service manager every 24 hours until a representative responds to the customer.

A service level agreement for an assignment may also affect the start of the goal and deadline intervals. For example, a stock brokerage establishes a deadline of two hours to price assets in customer accounts. However, associates cannot begin pricing assets until after the stock market closes for the day. In this case, the start time for the service level agreement occurs after the stock market closes for the day. If the stock market closes at 4:30 PM, the deadline is 6:30 PM, even if the case reaches the assignment at 1 PM.

For complex performance obligations such as these, you configure the service level agreement rule, rather than using the Goal & deadline tab in the Case Designer.

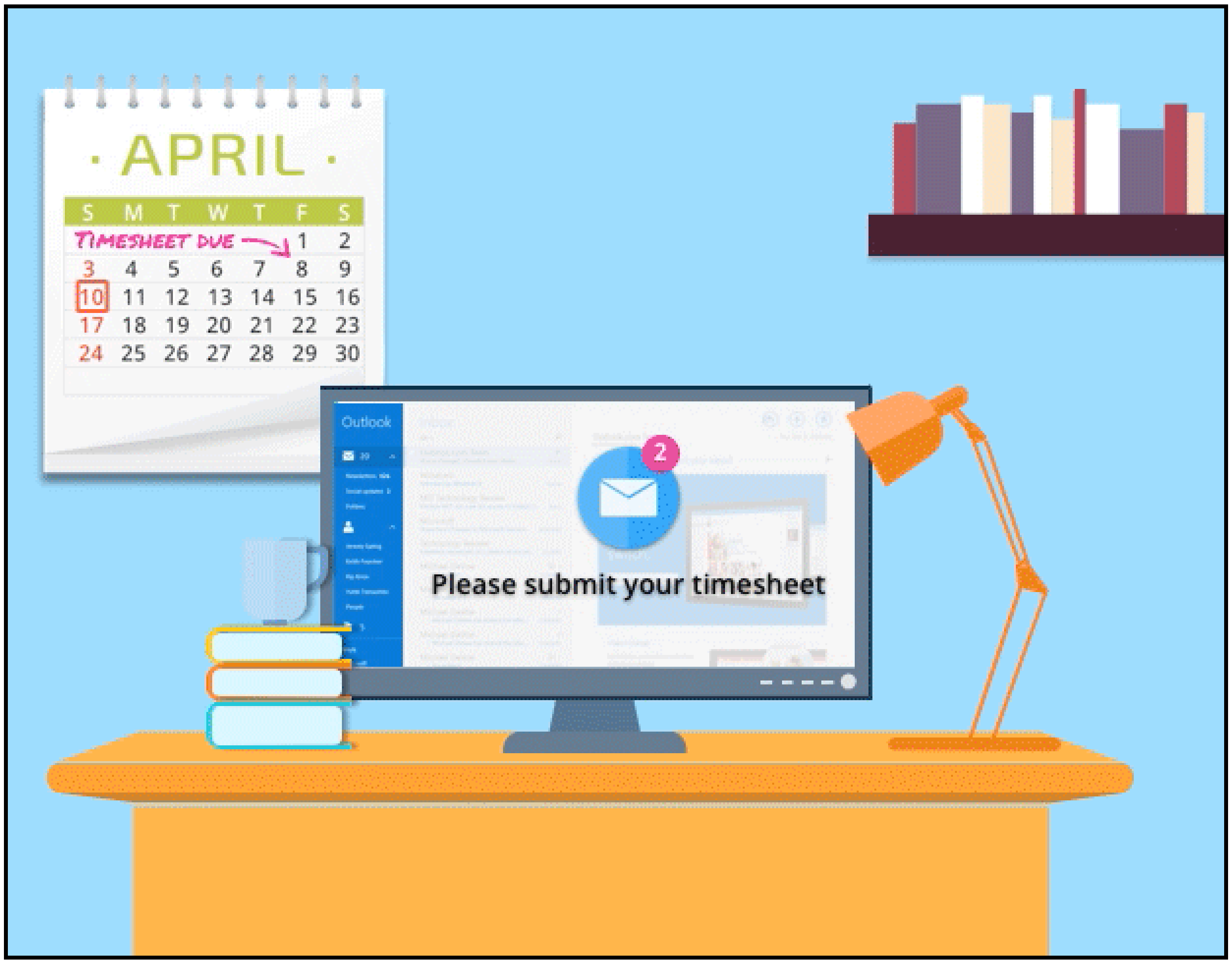
KNOWLEDGE CHECK

ANSWER What capabilities do you gain by configuring a service level agreement using the rule form, rather than using the Goal & deadline tab in the Case Designer?

The service level agreement rule form allows you to add behavior for assignments that are considered late, and to determine when an assignment is considered ready for the user to perform.

The Passed Deadline interval

Some service level agreements specify behavior for work that is considered "late". For example, a company requires that employees submit a record of their hours worked within two days of the end of the work week. This record of hours worked, called a time sheet, allows the Payroll department to credit the employee for hours worked during the week. Until an employee submits their time sheet, the Payroll department cannot pay the employee. To ensure that the employee is paid, the company must remind employees to submit their time sheet, even after the deadline to submit the time sheet is reached.



In Pega, you describe the behavior for late work using the **passed deadline** interval of a service level agreement rule. The passed deadline interval measures the time that has passed since the deadline for a still-open assignment.

Unlike the goal and deadline intervals, which occur once per assignment, the passed deadline interval repeats. This repetition allows you to continue to increase the assignment urgency and perhaps remind a user of a late assignment. You configure the passed deadline interval to repeat a fixed number of times, or repeat indefinitely until the user completes the assignment.

KNOWLEDGE CHECK

ANSWER How does the passed deadline interval differ from the goal and deadline intervals?

The passed deadline interval begins once the deadline interval ends. Also, the passed deadline can repeat, unlike the goal and deadline intervals.

How to adjust assignment urgency

To perform assignments in a timely manner, users must know the priority for each assignment. This allows users to perform the assignments that are most important first, before assignments that are of lower priority.

In Pega, the priority of an assignment is indicated by the assignment urgency. For each assignment, Pega calculates an urgency on the scale 0 to 100, with 100 the highest allowed urgency. The greater the urgency, the more pressing the assignment is. The assignment urgency allows users to determine which assignments to perform first. An assignment with an urgency of 60 is considered more pressing than an assignment with an urgency of 30.

The assignment urgency is also used by Pega to select assignments for users from team or department queues. When a user queries Pega for their next assignment, Pega identifies all of the assignments that the user is qualified to perform and selects the assignment with the greatest urgency.

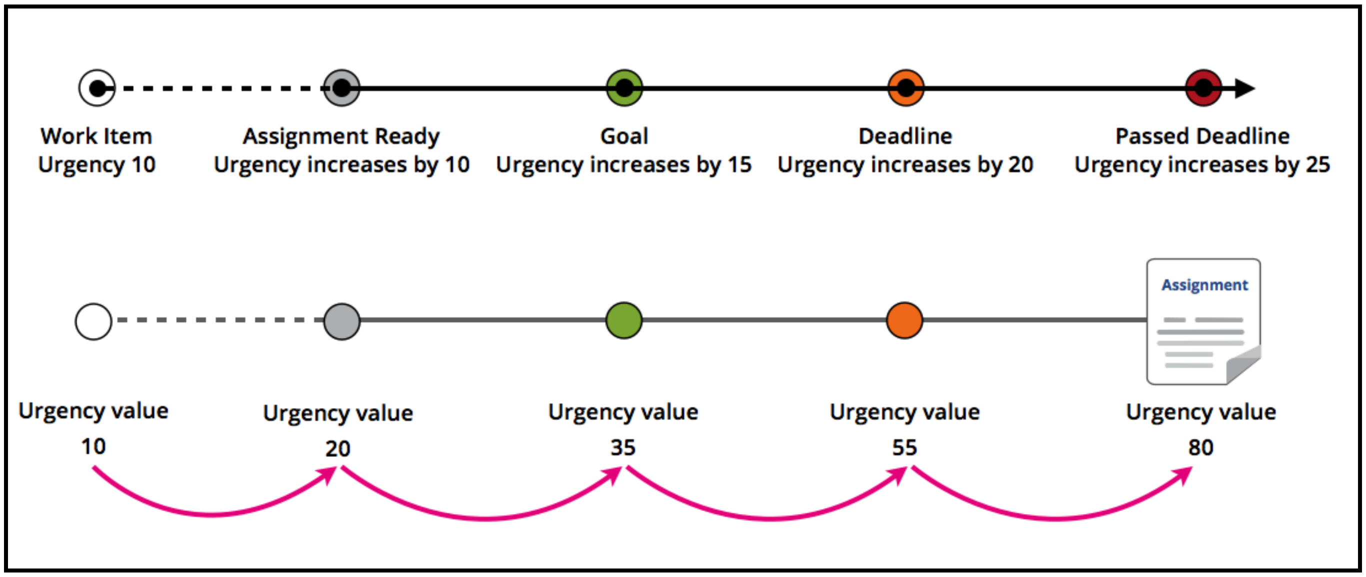
Once the urgency reaches 100, Pega ignores any further urgency adjustments. When this occurs, other service level agreement behavior is unaffected. For example, a deadline interval is configured to increment assignment urgency by 30 and notify the assigned user that the deadline has been exceeded. If the assignment urgency is 100 when the assignment reaches the deadline, Pega ignores the urgency increment but still sends the notification.

In Pega, the assignment urgency is recorded using the property *.pxUrgencyAssign*. Pega calculates *.pxUrgencyAssign*, the assignment urgency, as a sum of three input properties: *.pxUrgencyWork*, *.pxUrgencyAssignSLA*, and *.pyUrgencyAssignAdjust*.

- **.pxUrgencyWork** is the default urgency for the case type. The default value of *.pxUrgencyWork* is 10. This ensures that each assignment has a default urgency of 10, even when no service level is applied to the assignment. You change the value of *.pxUrgencyWork* to indicate that assignments for a specific type of case are more important than other cases. For example, if transaction dispute cases are a higher priority than other types of cases, you set the value of *.pxUrgencyWork* to 20 for transaction dispute cases. Assignments for transaction dispute cases then default to a greater urgency than assignments for other types of cases.
- **.pxUrgencyAssignSLA** is the urgency calculated from the service level rule. This value is the sum of

the initial urgency and the urgency increments for the goal, deadline, and passed deadline intervals. As an assignment ages in a workbasket or worklist, Pega increases the value of *.pxUrgencyAssignSLA* according to the configuration of the service level agreement.

For an Account review assignment, a service level agreement may establish an initial urgency of 10 and further increments of 15 for the goal, 20 for the deadline, and 25 for the passed deadline. When the user receives the Account review assignment, *.pxUrgencyAssignSLA* is set to 10, which increases *.pxUrgencyAssign* to 20. When each interval is exceeded, Pega increases *.pxUrgencyAssignSLA* as directed by the service level agreement, which further increases *.pxUrgencyAssign*.



- **.pyUrgencyAssignAdjust** is a manual adjustment for the assignment urgency. This value allows a user to increase the urgency of an assignment by running a local action. For example, a customer service representative (CSR) runs a local action to increase the urgency of an assignment if a customer reports that their credit card was stolen while on vacation. The CSR runs the local action, which increases the value of *.pyUrgencyAssignAdjust* to 50. This increases the overall assignment urgency, *.pxUrgencyAssign*, by 50 to increase the likelihood that the assignment is completed before other assignments.

Configuring a service level agreement rule

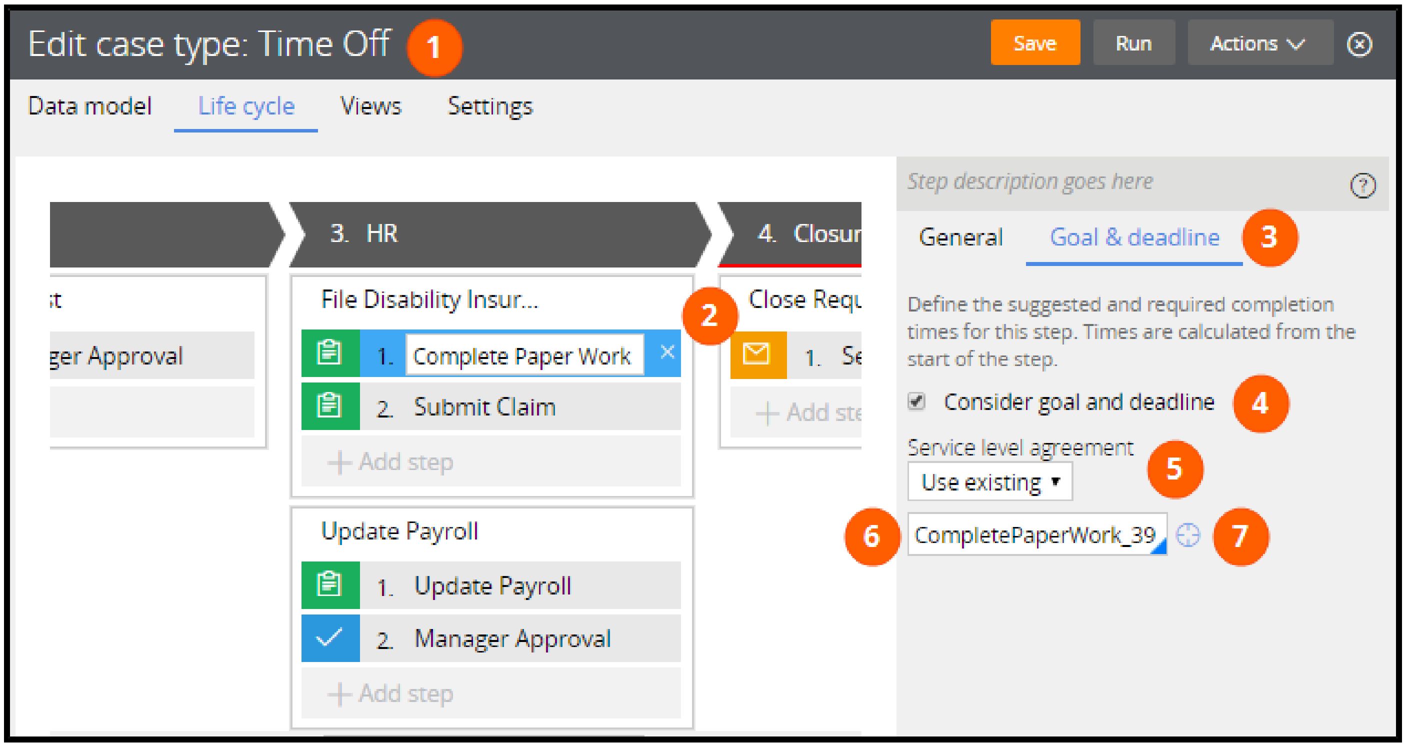
Configure a service level agreement rule to add a passed deadline interval for assignments considered late, or a delay before an assignment is considered ready for a user.

Add a service level agreement to an assignment in the Case Designer

Apply a custom service level agreement to an assignment or reuse a service level agreement already available in your application.

Follow these steps to add a service level agreement rule to an assignment:

1. Open the case type in the Case Designer.
2. Select the assignment to which to apply the service level.
3. In the properties panel for the step, click **Goal & deadline**.
4. Select the **Consider goal and deadline** check box.
5. From the **Service level agreement** drop-down list, select **Use Existing**.
6. In the empty field under the Service level agreement drop-down list, enter the name of the service level agreement rule to apply to the assignment.
7. To the right of the field containing the name of the service level agreement rule, click the **crosshair** icon.
8. If the rule has not been created, the New Record form opens. Click **Create and open** to create the service level agreement rule.



Configure the starting behavior for the service level

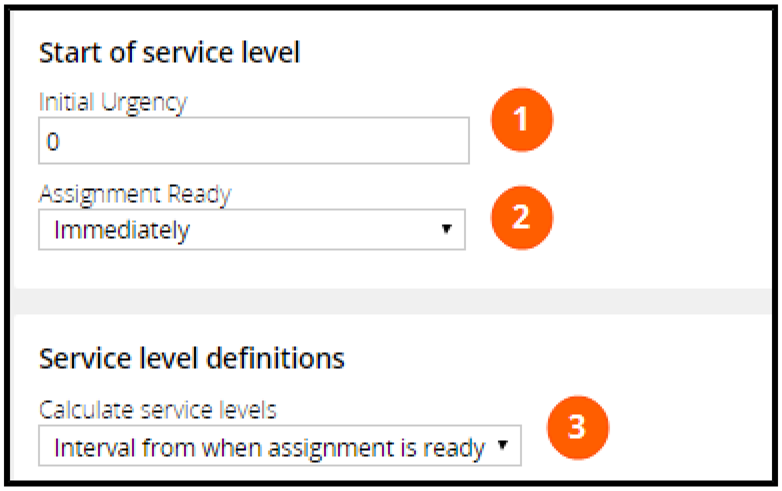
Specify an initial urgency for the service level agreement and any delay before tracking performance against the goal and deadline intervals.

Follow these steps to configure the starting behavior for service level agreement rule:

1. On the service level agreement rule form, in the **Initial Urgency** field, enter an initial urgency for the service level. The assignment urgency increments by the entered value when the assignment is ready for the user to perform.
2. From the **Assignment Ready** drop-down list, select when the assignment is considered available for a user to perform.

Option	Description	Usage
Immediately	Sends the assignment to a worklist or workbasket as soon as the case reaches the assignment. This is the default option.	Allow a user to perform the assignment immediately.
Dynamically defined on a Property	Delays sending the assignment to a worklist or workbasket until the specified delay interval elapses. Use the **Get Date Time From** field to specify a property that represents the optimal start time.	Delay the assignment until a specified time.
Timed delay	Delays sending the assignment to a worklist or workbasket until the specified delay interval elapses. Use the **Days**, **Hours**, and **Minutes** fields to enter the duration of the delay.	Delay the assignment for a specified amount of time.

3. From the **Calculate service levels** drop-down list, select whether to track the service level intervals against a fixed interval or against the value of a property. Use a property reference to adjust the intervals for each case. Use fixed intervals to ensure that the intervals are the same length of time for each case.


Start of service level

Initial Urgency: 0

Assignment Ready: Immediately

Service level definitions

Calculate service levels: Interval from when assignment is ready

Configure the goal and deadline intervals

Add a goal and deadline to measure whether the assignment is performed according to schedule.

Follow these steps to configure the behavior for either the goal or deadline interval:

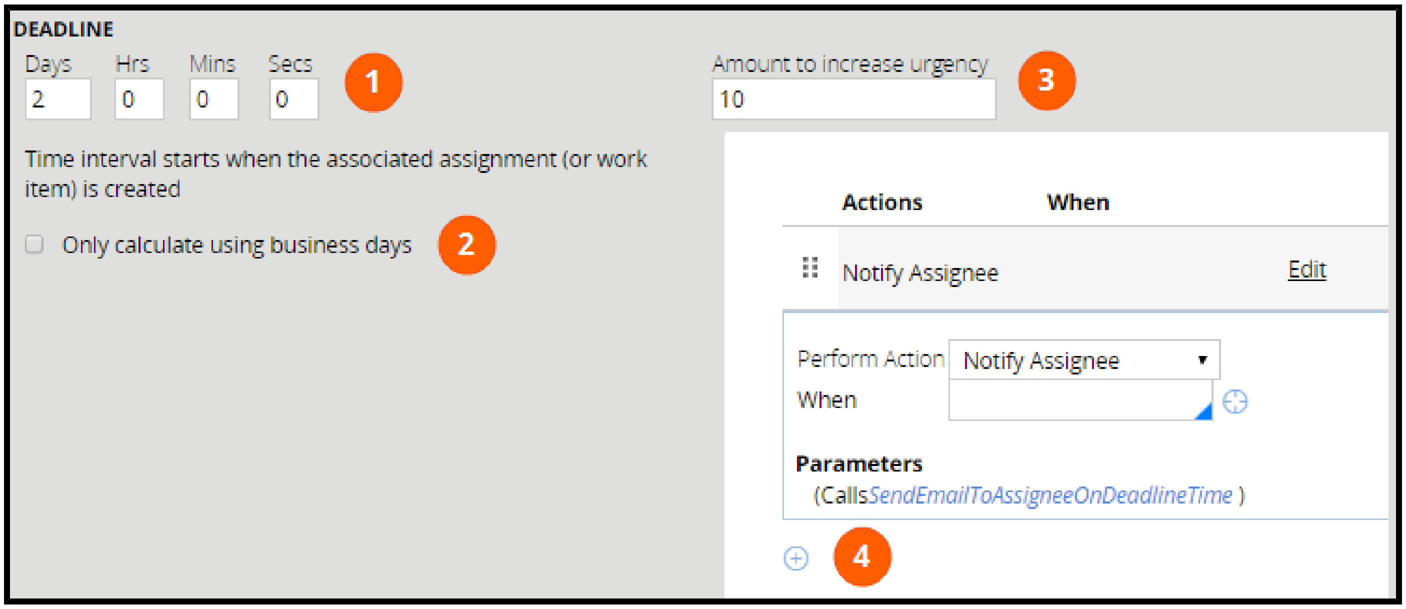
1. Enter a time for the interval. Depending on the selection in the **Calculate service levels** drop-down list, either specify the interval using the four fields labeled **Days**, **Hrs**, **Mins**, and **Secs**, or reference a property that represents the length of the interval.

2. Optional: click the **Only calculate using business days** check box to only measure elapsed time for the interval in business days. Enabling this option prevents Pega from counting non-work days, such

as weekends, against the interval limit. Pega determines non-work days from information in the user's operator ID record.

3. In the **Amount to increase urgency** field, enter an urgency adjustment for the interval. The assignment urgency increases by the specified amount until reaching 100.
4. Optional: click the **Add an action** icon to add an escalation action for the interval.

   From the **Perform Action** drop-down list, select an escalation action to perform when the interval ends. If necessary use the **When** field to use a when condition or when rule to determine whether to perform the escalation action.


DEADLINE
Days 2
Hrs 0
Mins 0
Secs 0
1
Amount to increase urgency 10
3
Time interval starts when the associated assignment (or work item) is created
Only calculate using business days
2
Actions
When
Notify Assignee
Edit
Perform Action Notify Assignee
When
Parameters
(Calls SendEmailToAssigneeOnDeadlineTime)
4

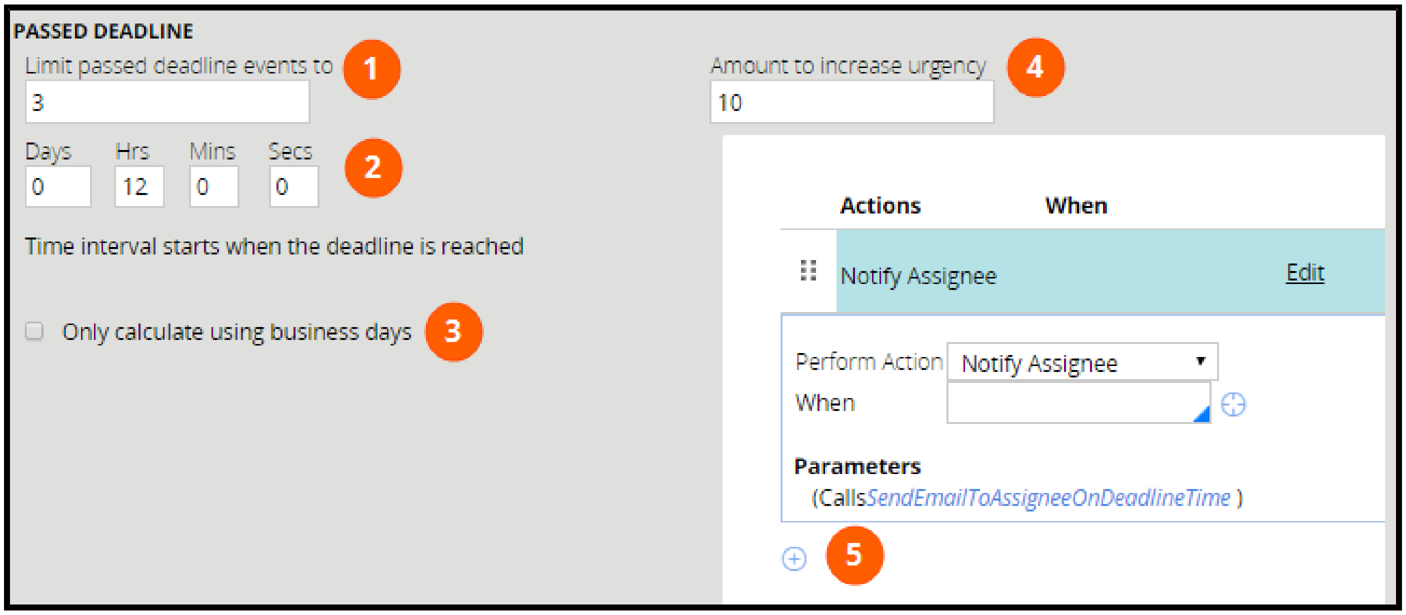
Configure the passed deadline interval

Add a passed deadline interval to describe behavior for assignments that are considered late.

Follow these steps to configure the behavior for the passed deadline interval:

1. In the **Limit passed deadline events to** field, enter the number of passed deadline events to apply to the assignment. To apply the passed deadline behavior indefinitely until the assignment is completed, leave the field empty.
2. Enter a time for the interval. Depending on the selection in the **Calculate service levels** drop-down list, either specify the interval using the four fields labeled **Days**, **Hrs**, **Mins**, and **Secs**, or reference a property that represents the length of the interval.
3. Optional: click the **Only calculate using business days** check box to only measure elapsed time for the interval in business days. Enabling this option prevents Pega from counting non-work days, such as weekends, against the interval limit.
4. In the **Amount to increase urgency** field, enter an urgency adjustment for the interval. The assignment urgency increases by this value each time a passed deadline cycle completes, until the assignment urgency reaches 100. Once the assignment urgency reaches 100, further urgency adjustments are ignored, though escalation actions are processed, and the Passed Deadline interval repeats if configured to do so.
5. Optional: click the **Add an action** icon to add an escalation action for the interval.

From the **Perform Action** drop-down list, select an escalation action to perform when the interval ends. If necessary use the **When** field to use a when condition or when rule to determine whether to perform the escalation action.



PASSED DEADLINE

Limit passed deadline events to 3

Days	Hrs	Mins	Secs
0	12	0	0

Time interval starts when the deadline is reached

Only calculate using business days

Amount to increase urgency 10

Actions	When	
Notify Assignee		Edit

Perform Action: Notify Assignee

When:

Parameters

(Calls *SendEmailToAssigneeOnDeadlineTime*)

Routing assignments

Introduction to Routing Assignments

An efficient process design routes assignments to users who can best perform the work. Sometimes, the correct user is an individual who has a specific role. At other times, anyone in a specific group of users can perform the assignment. Design your process so that it routes assignments to the best qualified users. This approach helps ensure that work is done correctly and completed on time.

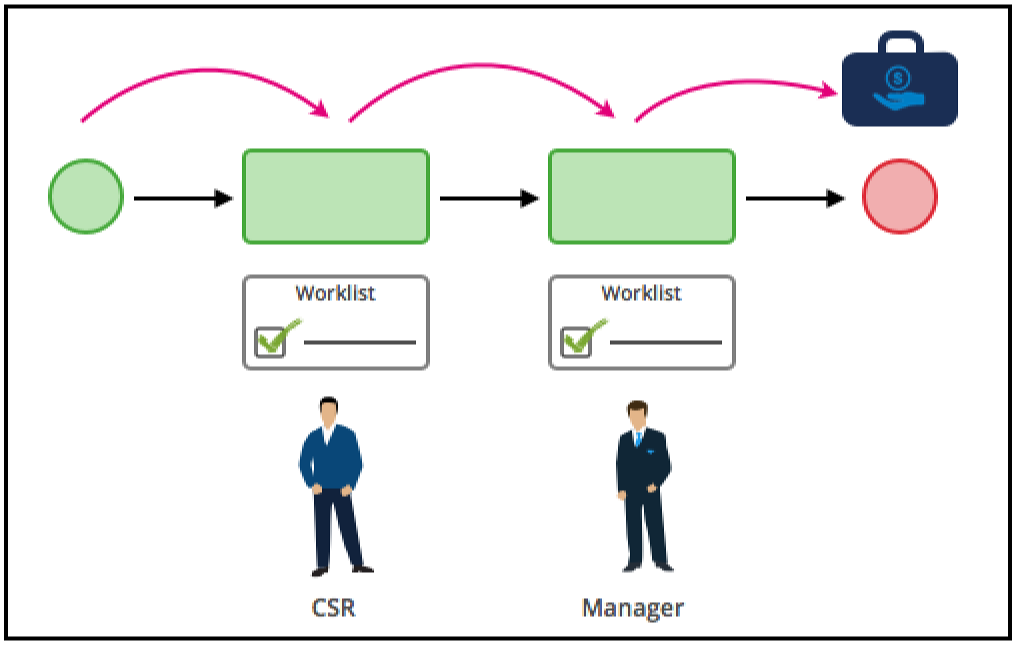
After this lesson, you should be able to:

- Explain the role of worklists in routing
- Explain the role of workbaskets in routing
- Explain the role of a router in routing assignments
- List common standard Pega routers
- Route an assignment to the appropriate user

Routing

Routing identifies who will work on an assignment as a case moves through a life cycle. When you create a case , the case initiates a starter flow. The starter flow moves from one step to another until it reaches an assignment. The starter flow stops at the assignment and does not continue unless a user performs an action. This action completes the assignment task. Routing instructions in the assignment control who performs the assignment task.

In most applications, more than one user works on a case until the case is resolved. For example, a customer requesting a car loan calls a customer service representative (CSR). The CSR enters the information collected from the customer. Then, the CSR routes the case to the manager to work on the customer’s request. The manager is now tasked to review the information. Routing information in the CSR assignment routes the assignment to the manager.



Worklists and workbaskets

Users complete assignments as a case moves toward resolution. When you configure the router setting in an assignment, you specify either a specific user or a queue accessible to a group of users.

Defining worklists

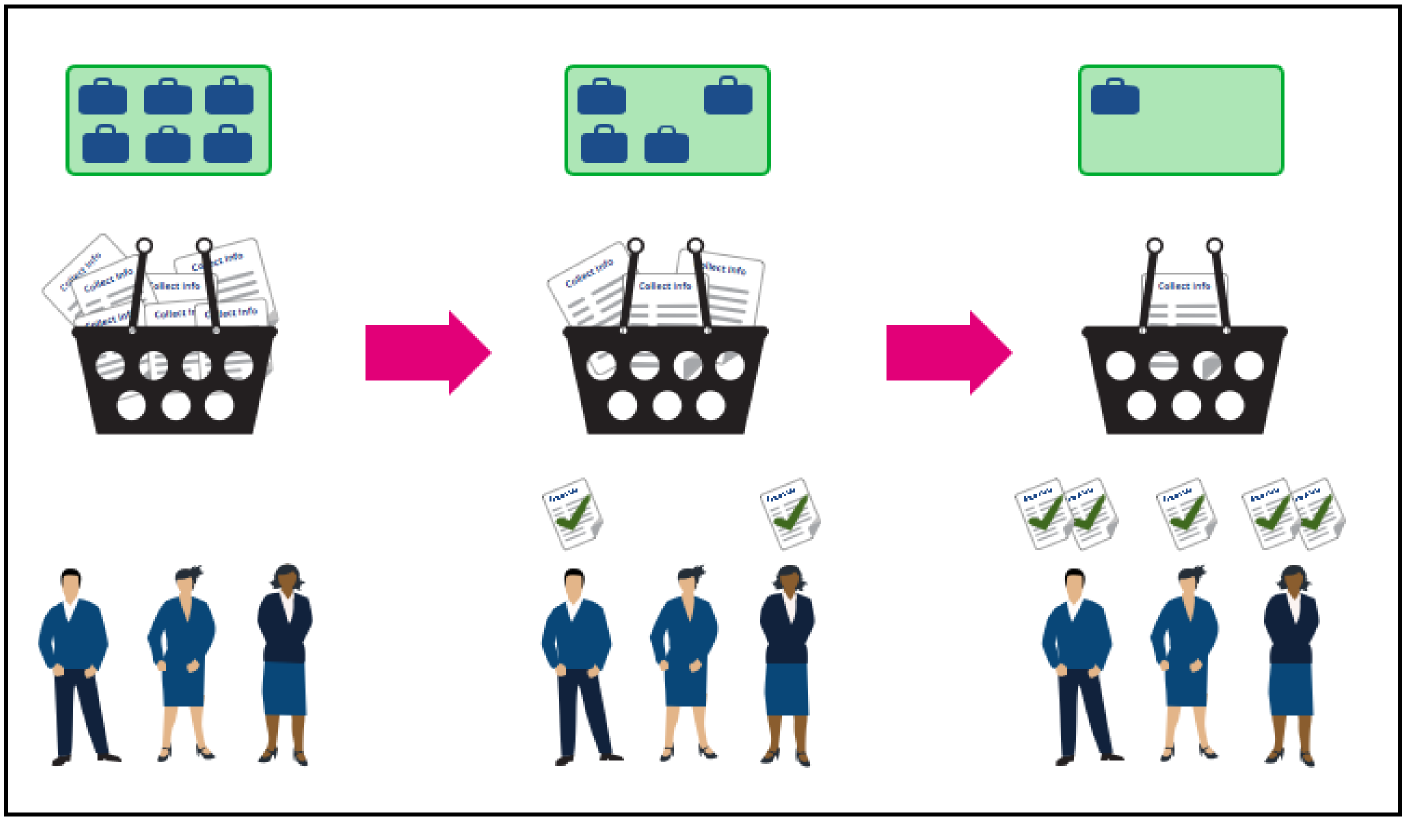
A **worklist**is a list of all open assignments for specific users. A user see an assignment on their worklist until the user performs an action that completes the assignment. Each user has a worklist that the user can access from user portals. For example, an assignment might allow only a human resources manager to approve employee time off requests. The assignment appears on the manager's worklist.

Note: Managers can access and assign work to the worklists of users who report to the managers.

Defining workbaskets

When assignments are queued for a team of users, the assignments are stored in **workbaskets**. A team associated with a workbasket is called a **work group**

Assignments stay in the workbasket until a team member selects an assignment or a manager sends an assignment to a specific user. For instance, any user who belongs to the company benefits team can add a dependent to an employee's medical insurance policy. Requests for updates to an employee's benefits would be queued and stored in the workbasket for the employee benefit team. A user selects an assignment from the workbasket and begins work.



Summary

The following table describes the relationships between who receives the work and how the assignment is accessed.

Who receives the assignment	How assignment is accessed
A user	Worklist
A team	Workbasket

Routers

When you add an assignment to your process, you specify the assignment's router. A router is a special type of activity that progresses an assignment based on the routing destination and assignment type. The activities use parameters to control routing behavior.

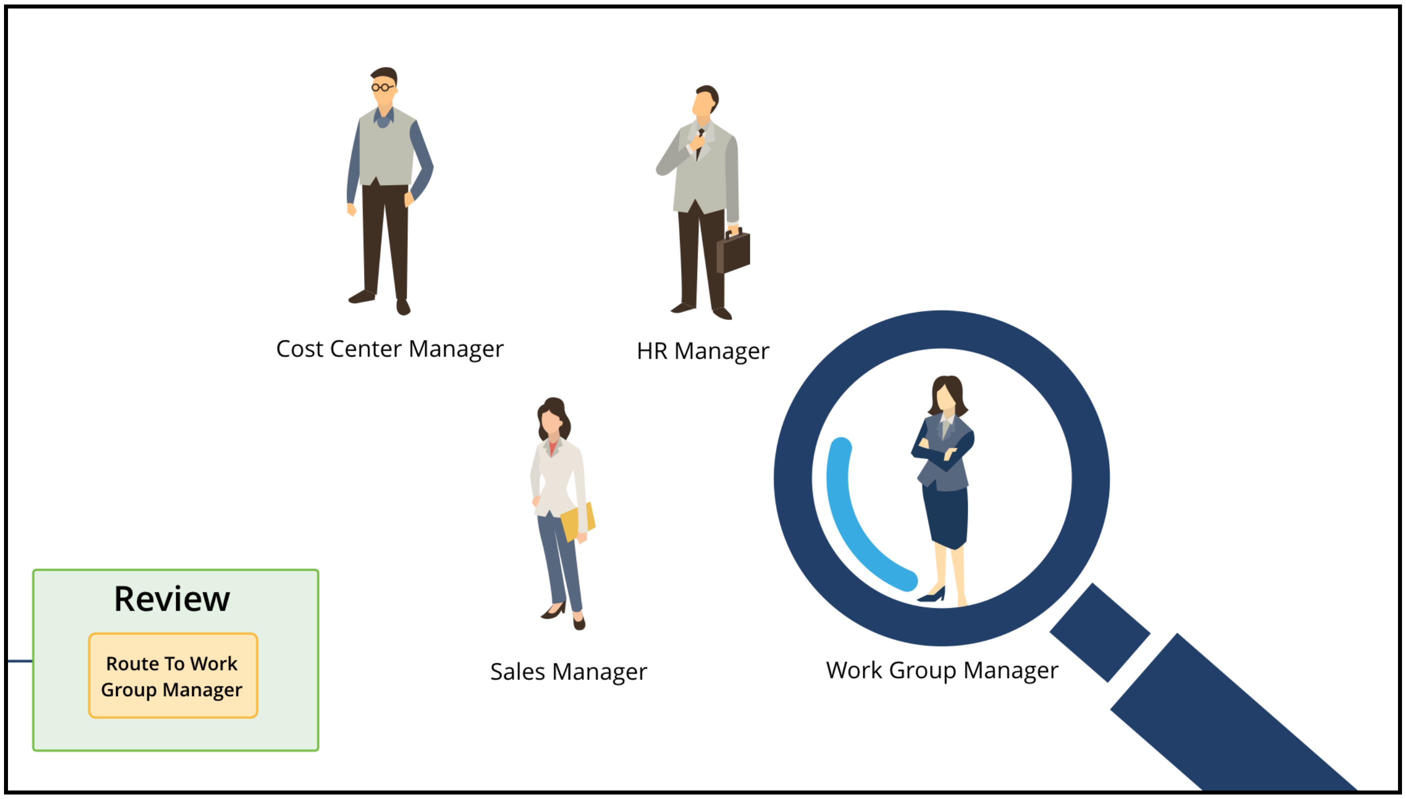
Standard routers

Two common routers send assignments to an individual (ToWorklist) or to a workbasket (ToWorkbasket). Worklist assignments use a user ID as the destination. Workbasket assignments use a workbasket name as the destination.

Other routers have parameters that are configured to suit specific requirements. Routing by user roles or user skills are two examples.

Routing by user roles

Routers can use user roles to control routing behavior. For example, use the ToWorkGroupManager router to route assignments to a work group manager defined in your application. The user with the work group manager role receives the assignment. You do not need a user ID to identify the manager.

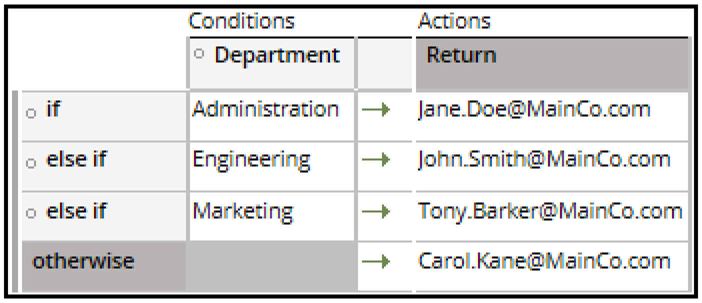


Routing by user skills

Routers can use user skills to direct assignments. For example, a routing activity named ToSkilledGroup has parameters for the skill rating of the users belonging to the group. For instance, a user in the group might speak German in order to process a loan request created in a German-speaking location. When you use the ToSkilledGroup activity, the user with the skill (German language) and the maximum skill rating performs the assignment.

Decision routing

Routers use the results of a process decision rule to route assignments. The ToDecisionTable router uses the results of a decision table to route an assignment. For example, assume you specify the following decision table in the ToDecisionTable router. If the Engineering department handles the case, the assignment routes to John Smith.



	Conditions		Actions
	Department		Return
if	Administration	→	Jane.Doe@MainCo.com
else if	Engineering	→	John.Smith@MainCo.com
else if	Marketing	→	Tony.Barker@MainCo.com
otherwise		→	Carol.Kane@MainCo.com

Identifying Pega routers

Pega provides a large set of preconfigured routers to suit specific requirements. For descriptions of the routers provided by Pega, see the Route Activities section in the help topic Standard activities for use in flows (interactive).

Configuring routing

You can specify routing directions in any one of the following locations:

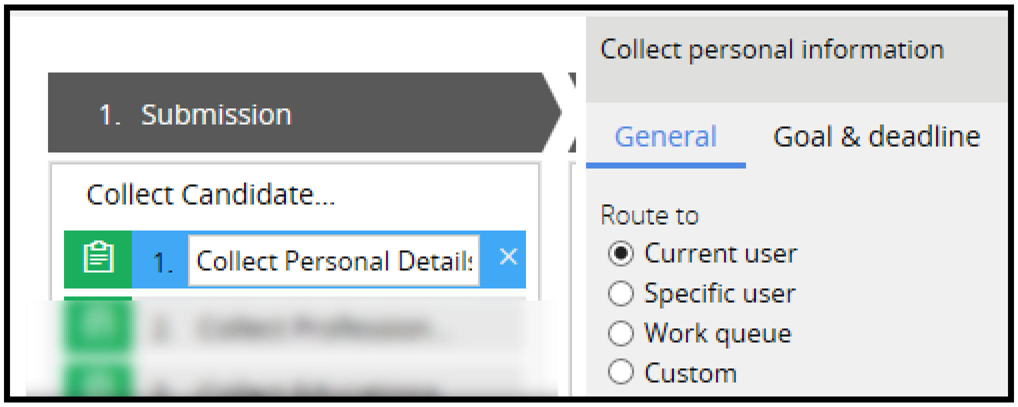
Where you define routing	When to use
Case life cycle	When adding assignment steps to your design
Assignment shape	When working with assignment shapes in an flow diagram
Approval process SmartShape	When determining where assignments will be routed in an Approval process

Configuring routing in the case life cycle

Specify assignment routing on a case life cycle diagram as follows:

1. Select an assignment step.
2. On the General tab of the contextual property panel, select a **Route to** option to indicate how to

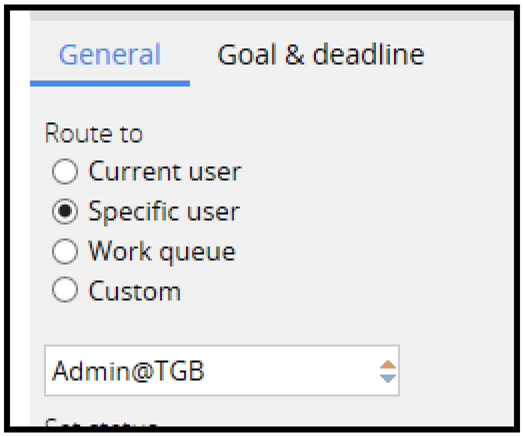
route the assignment at run time.



Route to
(●) Current user
( ) Specific user
( ) Work queue
( ) Custom

The **Route to** options are as follows:

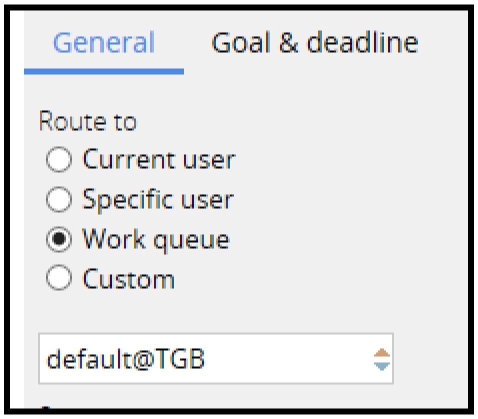
- Click **Current user** to route the assignment to the worklist of the user who last updated the case.
- Click **Specific user** to route the assignment to the worklist of another user in your application. Then, in the autocomplete field, press the **Down Arrow** key and select the name of a user.



Route to
( ) Current user
(●) Specific user
( ) Work queue
( ) Custom

Admin@TGB

- Click **Work queue** to route the assignment to a work queue that is processed by users with the same role. Then, in the autocomplete field, press the **Down Arrow** key and select the name of a work queue.



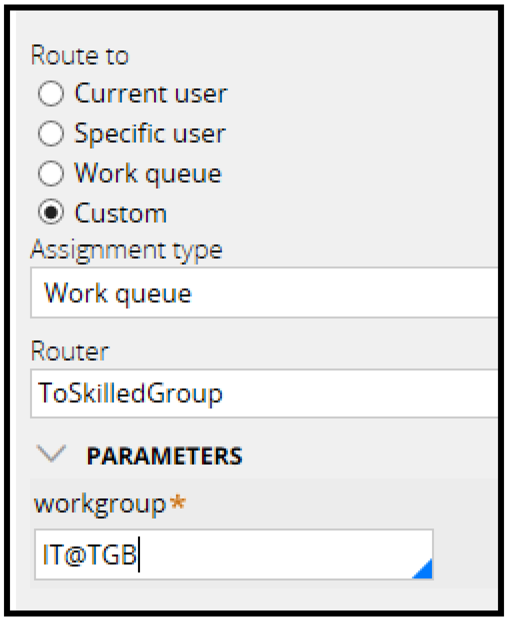
Route to
( ) Current user
( ) Specific user
(●) Work queue
( ) Custom

default@TGB

- Click **Custom** to provide custom routing options.

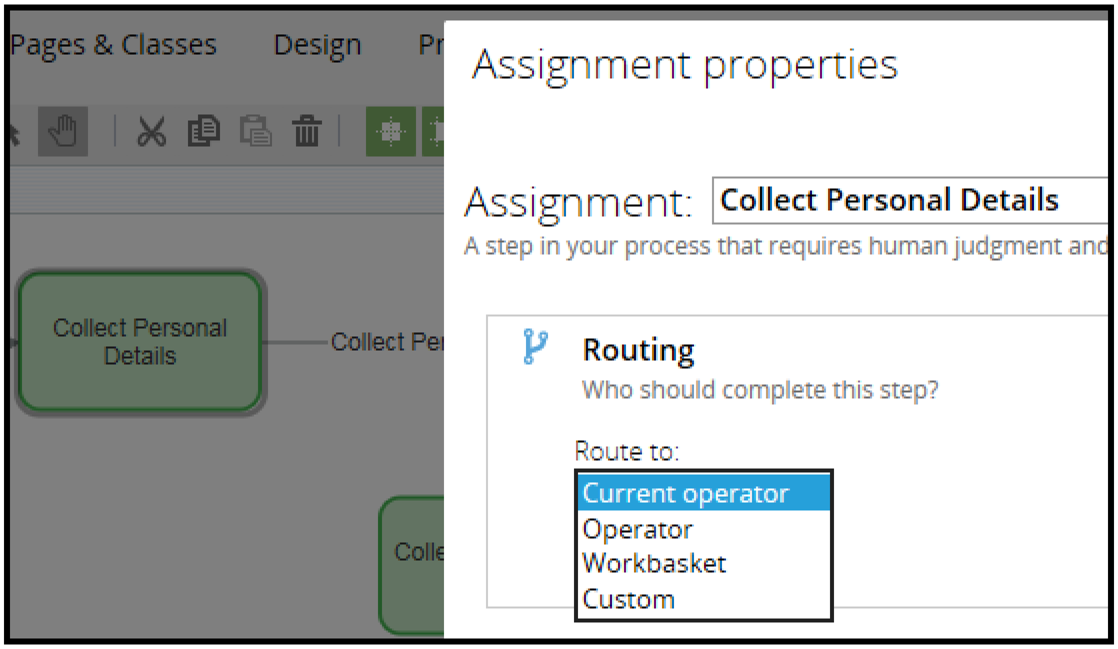
  Then, do the following:

3. In the Assignment type list, select an activity that creates an assignment. This list is populated with activities that have the **Usage** field set to Assign on the Activity form.
4. In the **Router** field, press the **Down Arrow** key and select the name of an activity that determines how the assignment is routed. For example, you can use the ToDecisionTree activity to route an assignment based on the value that is returned by a decision tree.
5. If the routing activity accepts parameters, pass values for those parameters by entering values in the fields in the Parameters section.


Route to
Current user
Specific user
Work queue
[✓ Custom]
Assignment type: Work queue
Router: ToSkilledGroup
PARAMETERS
workgroup*: IT@TGB

Configuring routing in an Assignment shape

To specify routing on an Assignment shape, right-click the shape to open the Assignment properties dialog. Use the drop-down list in the Routing section to select a routing option. The options are the same as the options that appear in the case life cycle General tab.



Assignment properties
Assignment: Collect Personal Details
A step in your process that requires human judgment and
Routing
Who should complete this step?
Route to:
Current operator
Operator
Workbasket
Custom

Note: The menu values Current operator, Operator, and Workbasket are the same as values named Current user, Specific user, and Work queue used in the case life cycle.

Configuring routing in an Approval process

You can specify routing for assignments within an Approval process.

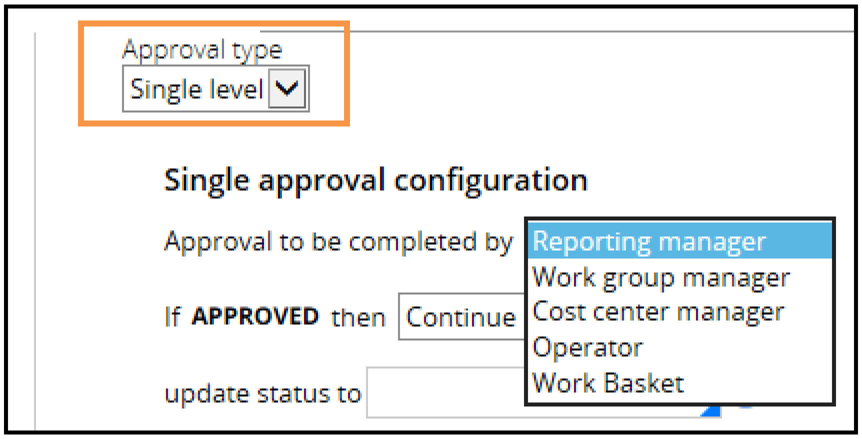
On an Approval Smart Shape, right-click to open the Approval properties panel to specify the routing behavior. The process has two levels of approval: single or cascading.

Single-level approval

When only one level of approval is required, you can route an assignment to a specific operator, workbasket, or type of approval manager.

To specify single-level routing, do the following:

1. In the **Approval type** field, select Single level.
2. In the **Approval to be complete by** field, select the user or workbasket that receives the approval assignment.



Approval type: Single level

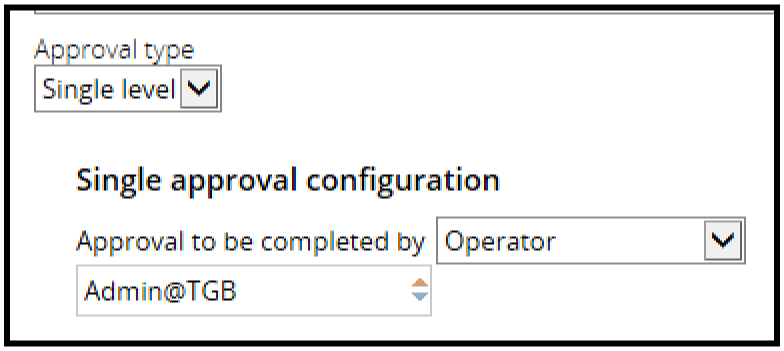
Single approval configuration

Approval to be completed by: Reporting manager

If **APPROVED** then Continue

update status to

3. If you select Operator or Work Basket, select the operator ID or workbasket name in the autocomplete field.



Approval type: Single level

Single approval configuration

Approval to be completed by: Operator

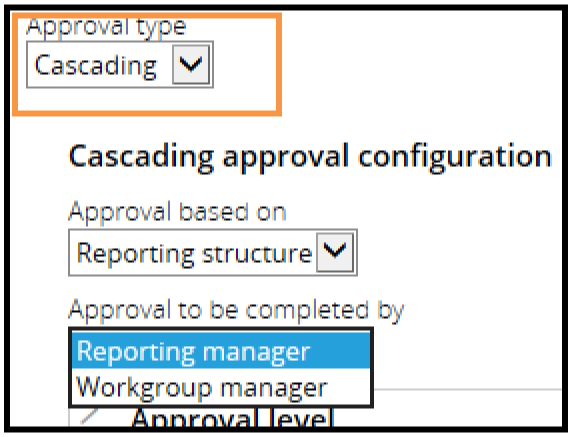
Admin@TGB

Cascading approval

When you use a cascading approval process, you route assignments to managers based on the current user's reporting structure, or using the results of an authority decision table.

To route assignments based on a cascading process, do the following:

1. In the **Approval type**field, select Cascading.
2. In the **Approval based on** field, select Reporting structure.
3. Select a manager in the **Approval to be completed by** field.



Approval type
Cascading

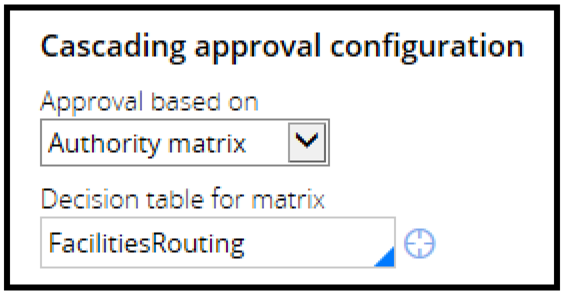
Cascading approval configuration

Approval based on
Reporting structure

Approval to be completed by
Reporting manager
Workgroup manager

Approval level

4. When you route assignments based on an authority matrix, select a decision table in the **Decision table for matrix** field. The outcome of the decision table determines where the assignment is routed.



Cascading approval configuration

Approval based on
Authority matrix

Decision table for matrix
FacilitiesRouting

Configuring correspondence

Introduction to Configuring Correspondence

During a business process, organizations often need to communicate with parties associated with a case. This communication ranges from simple notifications of assigned tasks to complex communications that contain case-specific data and calls-to-action. Adding correspondence to a business process keeps stakeholders and case workers engaged throughout the business process.

In Pega, you configure emails, letters, fax, and text messages with correspondence rules. You can configure your application so that the system can send correspondence automatically or enable users to send correspondence manually. Correspondence rules allow you to add case data to your communication to provide richer, more relevant communication to stakeholders and case workers.

After this lesson, you should be able to:

- Describe the process for creating correspondence.
- Configure a correspondence rule.
- Incorporate case content into correspondence.
- Send correspondence from a business process.

How to configure correspondence rules

Create correspondence rules to define, in HTML, templates for the content of outgoing correspondence. Each correspondence rule contains text for one type of correspondence such as email, letter, SMS phone text, or fax. JavaServer Pages (JSP) tags or directives allow correspondence to incorporate property values and calculations.

Informally, correspondence rules are sometimes called templates, as they define form letters for property values.

For simple notifications, you might add text directly in the rule. For richer content, you can enter dynamic fields that reference properties, or rules such as sections, paragraphs, or correspondence fragments. During flow processing, the system uses the source content in the correspondence rule to generate a customized message for the recipient.

Identify the correspondence type

When you create a correspondence rule, you specify the correspondence type as a key identifier. A correspondence type rule indicates whether a piece of correspondence is a printed letter, fax, email, or SMS phone text. Each type is associated with a different Data- subclass, such as Data-Corr-Email, that holds the content of correspondence items.

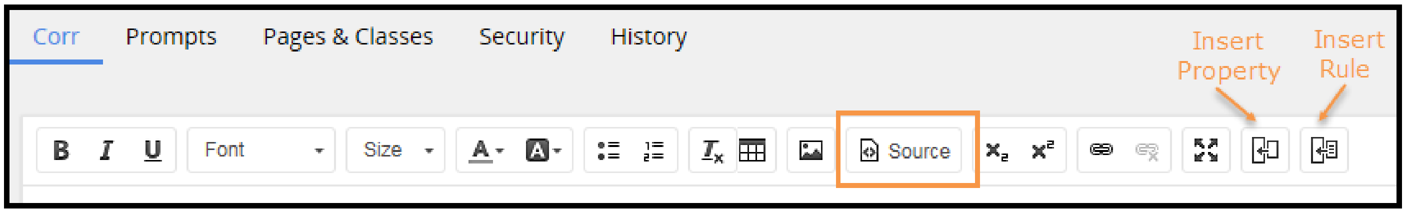
Be careful when selecting the correspondence type. Your specification should state who is receiving the correspondence and how the correspondence is sent. Before sending correspondence, the system references work party contact information to make sure the recipient can receive the correspondence. For example, the system cannot send email to a customer party that does not have an email address.

Add the content

Add the message for the correspondence on the **Corr** tab of the rule form. The tab presents a rich text editor in which you create the source content for the correspondence.

The toolbar includes controls for setting text styles, spell checking, list formatting, alignment, and for inserting images and graphic elements.

The toolbar also contains controls for viewing the source HTML in the text area, and for inserting properties and rules that contain correspondence content. You can use a combination of text and referenced content sources to create your correspondence.



- Click **Source** to view or update the source as HTML codes. In source mode, you can add HTML elements and JSP tags directly. For example, you can add the when JSP tag to conditionalize a portion of the HTML code.

  Note: When the correspondence is an email, the outgoing email includes HTML formatting, even if no HTML elements appear within the source. When the correspondence is phone text, the message does not contain HTML formatting.

- Click the **Insert Property** icon to include properties in your application such as pyID, LastName, and Department. For example, you may want to inform the recipient that the case is currently under review by the auditing department. You can reference the property .pyID to insert the case ID into the correspondence, rather than providing a generic message.

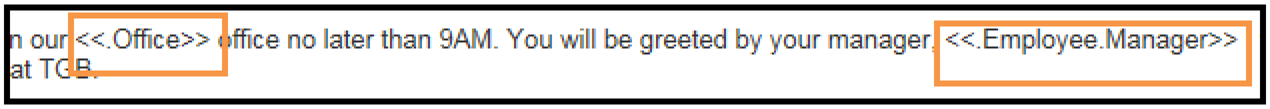
- Click the **Insert Rule** icon to include content in other rules such as paragraphs, sections, and correspondence fragments. You can also include other correspondence rules.

  Paragraphs present formatted text that can include colors, fonts, styles, and images. Paragraphs allow you to reuse content that is used elsewhere. For example, you use a paragraph rule to present instructions on a form. You can then use that paragraph rule in correspondence to describe the action the recipient is expected to perform. Referencing shared content ensures that your correspondence includes the most current version.

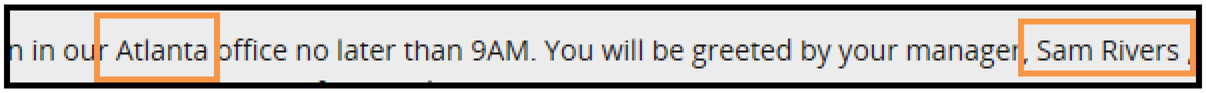
  Sections allow you to reproduce part or all of a form in the correspondence. Use a section to achieve greater control over the positioning of content in the correspondence.

  Correspondence fragments are useful for reusing boilerplate content, such as a mandatory disclosure or links to an organization's social media channels.

The editor uses angle << >> brackets to mark properties and rules. During flow processing, these elements are replaced with the values. The following example shows inserted properties .Office and .Employee.Manager.


n our <<.Office>> office no later than 9AM. You will be greeted by your manager, <<.Employee.Manager>>
at TGB.

When the correspondence is sent, the system replaces the properties with their values.


n in our Atlanta office no later than 9AM. You will be greeted by your manager, Sam Rivers ,

How to configure correspondence in a business process

When you have created your correspondence, you can configure your application to send the correspondence in various ways. For example, you can automatically send an email to a user when a new assignment has arrived in the user's worklist. You can automatically notify a customer when a request for a loan has been approved.

You can configure your application to automatically send:

- Emails, faxes, letters, or text messages when a case advances in the process using the Send Email Smart Shape or a correspondence Utility.
- Email notifications when a case reaches an assignment.
- Email service level (SLA) notifications when an assignment has gone past its goal or deadline.

You use flow actions so that users can select and send correspondence as needed.

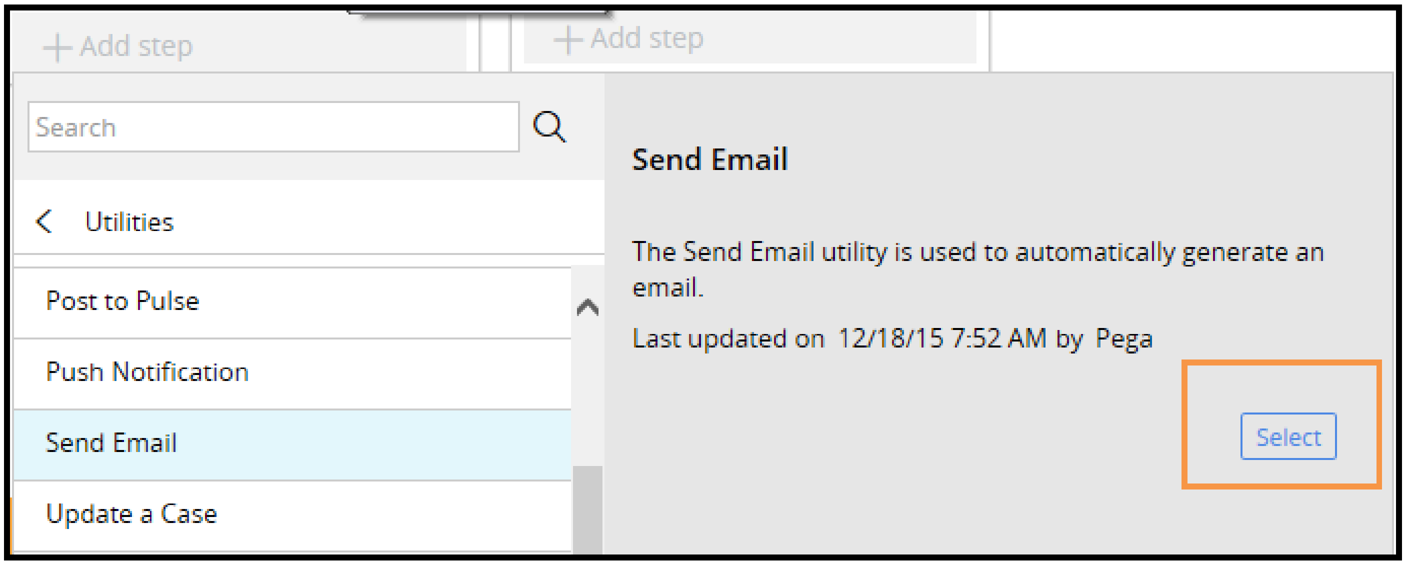
Using the Send Email Smart Shape

You can add the Send Email Smart Shape to your flow diagram to automatically send emails as a case advances during the business process. This Smart Shape is useful if you want to notify or send information to users after an action has been performed on a case. For example, the system can notify a manager that, as the result of an automated decision, a purchase request does not need approval.

You can use the Smart Shape to create simple text messages that you configured using the rich text editor. You can also send emails that you configured in a correspondence rule.

You can add the Smart Shape directly to your flow diagram. You can also add the Send Email step in the case life cycle — the system automatically adds the shape to your diagram.

To add a Send Email Smart Shape from the case life cycle, select **Utilities > Send Email**, and then click **Select**.

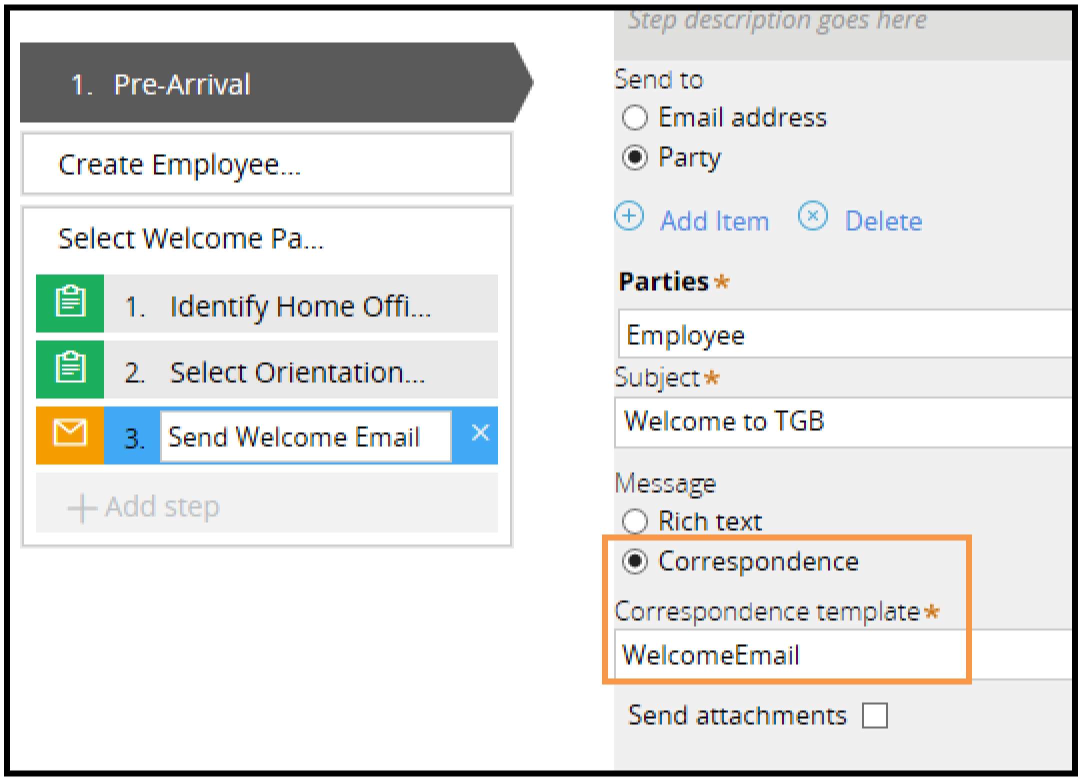


Send Email

The Send Email utility is used to automatically generate an email.

Last updated on 12/18/15 7:52 AM by Pega

The system displays the Send Email properties panel on the right side of the case life cycle. To use a correspondence template that was previously created, select **Correspondence**and select the rule name.



Send to
Email address [✓ Party]

Parties*
Employee

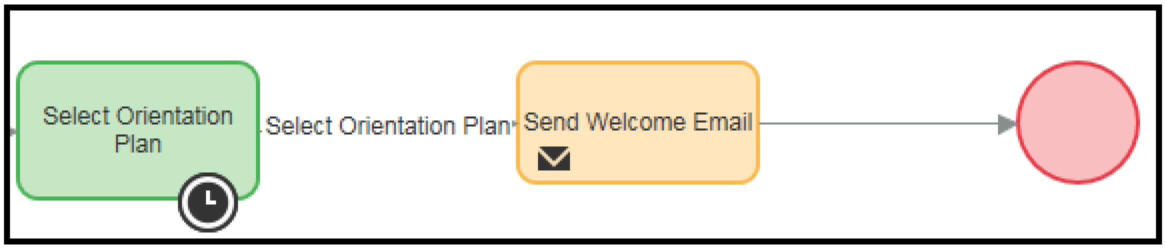
Subject*
Welcome to TGB

Message
Rich text [✓ Correspondence]

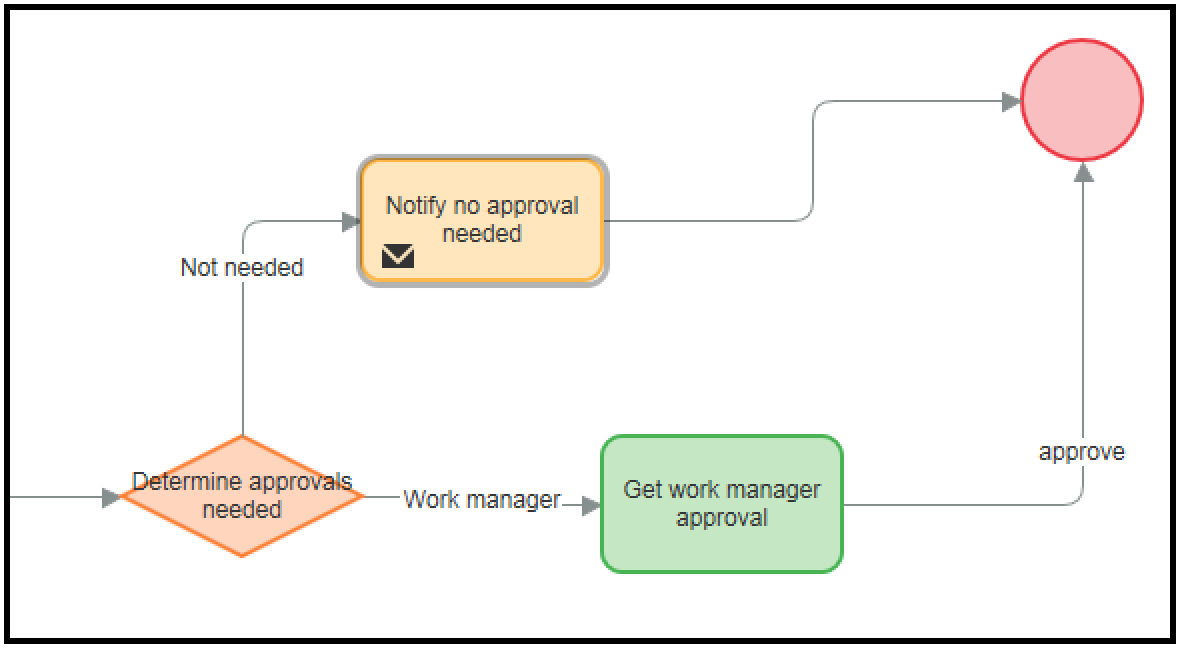
Correspondence template*
WelcomeEmail

Send attachments ☐

When you add the Send Email step, the system adds to the flow diagram a Send Email Smart Shape in the position you specified in the stage. For example, the above configuration inserts and connects the Send Welcome Email Smart Shape to the Select Orientation assignment, as shown in the following flow diagram.



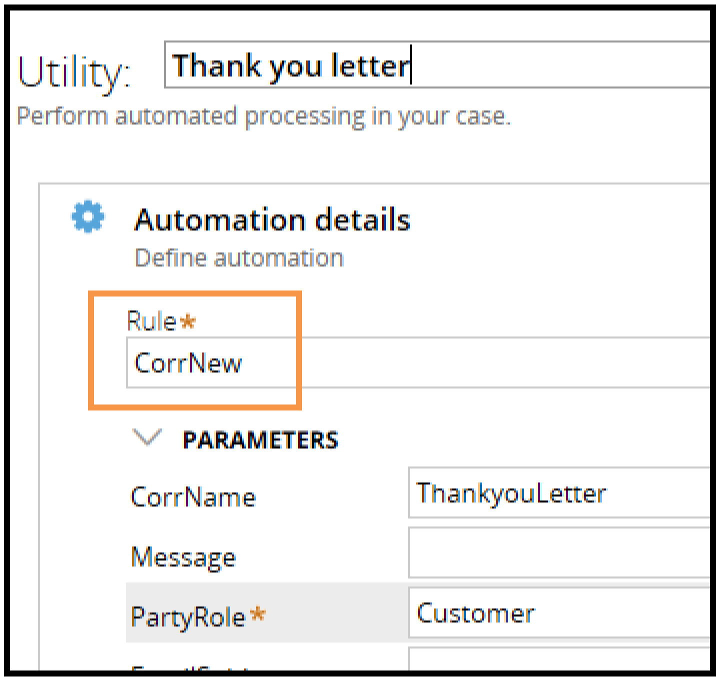
You can also add the Send Email Smart Shape directly to the flow diagram. In the following example, notice the Send Email Smart Shape is the outcome of a decision shape. The email automatically notifies a manager that the case did not require approval.


Notify no approval needed
Not needed
approve
Determine approvals needed
Work manager
Get work manager approval

Send correspondence using a Utility shape

Similar to the Send Email Smart Shape, you can use a Utility shape to automatically send correspondence. A Utility shape configured with the CorrNew activity offers greater flexibility than the Send Email Smart Shape. You can use the activity to send all types of correspondence including mail, fax, and text messages.

When you add a Utility shape to your process, open the properties panel and select CorrNew in the **Rule** field. Select your correspondence rule in the **CorrName** field as shown in the following example. When a case reaches the utility, the system sends the Thank you letter correspondence.


Utility: Thank you letter

Perform automated processing in your case.

Automation details

Define automation

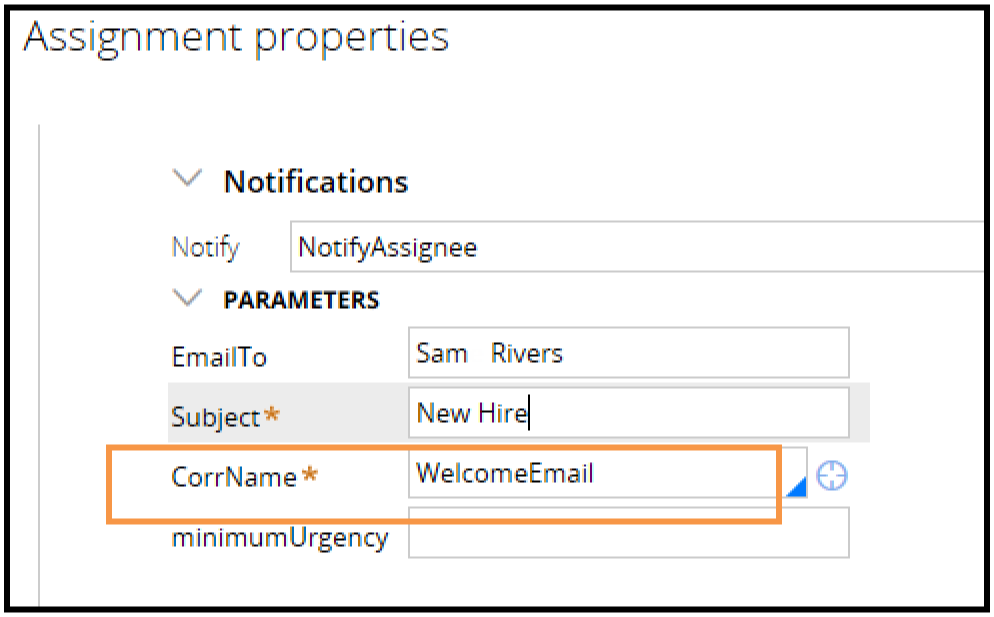
Rule* CorrNew

PARAMETERS

CorrName	ThankyouLetter
Message	
PartyRole*	Customer

Send notifications from assignments

You can automatically send email notifications when a case reaches an assignment using a Notify activity. This technique is useful if you want to notify users of new work that is waiting in their worklists. On the Assignment properties panel in the Notifications section, enter the party you want to notify. Then, enter the subject line text and select the correspondence rule in the CorrName field.


Assignment properties

Notifications

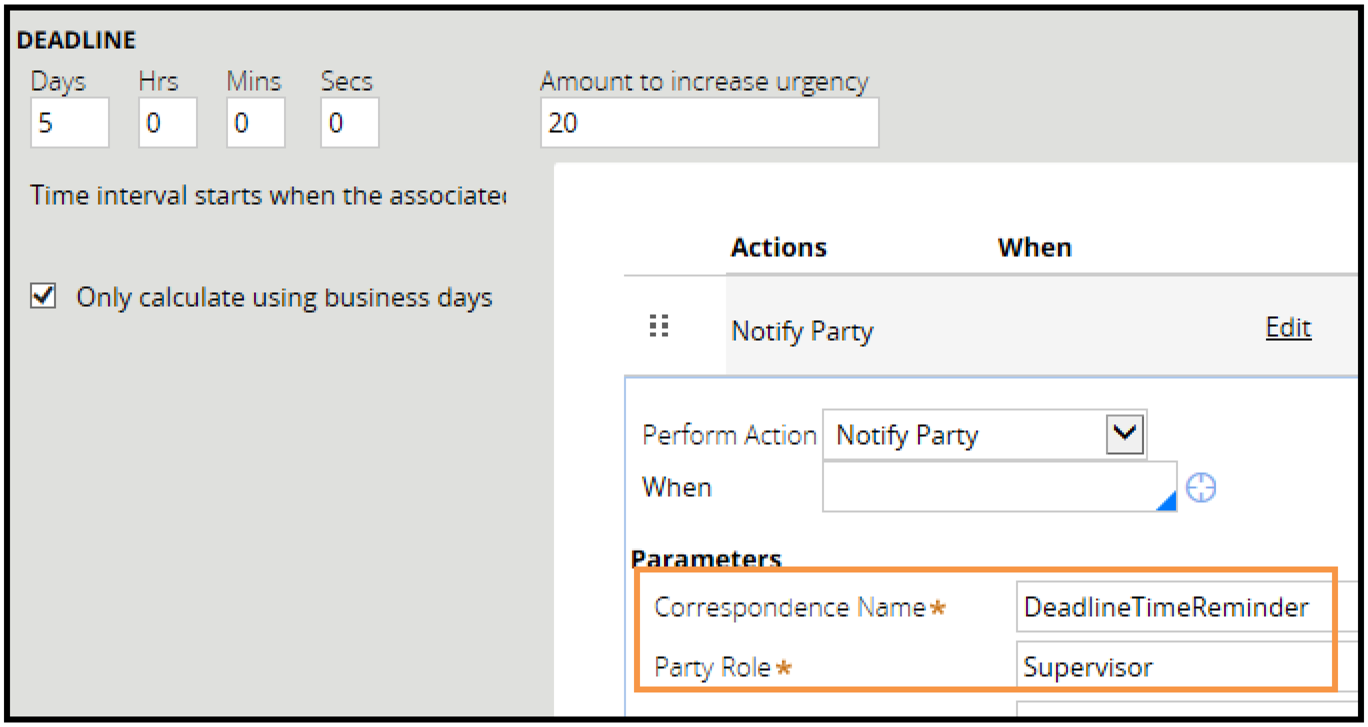
Notify NotifyAssignee

PARAMETERS

EmailTo	Sam Rivers
Subject*	New Hire
CorrName*	WelcomeEmail
minimumUrgency	

Send reminders from SLAs

You can set reminders in SLAs so that when the goal or deadline of a service level agreement has been reached, a notification email is sent out. You add the reminder by adding a row in the escalation actions area on the SLA form. In the Perform Action drop-down, you select a standard Notify notification. When you select Notify Party, you specify a party role and reference a correspondence rule.


DEADLINE

Days	Hrs	Mins	Secs	Amount to increase urgency
5	0	0	0	20

Time interval starts when the associate

☑ Only calculate using business days

Actions	When	
Notify Party		Edit

Perform Action: Notify Party
When:

Parameters

Correspondence Name*	DeadlineTimeReminder
Party Role*	Supervisor

Manually send emails using flow actions

Users often encounter questions or issues that are not part of the usual business process. For example, users, while working on an assignment, may need to send a request for sales receipts that were not attached to the user form. You can add the standard SendCorrespondence local action to your application so that users can select the correspondence they need when they need it.

KNOWLEDGE CHECK

ANSWER

Name the four standard types of correspondence you can use in Pega applications?

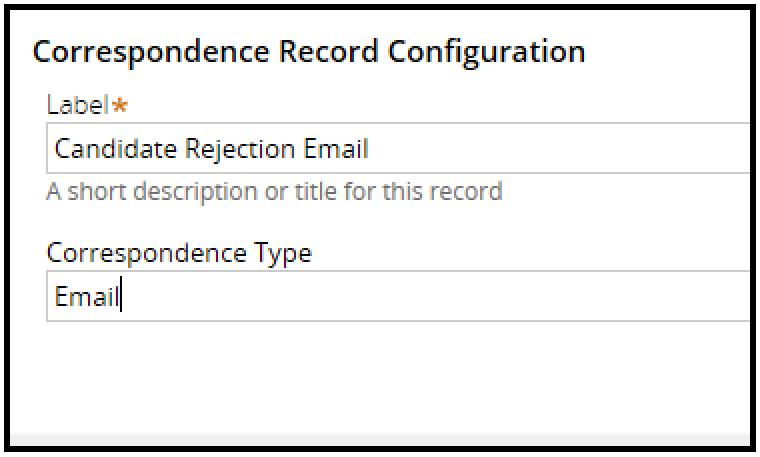
Emails, fax, phone text, and letters

Configuring correspondence rules

Configure a correspondence rule if you want to automatically or manually send email, fax, letters, or text messages during business processing.

Follow these steps in the Application Explorer to configure a correspondence rule:

1. Locate the class in which you want to create the correspondence rule.
2. Right-click on the class and select **+Create > Process > Correspondence** to open the Create Correspondence form.
3. Enter a name for the correspondence in the **Label** field and select the correspondence type, as shown in the following screenshot.



Correspondence Record Configuration

Label*

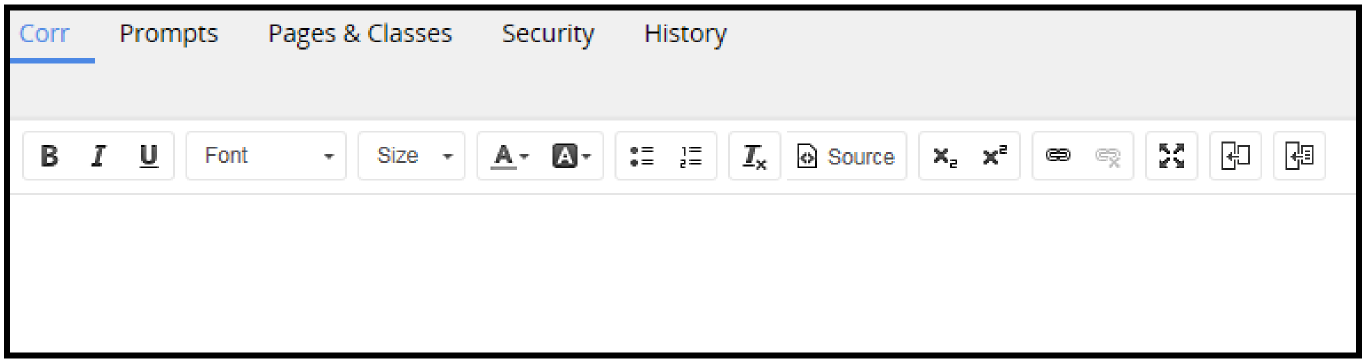
Candidate Rejection Email

A short description or title for this record

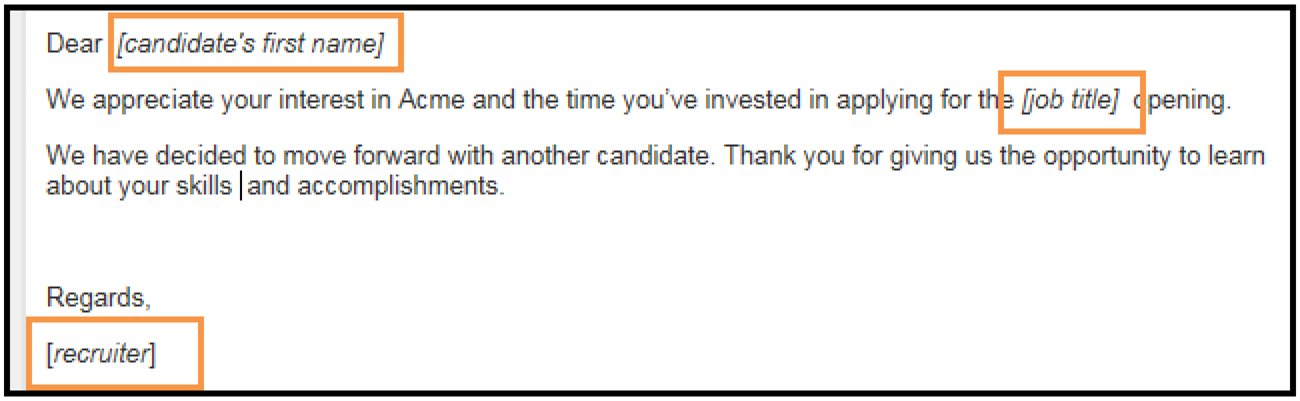
Correspondence Type

Email

4. Click **Create and Open**. The Correspondence form is displayed.



5. Enter the text you want to include in your email correspondence. In the following example, placeholder text shows where properties are inserted.



Dear *[candidate's first name]*

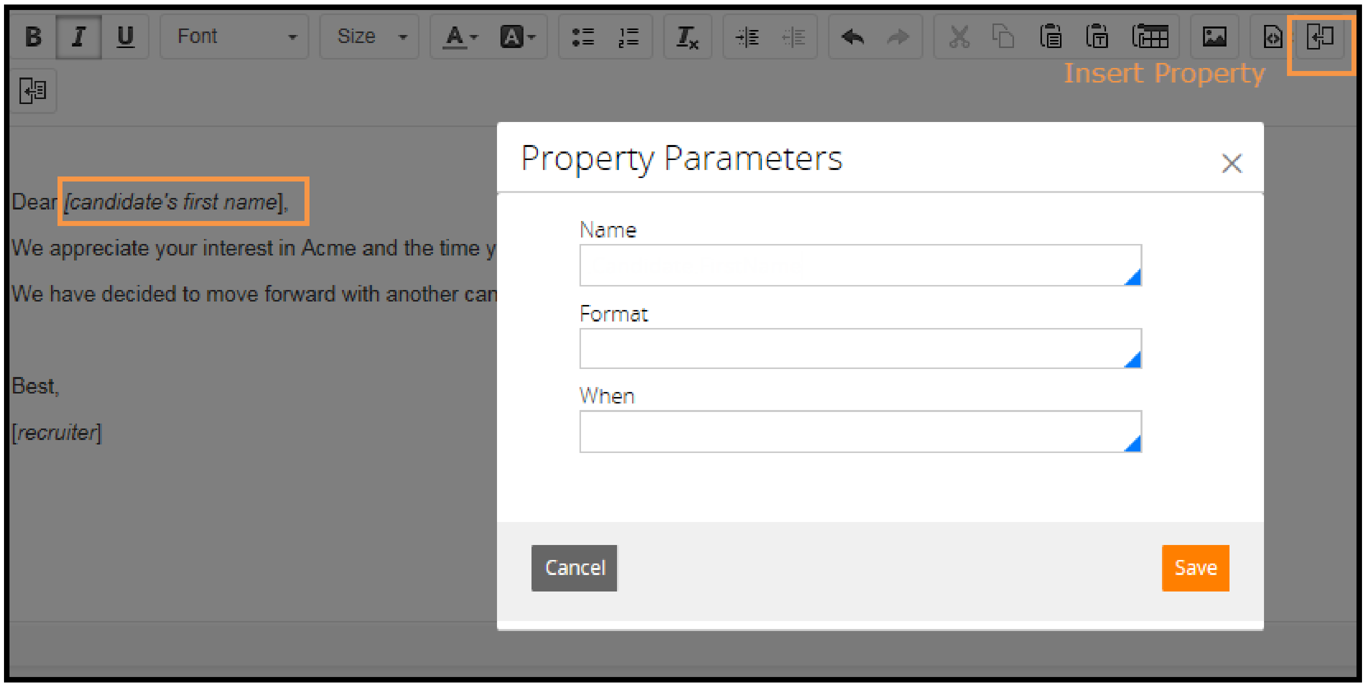
We appreciate your interest in Acme and the time you've invested in applying for the *[job title]* opening.

We have decided to move forward with another candidate. Thank you for giving us the opportunity to learn about your skills and accomplishments.

Regards,

[recruiter]

6. Select the field you want to replace with a dynamic reference to a property and click the **Insert Property** icon on the toolbar to open the Property Parameters dialog.



Dear [candidate's first name],

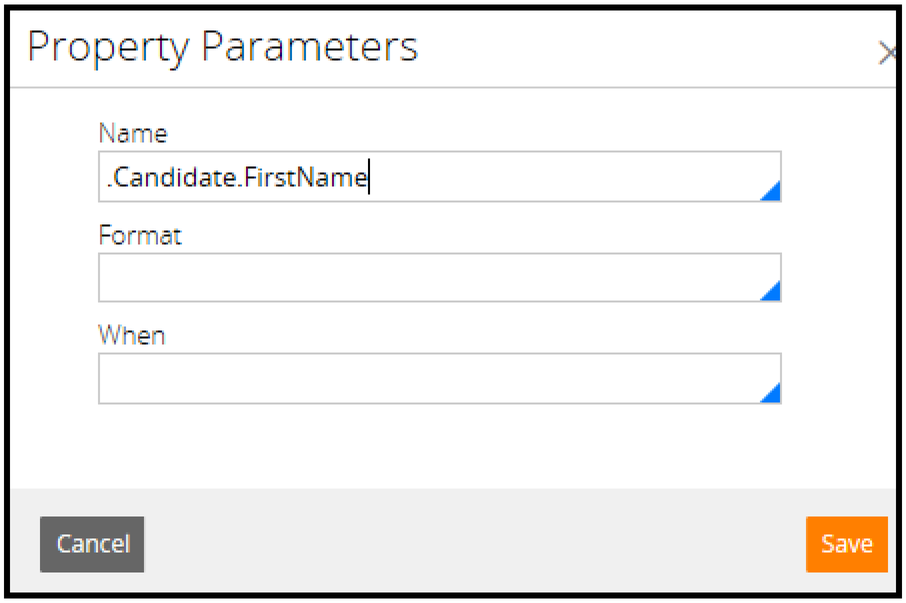
We appreciate your interest in Acme and the time y

We have decided to move forward with another can

Best,

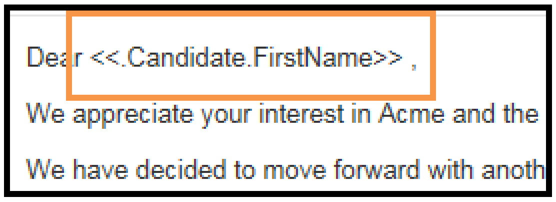
[recruiter]

7. Select the name of the property in the **Name** field and click **Save**.



Name: .Candidate.FirstName

The property name is displayed in brackets, as shown in the following example.

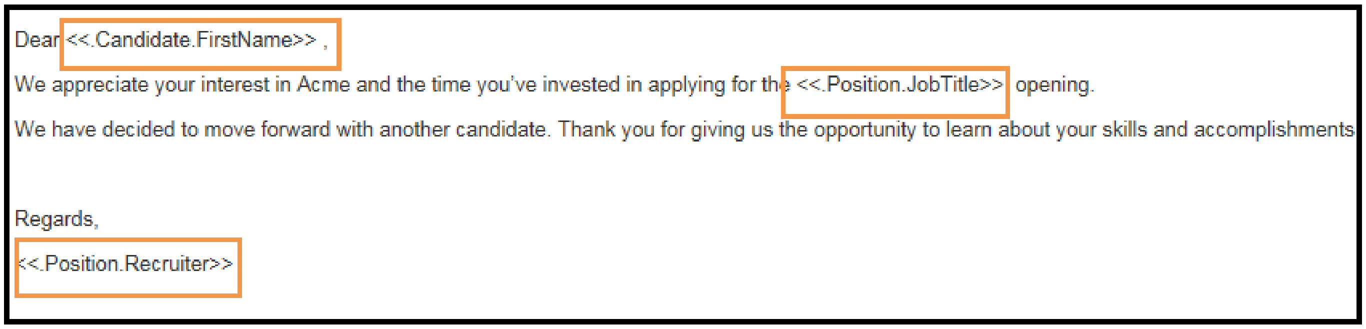


Dear <<.Candidate.FirstName>> ,

We appreciate your interest in Acme and the

We have decided to move forward with anoth

8. Insert the other properties in the form, as shown in the following example.



Dear <<.Candidate.FirstName>> ,

We appreciate your interest in Acme and the time you've invested in applying for the <<.Position.JobTitle>> opening.

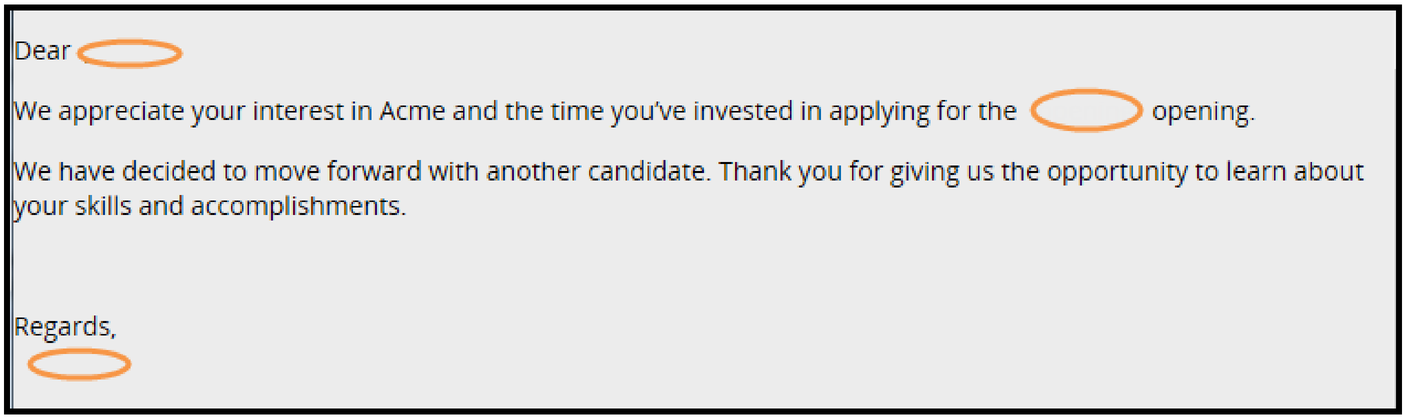
We have decided to move forward with another candidate. Thank you for giving us the opportunity to learn about your skills and accomplishments

Regards,

<<.Position.Recruiter>>

9. Click **Save** to commit your updates.

To preview the email, on the rule form header, select **Actions > Preview**. The message is displayed. Values do not appear in the reference property fields. The system populates the property fields during flow processing when the correspondence is sent.



Dear

We appreciate your interest in Acme and the time you've invested in applying for the opening.

We have decided to move forward with another candidate. Thank you for giving us the opportunity to learn about your skills and accomplishments.

Regards,

Circumstancing rules

Introduction to circumstancing rules

Applications often need to customize behavior to match the needs of a specific situation or circumstance. For example, a call center may need to enforce one set of performance objectives for clients with elite status, and a different set of performance objectives for clients without elite status.

In this lesson, you learn how to specialize case behavior through the use of circumstanced rules.

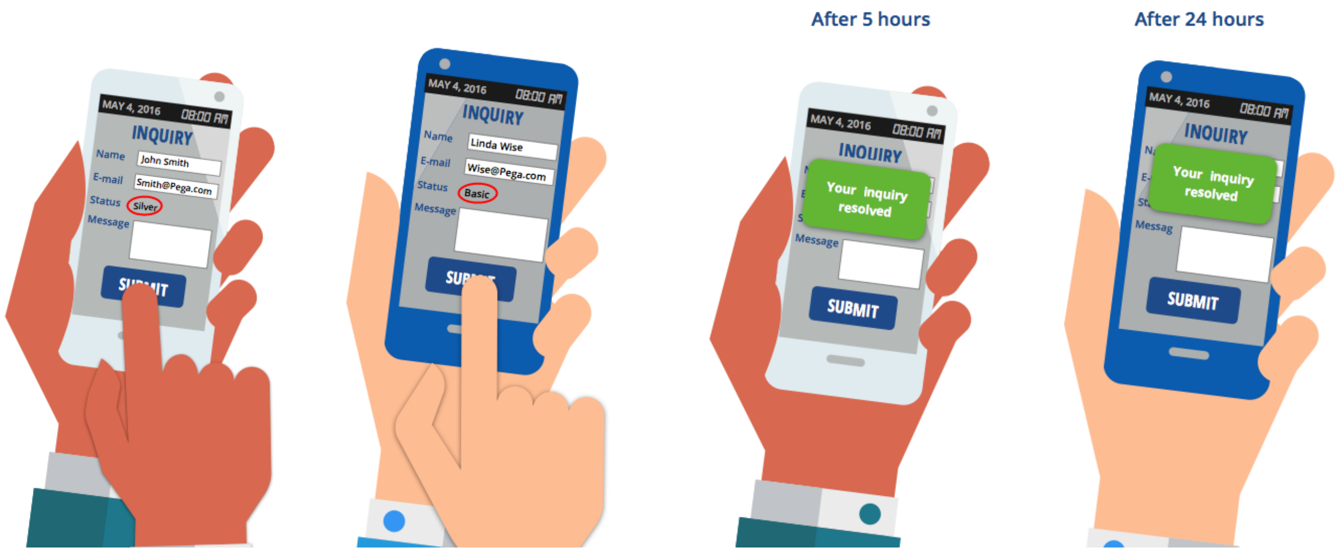
After this lesson, you should be able to:

- Explain how rule circumstancing supports rule specialization.
- Differentiate between base rules and circumstanced rules.
- Circumstance a rule.

Situational processing

Business processes must account for exceptions to typical case behavior. Exceptions make a business process more complex. This complexity makes processes difficult to maintain and update as business conditions change.

For example, a company promises to respond to customer complaints within one business day. For customers with silver status, the company promises a response in 12 hours. For customers with gold status, the company promises a response in only six hours. Reduced response times for customers with elite status are exceptions to normal business processing.



Simple exceptions like these can be difficult — or impossible — to model with a single rule. For example, a service level only defines one set of service expectations, and an assignment only applies one service level. To apply three different response intervals, you might design a process with three assignments, and apply the correct service level to each assignment. If the process changes, you need to update three assignments, instead of one.

Complex exceptions that depend on combinations of factors become difficult to maintain and update. Consider a bank that offers different promotions that reduce or waive fees for customers who meet specific conditions.

- A new customer receives 100 commission-free trades for the first three months after opening an investment account.
- A customer receives a rebate on commissions as long as the daily balance in their investment account exceeds certain thresholds — but the rebate amount, balance threshold, and number of rebate tiers vary by account type and country.
- A customer who refers a friend to the bank receives 10 commission free trades per month for six months.

A rule that models these commission discounts according to account type, account balance, and country can become complex. This complexity may lead to configuration errors and dissatisfied customers.

In Pega applications, you model complex exceptions through **circumstancing**. With circumstancing, you create a variant of a rule — such as a decision or a service level — tailored to a specific circumstance. When an application uses a circumstanced rule, the system determines which rule variant best satisfies the need. Circumstancing allows you to customize the behavior of your application to address each exception condition you identify using a collection of targeted rules rather than one complex, difficult-to-maintain rule.

KNOWLEDGE CHECK

ANSWER How does circumstancing solve the problem of configuring exception behavior in an application?

Circumstancing allows you to describe exception behavior with a set of targeted rules rather than one complex rule. Each targeted rule configures behavior to address a specific exception.

Rule circumstancing

Circumstancing establishes a baseline for expected case behavior, and adds variants to address exceptions to the behavior. The goal of circumstancing is to create a variant for each anticipated situation. Pega selects the appropriate variant, or circumstance, to use based on the details of the case.

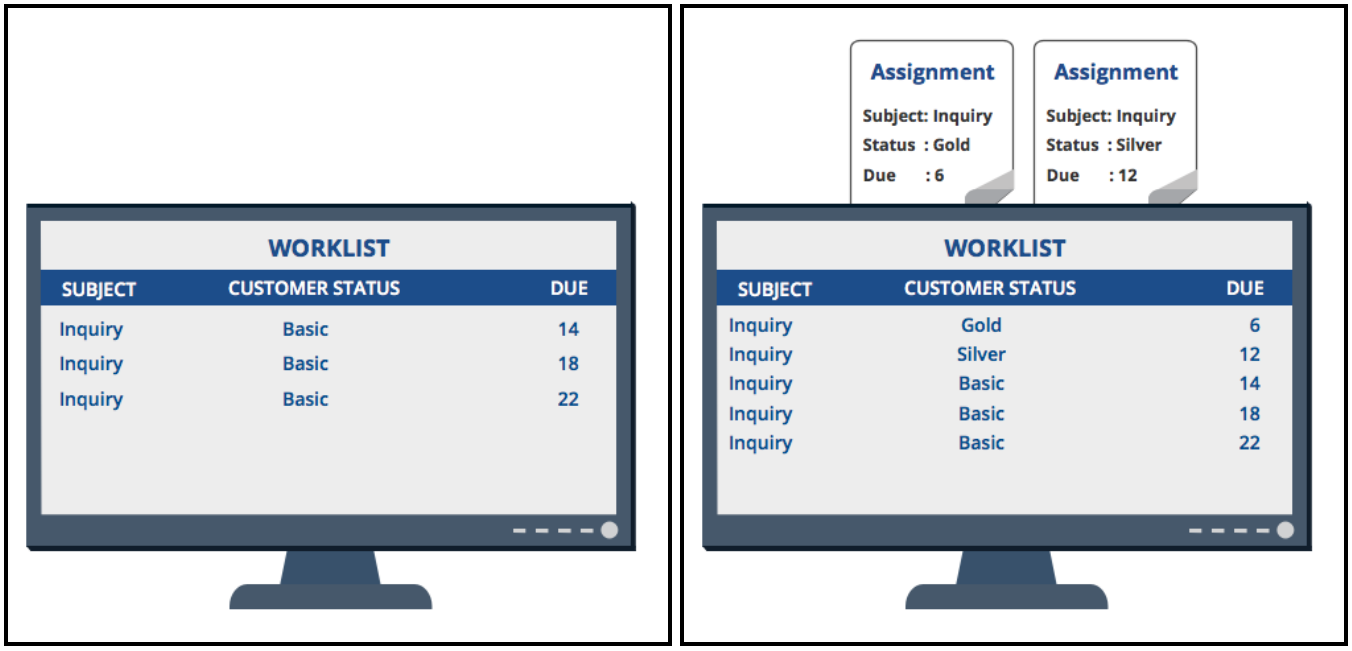
When you circumstance a rule, you create a set of focused rules to address exceptions to case processing, rather than one all-encompassing rule. Since each rule focuses on a specific exception, application maintenance and updates are easier and can be delegated to business users. And you can more easily reuse the rules you create at the application or enterprise level.

How to circumstance a rule

To circumstance a rule, you start by creating a base rule to define the expected behavior. Pega uses this base rule unless a circumstanced version is more appropriate.

Consider a company with different response-time obligations for elite and non-elite customers. The response time for non-elite customers is the expected behavior. In this situation, the response-time goal is 24 hours. So you create a base rule — in this case, a service level — to enforce the response-time goal for non-elite customers.

Then, you identify any exceptions to the expected behavior. For each exception, you circumstance the base rule to addresses the difference from the expected behavior. For example, elite customers with silver status have a response time goal of 12 hours. You circumstance the base rule to enforce the response-time goal for customers with silver status. You can then create another circumstance to address the goal time for customers with gold status, who have a service level response goal of 6 hours.



SUBJECT	CUSTOMER STATUS	DUE
Inquiry	Basic	14
Inquiry	Basic	18
Inquiry	Basic	22

Assignment
Subject: Inquiry
Status : Gold
Due : 6

Assignment
Subject: Inquiry
Status : Silver
Due : 12

SUBJECT	CUSTOMER STATUS	DUE
Inquiry	Gold	6
Inquiry	Silver	12
Inquiry	Basic	14
Inquiry	Basic	18
Inquiry	Basic	22

Types of circumstancing conditions

You can circumstance a rule according to the value of one or more conditions. You define a condition based on one variable, multiple variables, or the processing date, then apply the condition to a variant of the rule. When using the rule, the application evaluates the conditions defined on all the circumstanced variants. If one of the circumstancing conditions is satisfied, the application uses the corresponding rule variant. Otherwise the application uses the base rule.

Pega supports the following types of circumstance conditions.

- **Single value** — the rule variant is effective whenever the value of a single property satisfies the circumstancing condition. You specify the property to evaluate and a comparison value when circumstancing a rule. If the value of the property matches the specified value for a case, the application applies the circumstanced variant of the rule, rather than the base rule.

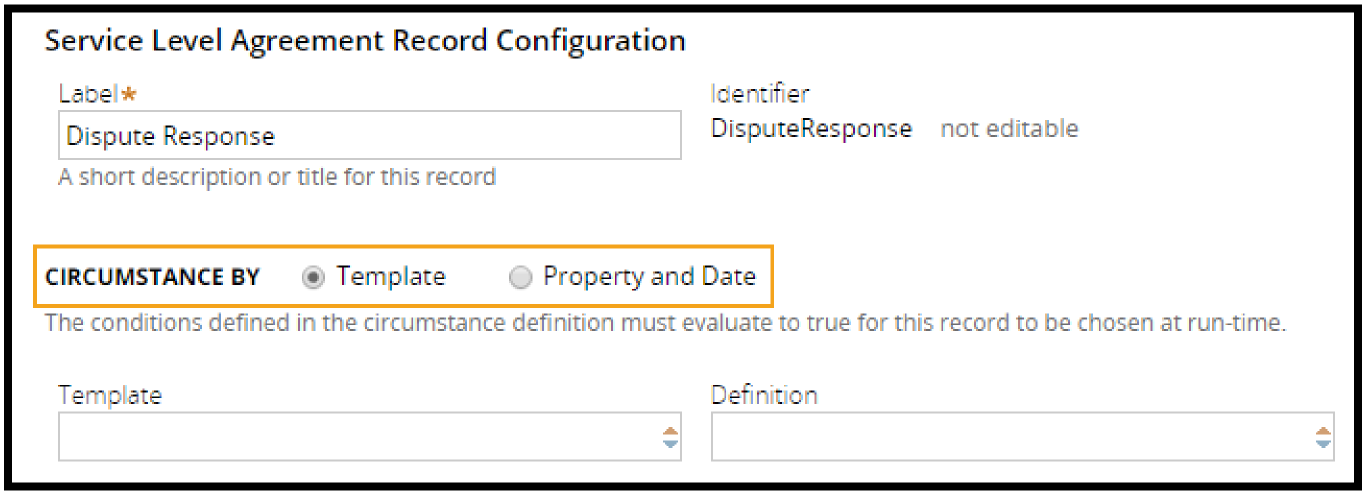
- **Multiple value** — the rule variant is effective whenever a combination of property values satisfies the circumstancing condition. Multiple value circumstances are based on a circumstance template and circumstance definition. The **circumstance template** defines the properties on which the rule is circumstanced. The **circumstance definition** defines the combination of conditions in which a variant of a rule is used. You apply the circumstance template and circumstance definition to the rule variant. If the case matches a combination in the circumstance definition, the application uses the circumstanced variant of the rule, rather than the base rule.
- **Date property** — the rule variant is effective whenever the value of a date property satisfies the circumstancing condition. This condition can be either a single date or a range of dates. If the value of the property is later than the specifies date or falls within the range of dates, the application uses the circumstanced variant of the rule, rather than the base rule.
- **As-of date** — the rule variant is effective after a certain date, or during a range of dates. After the specified date or during the specified range, the application applies the circumstanced variant of the rule, rather than the base rule.

Circumstancing a rule

To circumstance a rule, you first create a base rule and then create specialized versions of the rule. Each version is tailored to a specific exception in case behavior.

Follow these steps to circumstance a rule:

1. Open the base rule.
2. On the base rule, open the pull-down menu on the Save button and select **Specialize by circumstance**. The New Record form opens, with two circumstancing options: **Template** and **Property and Date**.



Service Level Agreement Record Configuration

Label* Dispute Response
A short description or title for this record

Identifier DisputeResponse not editable

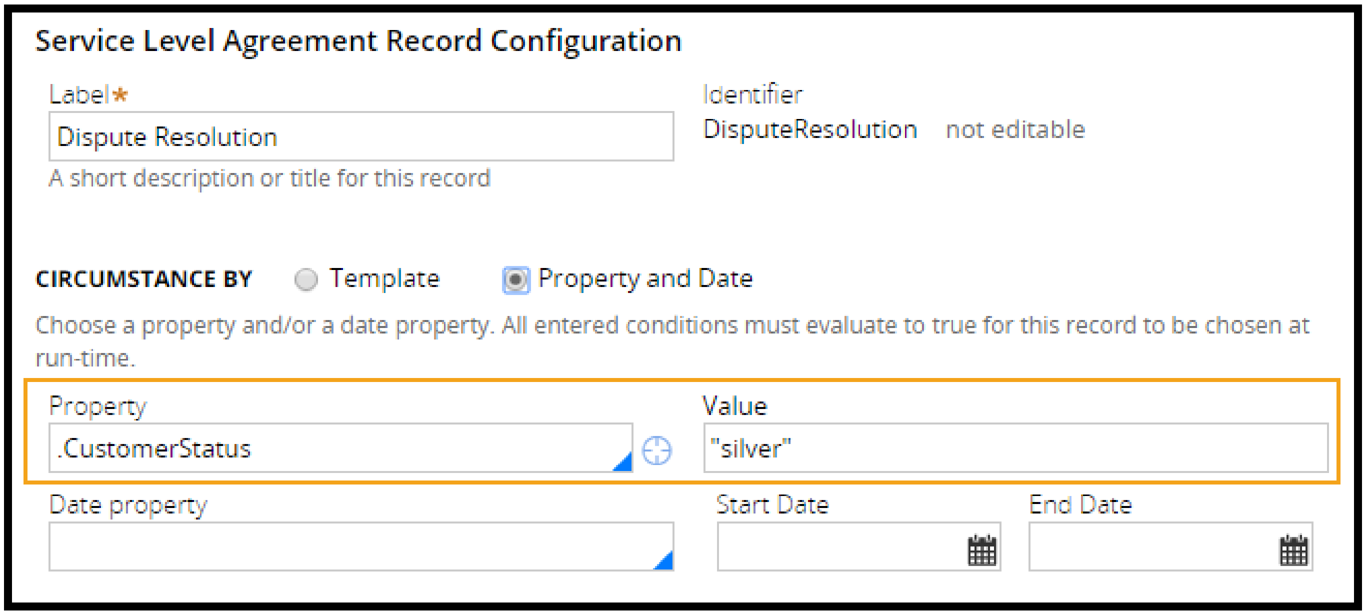
CIRCUMSTANCE BY [✓ Template] Property and Date
The conditions defined in the circumstance definition must evaluate to true for this record to be chosen at run-time.

Template

Definition

3. On the New Record form, identify the type of circumstance. To circumstance on one variable, select **Property and Date**. To circumstance on more than one variable, select **Template**.
4. Specify the condition under which the rule is used. The following example shows a service level

circumstanced to run whenever the value of *.CustomerStatus* is "silver". The value must be entered within quotation marks.


Service Level Agreement Record Configuration

Label* Dispute Resolution

A short description or title for this record

Identifier DisputeResolution not editable

CIRCUMSTANCE BY ○ Template ● Property and Date

Choose a property and/or a date property. All entered conditions must evaluate to true for this record to be chosen at run-time.

Property .CustomerStatus

Value "silver"

Date property

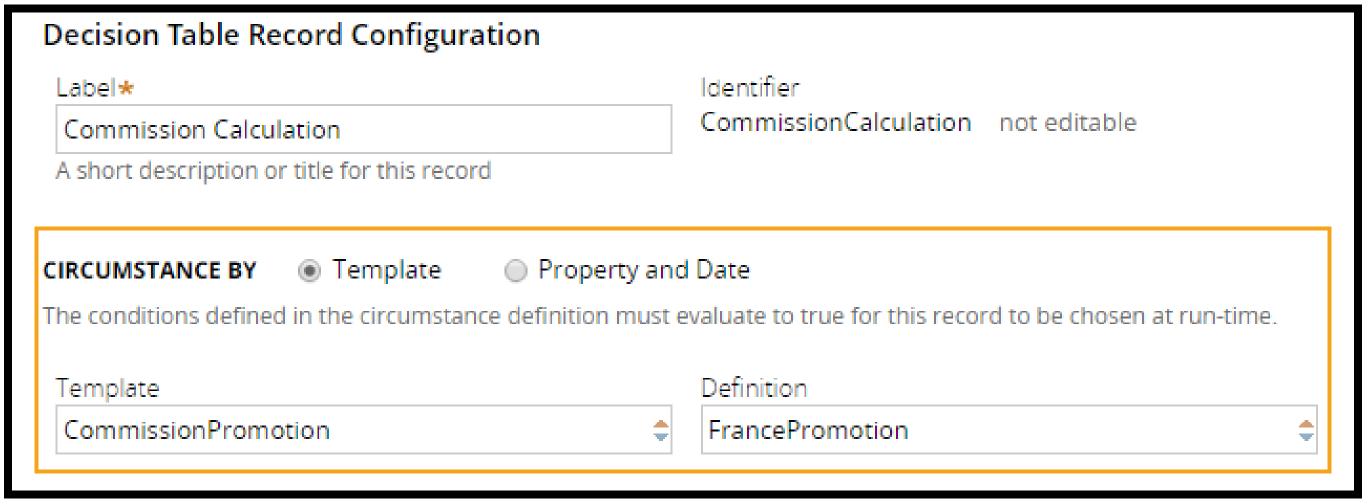
Start Date

End Date

To circumstance by date, use the following table to configure the circumstancing condition to meet various business requirements.

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

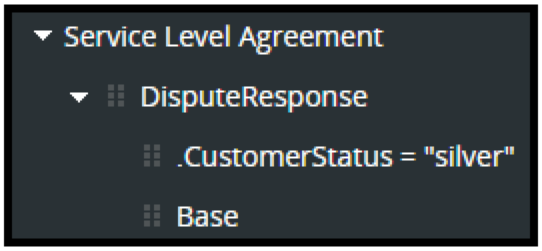
To circumstance by more than one property, specify the circumstance definition and circumstance template rules that define the combination of conditions. For each circumstanced rule, you must provide a unique circumstance definition.


Decision Table Record Configuration

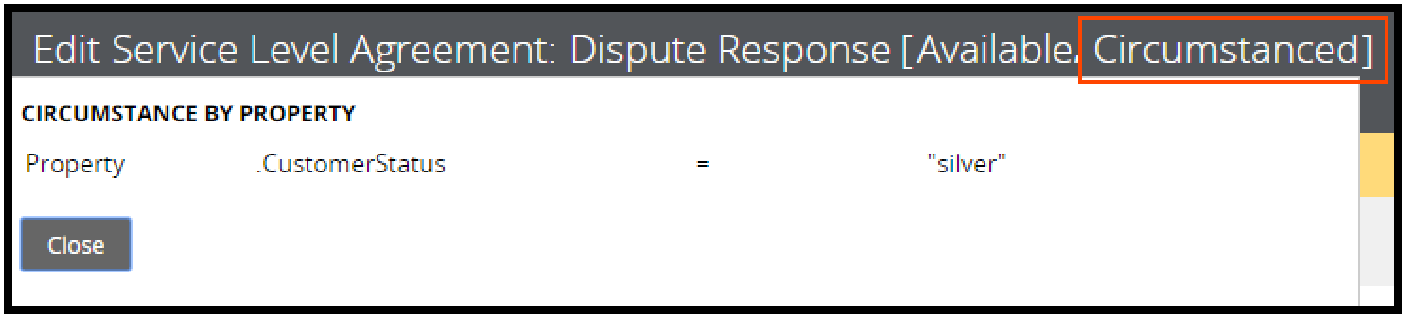
Label*	Commission Calculation
	A short description or title for this record
Identifier	CommissionCalculation not editable
CIRCUMSTANCE BY	[✓ Template] Property and Date
	The conditions defined in the circumstance definition must evaluate to true for this record to be chosen at run-time.
Template	CommissionPromotion
Definition	FrancePromotion

5. Click **Create and open** to open the rule form.
6. Customize rule behavior for the specified circumstance.

To view the circumstancing condition for a rule, locate the rule in the Application Explorer. Pega indicates a circumstanced rule with a collapse arrow. Clicking the arrow expands the rule entry to display the supported circumstances. In the following example, the *DisputeReponse* service level includes a circumstance used when the value of *.CustomerStatus* is silver.


```
▾ Service Level Agreement
    ▾ DisputeResponse
        .CustomerStatus = "silver"
        Base
```

You can also review the circumstancing condition for a rule by clicking the **Circumstanced** link in the rule header.


Edit Service Level Agreement: Dispute Response [Available, Circumstanced]

CIRCUMSTANCE BY PROPERTY

Property	.CustomerStatus	=	"silver"

DECISION DESIGN

Automated decisions in Pega applications

Introduction to Automated Decisions in Pega Applications

During the case life cycle, choices affect how each case progresses toward resolution. Choices range from deciding which processes to run to deciding which fields to complete.

By automating decisions, system architects significantly improve process efficiency. Automated decisions eliminate delays waiting for users to decide an appropriate outcome. Automated decisions ensure that decisions are evaluated consistently from case to case. For example, automating decisions can reduce the time needed to perform a credit check from three weeks to 15 minutes, dramatically reducing the time needed to process loan requests.

In this lesson, you learn about the types of decisions that Pega allows you to automate, from simple true/false conditional tests to complex strategies for improving the outcome of customer interactions.

After this lesson, you should be able to:

- Differentiate between the types of decision rules available in Pega applications

Types of decisions available in Pega applications

The Pega platform provides many options for automating decisions, from simple true/false tests to complex decision strategies that rely on predictive or adaptive analytics. The decision rules in Pega applications are divided into two categories: business rules and decision strategies.

Business rules

Business rules evaluate case data to determine outcomes that direct a business process. These rules are used in applications to direct flows, hide or display form elements, and even calculate property values. Pega provides four types of business rule decisions: **when conditions**, **map values**, **decision tables**, and **decision trees**.

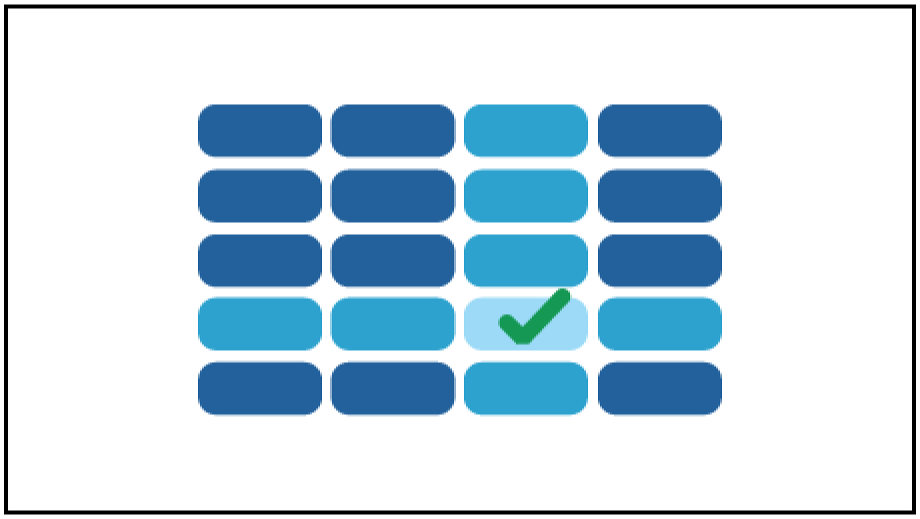
When conditions

When conditions are the simplest type of decision. A when condition evaluates a relationship among one or more property values to return a true or false result. When conditions correspond to the "if" statement in programming languages. When conditions are often used to determine whether an application performs some action, such as hiding the contents of a form. For example, use a when rule to test if a user selected "Married" as their marital status. If the result is true, then the form displays fields to obtain the name and birth date of the user's spouse.

Map values

Map values evaluate one or two criteria to return a result. Unlike a when condition, a map value can return numeric or text results. Map values are often called from a decision shape in a flow to direct flow processing, and can also be used as part of a declare expression to set the value of a property.

A map value uses a one- or two-dimensional matrix to derive a result. The inputs to a map value identify a row and column in the matrix, like latitude and longitude on a map. The intersection of the two inputs indicates the result of the decision.



For example, use a map value to test the options packages and colors offered for a vehicle to determine which combinations are allowed.

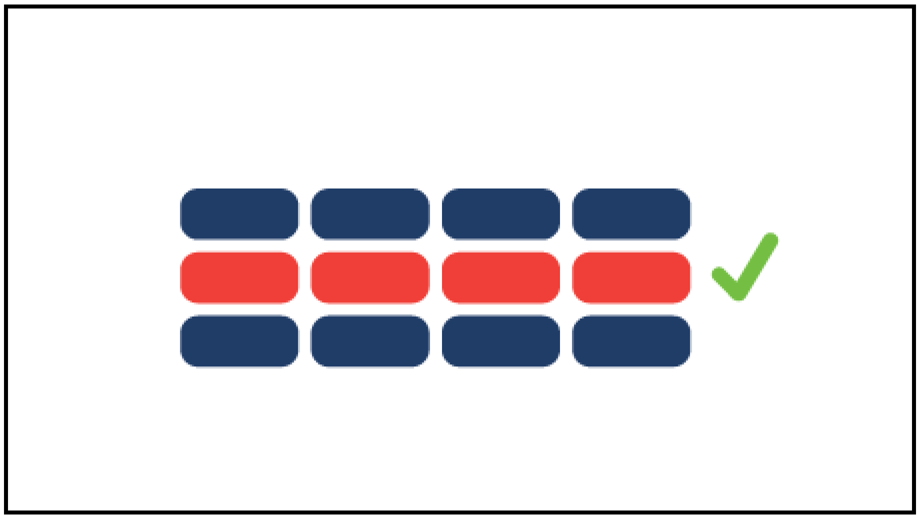
Decision tables and decision trees

Decision tables and decision trees evaluate a series of one or more conditions to return a result. Like a map value rule, decision tables and trees can return numeric or text results.

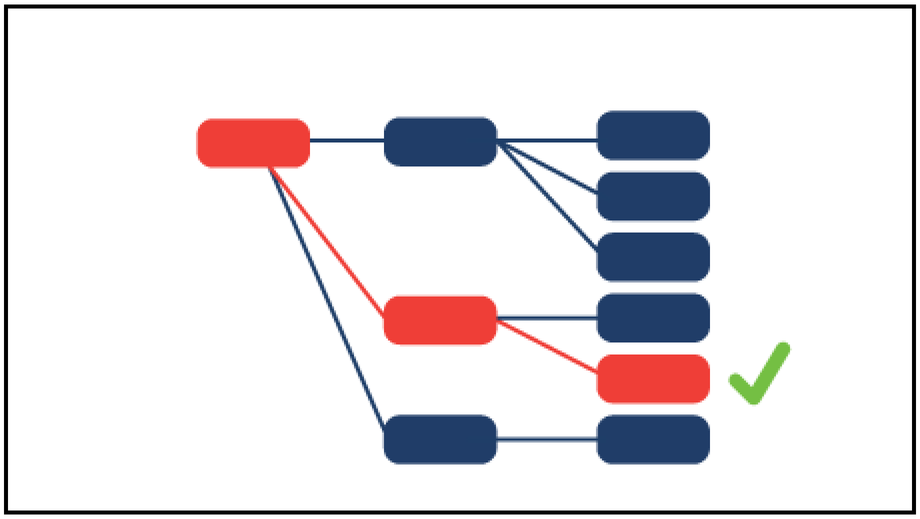
Decision tables and trees are often called from a decision shape in a process flow to direct case processing, and can also be used as part of a declare expression to set the value of a property.

Decision tables and trees behave similarly, but differ in how the decision logic is organized: in a table of conditions organized into rows, or in a tree structure with conditions organized into branches of the tree.

A decision table returns a result using a series of one or more conditions, organized as rows in a table. If all the conditions in a row evaluate to true, the decision table returns the result assigned to the row. If any of the conditions in a row evaluates to false, processing advances to the next row in the table. If no row in the table evaluates to true, the table returns a default result.



A decision tree returns a result using a series of if-then conditions, organized as a tree-like structure. Evaluation begins with the trunk of the tree and advances through a series of branches. If a branch evaluates to true, the decision tree returns the result assigned to the branch. If a false result occurs, processing returns advances to the next branch in the tree. If no branch evaluates to true, the tree returns a default result.



For example, use a decision table or tree to test the total amount of a purchase request and determine whether to

- approve the request automatically,
- forward the request to a manager for approval, or
- trigger an audit of the purchase request.

Unlike map value rules, which allow you to test one or two variables, decision tables and trees allow you to test any number of variables. The choice between a map value, decision table, and decision tree often depends on the number of conditions and the best format for presenting the decision logic: as a matrix, as a table, or as a tree.

Decision Management rules

Decision Management rule types, such as scorecards and predictive analytics, analyze customer behavior. These rules tailor offers or propositions to a customer, often as part of a retention or up-sell strategy.

Decision Management rules are available only in applications built on the PegaDM application or Decision Management rulesets.

Strategies

Strategies define a result personalized to the interests, risk, and eligibility of an each customer. A decision strategy consists of interactions, predictive and adaptive models, and scorecards. For example, you create a strategy rule to provide a customer with enticing offers for a new credit card based on their spending patterns, income, or other criteria.

Interactions

Interactions define the parameters for running a strategy and the possible outcomes.

Use an Interaction rule to execute a decision strategy and capture the details of the customer interaction. For example, you use an interaction rule to run a strategy designed to retain a customer calling to cancel their credit card. Delegate an interaction rule to allow process owners to switch to another strategy in an released application.

Predictive models

Predictive models predict behavior for one or more segments, based on customer data. Predictive models are used in strategies through predictive model components. For example, you use a predictive model rule to define characteristics that identify customers who are more receptive to a credit card with travel rewards or a low interest rate.

Adaptive models

Adaptive models capture customer responses in real-time to make and adapt predictions. Adaptive models are used in the absence of historical records.

You use an adaptive model to collect data from customer interactions, which you then use to generate predictive models. For example, you use an adaptive model in a retention strategy to determine if the age of a customer affects the response rate for a credit card with a cash-back reward. As customers respond to the available offers, the adaptive model collects data that you can use to create a predictive model for future marketing campaigns.

Scorecards

A scorecard uses one or more conditions and a combining method to return a score and a segment. You then define cut-off values to map each score range to a result. For example, a scorecard rule segments customers based on age and income, then maps the score ranges to recommended a different credit card to each segment. A scorecard allows you to recommend one credit card with a low interest rate to customers under the age of 25, and another credit card with travel perks to customers over the age of 40.

More information on Decision Management

For more information on Decision Management rules, see the PDN topic Decision Management.

Configuring when rules

Introduction to Configuring When Rules

Applications must evaluate case data and determine the appropriate response to automate case processing. For example, application logic must often determine whether to:

- Automatically approve a case or assign the case to a user for approval.
- Skip a processing step.
- Display information on a form.

Pega models true/false decisions such as these using a when rule. In this lesson, you learn how to automate decision-making with when rules.

After this lesson, you should be able to:

- Explain how when rules model true/false decisions.
- Describe the uses of when rules in an application.
- Configure a when rule to evaluate case data and return a result.

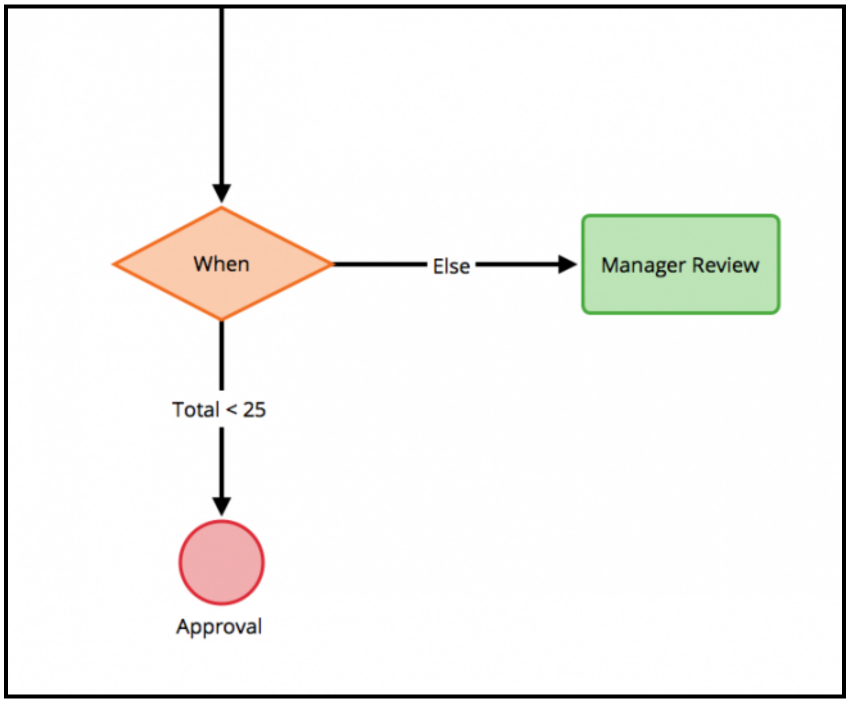
When conditions

Applications often need to decide whether to perform an action, such as skipping a process when a case satisfies some condition. For each decision, an application tests a condition to return a true or false result. If the result is true, the application performs the conditional action. If the result is false, the application skips the action. These true/false decisions allow applications to adapt a business process to the details of each case.

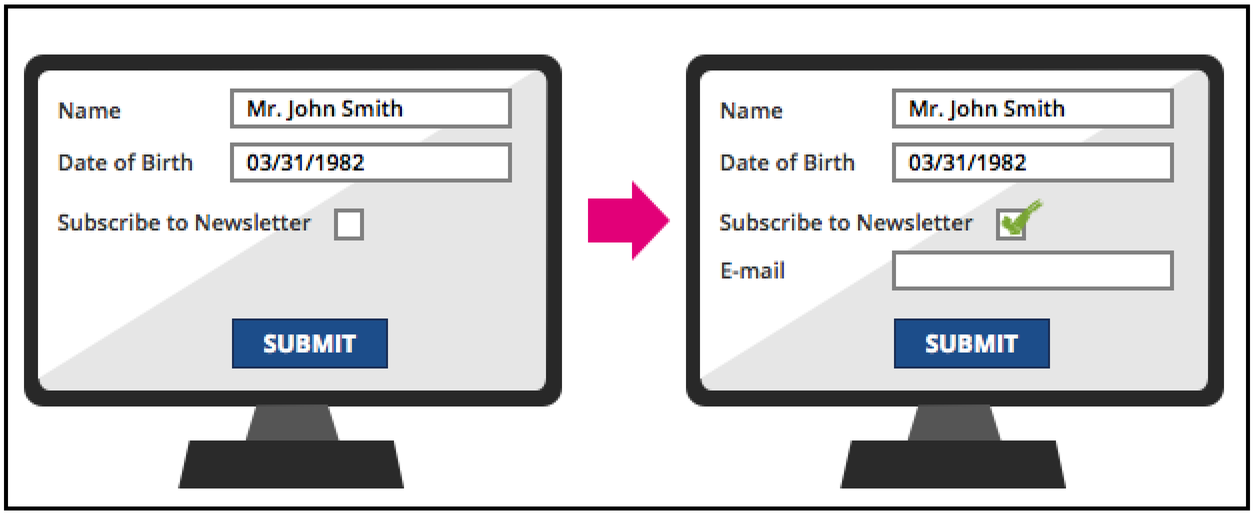
In Pega, a when condition describes a decision that returns a true or false result. You use a when condition to compare the value of one property against a constant or the value of another property. For example, use a when condition to determine whether the total value of an order exceeds EUR100. If true, the application can then apply a credit for the shipping charge to the order.

You can use when conditions whenever an application requires a true/false outcome. Use when conditions in flows, UI forms, and data transforms to adjust rule behavior in response to case data.

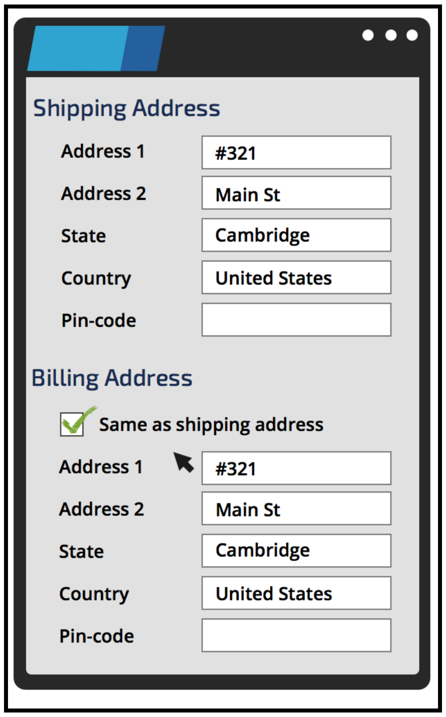
In flows, use a when condition to branch a flow based on case data. For example, if an employee submits a purchase request totaling less than USD25, then skip the approval step and automatically approve the request.



In UI forms, use a when condition to update the appearance of a form in response to user input. For example, if users indicate that they want to subscribe to a newsletter, an application provides a field for the users to enter their email address.



In data transforms, use a when condition to determine whether to perform a step or sequence of steps. For example, if users indicate on an order that their billing address is the same as their shipping address, then copy the shipping address information to the corresponding fields for the billing address.



KNOWLEDGE CHECK

ANSWER

What is the purpose of a when rule?

The purpose of a when rule is to evaluate one or more conditions to return a result of either true or false.

How to configure a when condition using a when rule

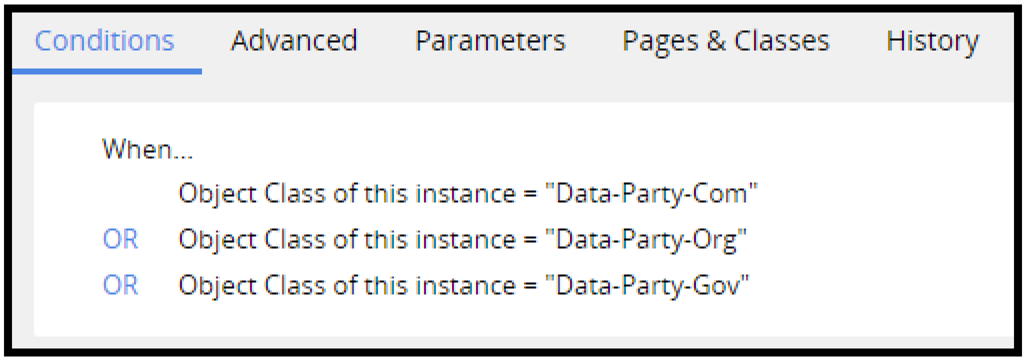
In Pega, you can apply a when condition where needed, or configure the when condition using a when rule. By using a when rule, you can reuse the when condition wherever necessary in the application.

The when rule form

A when rule organizes a set of one or more true/false tests into a tree-like structure. Each node on the tree represents either a single condition or a group of conditions related with a Boolean AND or OR operator. The entire conditions tree reduces to a single Boolean operation. The result of this operation is the result of the when rule.

Use the **Conditions** tab to enter or revise the conditions tree for a when rule. The when condition is expressed as either a single Boolean expression or a tree consisting of multiple expressions combined using Boolean AND or OR operators.

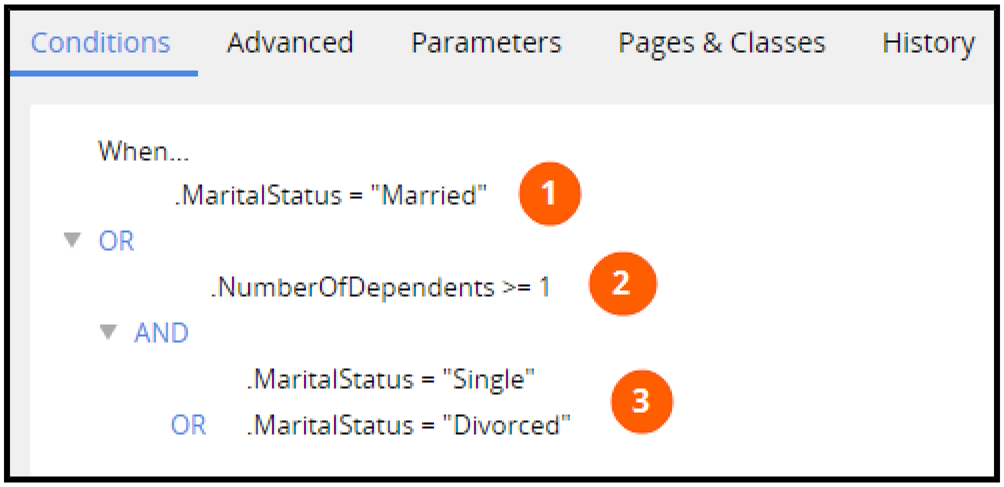
Each expression defines a single comparison that evaluates to **True** or **False**. The when rule evaluates to True only if the entire tree of conditions evaluates to true. In the following example, the when rule returns a result of True if a work party is an instance of any of the three listed classes: *Data-Party-Com*, *Data-Party-Org*, or *Data-Party-Gov*.



When...
Object Class of this instance = "Data-Party-Com"
OR Object Class of this instance = "Data-Party-Org"
OR Object Class of this instance = "Data-Party-Gov"

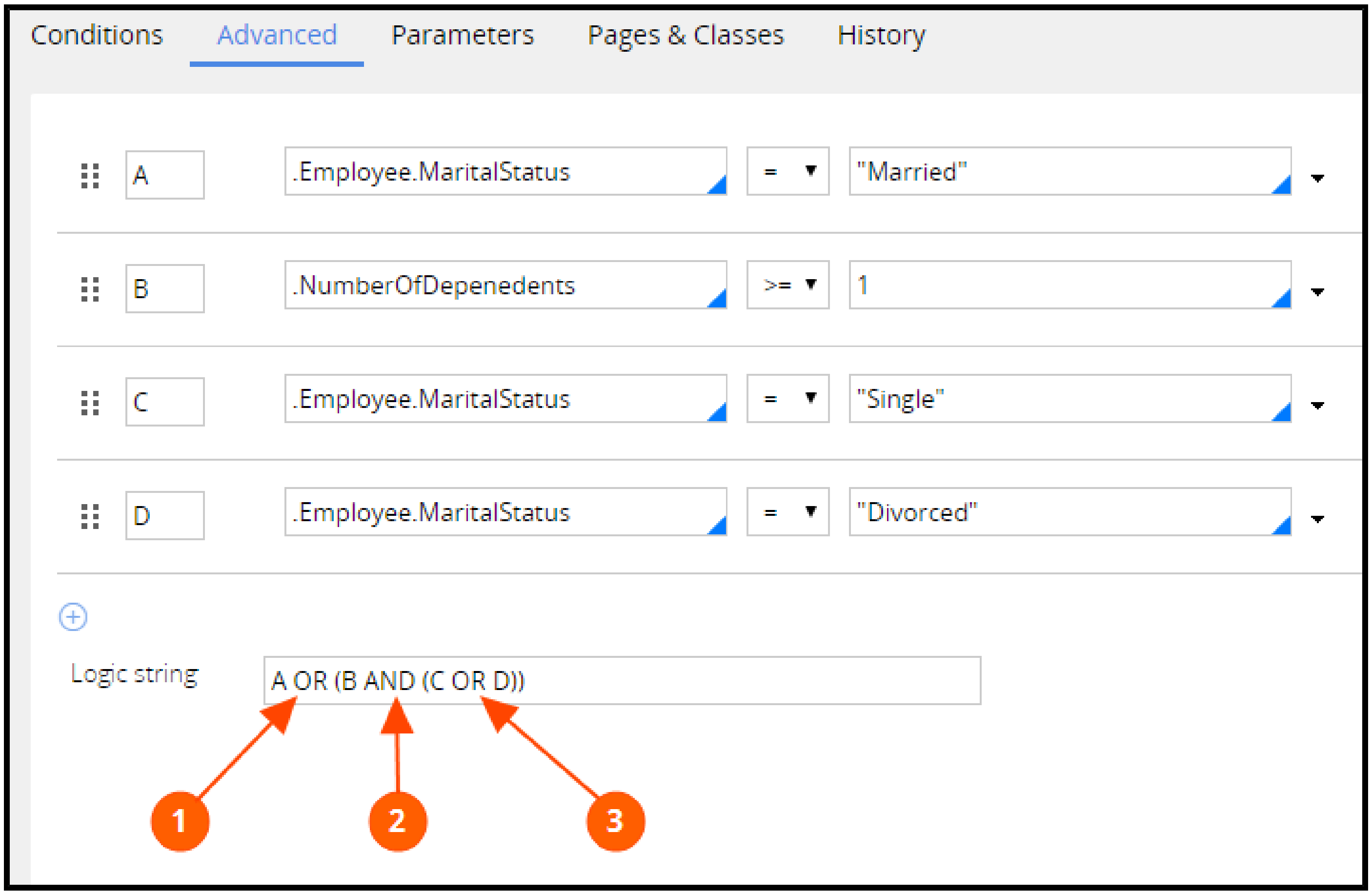
The when rule form starts you with a single condition to test. To further edit the conditions tree, use the context menu. From this menu, you either add a condition to the current node or add a new node to the conditions tree. Each node consists of one or more conditions linked by a Boolean AND or OR operator. The same operator is applied to all of the conditions within a node.

The following example demonstrates a when rule with three nodes.



When...
.MaritalStatus = "Married" 1
OR
.NumberOfDependents >= 1 2
AND
.MaritalStatus = "Single"
OR .MaritalStatus = "Divorced" 3

The conditions and logical operators you enter on the Conditions tab also appear on the **Advanced** tab, in the **Condition** array and **Logic String** field. Each set of parentheses in the logic string groups a set of conditions to represent a node. The following example shows the same when condition, viewed from the Advanced tab. The logic string groups each node with a set of parentheses.



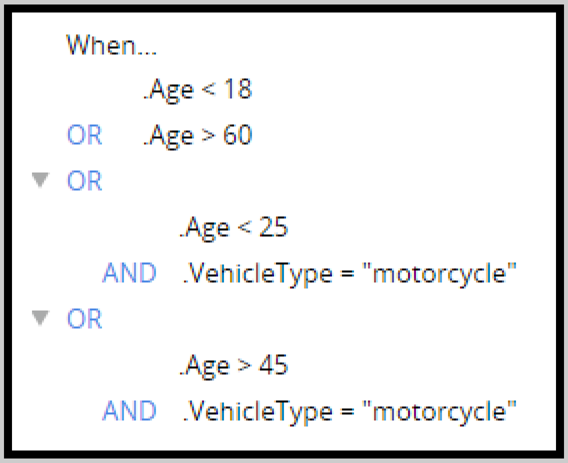
A	.Employee.MaritalStatus	=	"Married"
B	.NumberOfDepenedents	>=	1
C	.Employee.MaritalStatus	=	"Single"
D	.Employee.MaritalStatus	=	"Divorced"

Logic string: A OR (B AND (C OR D))

KNOWLEDGE CHECK

ANSWER

An application adds a risk premium to an auto insurance policy whenever the following when condition returns a result of true.



```
When...
      .Age < 18
OR    .Age > 60
OR
         .Age < 25
   AND   .VehicleType = "motorcycle"
OR
         .Age > 45
   AND   .VehicleType = "motorcycle"
```

Which of the following drivers has a risk premium added to the insurance policy?

- Motorcycle rider, age 30
- Any driver, age 67
- Motorcycle rider, age 22

The 67-year-old driver and the 22-year-old motorcycle rider are assessed a risk premium.

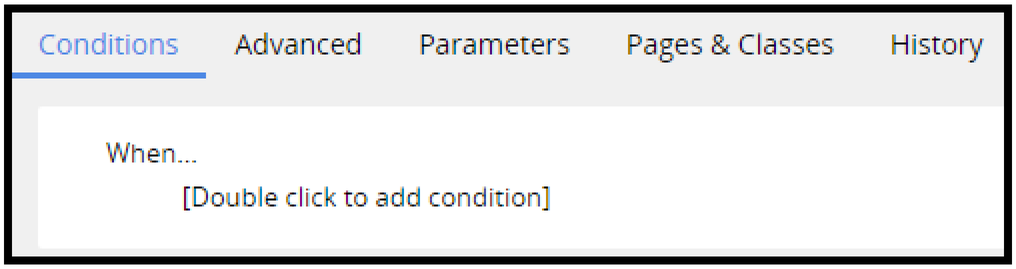
Configuring a when rule

Configure a when rule to perform a true/false test based on the relationship between one or more property values, literal constants, or functions.

Creating a when condition

The top node in a when conditions tree is labeled **When**. Follow these steps to configure the top node:

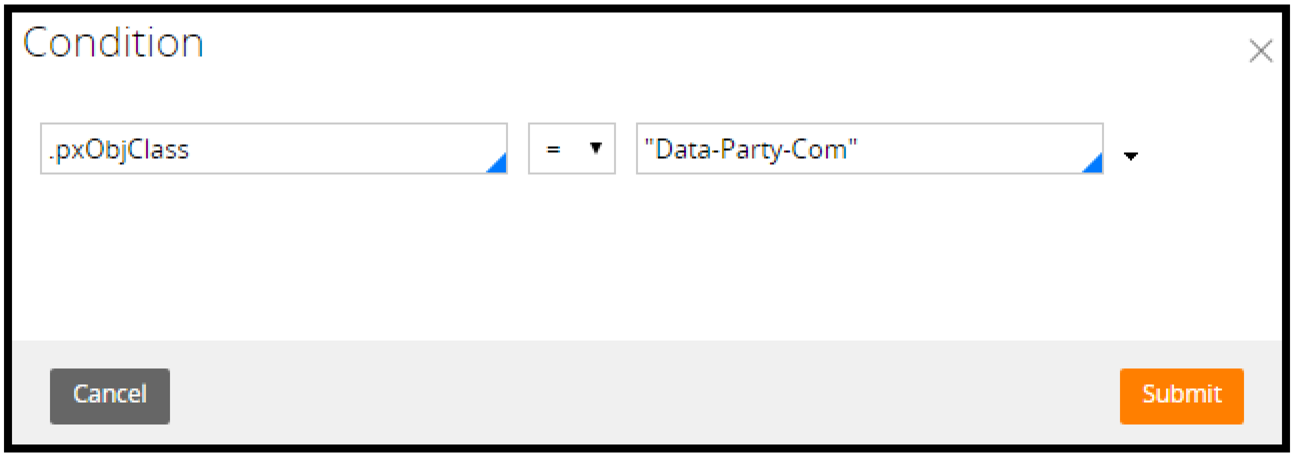
1. Double-click the link beneath the **When** node.



When...
[Double click to add condition]

   The system displays the Condition dialog that contains the default condition. The default condition compares two values.

2. Enter properties, literal constants, or function calls and parameters in the fields, and select a relational operator.



.pxObjClass = "Data-Party-Com"

   Optionally, click the **down** arrow at the end of the row and select from a list of standard conditions.

   Tip: If you select `[expression evaluates to true]`, use the Expression Builder for guided assistance. Click the **gear** icon to open the Expression Builder.

3. Click **Submit** to close the dialog and display the condition on the tree.

Adding more conditions to the conditions tree

To further edit the conditions tree, use the Actions menu. From this menu, you can edit the selected condition, delete the selected condition, or add conditions to the tree. To open the context menu, either right-click the expression or click **Actions** to the right of the node.

Follow these steps to add a condition to the current node:

1. From the **Actions** menu, select **Insert Condition**.
2. Enter the condition in the Condition dialog.
3. Click **Submit**. When the Condition dialog closes, the condition appears in the conditions tree, preceded by a Boolean AND or OR operator. To change the operator, click the operator and select the appropriate operator from the list.

Follow these steps to add a subnode with a condition:

1. From the **Actions** menu, select **Insert Group**.
2. Enter the condition in the Condition dialog and optionally add a label for the group.
3. Click **Submit**. When the Condition dialog closes, the node for the group is indented below the preceding node.

Configuring decision tables and decision trees

Introduction to Configuring Decision Tables and Decision Trees

Decision tables and decision trees are fundamental to enforcing business decisions. You can use these decision rules in flows, routers, activities, and declare expressions. For example, a decision rule can automatically select a connector to advance a case. A decision rule can also automatically route cases to the correct worklist or workbasket. Decision tables and decision trees allow you to design complex decision logic that goes far beyond simple yes/no decisioning.

After this lesson, you should be able to:

- Explain how decision tables can model decisions.
- Configure a decision table.
- Explain how decision trees can model decisions.
- Configure a decision tree.
- Describe the options for unit testing decision tables and trees.
- Describe how to test a decision for completeness, conflicts, and decision logic.

Decision tables

If you are asking a yes/no question when using an automated decision process, then a when rule serves the purpose. For example, a decision such as “Does this purchase order require additional approval?” works well with a when condition. However, if you need to test the values of multiple properties to answer questions such as “What promotional offer should the company offer?”, you can use a decision table.

For example, you can configure a decision table to determine the discount for customers at different spending thresholds. Using the decision table, customers who purchased more than USD1,000 in the previous year and have been a customer for more than five years are entitled to a 20 percent discount for purchases greater than USD50. Customers who purchase more than USD1,000 but have been a customer for less than five years are entitled to a 15 percent discount on purchases greater than USD100. Customers who do not meet either condition are not entitled to a discount.

Decision tables resemble a spreadsheet with rows and columns. This commonly used format helps non-technical users quickly understand how the decision logic works. Your organization may choose to delegate to business users responsibility for updating the decision table. For example, when the organization changes its discount rates, managers need to update the rate. A delegated decision table allows these users to quickly adjust the table to make the update, rather than waiting for IT to make the changes required.

You can reference decision tables in decision shapes to decide which connector to use when advancing a case in a process. You can also use decision tables in declare expressions, activities, or routers.

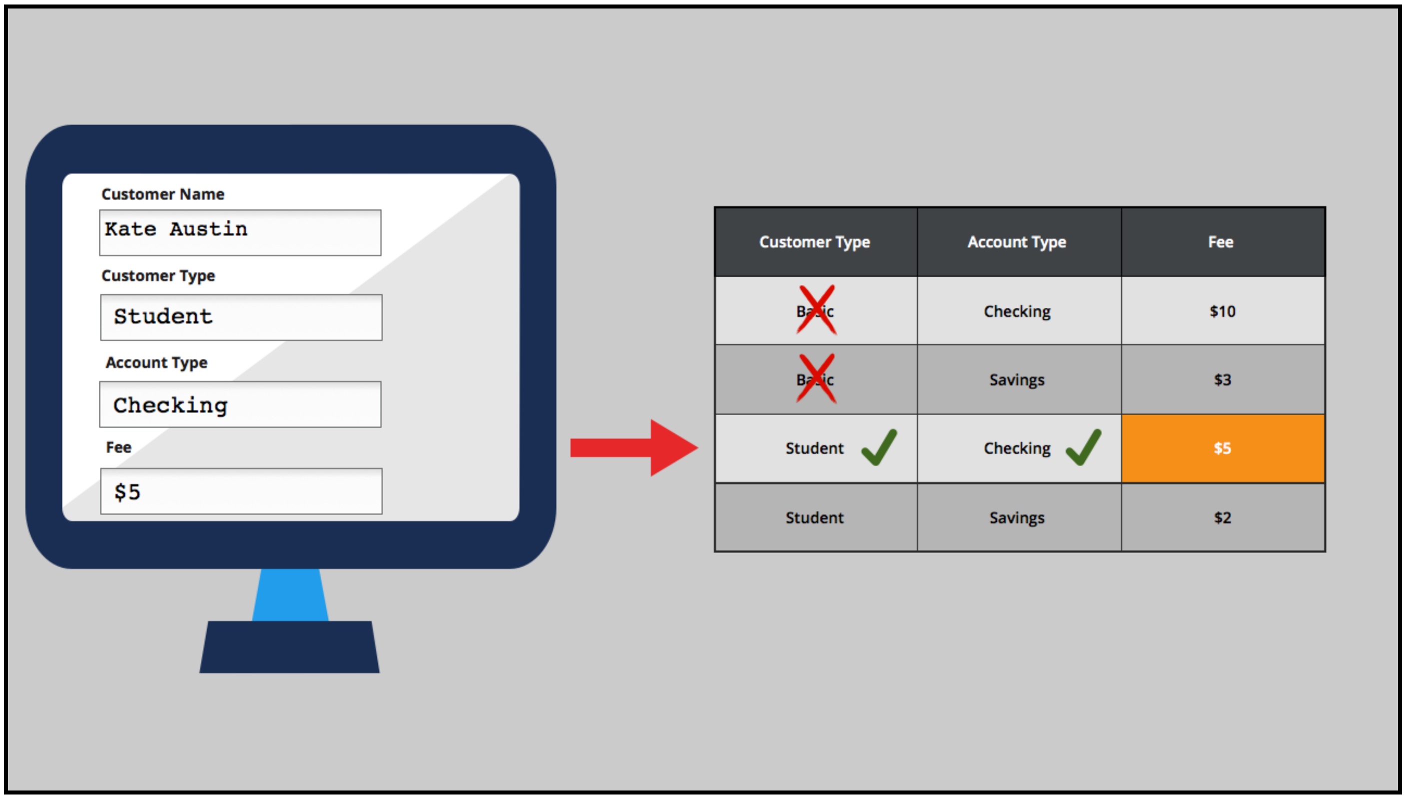
Decision table logic

Decision tables are a good approach when you use a set of properties or expressions to arrive at a decision. Watch the following video to see how the columns and rows are configured in a decision table.

In the following example, a banking application uses a decision table for determining monthly maintenance fees. To find the correct fee, the decision table compares the account type and customer type property values on the table to the input values.

The table has rows for evaluating the correct fee for each combination. For instance, if the account type equals Checking, and the customer type equals Basic, then the system returns a value of USD10.

The following example shows how the decision evaluation works based on the account type and customer type property values.



Customer Name	Kate Austin
Customer Type	Student
Account Type	Checking
Fee	$5

Customer Type	Account Type	Fee
~~Basic~~	Checking	$10
~~Basic~~	Savings	$3
Student ✓	Checking ✓	$5
Student	Savings	$2

By default, a condition uses an equal comparison operator (as configured in the previous example). If you are using numeric conditions, you can also specify greater than or less than comparison operators. For example, you can create a condition so that if the savings account balance is greater than USD1,000, then a customer is not charged any fees. You can also use value ranges to define the comparison. For instance, if the savings account balance is greater than USD500 but less than USD1,000, the customer can only be charged a checking account fee.

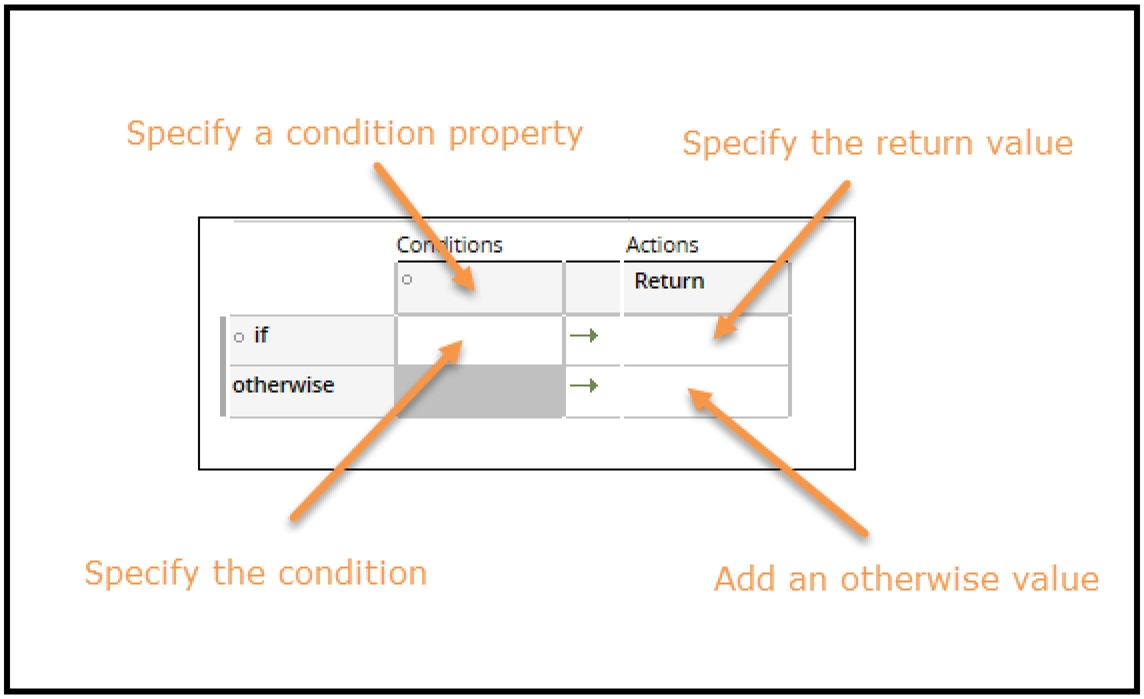
KNOWLEDGE CHECK

ANSWER What is the main reason for using a decision table rather than a when rule for automating a decision?

You need to test the values of multiple properties to make the decision.

How to configure a decision table

To design your decision table, you first specify a property or expression in the Conditions column header. Then, on the first row, you enter a value in the column that defines the condition. Under the **Return** column, enter the result that is returned by the table when the condition evaluates to true. Finally, in the **otherwise** row, enter a value that is returned if none of the conditions evaluate to true.



To create a decision table, in the Application Explorer, select a class. Then, right-click and select **+Create > Decision > Decision Table**.

Specify a condition property or expression

Configure a cell in the header row to define the property or expression used in the evaluation. Clicking the cell opens the **Decision Table property chooser** tool.

The tool allows you to:

- Select a property or create an expression used for the evaluation.
- Enter a label that appears on the table.
- Select the comparison operator. The default is the equals sign (=). If you select a numeric property, you can use greater than/less than operators. You can also use these operators to define a range.

You can add columns to create multiple conditions.

Specify the condition

In the rows under the conditions column, enter the value you want to compare during the evaluation. You can enter a literal value, a property, or an expression. For example, if the condition property is

Account Type, you can enter *checking* as a value. You can add multiple rows for each combination of conditions. If you have more than one condition column, you must enter a condition in at least one column.

Specify the return value

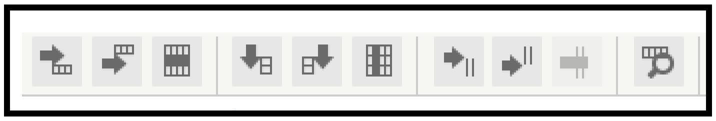
Under the **Return** column, enter a literal value, a property, or an expression. This is the result the table returns if all the conditions in the row evaluate to true.

Add an otherwise value

Be sure to add a value in the **otherwise** row to ensure that the decision always returns a result. A processing error can occur If there is no result.

Adding or deleting columns and rows

You can add or delete columns and rows using the following controls that are available above the table. Select any cell in the table to activate the controls.

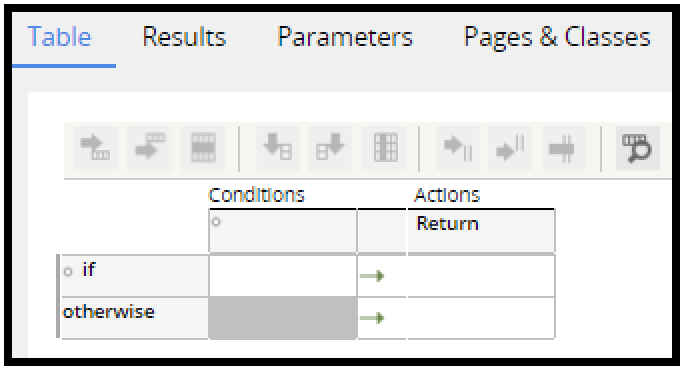


Configuring a decision table

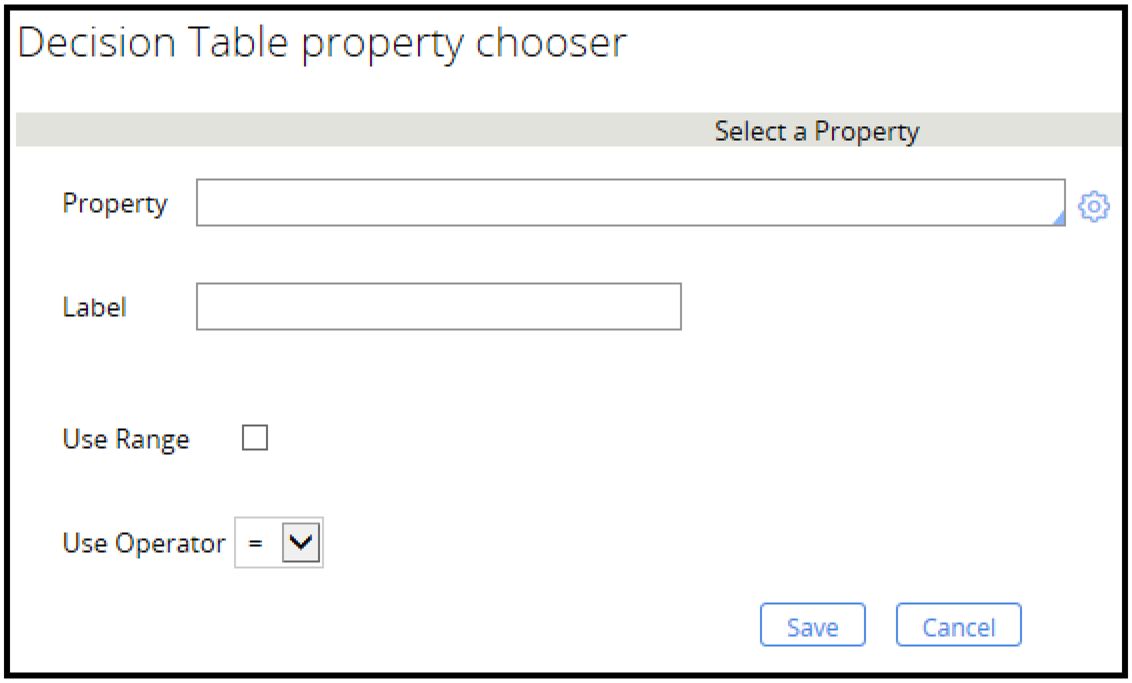
Create a decision table to derive a value that has one of a few possible results, where each result can be detected by a comparison condition. A decision table lists two or more rows, each containing one or more conditions and a result.

Follow these steps to create and configure a decision table for automating a decision:

1. Open the Application Explorer.
2. Select the class in which you want to create the decision table.
3. Right-click and select **+Create > Decision > Decision Table**. The New Record form opens.
4. In the **Label** field, enter a name that describes the purpose of the table.
5. Click **Create and open**. The decision table rule form opens.

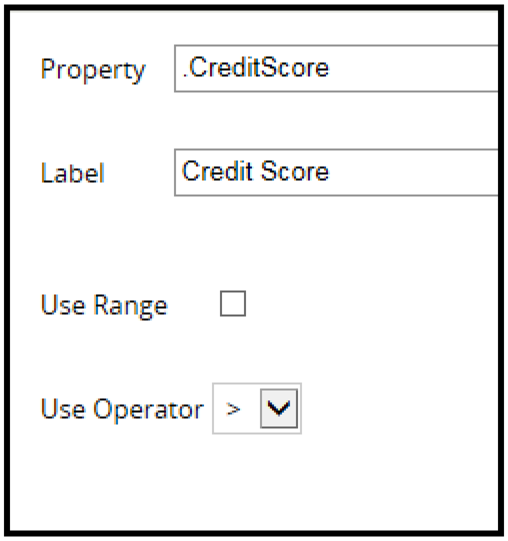


6. On the table under **Conditions**, click the empty header cell. The Decision Table property chooser dialog opens.



7. In the **Property** field, enter or select a property. You can alternately click the gear icon to build an expression.
8. In the **Label** field, enter the name of the property that you entered in the column header.
9. In the **Use Operator** drop-down, select a comparison operator.

   The following example shows a competed dialog.



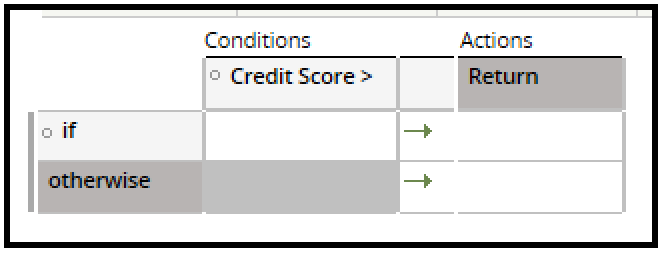
Property .CreditScore

Label Credit Score

Use Range ☐

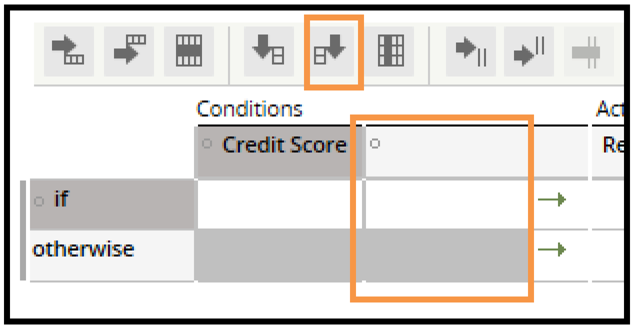
Use Operator >

10. Click **Save**. The Decision Table property chooser dialog closes, returning you to the decision table rule form.



	Conditions		Actions
	Credit Score >		Return
if		→	
otherwise		→	

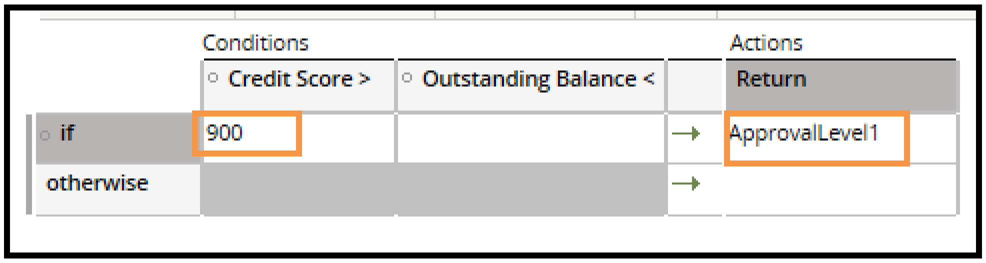
11. If you want to add another condition property, add a column to the right of the first column by selecting the **add column** icon on the control header.



	Conditions			Act
	Credit Score			Re
if			→	
otherwise			→	

12. Add a second property to the new column.
13. In the **if** row, click the empty cell under the first property and enter a value.

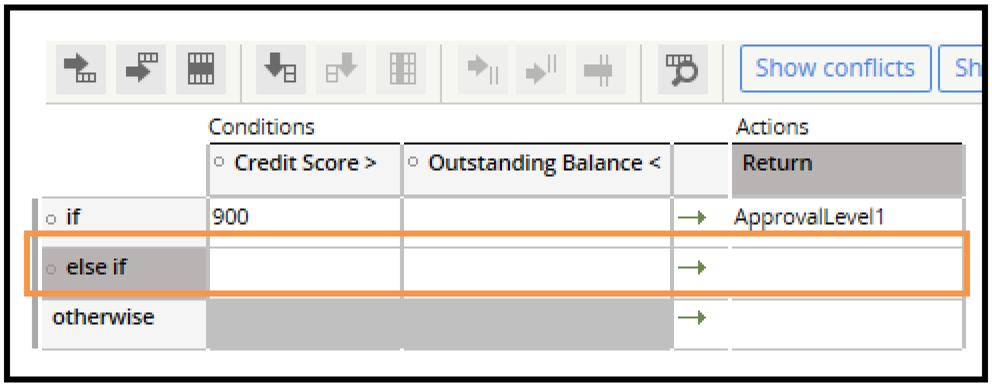
14. In the **Return** column, enter a return result. The following image shows the first condition set.



	Conditions			Actions
	Credit Score >	Outstanding Balance <		Return
if	900		→	ApprovalLevel1
otherwise			→	

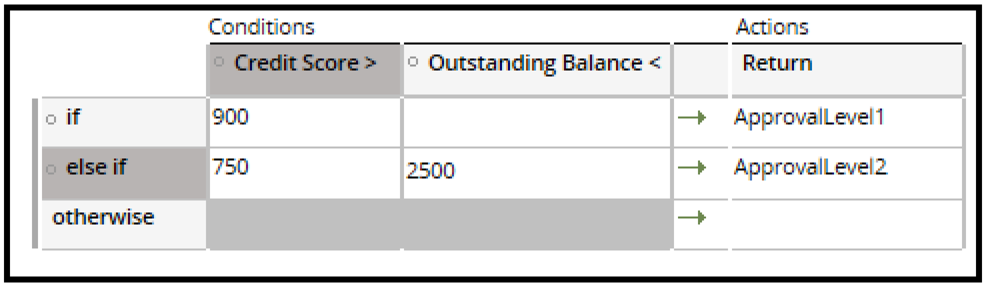
Note: If you are using two or more conditions, you must enter at least one condition in the row. In the previous example, only the Credit Score condition must be true in order to return Approval Level 1. The Outstanding Balance value does not affect the decision.

15. If you want to add conditions, select the **add row** icon on the control.



	Conditions			Actions
	Credit Score >	Outstanding Balance <		Return
if	900		→	ApprovalLevel1
else if			→	
otherwise			→	

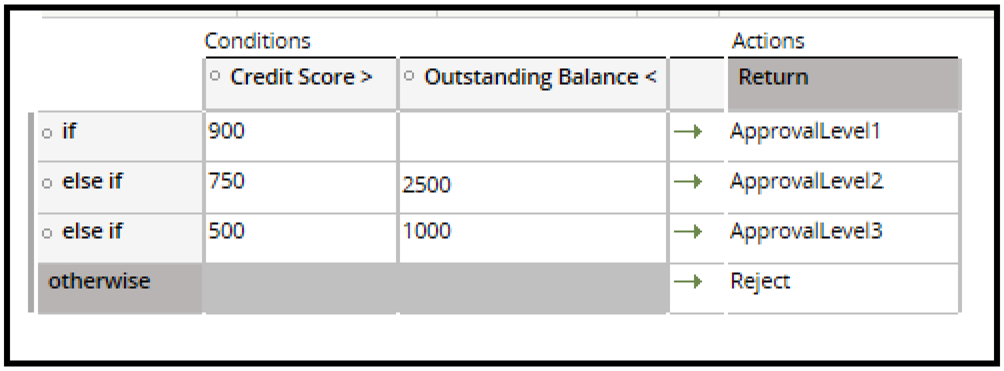
16. Enter values in the first and second columns and a return value. In this example, values for Credit Score and Outstanding Balance are entered.



	Conditions			Actions
	Credit Score >	Outstanding Balance <		Return
if	900		→	ApprovalLevel1
else if	750	2500	→	ApprovalLevel2
otherwise			→	

17. Enter another row and return action. Repeat this process until you add all the rows required to create the decision table.

18. Add a return value to the **otherwise** row.



	Conditions			Actions
	○ Credit Score >	○ Outstanding Balance <		Return
○ if	900		→	ApprovalLevel1
○ else if	750	2500	→	ApprovalLevel2
○ else if	500	1000	→	ApprovalLevel3
otherwise			→	Reject

19. Click **Save**.

Decision trees

As an alternative to decision tables, you can use decision trees to handle logic that calculates a value from a set of test conditions. Both decision tables and decision trees evaluate conditions and return results when a comparison evaluates to true. Only decision trees let you apply if... then... else logic. This means that a true comparison can result in more than one comparison.

For example, a human resources application contains a process for assessing a job candidate. The candidate receives a set of ratings during the interviews. These ratings are evaluated to determine whether to extend a job offer to the candidate. A decision tree can be configured to automatically use the ratings to decide whether the candidate is qualified. The decision starts at the top of the tree and proceeds downward. Each yes advances the evaluation.

1. Job history and reference rating must be greater than 60 percent.

   If yes, then continue to condition 2.

   Else, not qualified.
2. Interview rating must be greater than 40 percent.

   If yes, then continue to condition 3.

   Else, not qualified.
3. Interpersonal skills must be greater than 20 percent.

   If yes, then the candidate is eligible for a job offer.

   Else, not qualified.

Like decision tables, you can reference decision tables in flow shapes, declare expressions, activities, or routers.

The following video describes the structure of a decision tree.

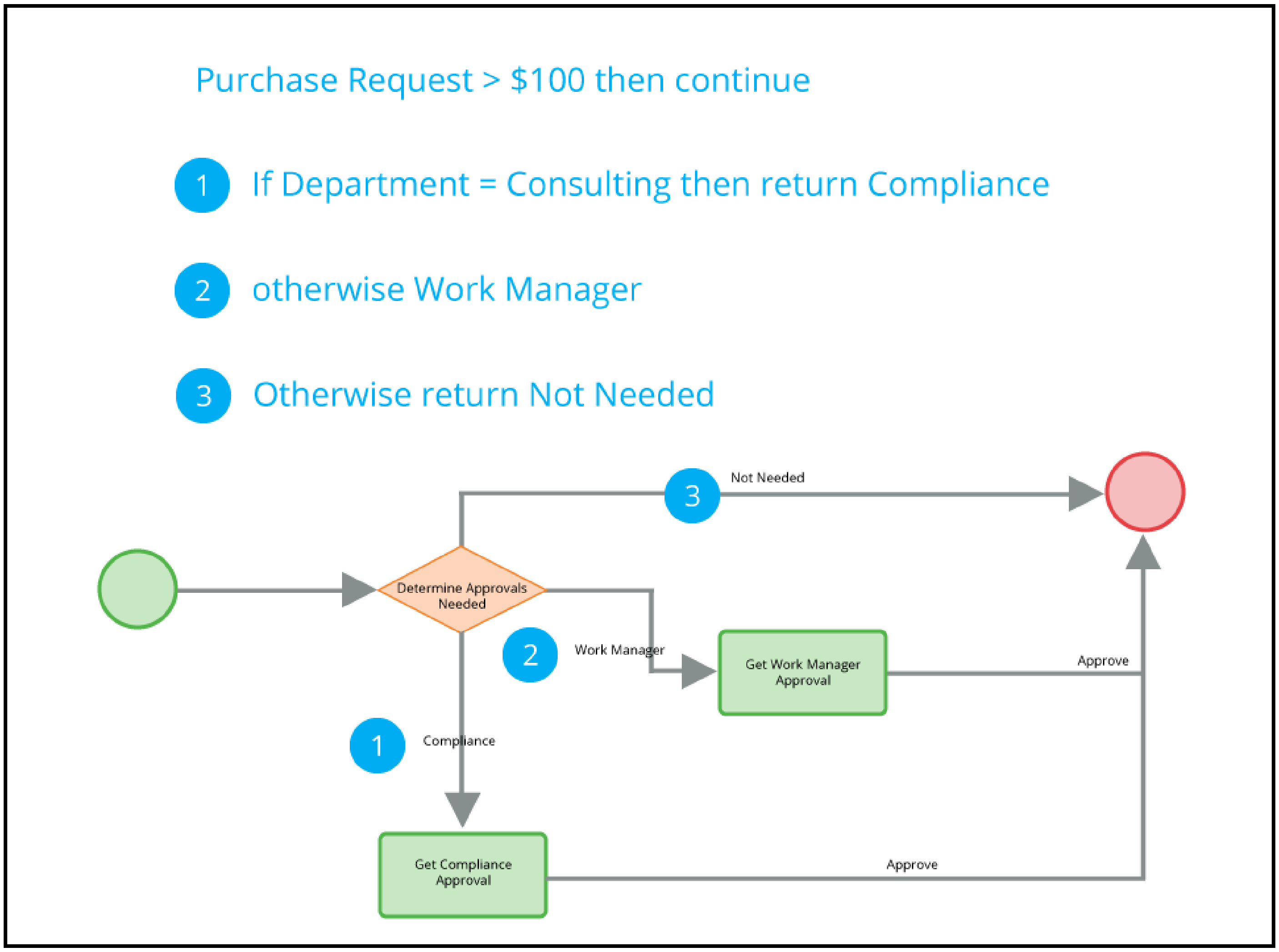
Decision tree logic

Decision trees contain condition branches — a comparison value, a comparison operator, and an action. The action can be to return a result, to continue the evaluation, or stop the evaluation. The branches are organized in a hierarchical tree structure. Typically, you specify common conditions and results at the trunk of the tree. You then extend the tree outward to more-specific conditions and their actions. When the decision tree is invoked, the system evaluates the top row, and continues until it reaches a result that evaluates to true. The result is returned to the system. If the system processes through all the branches but does not reach a returned result, the system returns the final otherwise value.

Nesting branches

You can organize decision tree branches in a nested structure. For example, assume that when a purchase request is submitted, three possible outcomes exist. The first condition states that if the request is for more than USD100, then the request must be approved. Two possible approval results exist. If the request is submitted by the Consulting department, the request advances to the Compliance department for approval. Otherwise, the request advances to the work manager for approval. If the request is for less than USD100, then approval is not needed.

The following image shows how the decision tree would be configured to advance the request to the correct connector. Note that the Compliance and Work Manager approval conditions are nested beneath the purchase request condition. This condition must be true before the other conditions are evaluated. If the request is for less than USD100, the tree does not need to evaluate the request any further and returns the result Not Needed.

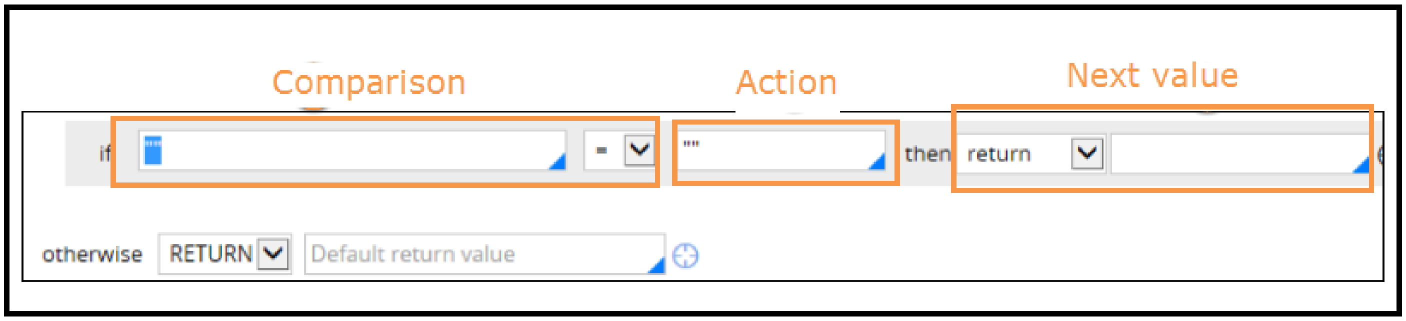

Purchase Request > $100 then continue
1 If Department = Consulting then return Compliance
2 otherwise Work Manager
3 Otherwise return Not Needed
3 Not Needed
Determine Approvals Needed
2 Work Manager
Get Work Manager Approval
Approve
1 Compliance
Get Compliance Approval
Approve

KNOWLEDGE CHECK

ANSWER	When would you use a decision tree rather than a decision table to automate a decision?
When you want to apply if...then...else logic to evaluate a set of conditions.	

How to configure a decision tree

To design your decision tree, enter the If/Then logic in the three-column array for each branch. Each column consists of an unlabeled field. The columns are comparison, action, and next value.



To create a decision tree, in the Application Explorer, select a class. Then, right-click and select **+Create > Decision > Decision Tree**.

Configure the columns

In the comparison column, select the property to evaluate. Use the drop-down to select the comparison operator.

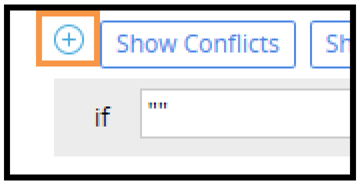
In the action column, specify the literal value or property to compare against.

In the next value column, use the drop-down to select the outcome of the comparison. The options are:

- return — If the condition evaluates to true, the system returns a result value that you define in the field to the right of the drop-down.
- continue — Causes the next branch of the decision tree to nest within this branch. The system indents the next branch on the form.
- otherwise — Select only as the bottom of the tree. The value in the right column of this row becomes the result of this decision tree evaluation. Typically, you use this option when configuring indented branches.

As a best practice, enter a default return value in the **otherwise** row at the bottom of the tree. This helps ensure that there is a returned value if no other conditions evaluate to true.

You can add rows to by selecting the **crosshair** icon next to the **Show Conflicts**button on the form.



To reorder rows, hover your mouse over a row, and then drag and drop it.

Configuring a decision tree

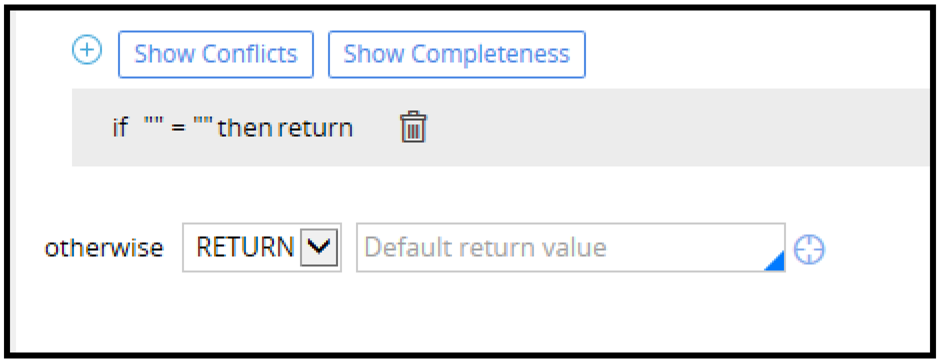
Create a decision tree to use if... then... else decision logic that calculates a value from a set of test conditions organized as a tree structure.

Follow these steps to create and configure a decision table for automating a decision:

1. Create the decision tree.
2. Add a condition and result.
3. Optionally, nest the conditions.

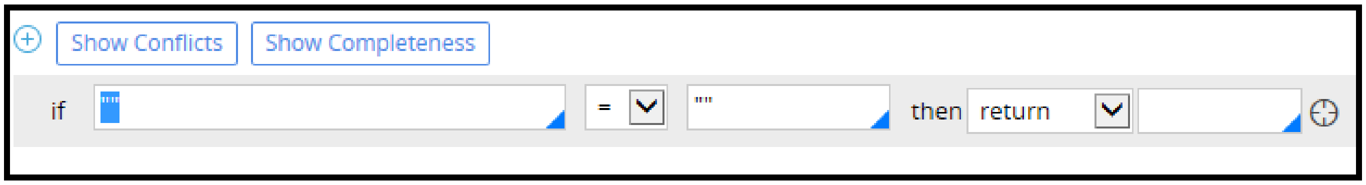
Create the decision tree

1. Open the Application Explorer.
2. Select the class in which you want to create the decision table.
3. Right-click and select **+Create > Decision > Decision Tree**. The New Record form opens.
4. In the **Label** field, enter a name that describes the purpose of the tree.
5. Click **Create and open**. The decision tree rule form opens.



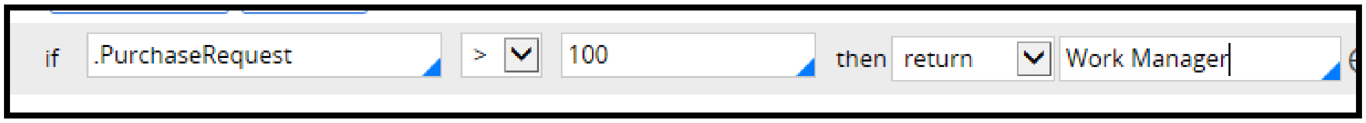
Add a condition and the result

1. On the form, select the branch to display the columns.

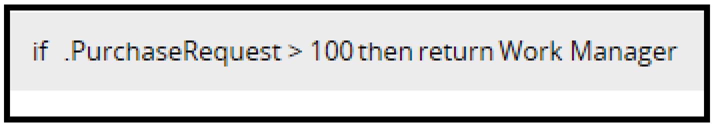


2. In the first field, enter a property or a literal value.
3. In the drop-down to the right of the field, select a comparison operator.
4. In the next field, enter a property or literal value used in the comparison.

5. In the **then** drop-down, select the action you want the system to perform when the condition evaluates to true. To return a result when the condition evaluates to true, select return.
6. In the field to the right of the drop-down, enter a property or value result that you want the system to return. The following shows a completed condition.

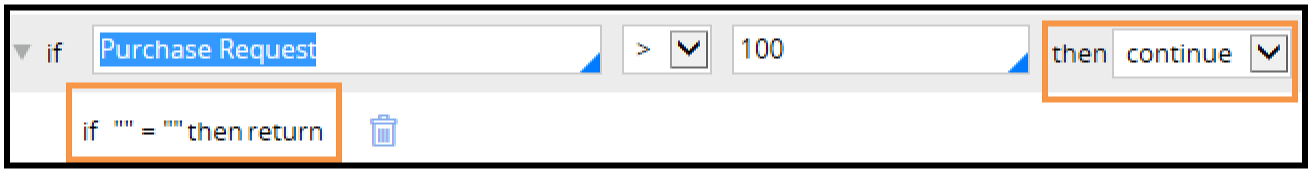

if .PurchaseRequest > 100 then return Work Manager

When you click out of the branch, the condition is displayed on the form.

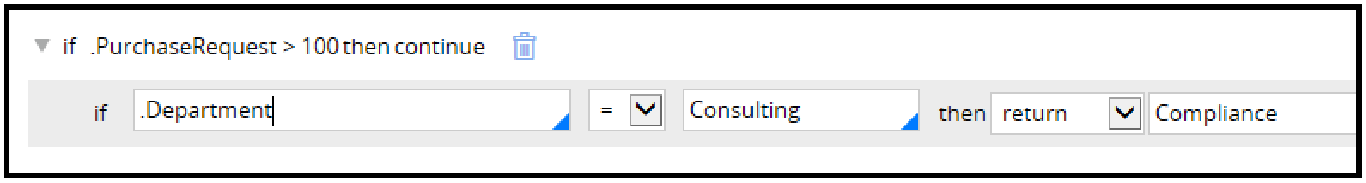

if .PurchaseRequest > 100 then return Work Manager

Nest the conditions

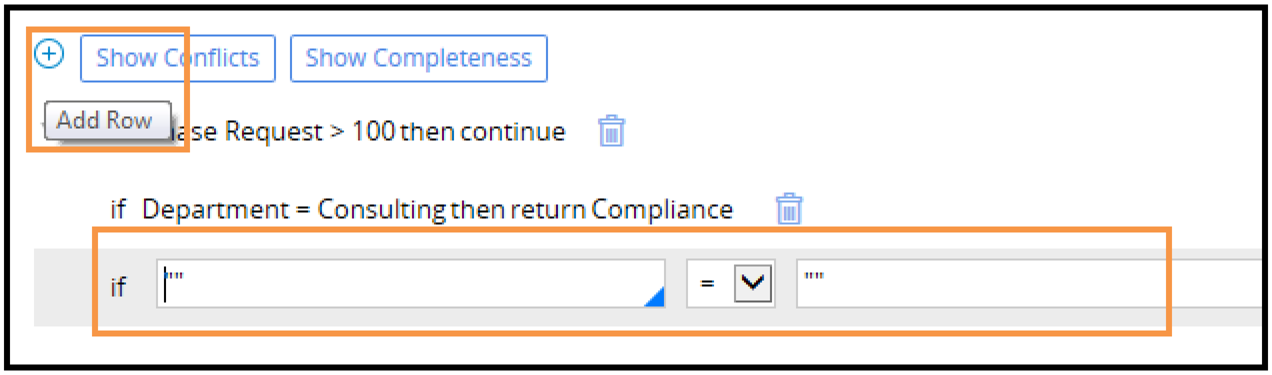
1. When you create a condition, in the **then** drop-down, select continue to create a nested branch. The return field to the right of the drop-down is removed. An indented condition branch is displayed under the first row.


if Purchase Request > 100 then continue
if "" = "" then return

2. Select the second branch to display the columns.
3. Specify a condition and result in this branch. The following condition is an example of an indented branch.


if .PurchaseRequest > 100 then continue
if .Department = Consulting then return Compliance

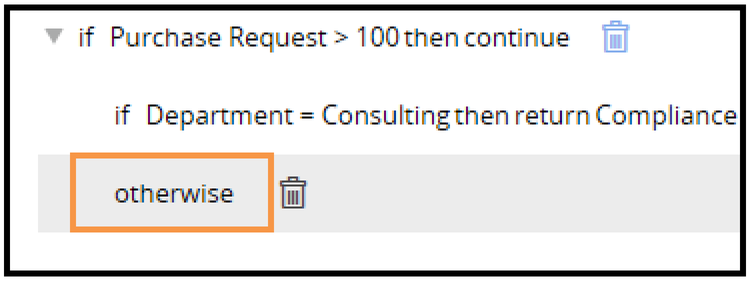
4. To add an otherwise condition to the indented branch, select the **crosshair** icon.



...ase Request > 100 then continue
if Department = Consulting then return Compliance
if "" = ""

5. In the second branch, select otherwisein the then drop-down. The system changes the branch to

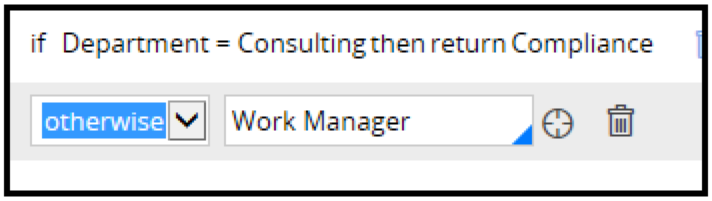
otherwise.


if Purchase Request > 100 then continue

if Department = Consulting then return Compliance

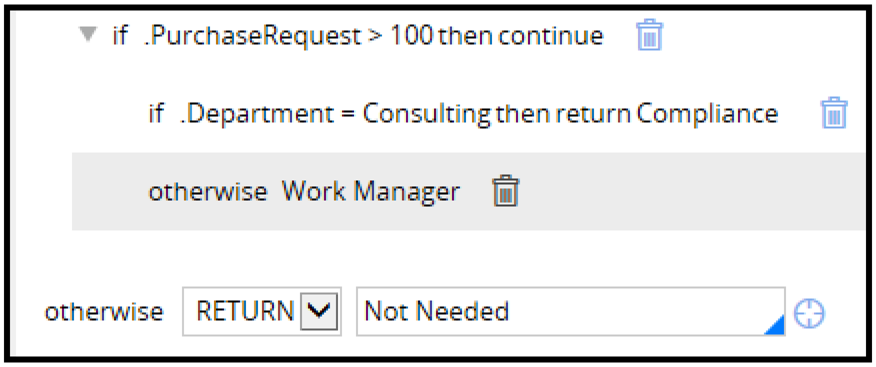
otherwise

6. Select the **otherwise** row to display the result field and to enter a value.


if Department = Consulting then return Compliance

otherwise Work Manager

7. Enter a value in the otherwise field at the bottom of the table. This is the result when neither of the nested conditions evaluate to true. The following image provides an example of a completed decision tree.


if .PurchaseRequest > 100 then continue

if .Department = Consulting then return Compliance

otherwise Work Manager

otherwise RETURN Not Needed

8. Click **Save**.

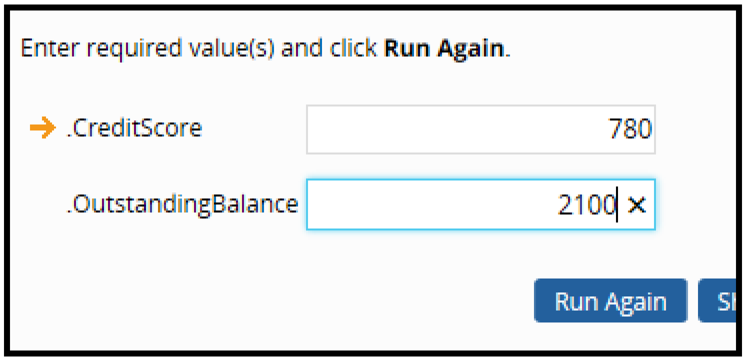
How to unit test a decision table or decision tree

Testing a decision table or decision tree on its own before testing it in the context of the entire application is called **unit testing**. Since decisions are evaluated automatically, they can have a significant impact on case processing. Ensuring that the decision logic is correct helps avoid troubleshooting the process if you get unexpected results. You can unit test decision rules by testing the logic, checking for conflicts in the logic, and checking for completeness.

Test for logic

You can test the logic of a decision rule by entering test values and running the rule to observe the results. If you do not see the expected results, make sure that the properties and comparison operators are correct.

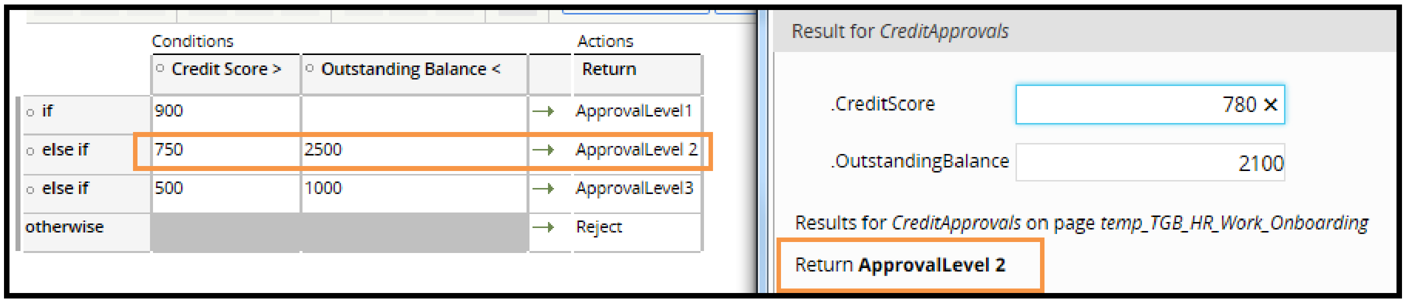
To test for logic, on a decision rule form, select the **Actions > Run**. The system displays a test page for entering test values. On the form, click **Run Again** after you enter each value as shown in the following example.



Enter required value(s) and click **Run Again**.

.CreditScore	780
.OutstandingBalance	2100

After you have entered values for all the conditions and click **Run Again**, the form returns the corresponding result from the decision table or tree. The following logic test shows that the input values returned the correct result.



	Conditions		Actions
	Credit Score >	Outstanding Balance <	Return
if	900		ApprovalLevel1
else if	750	2500	ApprovalLevel 2
else if	500	1000	ApprovalLevel3
otherwise			Reject

Result for *CreditApprovals*

.CreditScore	780
.OutstandingBalance	2100

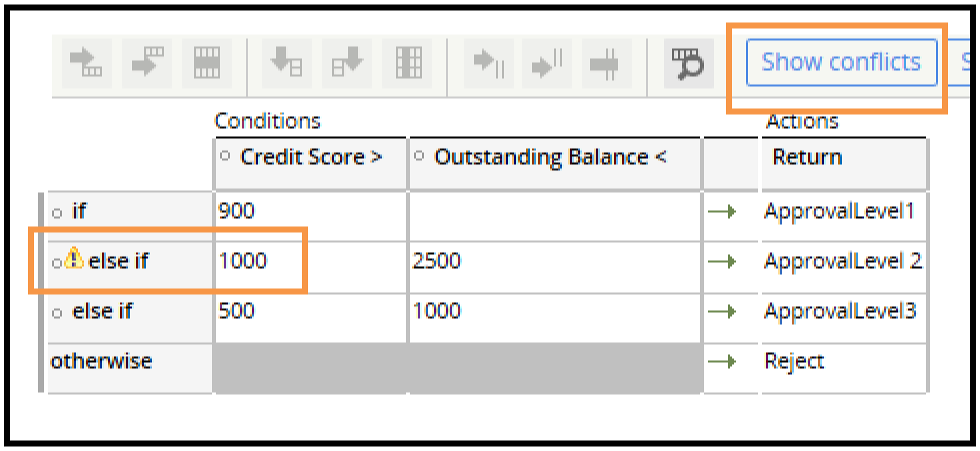
Results for *CreditApprovals* on page *temp_TGB_HR_Work_Onboarding*

Return **ApprovalLevel 2**

Test for conflicts

Checking for conflicts shows you if your decision rule prevents one or more of its rows or branches from ever being used. For example, assume your decision table contains a row that tests for purchase requests that exceed USD300. The next row tests for purchase requests that exceed USD500. The second row may never be evaluated, because the upper row includes that condition.

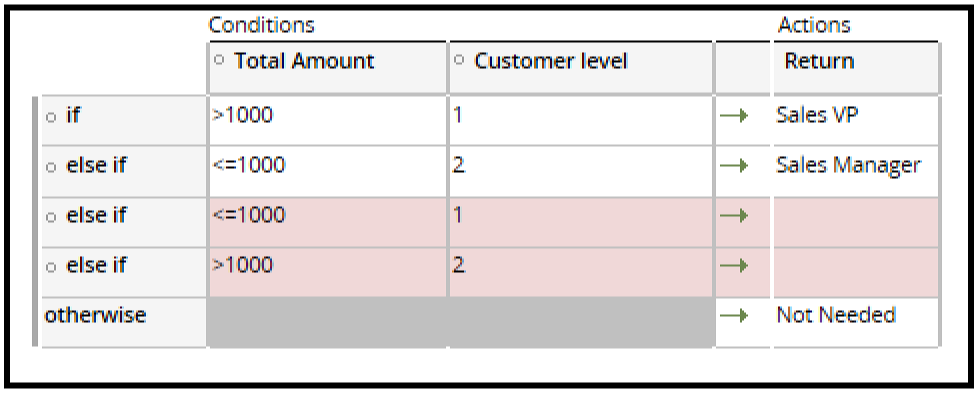
To test for conflicts, on the decision form, click **Show Conflicts**. If a conflict exists, a warning is displayed on the row causing the conflict. In the following example, the condition Credit Score >1000 cannot be evaluated because it is a larger value than the 900 condition that is evaluated first.



	Conditions			Actions
	Credit Score >	Outstanding Balance <		Return
if	900		→	ApprovalLevel1
else if	1000	2500	→	ApprovalLevel 2
else if	500	1000	→	ApprovalLevel3
otherwise			→	Reject

Test for completeness

To test for completeness, on the decision form, click **Show completeness**. The system adds rows to indicate values that will not be evaluated. The results are suggestions. You can add return results to additional rows if you think the decision rule needs a more detailed evaluation of the values.



	Conditions			Actions
	Total Amount	Customer level		Return
if	>1000	1	→	Sales VP
else if	<=1000	2	→	Sales Manager
else if	<=1000	1	→	
else if	>1000	2	→	
otherwise			→	Not Needed

UI DESIGN

Designing a UI form

Introduction to Designing a UI Form

A user interface (UI) is the means by which users interact with a system. A UI can be a data entry form in which users provide information for filing a claim, a screen that displays the legal terms that users must accept before opening a bank account, or a list of transactions during the past month. Users receive information from and provide data to an application through a UI.

In this lesson, you learn how to build a Pega UI with sections, and how to arrange UI elements in a section with layouts.

After this lesson, you should be able to:

- Understand the hierarchical structure of a Pega UI.
- Explain the role of a section in UI design.
- Construct a section.
- Create a dynamic layout inside a section.
- Articulate some guidelines for UI design.
- Inspect a UI with the Live UI tool.

User interface structure

When you build end-user forms, you need to focus on the user interface (UI) rules you are most likely to configure. Assume you are developing an application for hiring employees. This process has many steps, -- for example, collecting applicant information, reviewing the information, and scheduling interviews. Each step requires its own user form so that users can perform their task. In most applications, you will create and update many user forms.

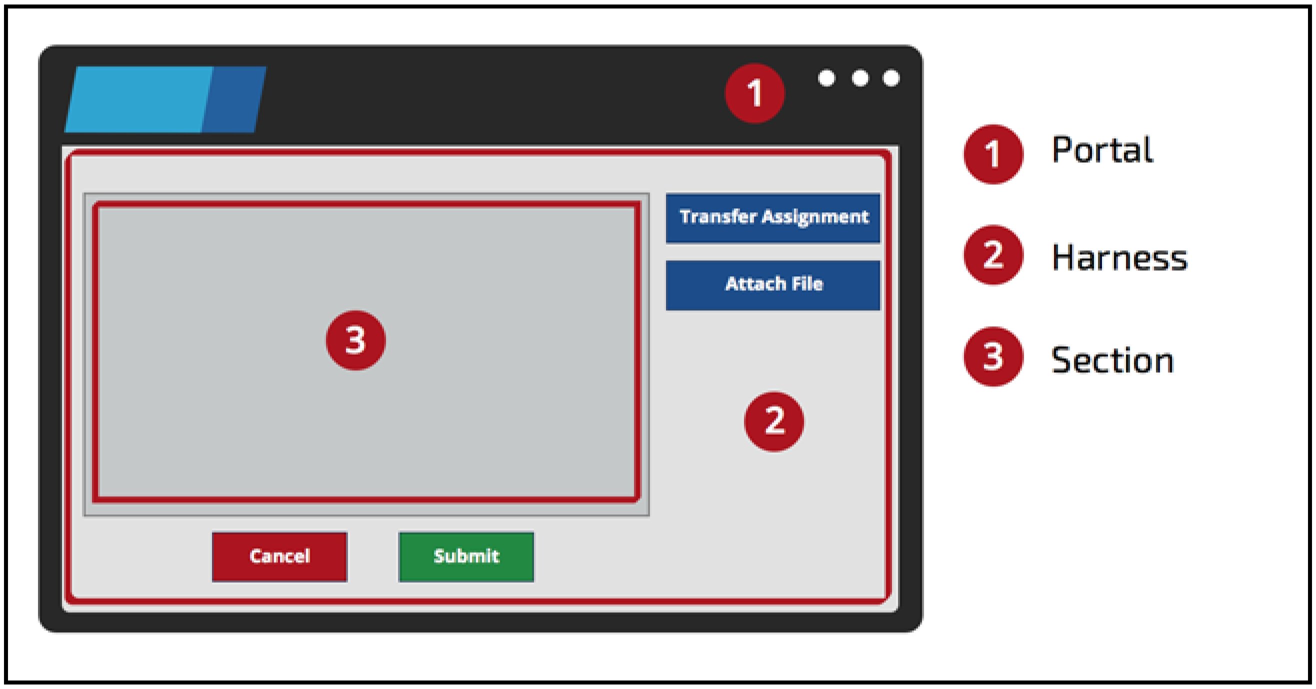
To help make designing forms easier, Pega provides a specific set of UI rules. You focus on only two types of rules when building forms. First, you create a rule that defines the form and contains the form's contents. Then, inside that rule, you add rules such as fields or buttons so that users can interact with the form and complete their work.

Then, Pega organizes all UI rules in a nested structure. This structure shows you how all the UI elements fit together. Most of the UI structure is already built for you. For example, the structure contains areas in which users monitor reports or manage worklists. Nested within the structure are the rules you use when you design your forms. You do not need to update any of the other UI rules. The nested structure helps you find the rules you need when you build forms.

The UI rules

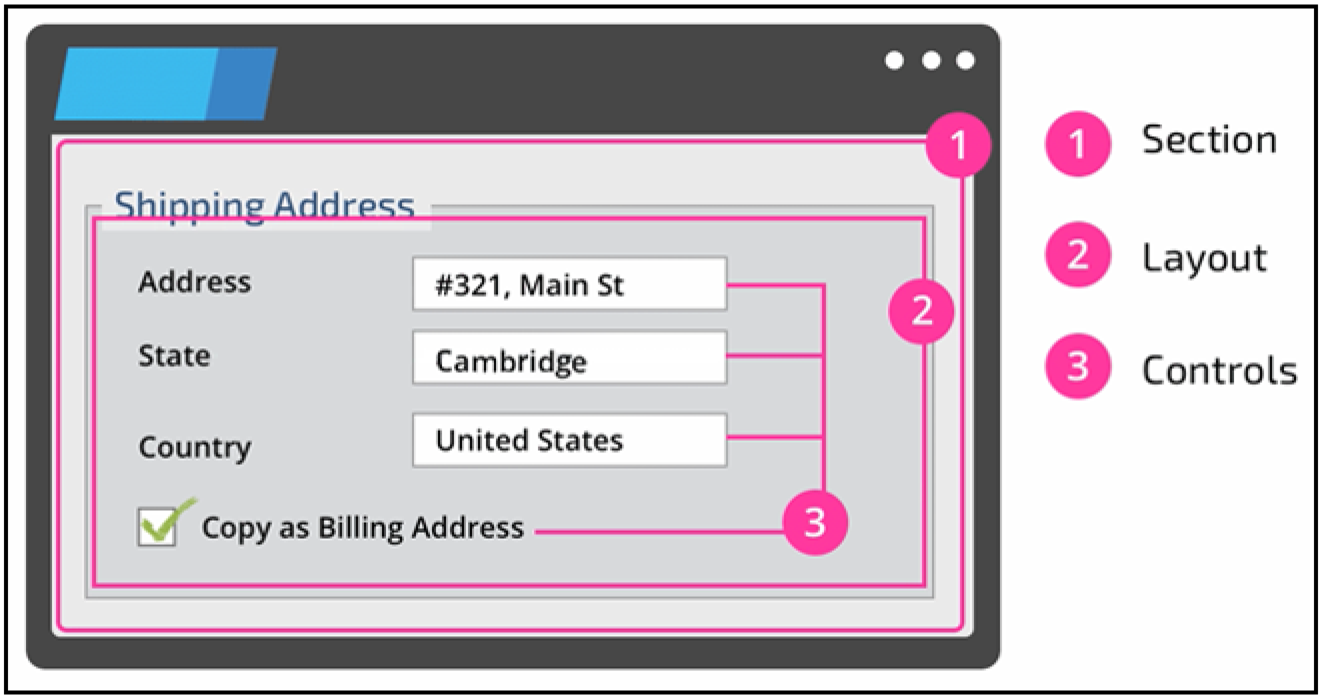
Every Pega UI is built inside a **portal rule**. A portal rule does not hold any visual elements such as other UI rules. The main purpose of a portal rule is to set up workspaces for users. Pega provides standard portals that are built with portal rules. The standard user portal is a workspace that supports users and managers as they create, update, route, and resolve work items. Portal rules reference **harness rules** for content. Harnesses frame the work areas in which users process cases. A harness provides tools that let users manage the assignment process. Harnesses give users the ability to cancel, save, or submit their work. While working in the assignment, users can transfer their assignments to other users, attach files to the case, or send email correspondence.

When a case reaches an assignment, the flow action presents the appropriate harness that allows users to perform tasks defined for the assignment. The flow action also references a **section rule** and displays it in the harness. The section is the form in which users work when they perform their task and complete their assignment. When you build a user form you create a section rule.



Pega provides rules called **controls** that you add to a section to help users interact with the form. For example, assume you are designing a form for collecting information when users apply for loans. You add controls that allow users to provide the information. For example, you add text box controls so that users can enter their name and address, a drop-down control that allows users to select a type of loan, and a check box that allows users to indicate whether they are existing customers.

Sections use **Layouts** that organize the controls in a series of rows and columns. Each cell within a layout can contain a control. You can configure many layout designs in order to make user interactions intuitive and efficient.

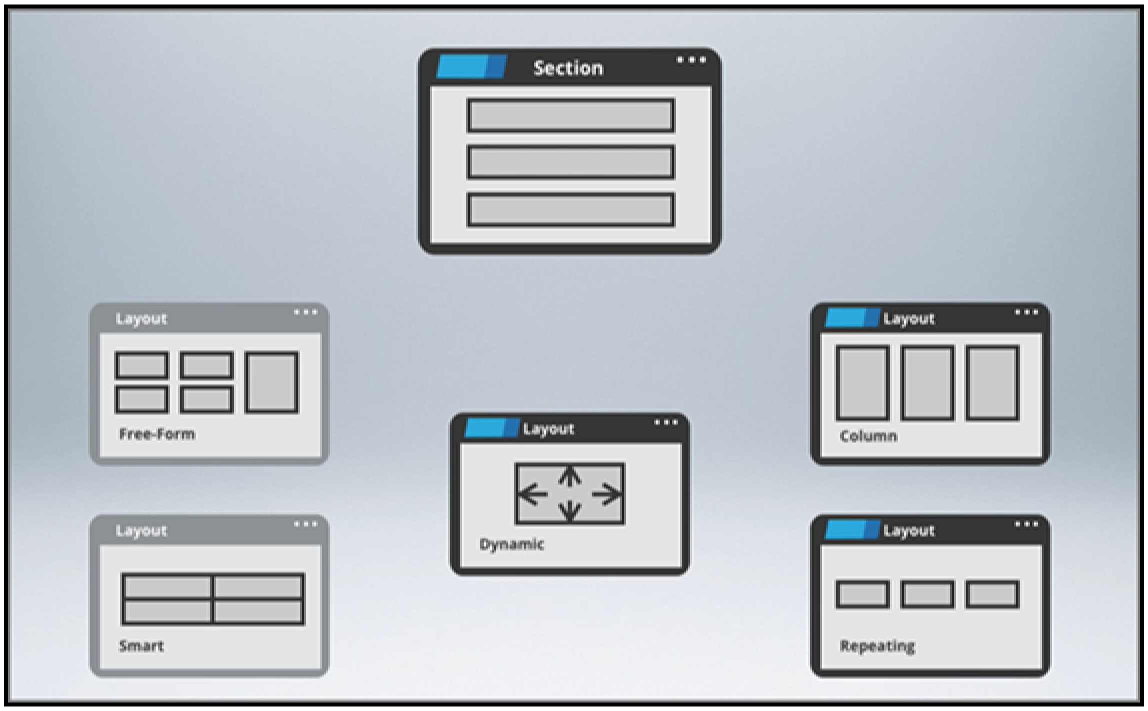


Sections and layouts

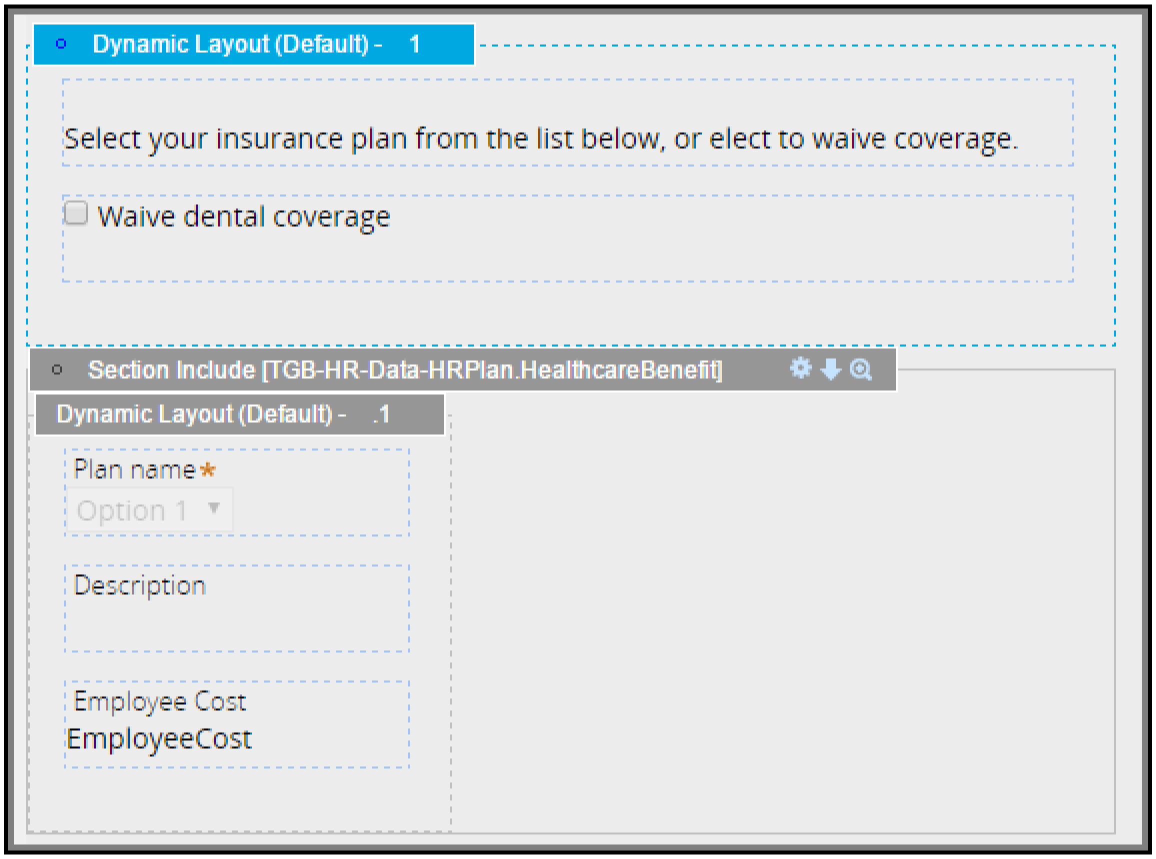
Users interact with an application and perform tasks through user forms. A user form can be a data entry form in which users provide information for filing an insurance claim, a display of the legal terms that users must accept before opening a bank account, or a list of bank account transactions during the past month.

In Pega, you build user forms with sections. Sections group information and functionality by context and purpose. Inside a section, you organize UI elements with layouts. Layouts contain rows and columns, defining a set of cells. A cell can be empty or contain any of various fields and controls.

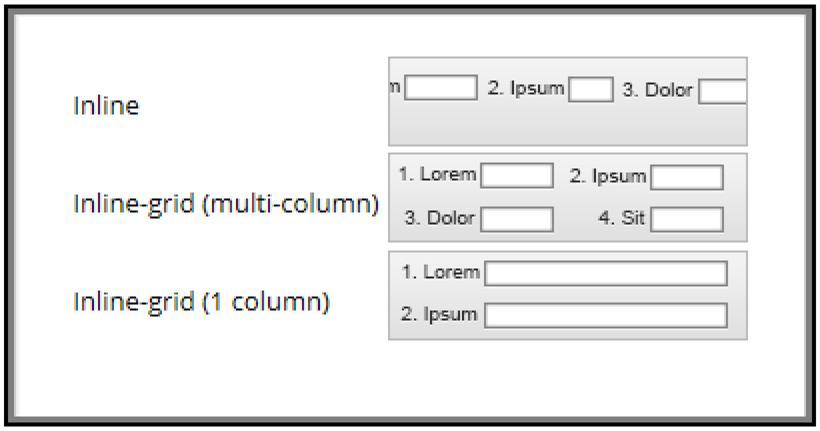
The following picture shows several types of layouts. Different layouts arrange UI elements in different fashions. A column layout arranges items in a set number of columns. A dynamic layout arranges items in a flexible form that automatically adjusts to screen size. To display a collection of data that belongs to a page list or a page group, you can use a repeating layout.



Structurally, a section consists of one or more layouts and embedded sections. The following image shows a section displaying healthcare coverage for employees. Inside the section is a dynamic layout at the top that contains a read-only text and a check box. Below the dynamic layout, a section defined in another class is included. This included section contains a dynamic layout that displays the coverage plan information.


Dynamic Layout (Default) - 1
Select your insurance plan from the list below, or elect to waive coverage.
Waive dental coverage
Section Include [TGB-HR-Data-HRPlan.HealthcareBenefit]
Dynamic Layout (Default) - .1
Plan name*
Option 1
Description
Employee Cost
EmployeeCost

The most commonly used layout in building sections is the dynamic layout because it enables a highly flexible display of UI content. In a dynamic layout, the item arrangement can be one of two types: inline or inline-grid. The inline arrangement displays items in a row like words in a sentence. The inline-grid arrangement displays items in a multi-column grid. The inline-grid with one column is equivalent to a staked format.



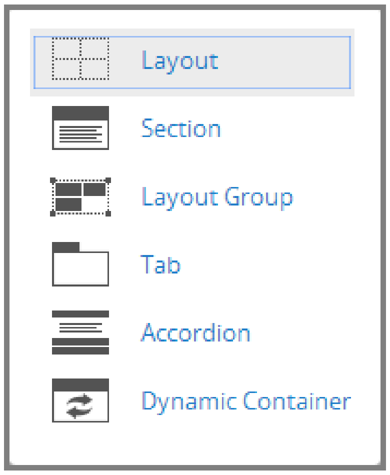
Several formats of dynamic layout styles are available, including Default, Stacked, Inline, Inline-grid double, and Inline-grid triple. You can modify and create additional formats. Changing the format automatically affects all sections using that format.

How to build a section

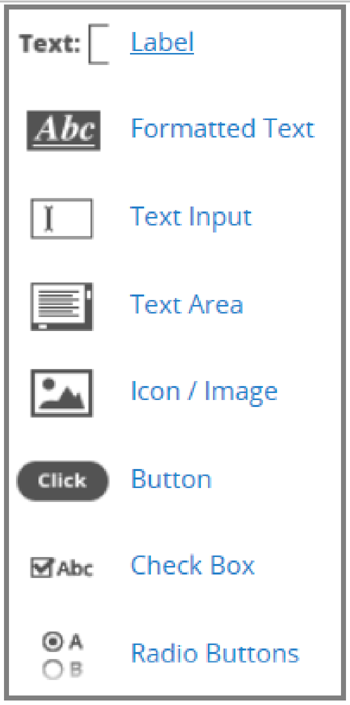
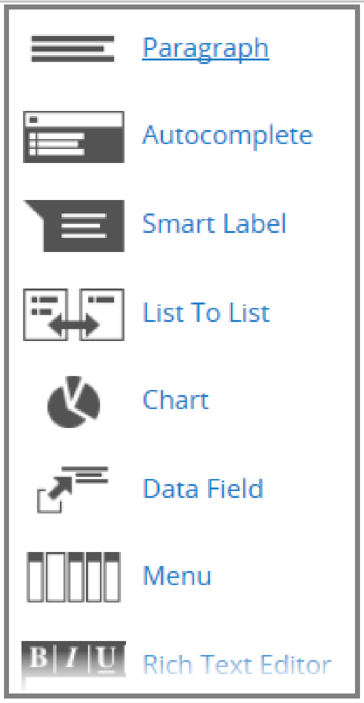
To build a section, you first select a layout type to give the section a skeletal structure. Then, you populate the cells of the layout with elements such as properties, controls, other layouts, or other sections.

In the Design tab of a Section rule, the toolbar provides various action icons to perform edit operations such as cut, copy, paste, add/delete rows or columns, and merge rows or columns. The control groups (Layout, Basic, and Advanced) contain UI elements you can drag and drop to construct the section.

The Layout group provides various structural elements for organizing UI content.



The Basic and Advanced groups list all the controls that you can use to present application data.

Basic	Advanced
	

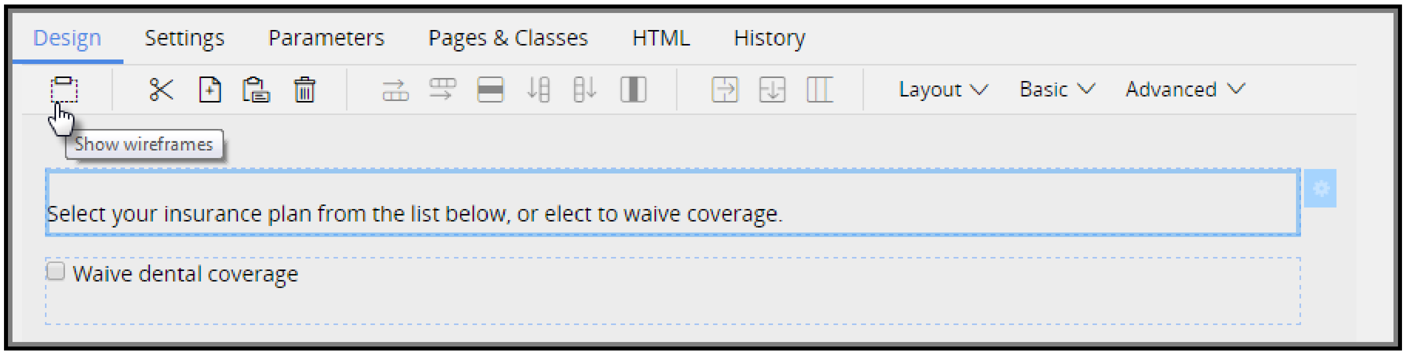
Creating a dynamic layout in a section

A dynamic layout arranges items in a flexible form that automatically adjusts to screen size. This is useful when your application my be accessed from computers, laptops, tables, or mobile devices.

Follow these steps to create a dynamic layout in a section:

1. In the **Design** tab of a section, click the **Layout** group and select **Layout**.
2. Drag and drop the layout to the desired position in the design area.
3. On the**Set layout type** dialog, select **Dynamic Layout**.
4. Click **OK**. The dynamic layout displays in the section.
5. Click the **configuration gear** icon to open the **Dynamic layout properties** panel.
6. Configure settings for the dynamic layout in the **General**, **Presentation**, and **Actions** tabs.
7. Click **Save** to save your changes.

To inspect the structure of the section, click **Show wireframes** at the left of the tool bar. The wireframes display the names of rules referenced and the relationships of elements in the section.



Select your insurance plan from the list below, or elect to waive coverage.

☐ Waive dental coverage

Creating a repeating layout in a section

When you want to display a collection of data that belongs to a page list or a page group, you use a repeating layout.

Follow these steps to create a repeating layout in a section:

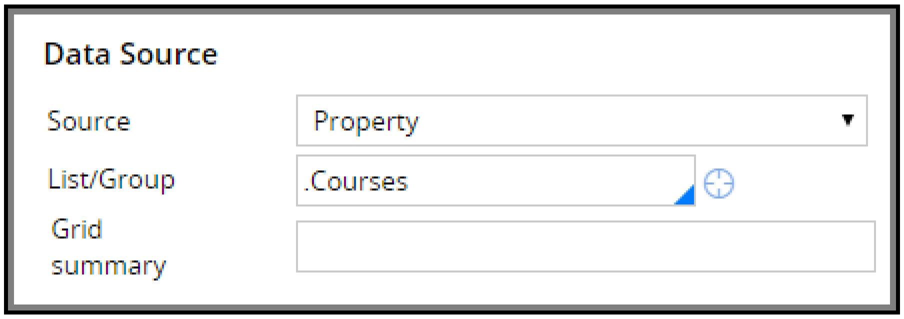
1. In the **Design** tab of a section, click the **Layout**group and select **Layout**.
2. Drag and drop the layout to the desired position in the design area.
3. On the **Set layout type** dialog, select **Repeating Layout**. Choose one of the following repeating layout types from the Repeating Layout drop-down.

Select...	**to present...**
Grid	items in a spreadsheet format.
Tree	in a tree format that enables users to expand and collapse branches to find entries of interest.
Tree Grid	a tree navigation control in combination with a grid display of items.
Dynamic	each item from the source in the dynamic layout format specified in the skin.
Column	each item from the source in vertical columns, typically with labels in the left most column.
Tabbed	each item from the source as a tab, typically with labels in the tab button.

   Click **OK**. The repeating layout of the selected type displays in the section.

4. Click the **configuration gear** icon to open the **Dynamic layout properties** panel. Configure settings for the dynamic layout in the **General**, **Presentation**, and **Actions**tabs.

   For a repeating layout, you need to specify the data source in the General tab. The data source can be a Property, list-type Data Page, or a Report Definition. The following image shows the configuration for a grid repeating layout with data provided from a property named Courses.



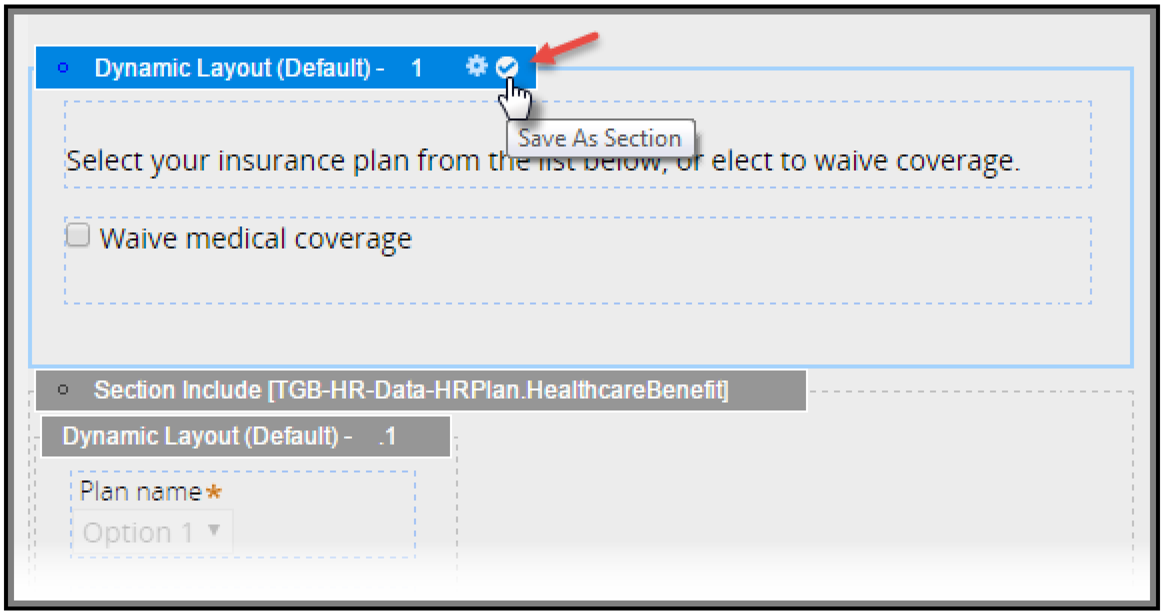
Data Source

Source	Property
List/Group	.Courses
Grid summary	

How to build sections for reuse

When designing a section, ensure that the section contains enough useful information but is small enough to be reusable. If a section is packed with so much functionality that the section can only be used for a particular use case, then consider separating it into smaller sections. Reconstruct the section to render the same amount of information by embedding the smaller sections inside.

You can convert a layout into a section so that the layout can be reused in other harnesses, sections, or flow actions. Simply click the **Save as Section** icon, displayed in the following healthcare coverage section example.


Dynamic Layout (Default) - 1
Save As Section
Select your insurance plan from the list below, or elect to waive coverage.
Waive medical coverage
Section Include [TGB-HR-Data-HRPlan.HealthcareBenefit]
Dynamic Layout (Default) - .1
Plan name*
Option 1

Whenever possible, collocate sections with data classes. A section displaying property content should be defined in the class where the properties are located. Collocating sections and data classes makes sections available for reference wherever data class instances are used.

In the above healthcare coverage section example, the section is defined on a work class for benefits enrollment. The work class has an embedded page containing information about a specific healthcare plan. The embedded page is of a data class named HRPlan. The portion of the screen displaying plan information is defined in a section that belongs to the data class HRPlan, instead of the work class, as indicated by the section include of HealthcareBenefit.

With the section HealthcareBenefit defined on the data class HRPlan, you can use the section anywhere the data class is referenced during the work processing. You can do this whether the section is an editable data entry form for employee benefit enrollment or read-only display for HR review. Any changes you make to the data class UI section automatically propagate to wherever the section is used.

Live UI

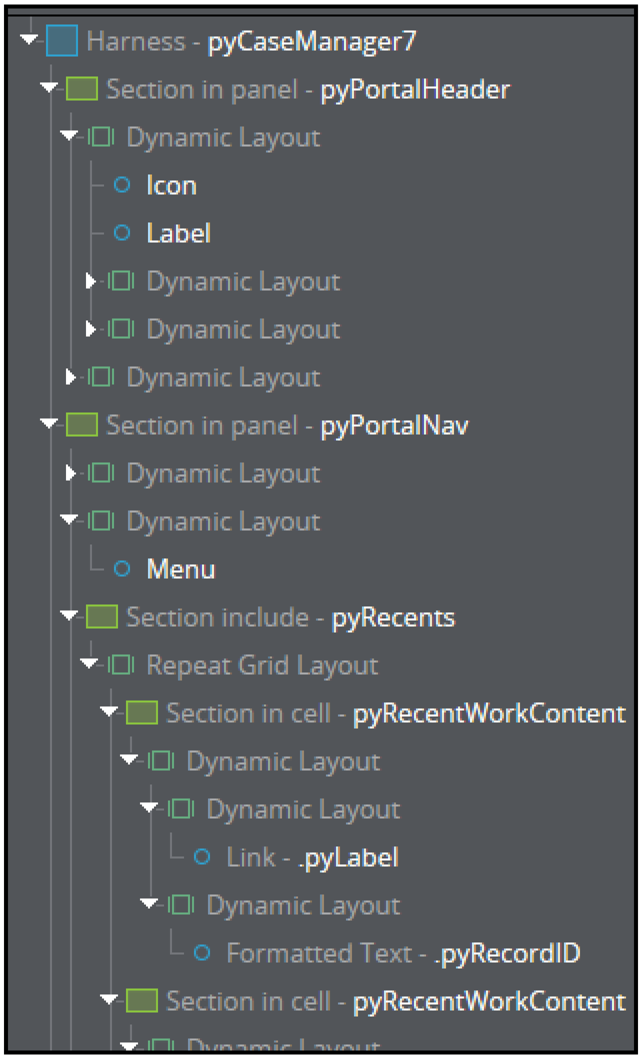
To examine and edit the rule structure of your user interface designs, use the **Live UI**tool. This tool lets you examine UI rules and elements such as sections, controls, or layouts that you used to build the form. LiveUI also allows you to locate properties that use declarative values. Rather than examine the rules individually in Designer Studio, use Live UI to view the form as users see the form. Live UI lets you review how all the rules and elements fit together.

You can use Live UI to do the following:

- Quickly identify and open a UI rule
- Change the presentation of a layout or a field
- Add or delete elements
- Move elements within the structure

How to use Live UI

The Live UI tool is located on the bottom of a user form. Click the Live UI icon to start the tool. When you start Live UI, a panel on the right side of the form displays the hierarchy of elements in a tree.



You can select an element in the tree to highlight the element on the form. You can also select an element on the form to show where the element is located in the hierarchy.

To relocate elements on the form, select it in the tree and drag and drop it onto the new location.

KNOWLEDGE CHECK

ANSWER Where and how do you start the Live UI tool?

The Live UI tool is located at the bottom of a user form. Start the tool by clicking the Live UI icon.

Using Live UI

The following examples show you how to use Live UI to:

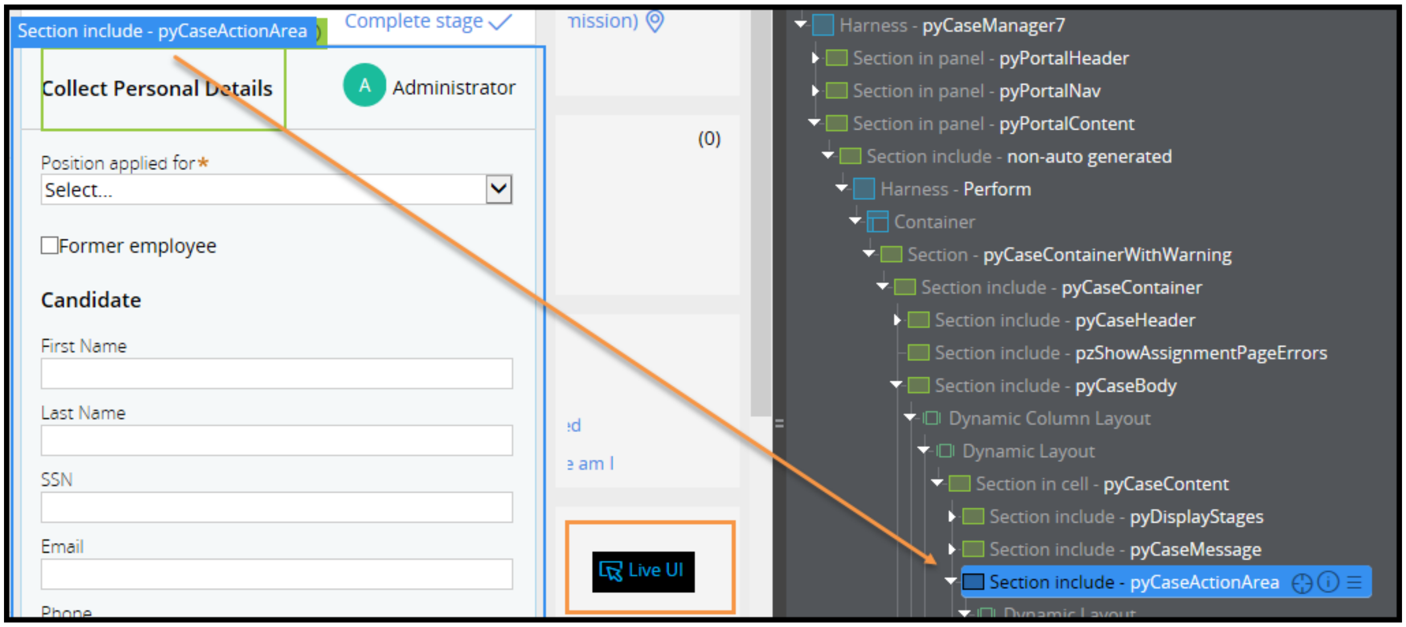
- Identify UI rules and change their position on a form.
- Add or delete UI rules.
- Locate properties that use declarative values.

Identifying a UI rule

In the following example, assume you want to identify and examine the section in the harness action area named pyCaseActionArea.

Follow these steps to find the section in the user form:

1. In the user portal, open the user form.
2. Start Live UI by clicking the **Live UI** icon.
3. On the user form, hover the mouse pointer over the section you want to review. Note that the section is also highlighted in the hierarchy tree.

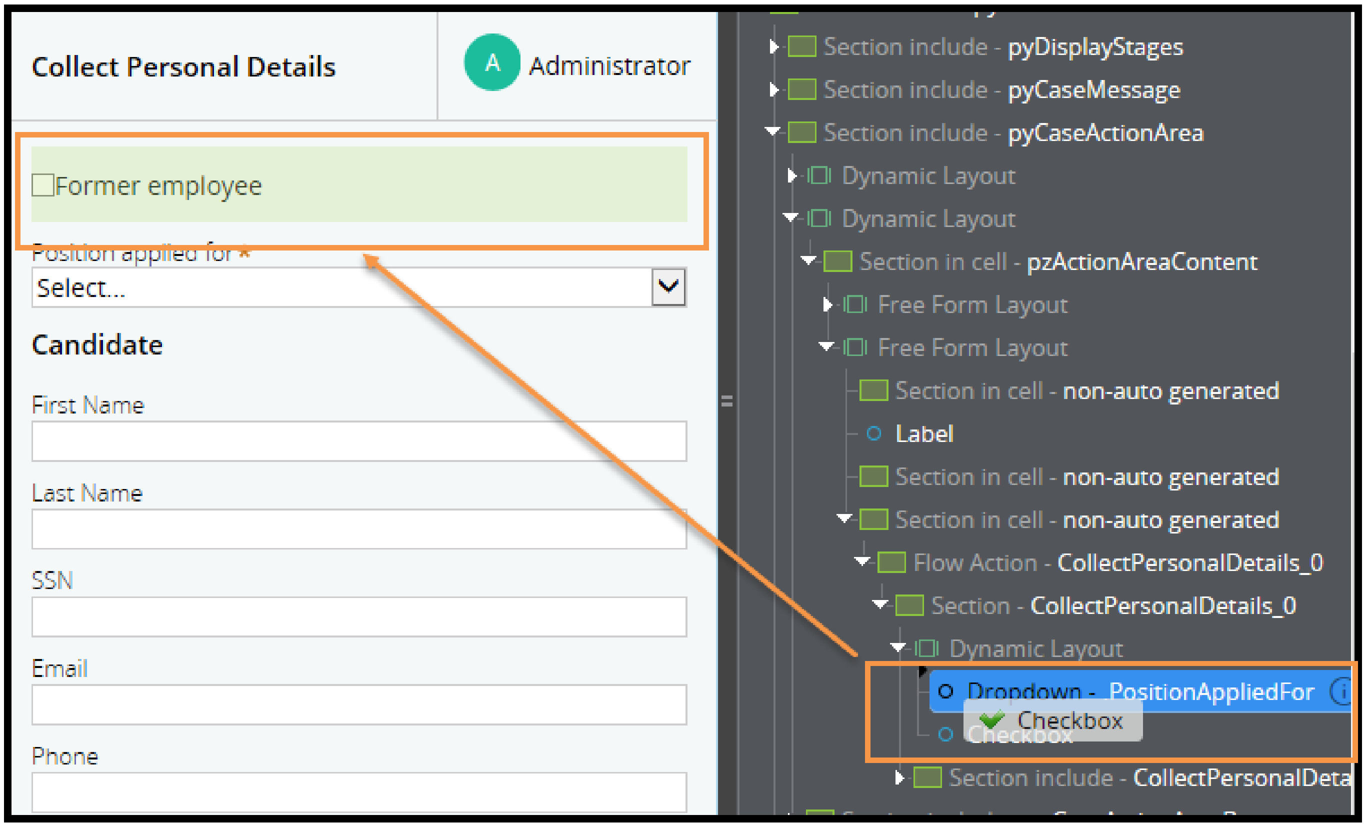

Section include - pyCaseActionArea
Complete stage
Collect Personal Details
Administrator
Position applied for*
Select...
Former employee
Candidate
First Name
Last Name
SSN
Email
Phone
(0)
Live UI
Harness - pyCaseManager7
Section in panel - pyPortalHeader
Section in panel - pyPortalNav
Section in panel - pyPortalContent
Section include - non-auto generated
Harness - Perform
Container
Section - pyCaseContainerWithWarning
Section include - pyCaseContainer
Section include - pyCaseHeader
Section include - pzShowAssignmentPageErrors
Section include - pyCaseBody
Dynamic Column Layout
Dynamic Layout
Section in cell - pyCaseContent
Section include - pyDisplayStages
Section include - pyCaseMessage
Section include - pyCaseActionArea
Dynamic Layout

Changing the position of a UI rule

In the following example, assume you want to change the position of a check box control on a form.

1. Start Live UI by clicking the Live UI icon.

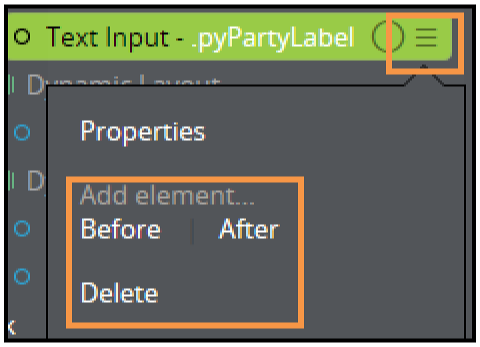
2. In the tree, click the control you want to move. Notice the control highlights on form and in the tree. In this example, you want to relocate the **Former employee** check box below the **Position applied for** drop-down control so that the check box is above the drop-down control.
3. In the tree, select the control you want to move, then drag and drop it onto the new location. In this example, you move the check box so it is located above Dropdown-PositionAppliedFor as shown below.


Collect Personal Details
Administrator
Former employee
Position applied for *
Select...
Candidate
First Name
Last Name
SSN
Email
Phone
Section include - pyDisplayStages
Section include - pyCaseMessage
Section include - pyCaseActionArea
Dynamic Layout
Dynamic Layout
Section in cell - pzActionAreaContent
Free Form Layout
Free Form Layout
Section in cell - non-auto generated
Label
Section in cell - non-auto generated
Section in cell - non-auto generated
Flow Action - CollectPersonalDetails_0
Section - CollectPersonalDetails_0
Dynamic Layout
Dropdown - PositionAppliedFor
Checkbox
Checkbox
Section include - CollectPersonalDeta

Adding or deleting UI rules

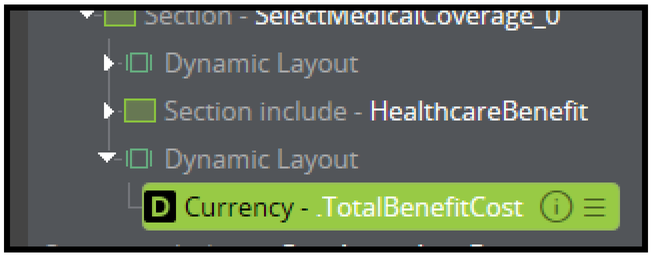
To add or delete a UI rule, do the following:

1. Select an item in the tree or on the form and click the slide-out icon to open the configuration menu.
2. In the menu do either of the following:
   a. Select **Before** or **After** to add an item.
   b. Select **Delete** to delete the item.

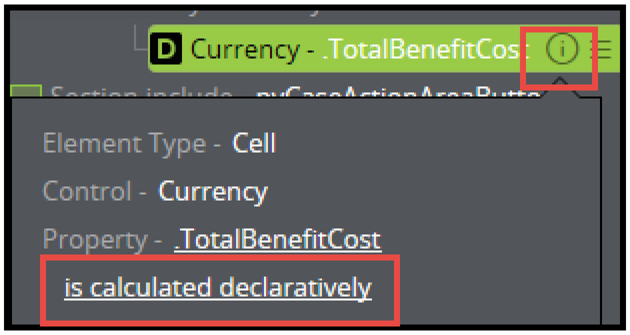


Locate declarative properties

Declarative items are listed with a "D" next to the element's name in the tree as shown in the following example.


Section - SelectMedicalCoverage_0
Dynamic Layout
Section include - HealthcareBenefit
Dynamic Layout
D Currency - .TotalBenefitCost

Clicking the **is calculated declaratively** link on the **Info** icon popup menu opens the declarative network display.


D Currency - .TotalBenefitCost
Element Type - Cell
Control - Currency
Property - .TotalBenefitCost
is calculated declaratively

For more information about using Live UI, see the PDN article Using the Live UI tool.

Guidelines for designing user forms

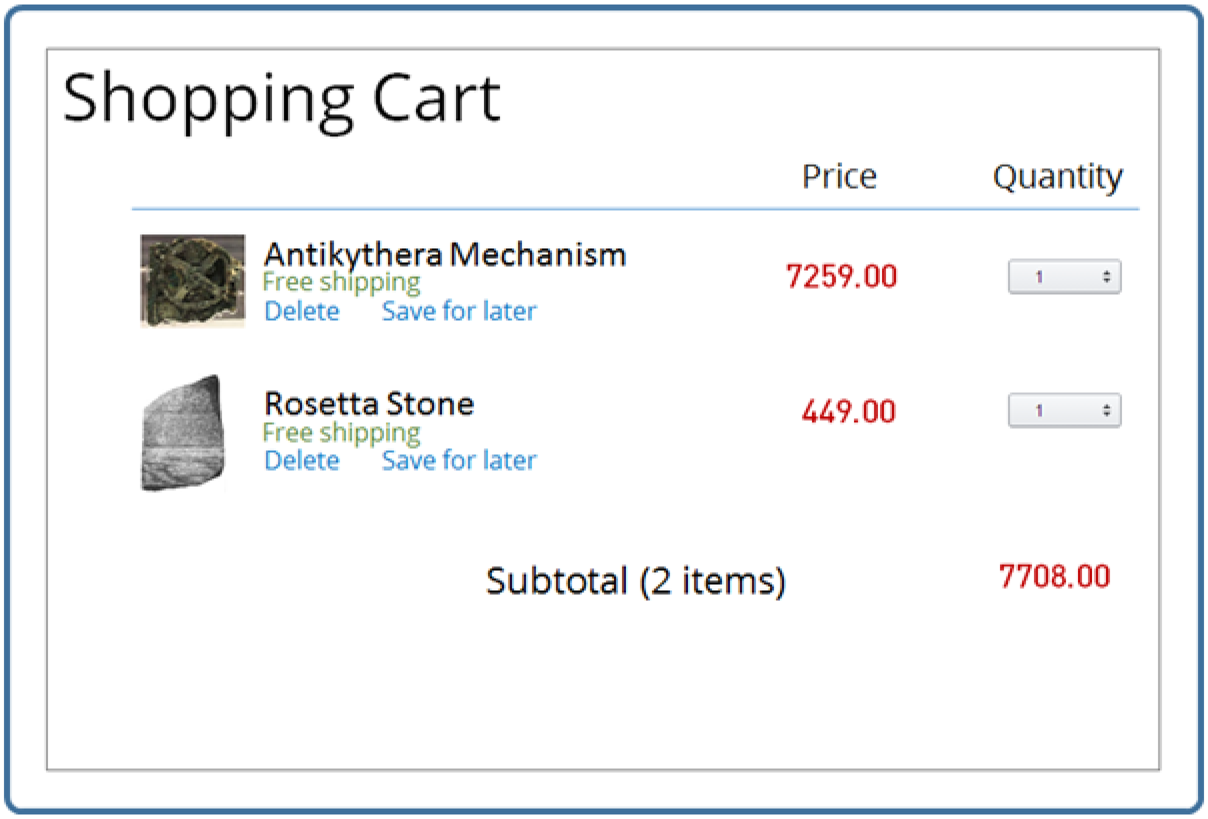
When designing and building a business application, remember that the user interface (UI) is the end users' view of the application. To be effective, the application UI must meet the needs of the end users, and be easy to use.

A well-designed application UI provides end users with a better understanding of what the application is intended to deliver. A good application UI provides the right functionality at the right time to the right people.

Consider the following guidelines when planning the application UI.

Design an intent-driven UI

This consideration is about providing end users the necessary information when they need it. An intent-driven UI is a screen where the end users have no problem understanding what they need to do.


Shopping Cart

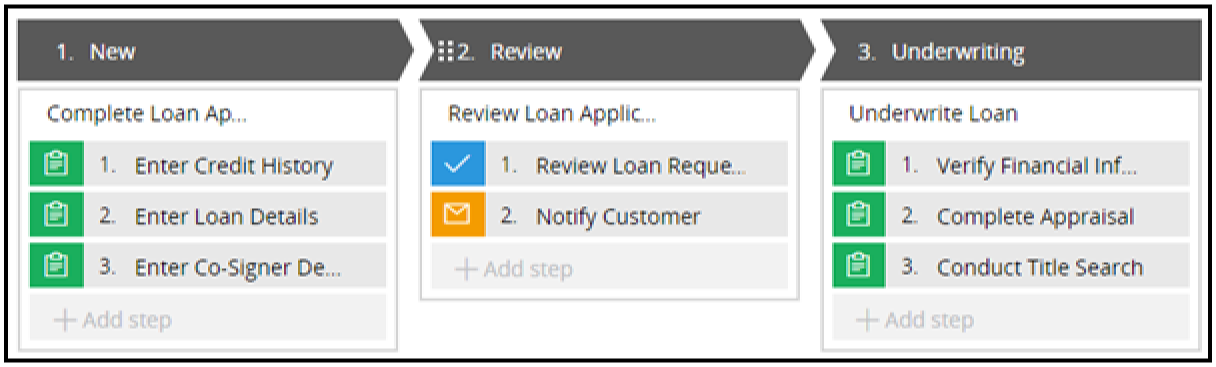
	Price	Quantity
Antikythera Mechanism Free shipping Delete Save for later	7259.00	1
Rosetta Stone Free shipping Delete Save for later	449.00	1
Subtotal (2 items)	7708.00	

For example, the form used to review a shopping cart for an online order should only display the relevant information such as the items to purchase, the quantity, and the price of each item. The form should not contain information for options such as making changes to the billing address. Changing the billing address is not relevant when reviewing a shopping cart. Making such changes could distract the user from the task at hand.

By focusing on the intent at any one step in the case life cycle, business users spend less time searching for the relevant information, thereby becoming more productive.

Design a model-driven UI

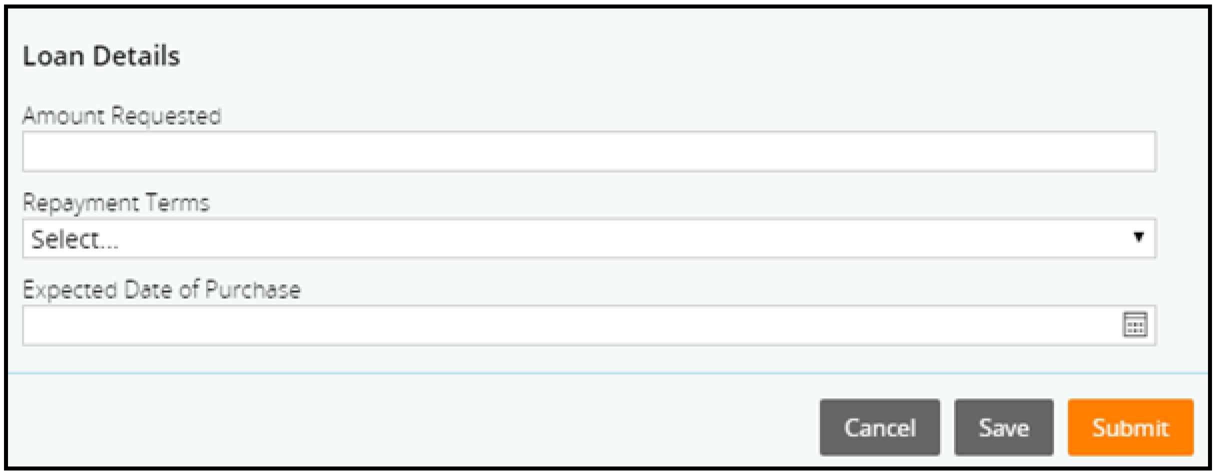
The user interface should be model-driven. This means that UI forms are tightly coupled with the business process.



The business process determines which user interface is rendered at each step in the process. A model-driven UI has several benefits, including speedier application development, a UI that is contextually sensitive to the type of business process you are mapping out, and a UI that responds rapidly to changes in business rules.

Keep the UI simple and obvious

The best interfaces are almost invisible to the end user. Avoid unnecessary elements and use meaningful labels that clearly describe the information the business user is working with.

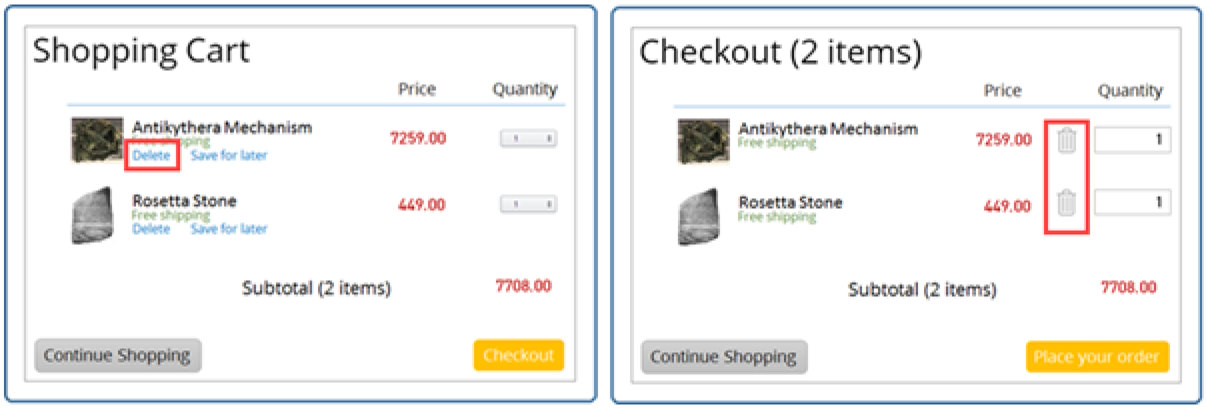


Minimize data elements on the screen to an optimal set that is critical to the intent-driven task.

Focus on what the end users are trying to accomplish. Be aware of where end users are in the process of achieving their tasks — and what the next preferred action is. End users should not have to guess how to proceed while looking at a screen.

Use common UI labels and elements

By using common labels and elements in the application UI, end users feel more comfortable and can get things done more quickly.



Create patterns in language, layout, and design to help end users complete their tasks. Once end users learn how to do something, they should be able to transfer that skill to other parts of the application.

Reusing text with paragraph rules

Introduction to Reusing Text with Paragraph Rules

Paragraphs enable you to create reusable text pieces that can be used in several different rule types. For example, you might want to add instructions for the user filling out a form. If the instructions are defined as paragraphs, those instructions can be reused in other forms or in emails sent by your application.

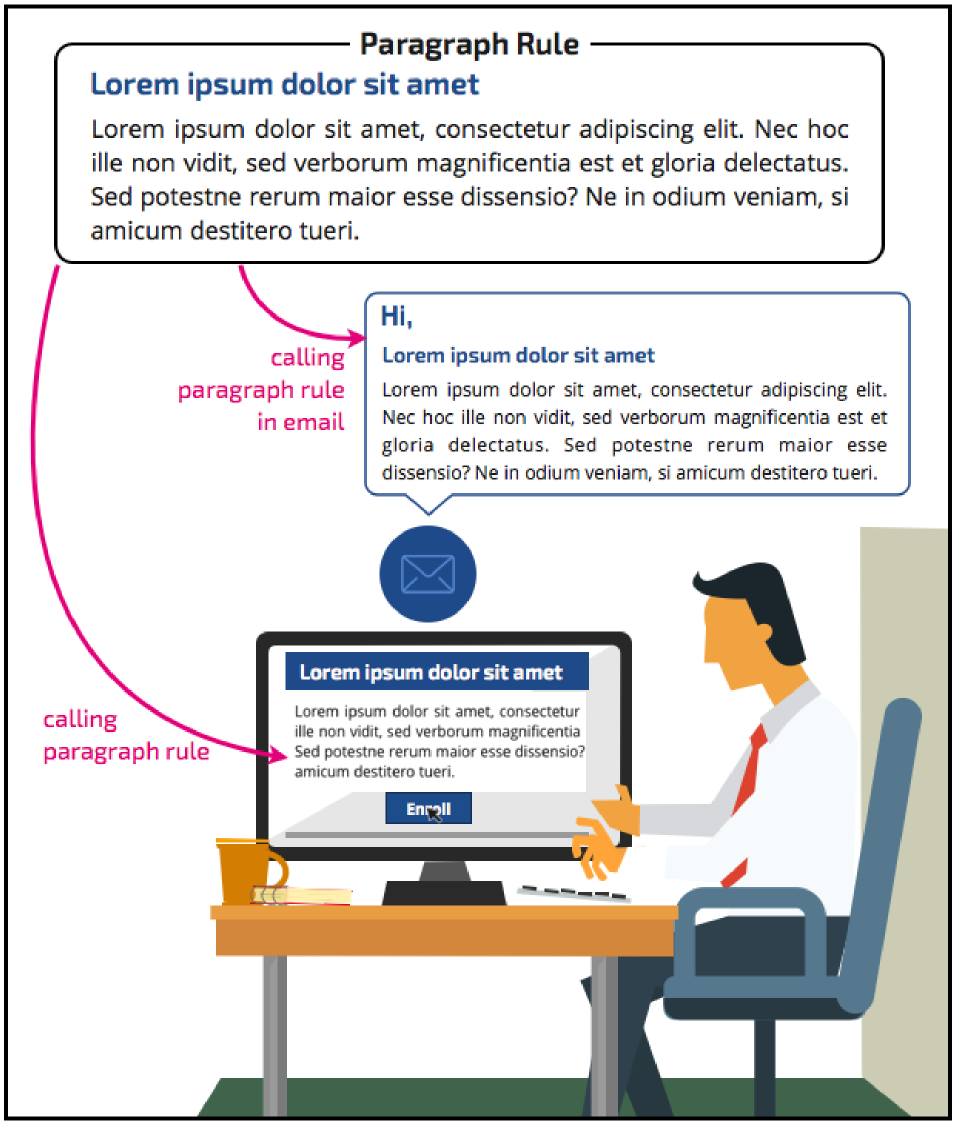
After this lesson, you should be able to:

- Identify opportunities for using paragraphs.
- Explain the purpose of paragraphs.
- Configure paragraphs.
- Reuse text with a paragraphs.

Paragraph rules

To minimize the amount of training that users need to become productive with an application, you want to add instructions to forms. You also want to notify users of pending assignments with instructions on what they are expected to do. You can write static text, such as instructions, directly into the UI and the notification.

Defining static text in paragraph rules allows you to reuse the text across your application. For example, you can create a paragraph with an instruction and add that paragraph to the form. The same paragraph can be reused in correspondence.


Paragraph Rule
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nec hoc ille non vidit, sed verborum magnificentia est et gloria delectatus. Sed potestne rerum maior esse dissensio? Ne in odium veniam, si amicum destitero tueri.
calling paragraph rule in email
Hi,
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nec hoc ille non vidit, sed verborum magnificentia est et gloria delectatus. Sed potestne rerum maior esse dissensio? Ne in odium veniam, si amicum destitero tueri.
calling paragraph rule
Lorem ipsum dolor sit amet
Lorem ipsum dolor sit amet, consectetur ille non vidit, sed verborum magnificentia Sed potestne rerum maior esse dissensio? amicum destitero tueri.
Enroll

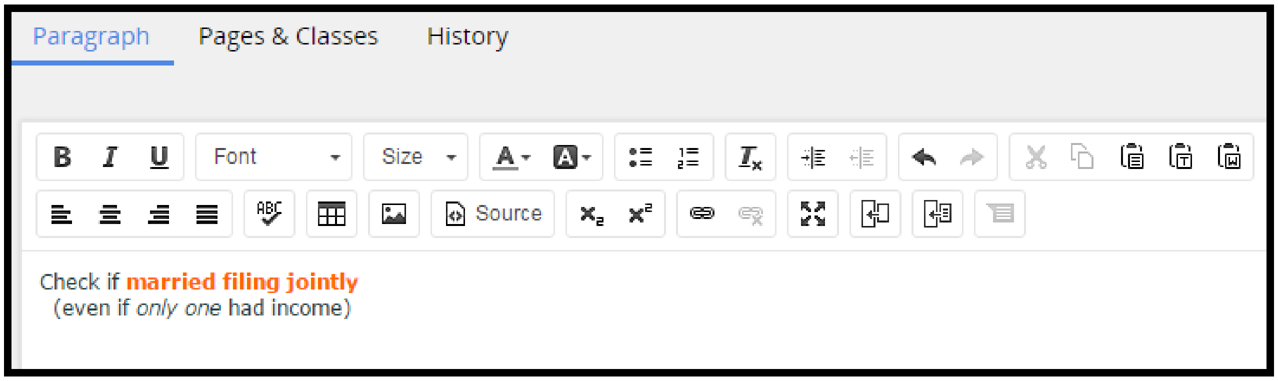
In addition to reuse, paragraph rules provide the following benefits:

- Paragraphs can convey entire sentences or paragraphs.
- Paragraphs support rich texts including images and links.

- Paragraphs can include property references. For example, you can include a customer's name in a paragraph.

The paragraph rule form

Paragraphs are available under the user interface category. Use the paragraph tab to define the content and appearance of your paragraph. The source button allows you to switch to the HTML source code. Use the pages and classes tab to identify any clipboard pages from which you want to include property references.



Check if **married filing jointly**
(even if *only one* had income)

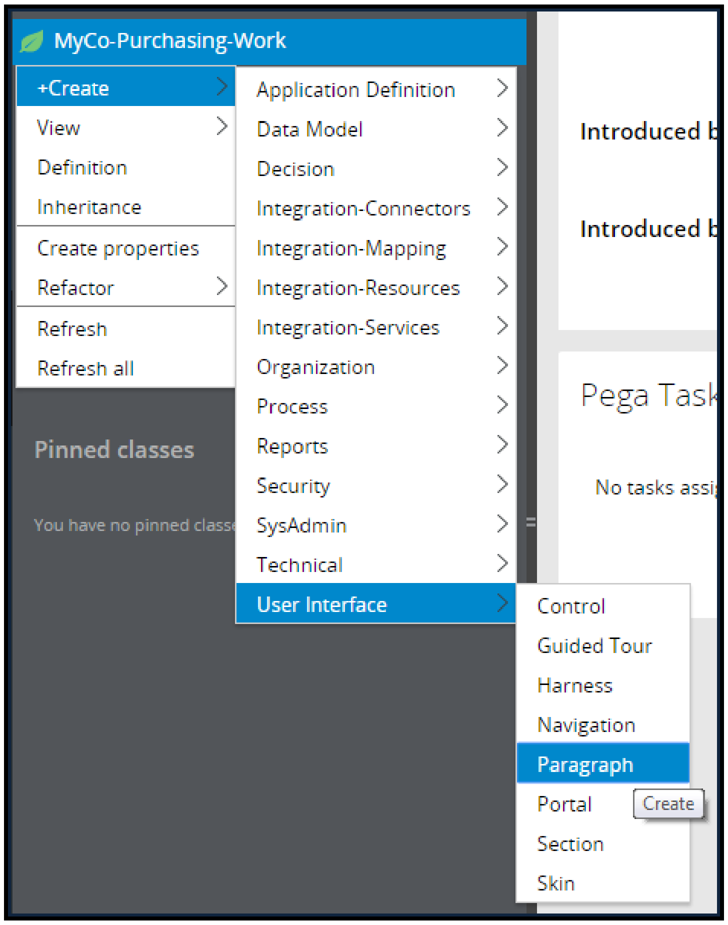
Reusing text with paragraph rules

This procedure shows you how to create, reference, and use properties with paragraph rules. In the purchase application, you want to help users place the order. You create a paragraph containing the phone number and email of the hotline. The paragraph can then be reused where needed.

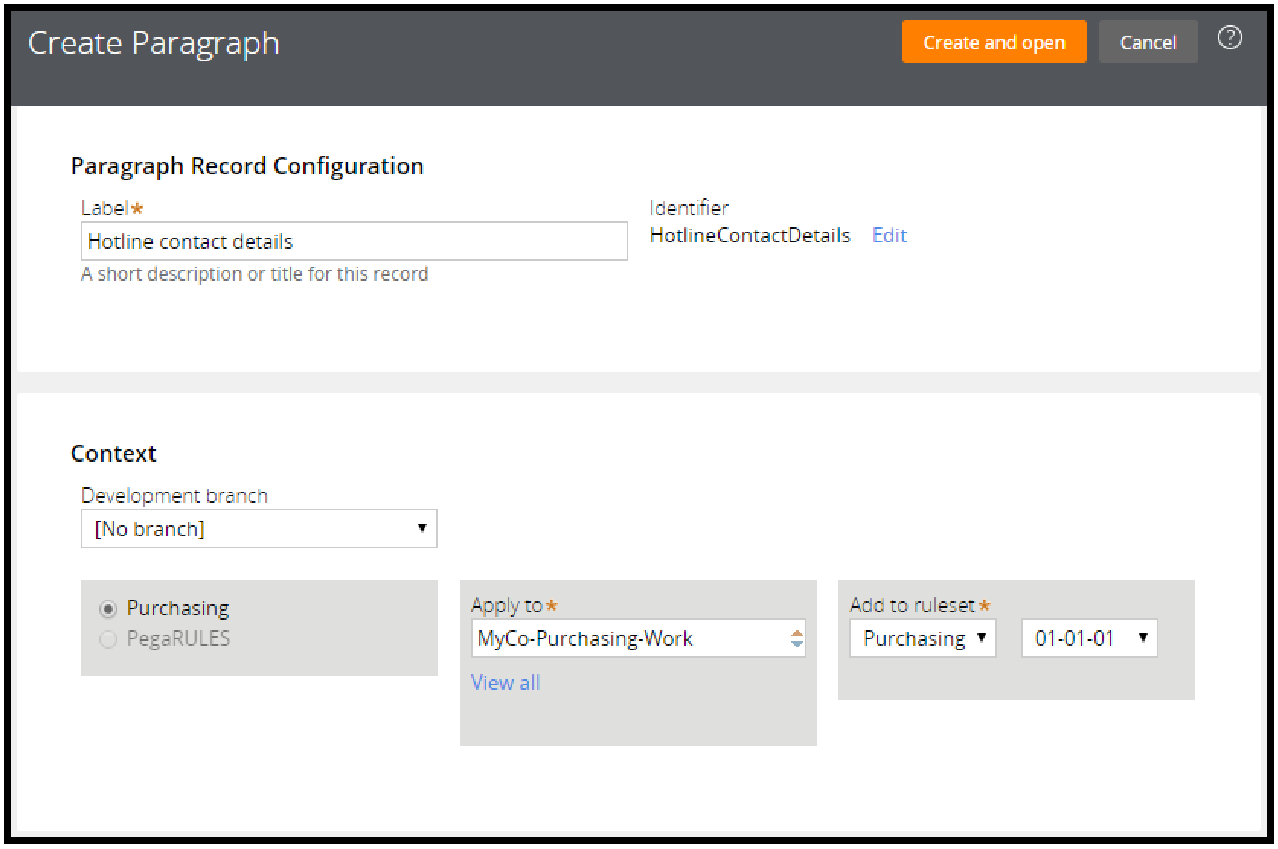
Create a paragraph

Follow these steps to create a paragraph:

1. In the User Interface category, select **Paragraph**.



2. In the **Label** field, enter a name for your paragraph. Click **Create and open**.



Paragraph Record Configuration

Label* Hotline contact details

A short description or title for this record

Identifier HotlineContactDetails Edit

Context

Development branch [No branch]

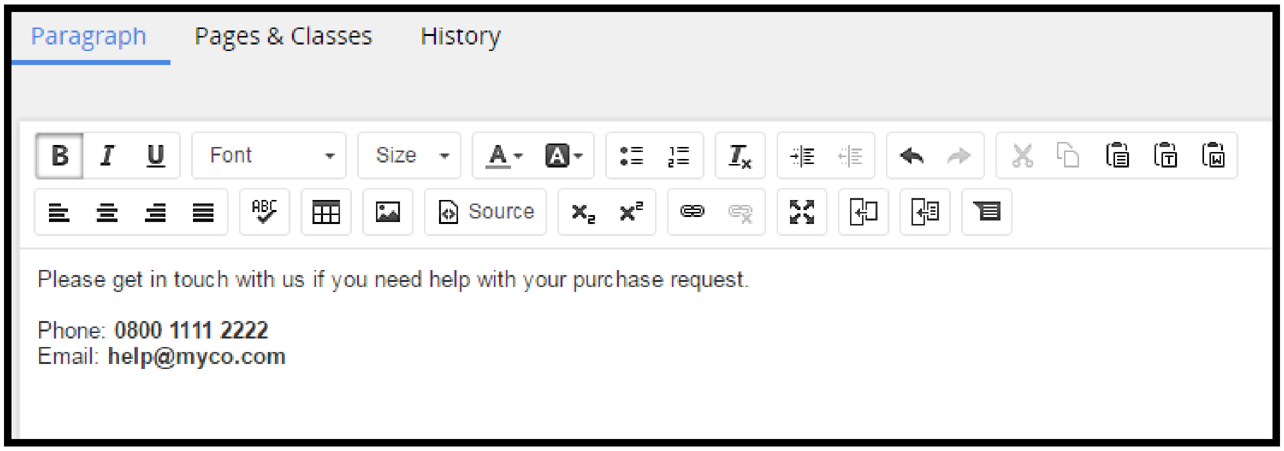
Purchasing ● PegaRULES

Apply to* MyCo-Purchasing-Work

View all

Add to ruleset* Purchasing 01-01-01

3. Enter your text on the Paragraph tab.

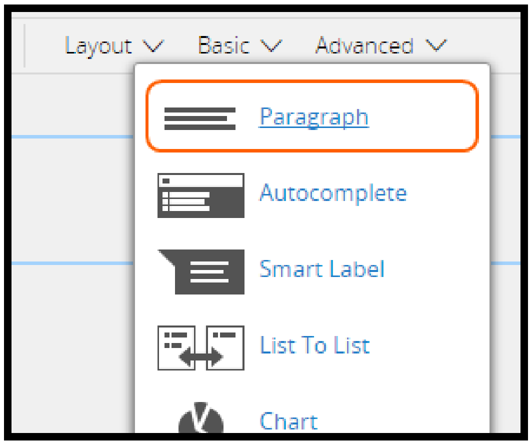


Please get in touch with us if you need help with your purchase request.

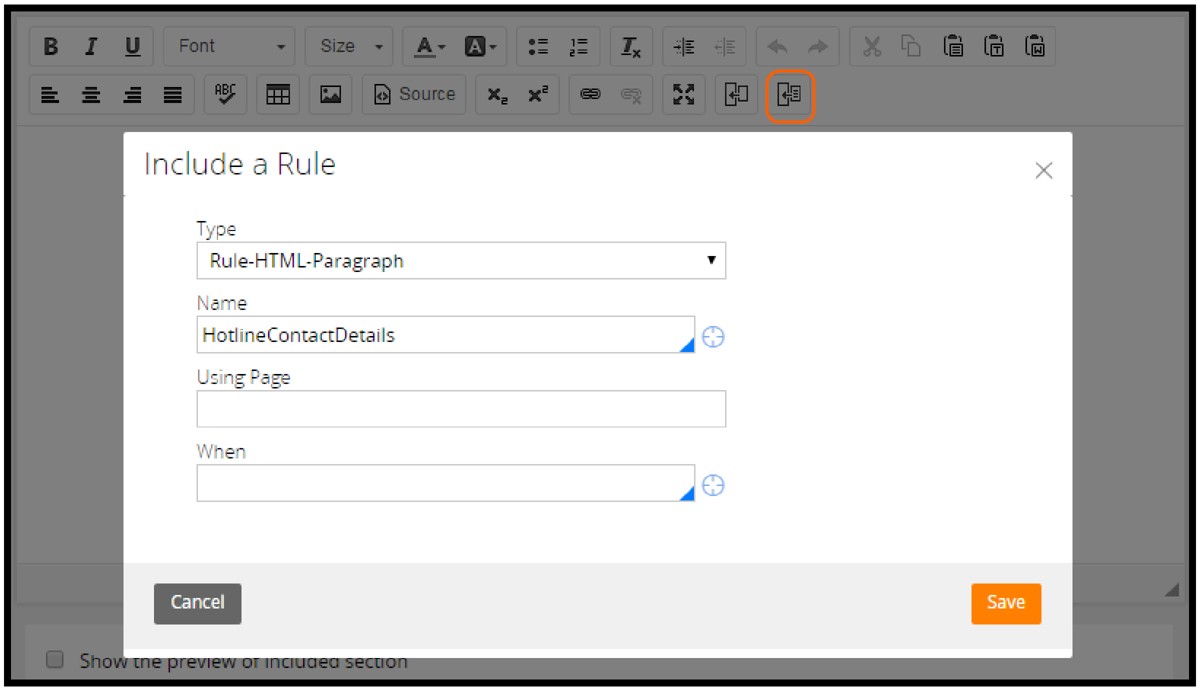
Phone: **0800 1111 2222**
Email: **help@myco.com**

Referencing a paragraph

Paragraphs can be inserted in sections, harnesses, and within other paragraph rules. This allows you to create reusable building blocks of text. In the advanced menu, select **Paragraph** to include a paragraph in a section or harness.



When you include a paragraph within another paragraph or in a correspondence, insert a rule and select **Rule-HTML-Paragraph**.



Include a Rule

Type
Rule-HTML-Paragraph

Name
HotlineContactDetails

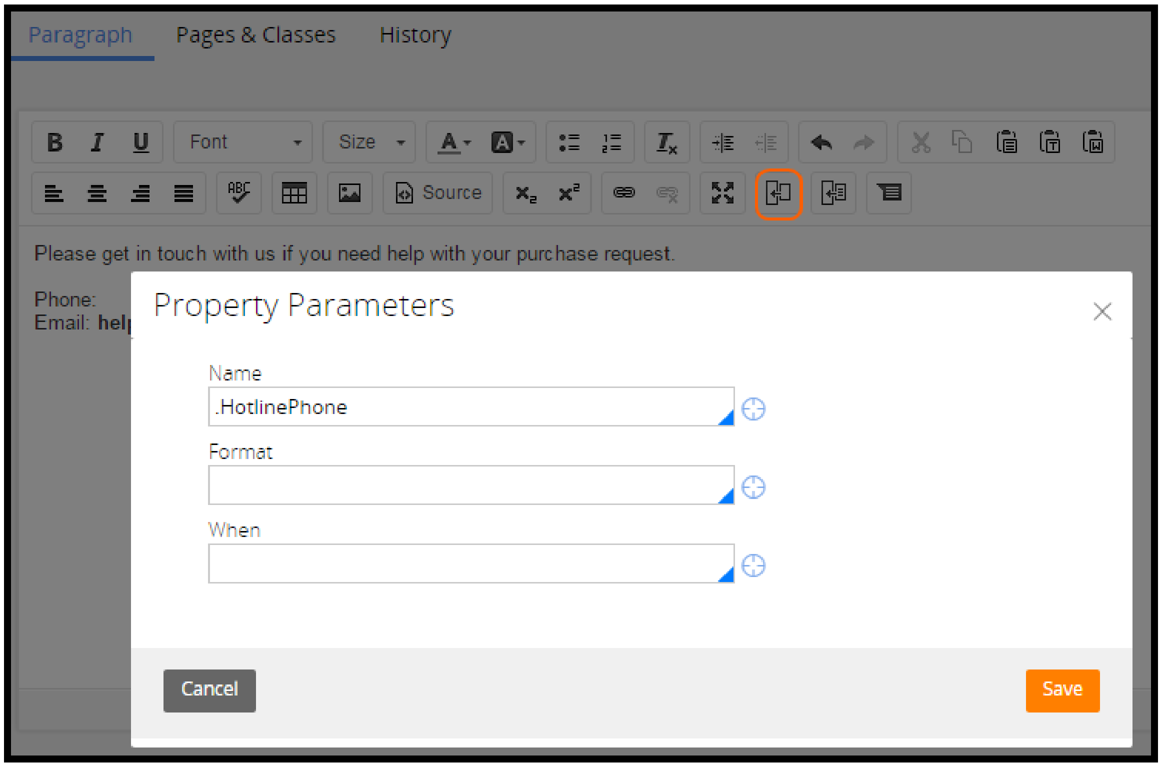
Using Page

When

Inserting a property

Instead of hard-coding the phone and email numbers, you can reference properties holding the values. By referencing properties, you can define the values in a central point.

Click **Insert property** to specify the parameters for the property you want to insert.


Paragraph Pages & Classes History

Please get in touch with us if you need help with your purchase request.

Phone:
Email: hel

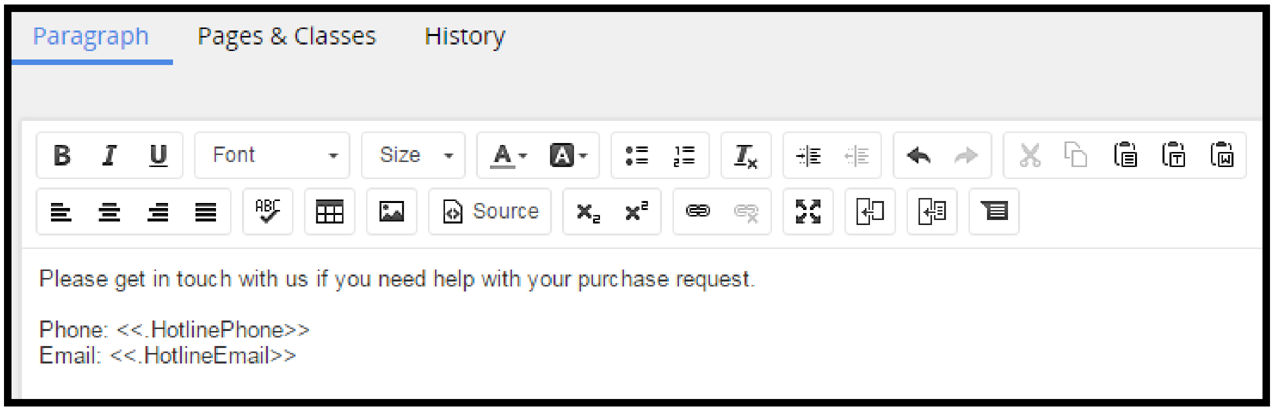
Property Parameters

Name: .HotlinePhone

Format:

When:

The values of the properties specified are displayed in the paragraph.


Paragraph Pages & Classes History

Please get in touch with us if you need help with your purchase request.

Phone: <<.HotlinePhone>>
Email: <<.HotlineEmail>>

Configuring responsive UI behavior

Introduction to Configuring Responsive UI Behavior

Ever-increasing popularity of mobile devices puts high demands on user interface (UI) design. Applications can no longer restrict themselves to desktop computers. Applications must run on tablets and smart phones as well.

A responsive UI adapts an application to diverse screen resolutions and sizes. Responsive UI allows developers to create an optimal user experience regardless of the device.

After this lesson, you should be able to:

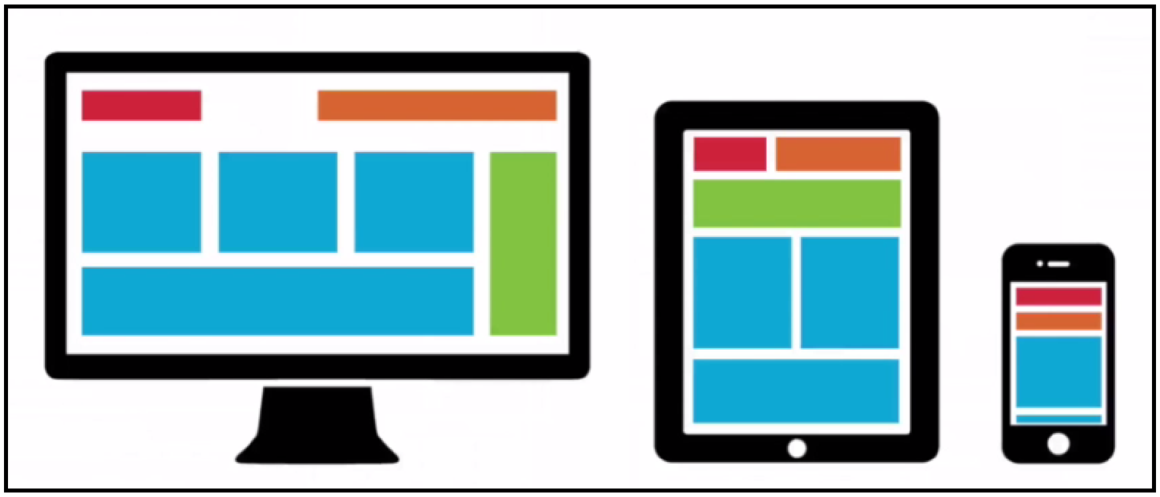
- Explain the benefits of a responsive UI.
- Explain the role of skin in UI design.
- Explain how a responsive UI adapts to different display configurations.
- Configure responsive behavior for layouts.

Responsive user interface

People expect to be able to access applications at any time, not only in the office. Some days a business person might sit at a desk and enter her expense report on her desktop. Other days, she may use her smart phone to enter an expense report while waiting at the airport.

You need to account for variations in how users will access an application when designing applications. Some application development environments require you to create an application for every device so that the UI renders correctly.

With Pega, you create a single application with a responsive user interface (UI) that adapts to multiple devices. A **Responsive UI** enables a layout to automatically adjust to rendering devices. Elements can move around, resize, or completely disappear depending on the resolution and size of a screen.



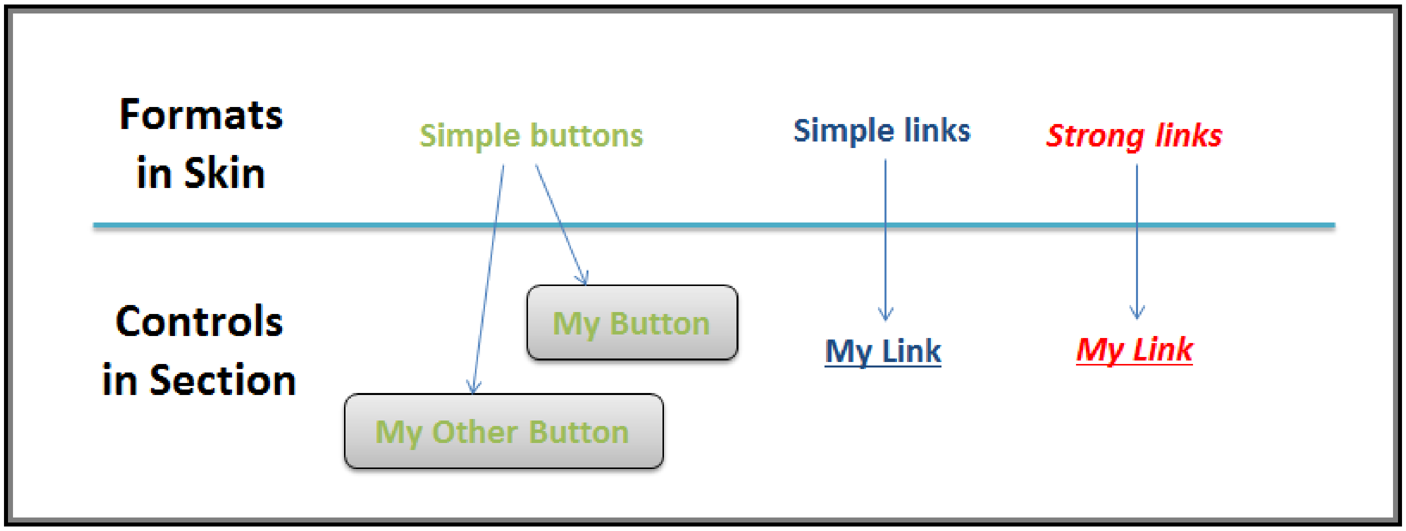
There are many benefits of a Responsive UI. You should design applications with a Responsive UI for the following reasons:

- The layout adapts according to the screen size, and the majority of the application is the same across all devices. Consistent UI behavior across devices results in a consistent user experience, easier adoption, faster learning, and fewer errors.
- A Responsive UI leads to lower maintenance costs and easier management. Instead of creating different layouts for smart phones, tablets, and wide-screen desktops, developers configure a single form to automatically adjust to different screen sizes.
- With a single code set, a Responsive UI provides an optimal user experience on different devices. While one user accesses an application on a desktop computer in the office, another user can use a mobile device during a business trip. This increased accessibility enables an application to reach a wider user base.

Presentation layer and UI skins

In Pega UI, a separation between content and presentation exists. The content layer contains data and structural elements such as sections, layouts, and individual controls. The presentation layer consists of Skin rules. A **Skin**rule specifies the visual styling as well as the responsive behavior of the UI.

In a Skin rule, you define style formats that can be applied to UI elements in a section. A style format specifies a group of styling attributes such as typography, borders, backgrounds, placement, and alignment.

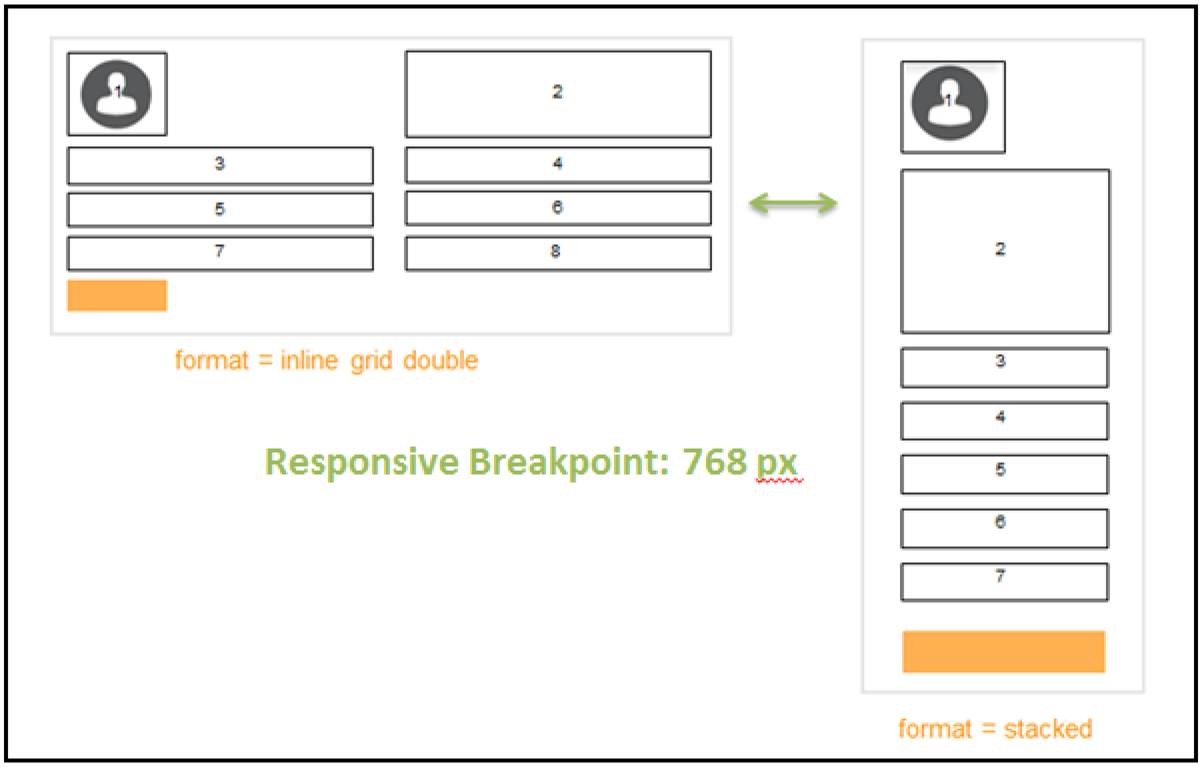


You can define style formats for UI controls like buttons and links. You can also define style formats for structural elements like layouts. The layout formats contain configurations for responsive behavior.

How to trigger responsive behavior with responsive breakpoints

You build Pega UI with sections. In each section, you arrange the UI elements by placing them inside layouts. Many layouts in Pega 7 -- such as screen layouts, dynamic layouts, and column layouts -- support configuration of responsive behavior.

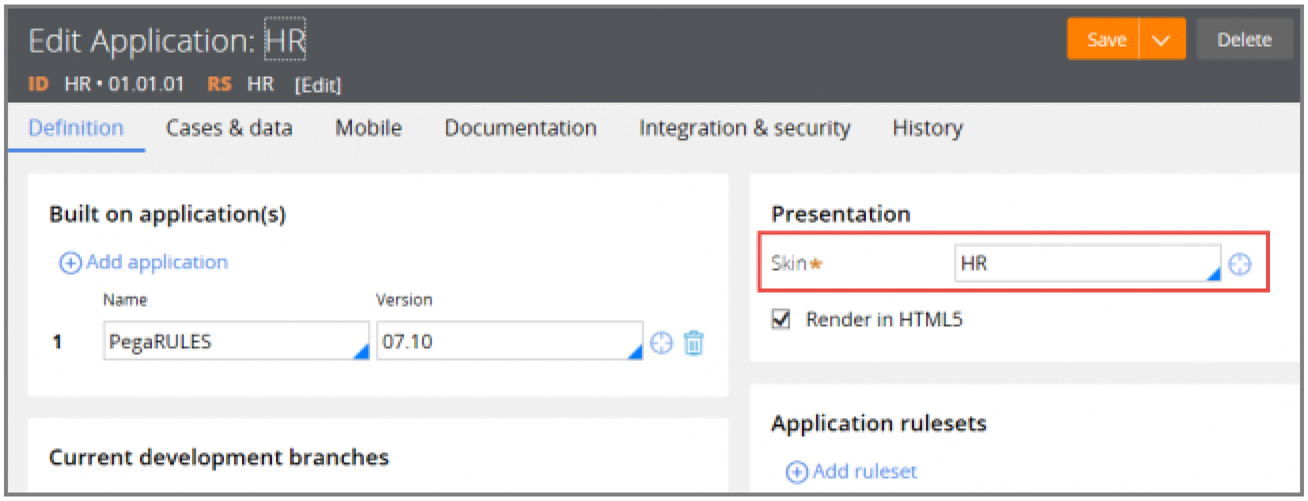
For example, you can configure a dynamic layout to arrange UI elements in different formats at different screen sizes. With dynamic layout configured, the layout arranges content into a double column on a screen wider than 768 pixels. On a screen narrower than 768 pixels wide the layout arranges its content in a vertical stack. The screen size thresholds that trigger different UI behaviors are called **responsive breakpoints**. You configure these breakpoints in the layout formats defined in the skin.



Responsive behavior has different meanings in the context of different layouts. When screen size narrows down, a dynamic layout may change from inline-grid-triple to inline-grid-double, and from inline-grid-double to stacked. Meanwhile, a repeating grid layout may drop certain columns from the grid to avoid horizontal scrolling. Refer to the product document when configuring responsive breakpoints for different layout types.

How to style applications with UI skins

A UI skin defines the presentation layer of an application. You can associate a Skin rule with an application or with a portal. The following image shows the skin reference in an Application rule.


Edit Application: HR

ID HR • 01.01.01 RS HR [Edit]

Built on application(s)

	Name	Version
1	PegaRULES	07.10

Presentation

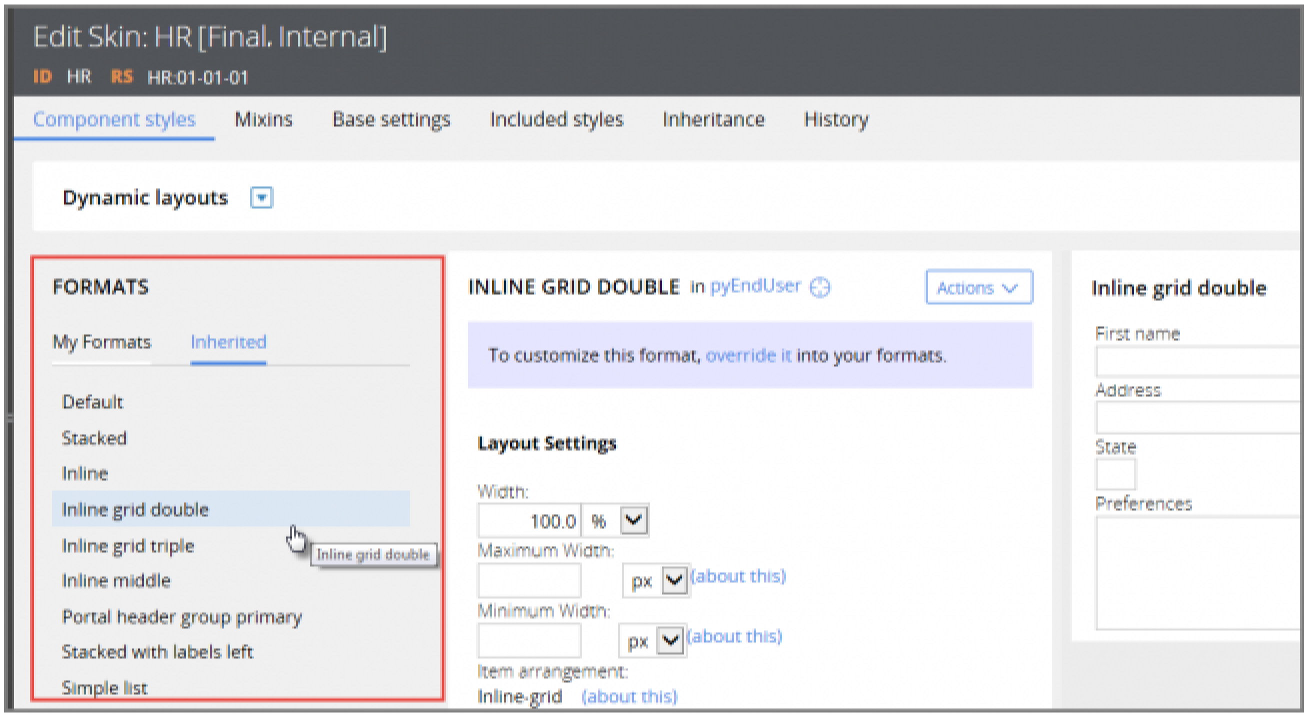
Skin* HR

☑ Render in HTML5

Current development branches

Application rulesets

The presentation configuration is defined in a Skin rule, in the form of style formats for UI components. In this example, the skin specifies a list of dynamic layout style formats such as Default, Stacked, and Inline.


Edit Skin: HR [Final, Internal]

ID HR RS HR:01-01-01

INLINE GRID DOUBLE in pyEndUser

To customize this format, override it into your formats.

Layout Settings

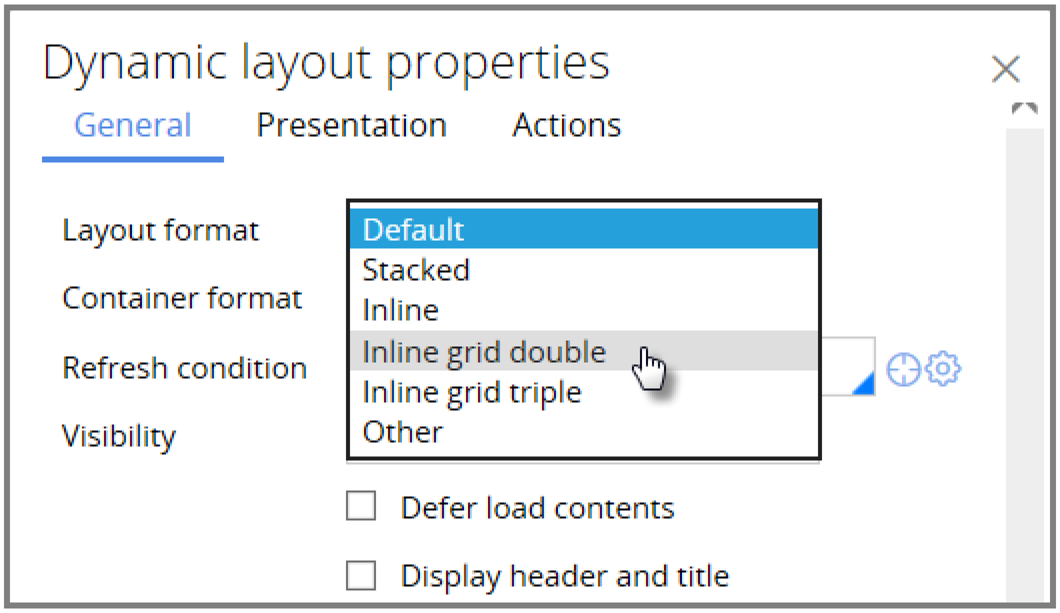
Width: 100.0 %

Maximum Width: px (about this)

Minimum Width: px (about this)

Item arrangement: Inline-grid (about this)

For a style format to take effect, you need to apply the format to a UI element. In the above example, one of the style formats defined in the skin is named Inline grid double. To apply this format to a dynamic layout, you reference it from the layout configuration in a section.


Dynamic layout properties

General | Presentation | Actions

Layout format: Default

Container format

Refresh condition

Visibility

☐ Defer load contents

☐ Display header and title

Skin inheritance allows a dependent skin to inherit formats from a parent skin. When a format on the parent skin is modified, the dependent skin automatically inherits those changes unless the format is overridden in the dependent skin.

Configuring responsive breakpoints on a dynamic layout format

To configure responsive behavior for a dynamic layout, you need to first add and configure responsive breakpoints on a dynamic layout format. Then, you apply this format to the dynamic layout in a section.

Follow these steps to add and configure responsive breakpoints on a dynamic layout format:

1. To open the skin of the application, select **Open Application Skin** from the Application menu in the Designer Studio.

   If the style format you want to configure is defined in the parent of the application skin, then open the parent skin from the **Inheritance**tab.

2. On the **Component Styles** tab, select **Dynamic layouts** from the component drop-down.
3. Select the style format you want to configure in the **My Formats** column.
4. In the **Responsive breakpoints** section, check the **Enable support for responsive breakpoints**.
5. Configure **Breakpoint1**. Select the format the dynamic layout should use when rendering at the dimensions specified for this breakpoint. Specify the screen dimension for this breakpoint:

max-width	The maximum width at which the dynamic layout will display in the format you specified for this breakpoint.
min-width	The minimum width at which the dynamic layout will display in the format you specified for this breakpoint. Leave min-width empty when a range is not desired.

6. Click **Add breakpoint** to add another responsive breakpoint.
7. Save the change.

Applying the dynamic layout format

To apply the style format to a dynamic layout in a section:

1. Open the configuration panel for a dynamic layout in a section rule.
2. From the **Layout format** drop-down on the **General** tab, select the dynamic layout format that you just configured.
3. Save the change.

Designing a dynamic UI

Introduction to Designing a Dynamic UI

A simple and focused user interface (UI) is immediately intuitive — users know exactly what to do, so they do not need to guess. Building a dynamic UI is a key component of that simplicity.

A dynamic UI predicts the intention of the user and adjusts the application display to the user context. The purpose of a dynamic UI is to provide the user with the correct functionality at the correct time. A dynamic UI efficiently guides the user toward task completion.

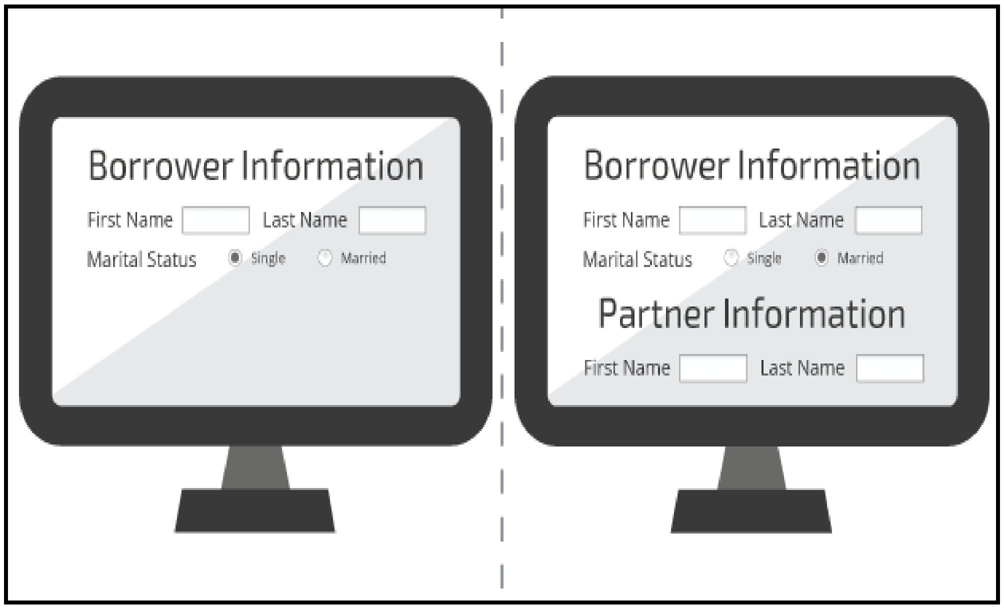
After this lesson, you should be able to:

- Describe how a dynamic UI works.
- Configure Visible When conditions to hide/show UI elements.
- Configure action sets for UI interaction.

Dynamic user interface behavior

In a dynamic user interface (UI), the UI content changes based on a user's interaction with the content. These changes reduce the UI to the fields essential for the user to maximize efficiency.

While enrolling for a new mobile phone plan, customers may be asked if they are married and if they have any children. If the answer is yes, the system knows to ask if they are interested in enrolling for a family plan. The system can then collect personal information for each of their dependents before moving on to the next question. For single customers with no children, the system skips the family plan section of the sign-up form and displays the next step, saving customers time and effort.



Using a dynamic UI has many benefits, including the following:

- Real-time response to end-user behavior
- Robust functionality available for most user interactions
- Reduced visual clutter on the screen
- Fewer full page refreshes, resulting in improved UI responsiveness

These benefits lead to a more compelling, modern user experience.

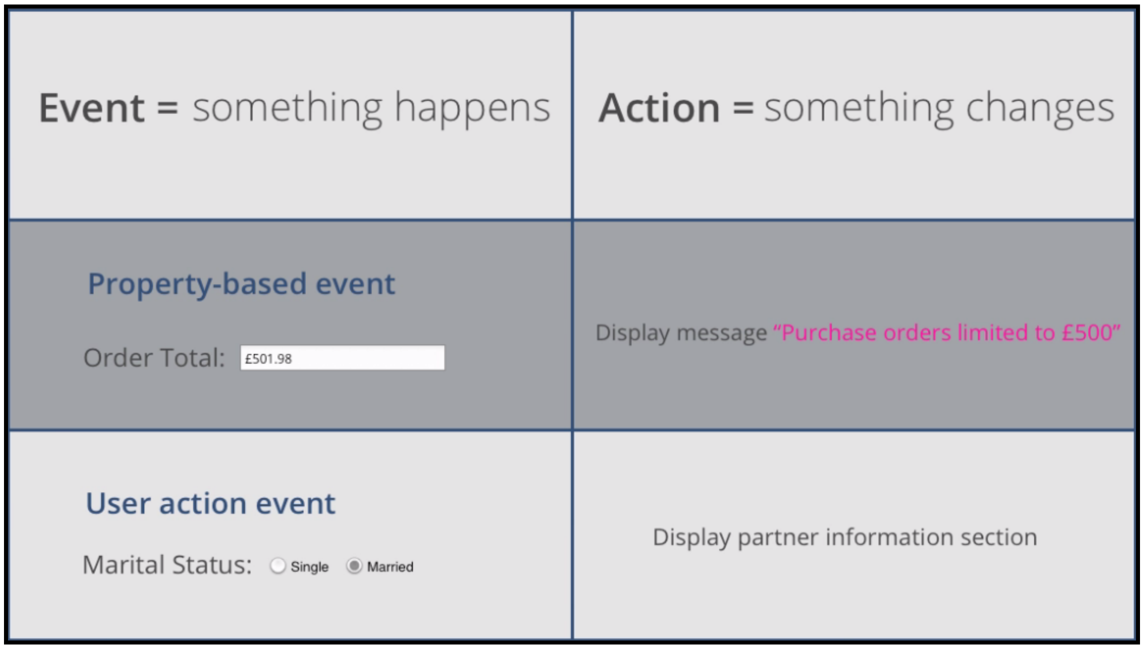
Event-action model

When designing a dynamic UI, you are using an event-action model in the browser-based application. Think of event and action as a cause-and-effect pair. An event is performed by a user that triggers changes on the UI. These changes are the action.

During online shopping checkout, users enter the shipping and billing addresses in sequence. After providing the shipping address, users click a check box labeled **use same address for billing**. The section for the billing address then disappears. In this case, clicking the check box is the event, and hiding the billing address section is the resulting action.

Two types of events exist — property-based events and user events.

- Property-based events occur either when a data value changes or when a value meets a specific criteria.
- A user event occurs when an end user takes some action on the page such as selecting an option or clicking on a link.



Event = something happens	**Action =** something changes
Property-based event Order Total: £501.98	Display message "Purchase orders limited to £500"
User action event Marital Status: ○ Single ● Married	Display partner information section

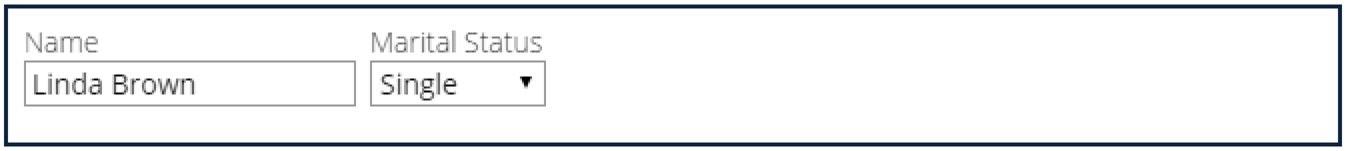
Categorizing events into two types simplifies the event-action concept. In practice, these two event types often overlap. For example, when a user clicks a button (a user event), the action is to set a value on a property. That action then triggers a property change event.

Hiding and showing UI elements

Controlling the fields displayed on screen removes irrelevant elements, and that can simplify the UI. You use **visible when conditions** to hide or display data fields based on a value entered by the user.

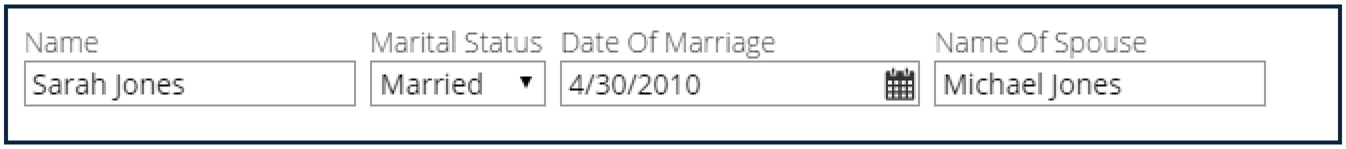
In the following example, users enter marital status in a form. Depending on the value entered in the **Marital Status** field, additional fields may be displayed on the screen.

If the user selects Single, no additional input is required.



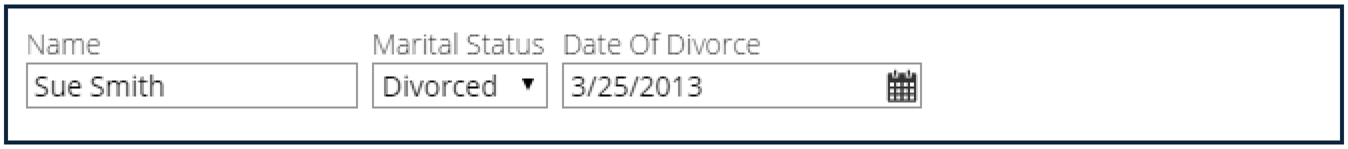
Name	Marital Status
Linda Brown	Single

If the user selects Married, the **Date of Marriage**and **Name of Spouse** fields are displayed.



Name	Marital Status	Date Of Marriage	Name Of Spouse
Sarah Jones	Married	4/30/2010	Michael Jones

If the user selects Divorced, the **Date of Divorce** field is displayed.



Name	Marital Status	Date Of Divorce
Sue Smith	Divorced	3/25/2013

In the above example, visible when conditions use the **Marital Status** value to determine when the **Date of Marriage**, **Name of Spouse**, and **Date of Divorce** fields are displayed.

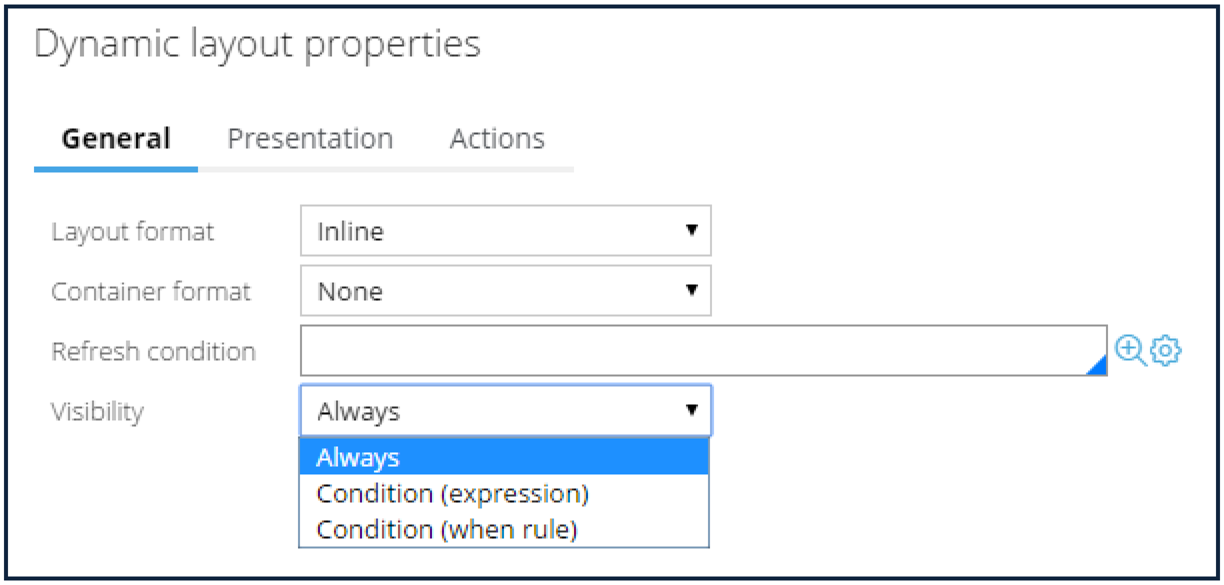
Configuration options for visible when conditions

You can set Visible When conditions on sections included in another section, on layouts, and on cells. The conditions are configured in the **Visibility** field on the General tab of the property panel.

The **Visibility** options are:

- Always — No visibility condition is on this field, layout, or section; the UI element is always displayed.
- If not blank — Visible if the value of that field is not blank.
- If not zero — Visible if the value of that field is not zero.
- Condition (expression) — Uses a boolean expression to determine visibility; visible when the expression returns true.
- Condition (when rule) — Uses a when rule to determine visibility; visible if the when rule returns true.

Layouts and sections include Always, Condition (expression), and Condition (when rule)options.

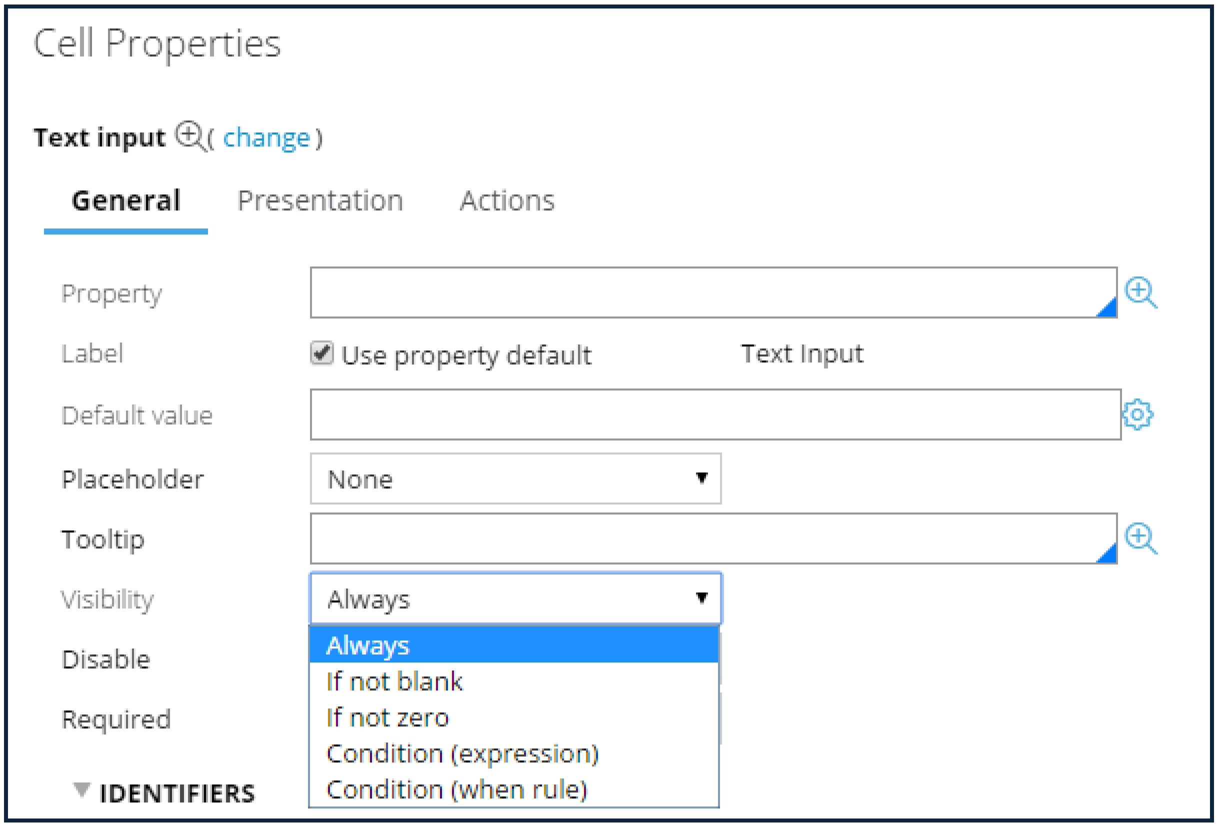


Dynamic layout properties

General | Presentation | Actions

Layout format	Inline
Container format	None
Refresh condition	
Visibility	Always

Cells also include If not blank and If not zero options.



Cell Properties

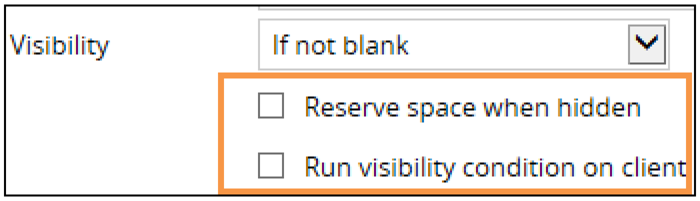
Text input (change)

General | Presentation | Actions

Property	
Label	✓ Use property default — Text Input
Default value	
Placeholder	None
Tooltip	
Visibility	Always
Disable	
Required	

Additional options

There are two additional options available when you select a visible when condition.



The **Reserve space when hidden** option keeps the space surrounding the control open. This prevents the UI elements on the screen from repositioning when the visible content is displayed.

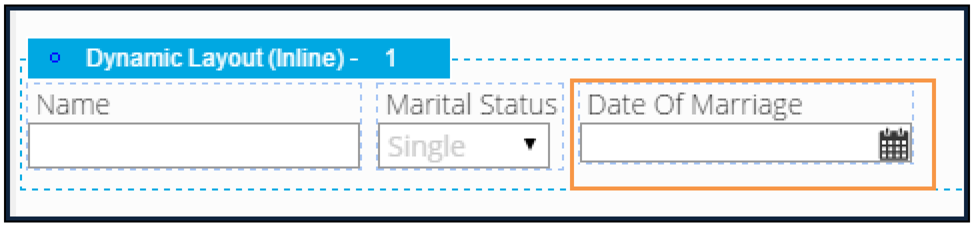
The **Run visibility condition on client**option is displayed when you use the If not blank, If not zero, or Condition (expression)visibility options. When you select the **Run visibility condition on client**option, all of the possible data that can be displayed is included in the clipboard page.

The system uses the data on the page to refresh the section when the visibility condition is met. If you do not select this option, the page does not contain the hidden data. The client communicates with the server to refresh the section. If the hidden content is not likely to change during case processing, select the **Run visibility condition on client** option to reduce the number of server trips and avoid page refreshes.

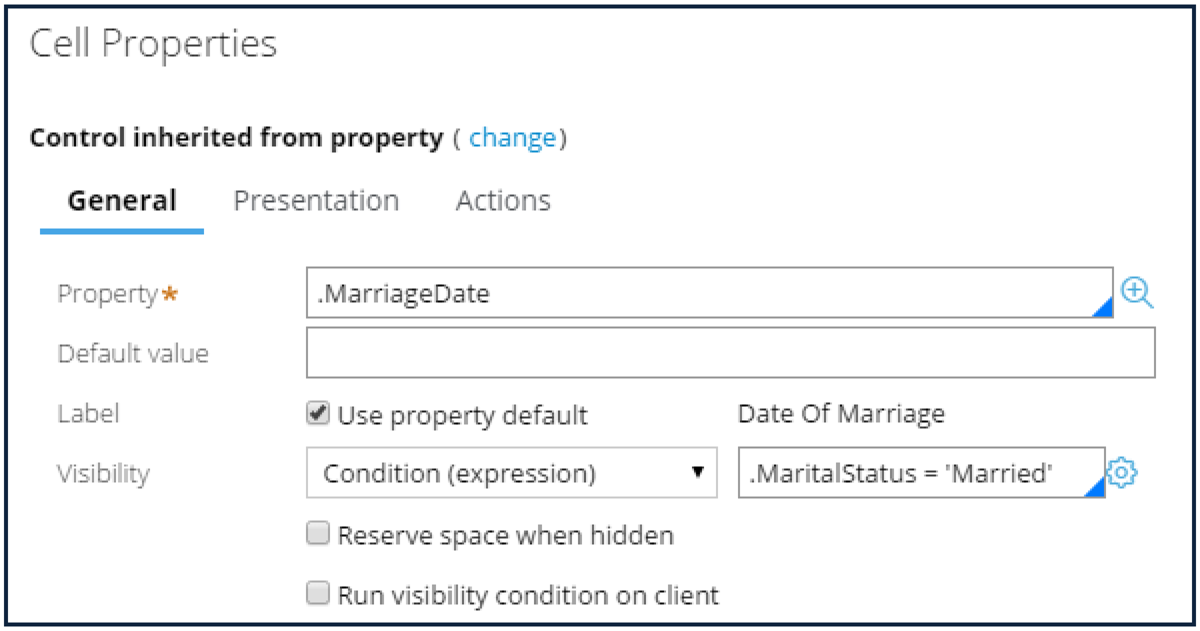
Configuring Visible When conditions on a UI element

First, identify the UI element target that you want to dynamically show and hide. Then, decide at which level — section, layout, or field — to apply the visible when condition.

In the following example, you want to display the **Date Of Marriage** field only when the martial status is set to Married. You configure the cell containing the field.



1. Open the configuration panel for the cell containing the marriage date property.
2. Click the **Visibility** drop-down and select **Condition (expression)** to control the visibility of the marriage date property.
3. Configure the visibility expression for the marriage date property so that the expression value is driven by the marital status value.


Cell Properties

Control inherited from property (change)

General Presentation Actions

Property* .MarriageDate

Default value

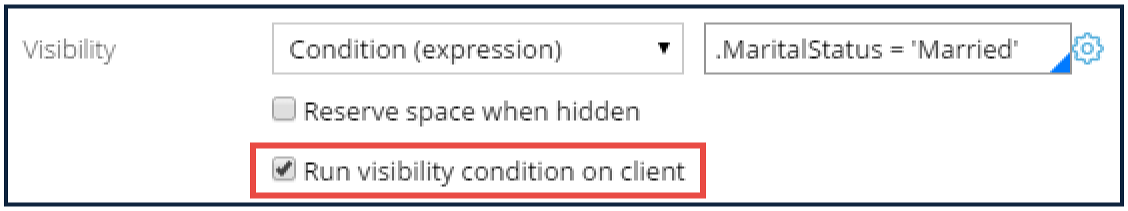
Label [✓] Use property default Date Of Marriage

Visibility Condition (expression) .MaritalStatus = 'Married'

[ ] Reserve space when hidden

[ ] Run visibility condition on client

4. Select **Run visibility condition on client**.


Visibility Condition (expression) .MaritalStatus = 'Married'

[ ] Reserve space when hidden

[✓] Run visibility condition on client

5. Click **Submit.**

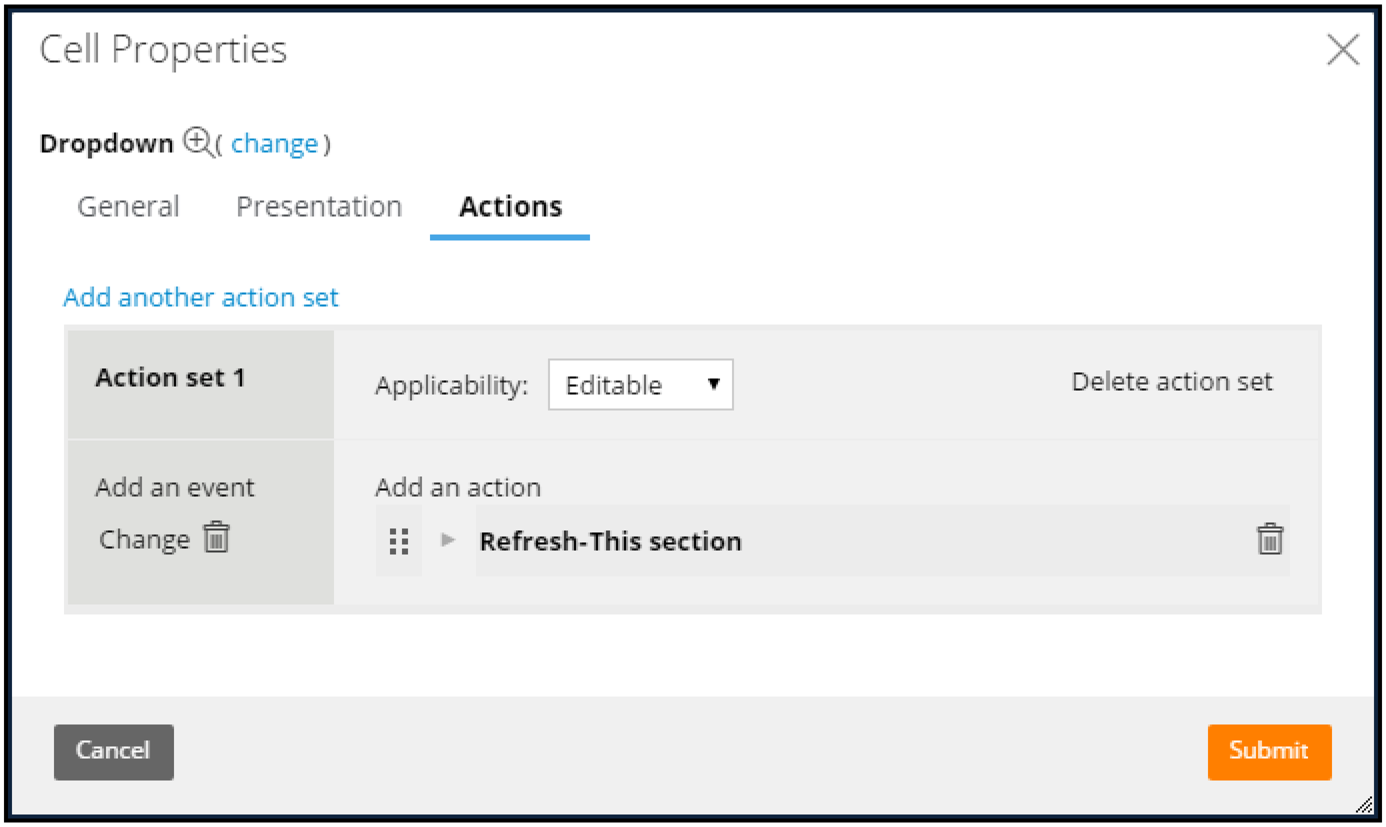
Action sets

In addition to visible when conditions for showing or hiding fields, you use **action sets** to configure a dynamic UI. An action set consists of an event, an action, and (optionally) conditions.

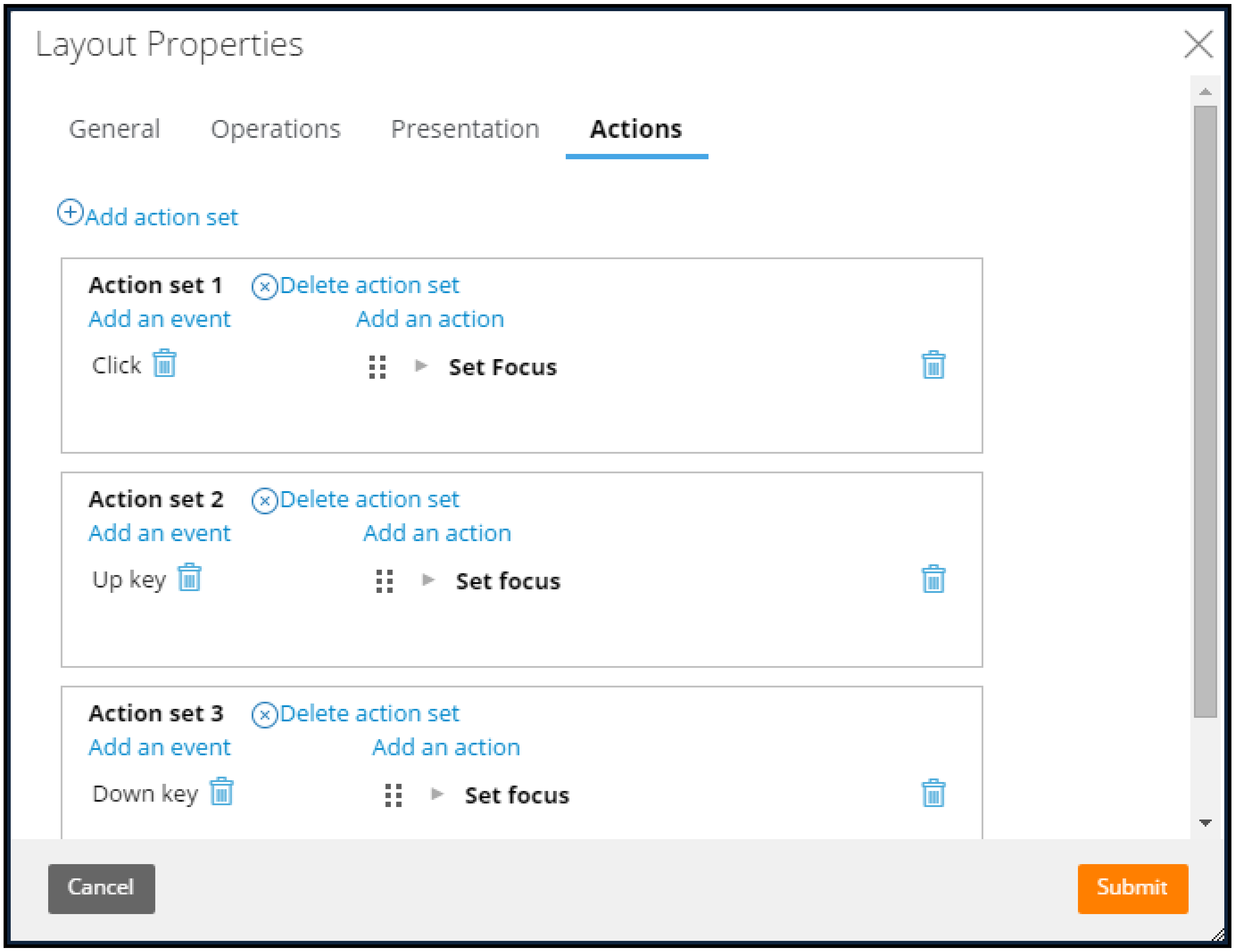
- **Event** — A trigger performed by the user, such as clicking a button, hovering a mouse pointer over a field, or entering a value in a grid
- **Action** — A response performed by the system as a result of the user event (for example, when the user clicks a button, a case is created.)
- **Conditions** — Restrictions such as when rules, which can be applied to an event and action combination (for example, you can configure conditions so that hovering over a field displays a smart tip message only if the field contains a property value.)

For each action set, you must define at least one event and one action. You define action sets on UI controls and grids. You can create multiple action sets for a single control or grid.

In most cases, you define action sets on controls. Action sets are configured on the Actions tab of the control's Cell Properties form.



For grids, action sets are configured on the Actions tab of the Layout Properties form.


Layout Properties

General Operations Presentation **Actions**

Add action set

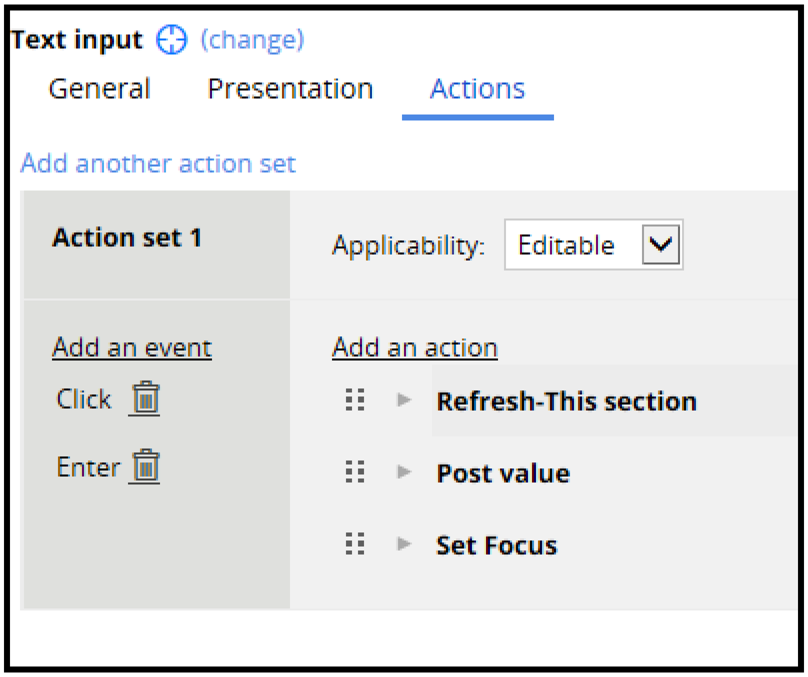
Action set 1 Delete action set
Add an event Add an action
Click **Set Focus**

Action set 2 Delete action set
Add an event Add an action
Up key **Set focus**

Action set 3 Delete action set
Add an event Add an action
Down key **Set focus**

Cancel Submit

You can define multiple events and actions within an action set. The system executes the actions in the order the actions are listed.


Text input (change)

General Presentation Actions

Add another action set

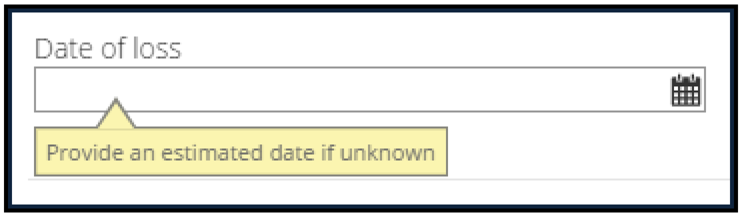
Action set 1 Applicability: Editable

Add an event
Click
Enter

Add an action
Refresh-This section
Post value
Set Focus

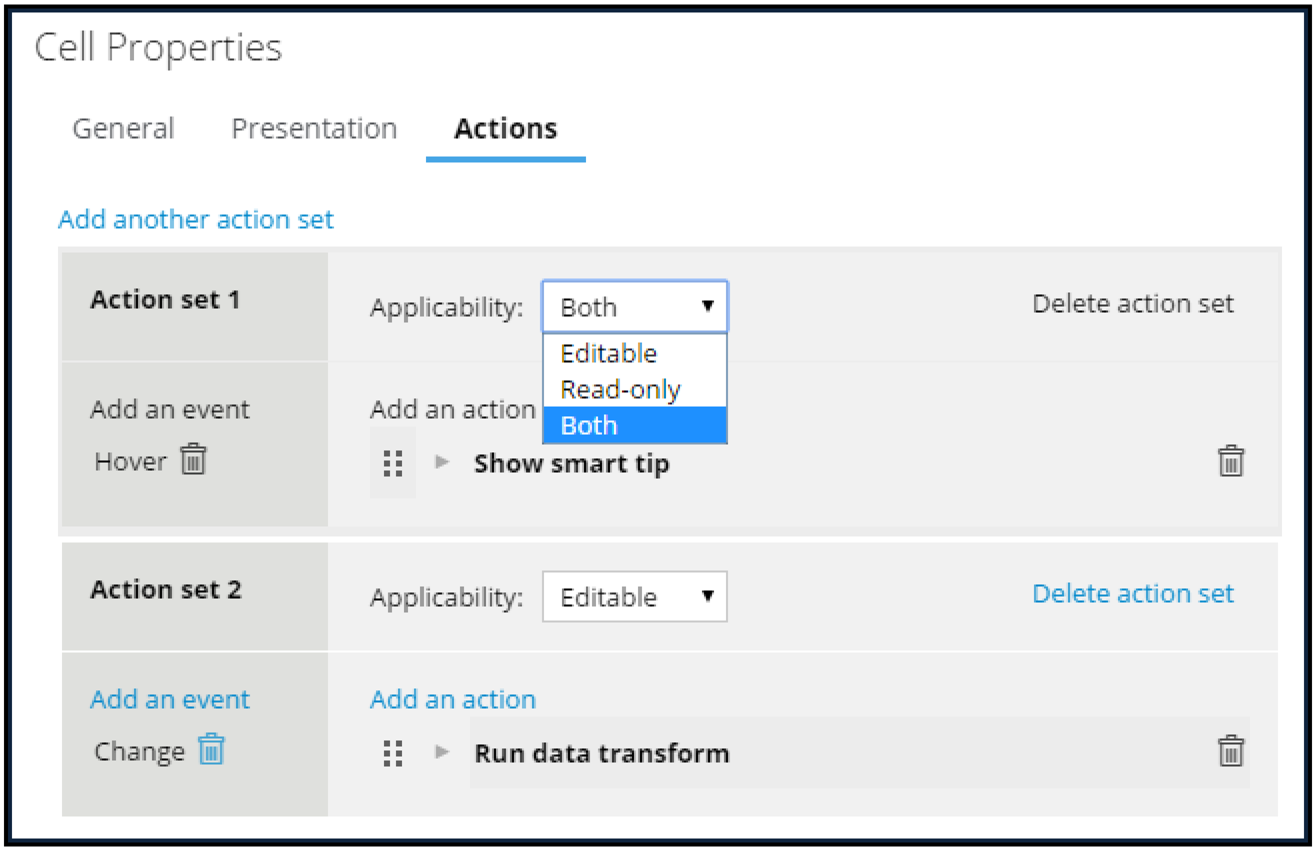
An event triggers an action

An insurance claims application includes a Date of loss field. You want to provide a smart tip to explain the purpose of the field to users. Depending on how long ago the loss occurred, you also want to set a few property values based on that date.


Date of loss

Provide an estimated date if unknown

To implement this behavior, you define one action set for the hover event showing the smart tip, and another action set to run a data transform that sets the values.


Cell Properties

General Presentation **Actions**

Add another action set

Action set 1 Applicability: Both (Editable / Read-only / Both) Delete action set

Add an event: Hover — Add an action: **Show smart tip**

Action set 2 Applicability: Editable Delete action set

Add an event: Change — Add an action: **Run data transform**

The smart tip is displayed when the mouse hovers over the date field — regardless of whether the field is read-only or editable — since you have set the visibility of the hover action set to Both.

When users change the value of the date field that is editable, the change event triggers a data transform that set the property values.

Implementing action sets

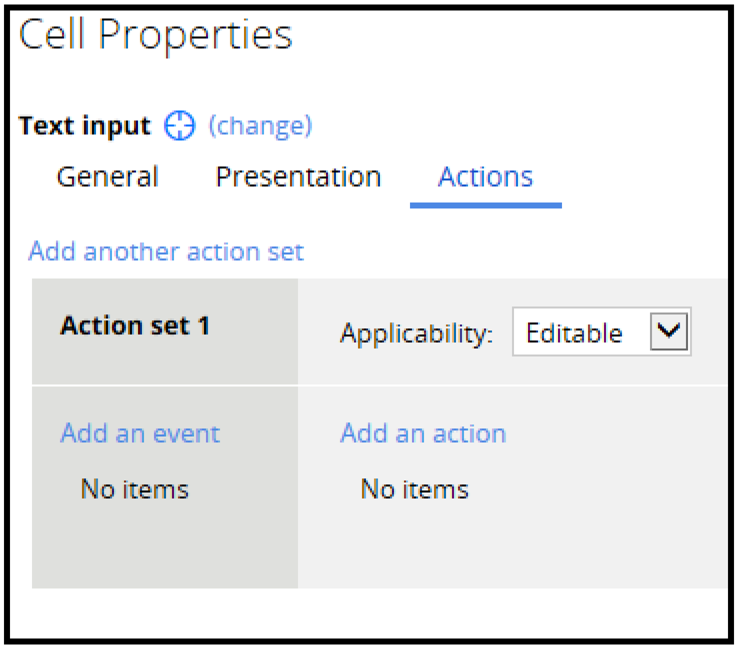
The following table shows a few examples of how you might implement action sets in your UI.

Event	Action
Click a control such as button, link, or icon.	Opens a new window
Double-click a row in the grid.	Opens the row in edit mode
Right-click the entire grid.	Shows a menu
Press the Esc key in the keyboard.	Closes the assignment and returns to the home page
Select a value from the state drop-down.	Updates the list of counties
Click a check box.	Unmasks the password
Enter a value in the quantity field.	Calculates the total

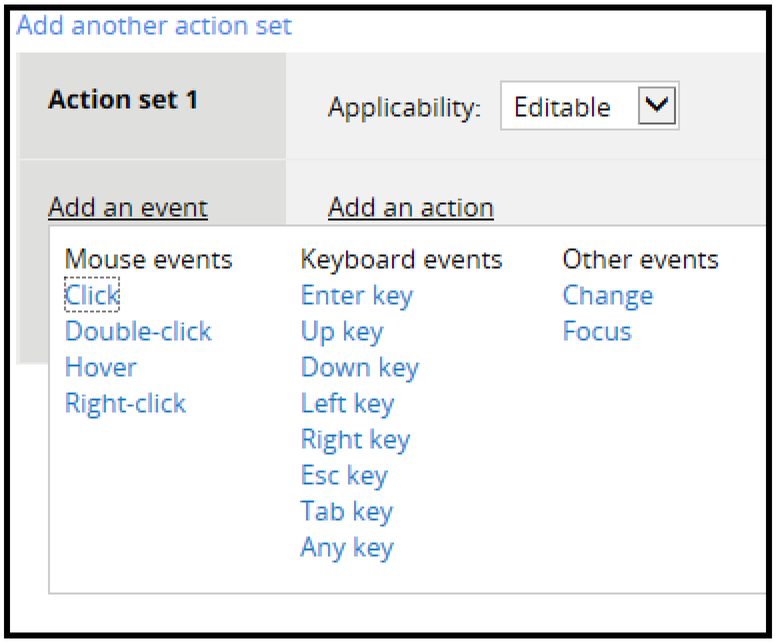
Configuring an action set

Configure an action set if you want a user action (such as clicking a button) to trigger an action (such as displaying a smart tip message).

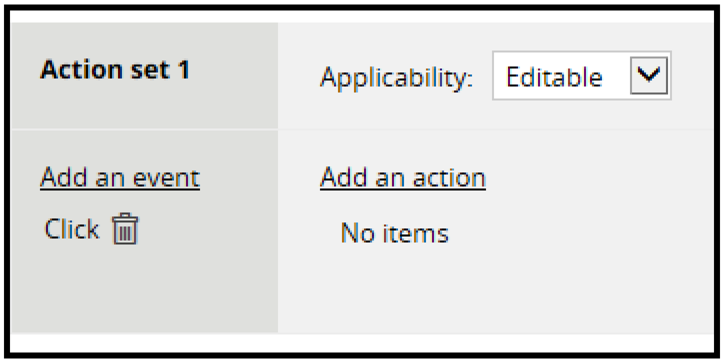
1. Open the Properties panel of a UI control.
2. Select the **Actions**tab.
3. Click **Create an action set**. An action set section is displayed.



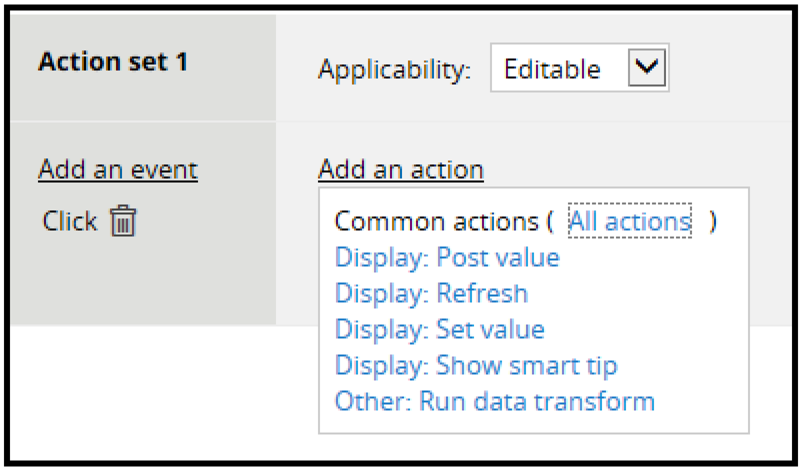
4. Click **Add an event** in the left column. A menu of events is displayed.



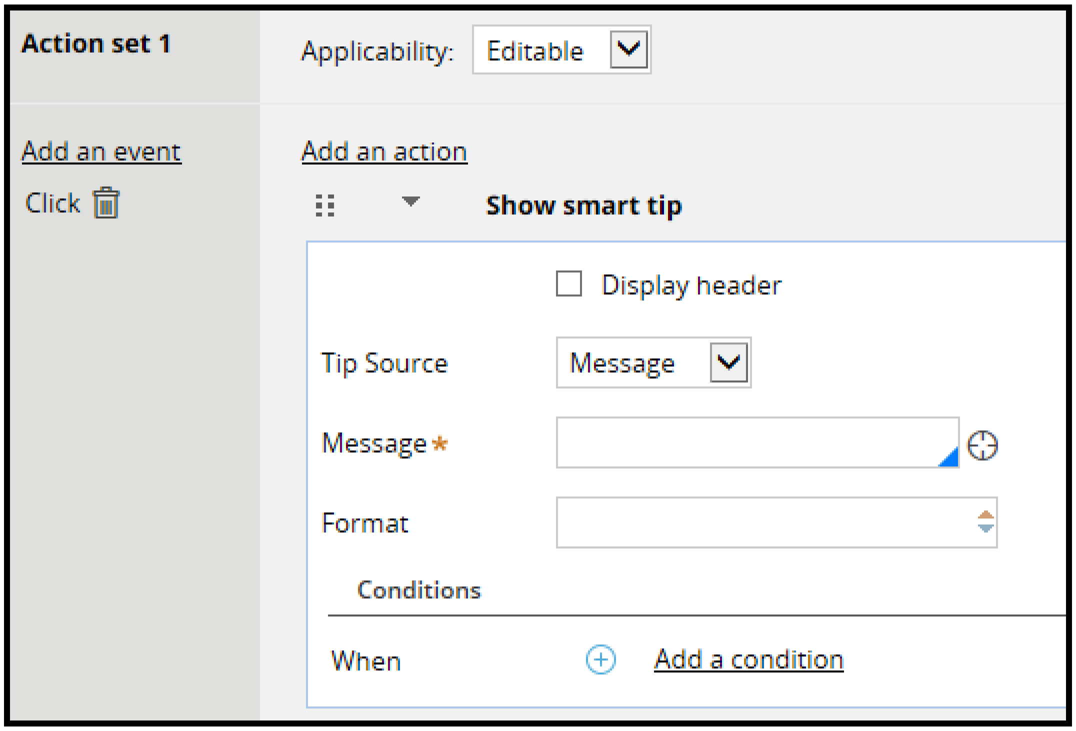
5. Select an event. The event appears in the left column.



6. Click **Add an action** in the right column. A menu of actions is displayed.



7. Select an action. The action is displayed in the right column. A section for adding information that is relevant to the action may also be displayed, as shown in the following example.



8. Enter the information.

   Note: Use the **Add an event** and **Add an action** links to include more than one event or action in a single action set.

9. Click **Submit**.

Editing or deleting action sets

To edit an action set, double-click the row.

To delete an action set, select the row and click the **X**.

Validating user data

Introduction to Validating User Data

When you design a user form, you add all the fields and controls required by the specification. However, users must enter data that use a format or contain a value the system can process correctly. Pega provides rules that validate the data and help prevent processing errors when a form is submitted.

After this lesson, you should be able to:

- Explain the options for ensuring valid data entry by users.
- List the type of user interface (UI) controls that provide data validation.
- Present a dynamic list of data entry options.
- Validate user data using a validate or an edit validate rule.

Methods of data validation

When you design a user form, you add all the fields and controls that the specification requires. You must also consider how to ensure that the data values entered by users are valid. Valid data is required so that the system can process the information without error.

The following list describes a few important design requirements.

- The data must be the right type. For example, a user must enter a number in a total purchase amount field.
- The data must fit the business logic. For example, a date of birth field is usually in the past.
- The data must be restricted to possible values. For example, a user can only select a valid loan type by selecting the type from a list of options.

To prevent processing errors, Pega provides property types, controls, and rules that support most validation requirements.

Properties

Single value properties have property types such as date, decimal, integer, text, or true/false. Selecting the appropriate property type ensures that users enter a valid value. For example, a purchase price field that uses a decimal property type ensures that users can enter only numeric values and cannot enter text.

Controls

Controls are another way you restrict users from entering or selecting invalid values on a form. For example, when a form requires a date, using a calendar control ensures that users enter a date value.

You can also use controls to allow users to select only valid values. For example, you can use drop-down lists or radio buttons so that users can only select the available values.

In addition to ensuring valid values, you can make fields required. This ensures that users enter a value before they can complete an assignment.

Validation rules

You use validation rules when you cannot predict or control the value a user will enter in a form. There are two types of validation rules: **validate** and **edit validate**.

You use validate rules to compare a property against a condition when the user submits a form. A validate rule is typically referenced from a flow action. If the user enters a value that fails to meet the condition, the form displays an error when the form is submitted. For example, assume your form contains a field for date of birth. The property type and control cannot prevent users from entering a date that is in the future. However, you can design a validate rule to display an error if the user submits a date that is in the future.

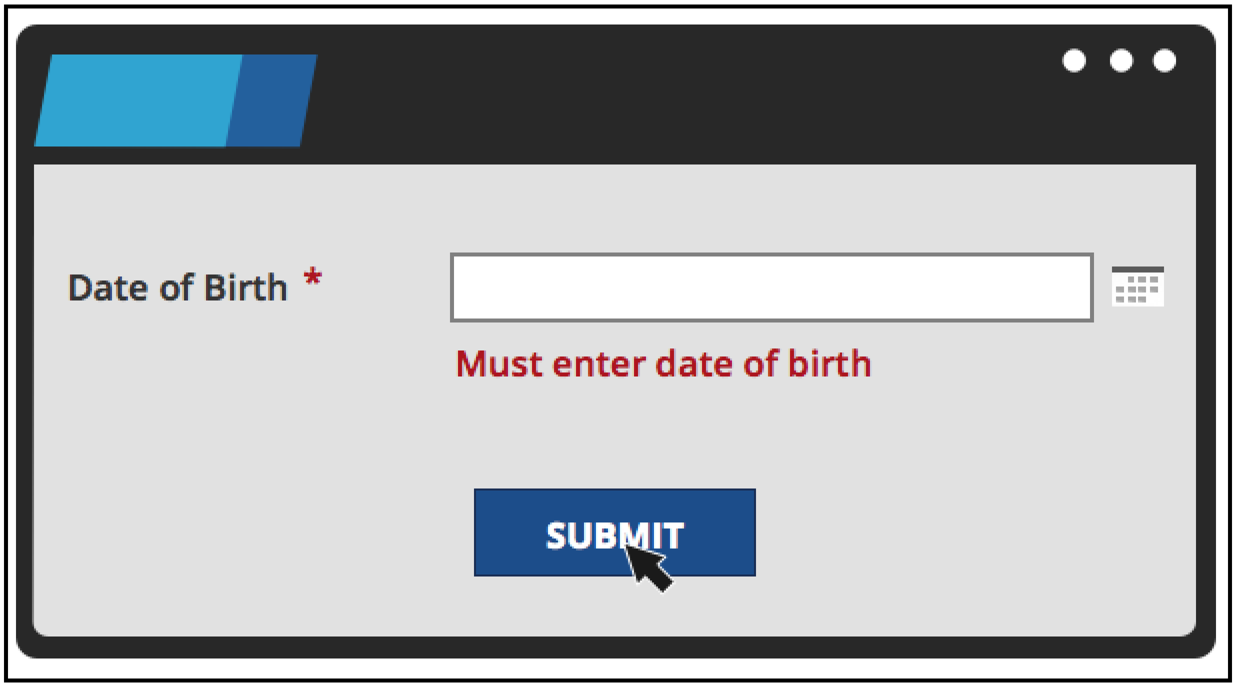
You use edit validate rules with single value, value list, and value group properties to test for patterns. Edit validate rules are referenced from a property rule. For example, you can configure a zip code property to reference an edit validate rule that tests whether the entered value has five digits. In another example, an email address can reference an edit rule to test whether the entered value contains an "at" (@) symbol. If the submitted value is invalid, the field displays an error. Edit validate rules run when the user exits a field if the harness rule is configured to support client-side validation. Otherwise, edit validate rules are run when the user submits a form.

Note: The standard harnesses provided with the Pega Platform are configured to support client-side validation.

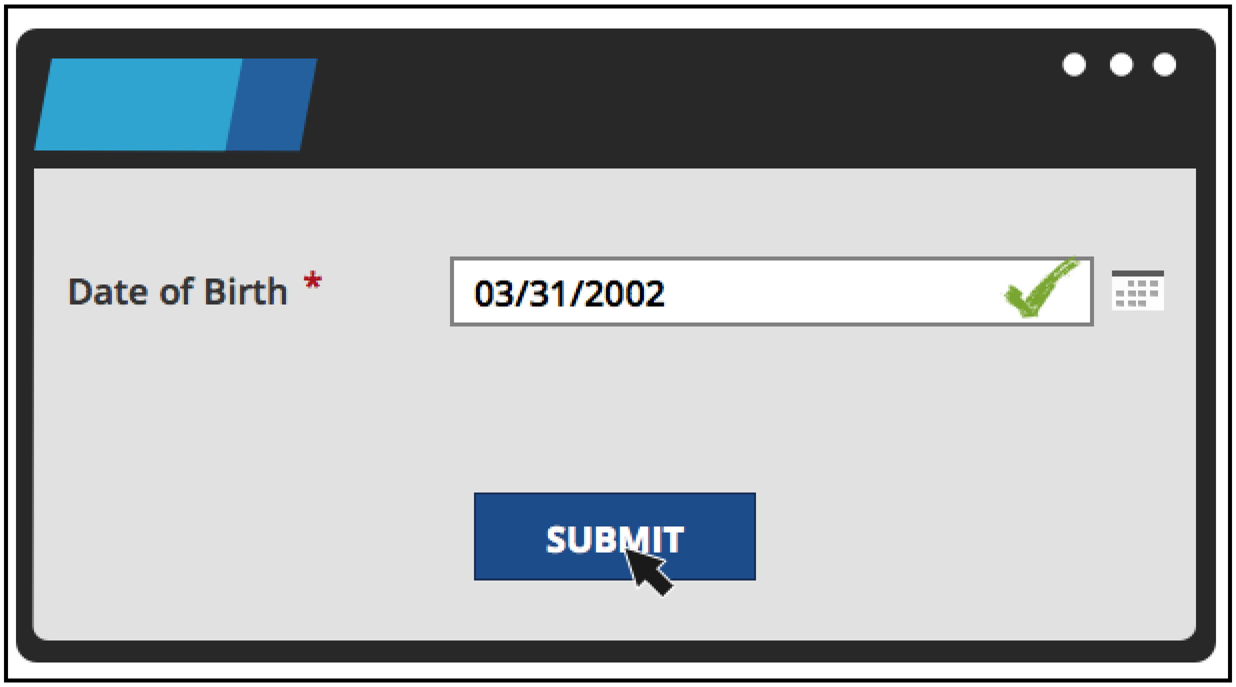
Controls

Controls used on forms provide the most common approach to validation. The three most common ways you can use controls for validation are required fields, editable settings, and control types.

Required fields — Configuring a control as a required field ensures that the user enters a value. If there is no value, users get an error when they try to submit a form. For instance, assume you design a form in which users enter a date of birth to qualify for a discounted auto insurance policy. You configure the date of birth number control as a required field. If the user does not enter a date in the field, an error message appears when the user attempts to submit the form.



The error message does not appear if there is a date in the field.


Date of Birth * 03/31/2002

Editable settings — You can use editable settings on controls to restrict the input values to valid formats. The settings are specific to the control type. For example, you can specify the minimum and maximum of characters allowed in a text input control. You can also specify that users cannot enter dates as text — users must select a date from a calender icon control.

Control types — Using the correct control for a specific purpose helps users enter valid values. The following table shows some example use cases for the different Control types.

Use case	Control type	How the control helps validation
Users must enter a date that includes day, month, and year.	Calendar	Selecting a date from a calendar icon helps ensure that the user enters a date in a valid format.
Users must select one of three possible loan types. The user must see all types on the form.	Radio buttons	Restrict choices to a set of valid values and allows users to select only one value. Generally, you use radio buttons when only a small number of options (for example, fewer than five) is available.
Users must select one of 10 types of office chairs from a list. The options do not need to be displayed on the form.	Drop-down	Restricts valid values to the ones that appear in the list. A drop-down list presents the options only when the user clicks the control. This helps reduce the clutter on the form. Unlike radio buttons, you can configure the drop-down control so that users can select multiple values.

Use case	Control type	How the control helps validation
Users must select the country in which they reside from a list. The user can enter text in the control to help find the right country.	Autocomplete	When a user enters one or more values in the control, the control filters the available options. This helps users find an option in a list if there is a large number (for example, more than 20) of possible options.
Users select an option to purchase extra travel insurance.	Check box	The user can select the check box or leave it blank. This ensures that a true/false property is either true (selected) or false (unselected).

For more information about control rules, see the help topic About Controls.

Validating with controls

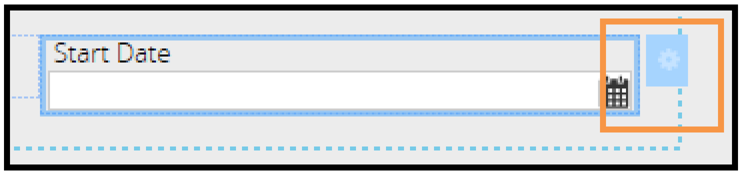
Validating input fields using controls is a simple way to prevent users from entering invalid values in input fields. Requiring data entry in a field or defining character limits are common methods for helping users enter the correct values.

Specifying required fields

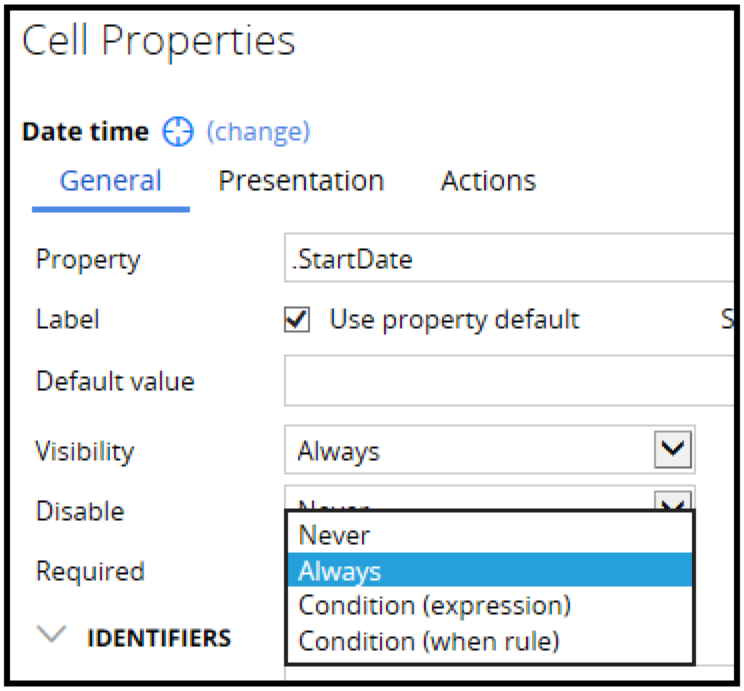
You can specify one or more controls on a form that requires a user to enter a value. If a user does not enter or select a value, the form displays an error when the user attempts to submit the form.

Follow these steps to specify a required field.

1. In a section, select the control in which the user must enter a value.
2. Click the **gear** icon to open the field's properties panel.



3. In the General tab on the Cell Properties panel in the **Required** drop-down list, select Always so the user must enter a value under any condition. If you want to make the field required under specific conditions, select either Condition (expression) or Condition (when rule).


Cell Properties

Date time (change)

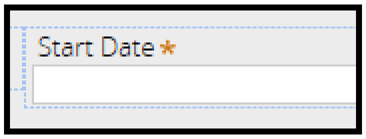
General | Presentation | Actions

Property	.StartDate
Label	☑ Use property default
Default value	
Visibility	Always
Disable	Never
Required	

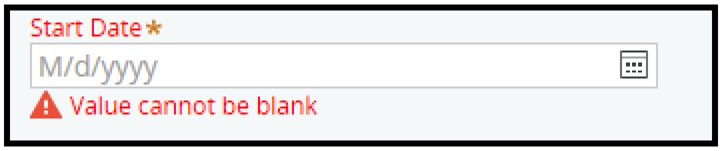
IDENTIFIERS

4. In the panel, click **Submit.**

   The field displays an asterisk (*) to indicate that the user must enter a value.



When a user leaves the field blank and attempts to submit the form, the system displays the following error message.



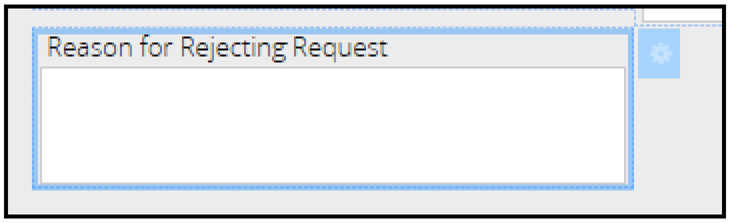
Value cannot be blank

Specifying character limits in text controls

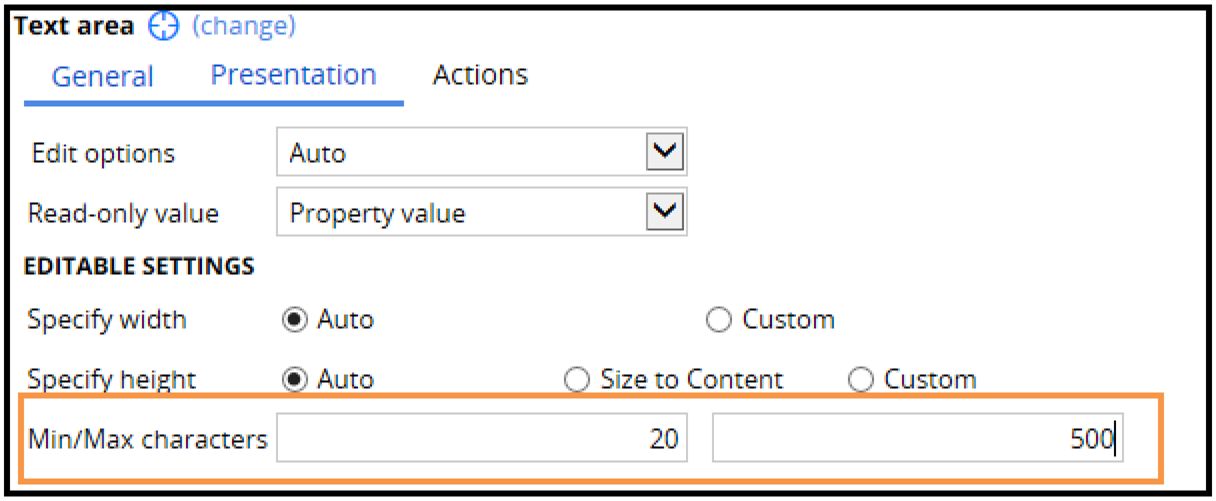
You can use editable settings in Text Input or Text Area controls to limit users to a minimum and/or maximum number of characters.

Follow these steps to specify character limits.

1. In a appropriate section, select the text field you want update as shown in the following example.



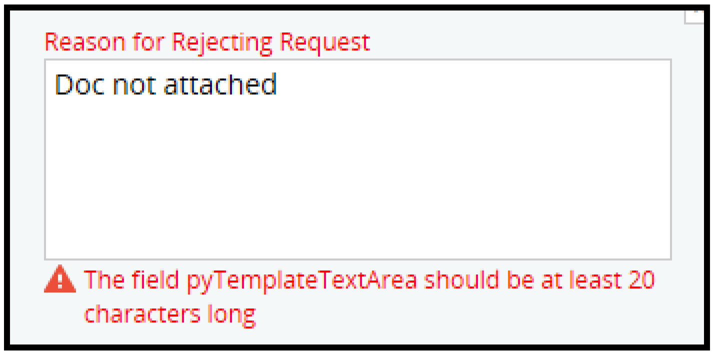
2. Open the field's properties panel.
3. In the Presentation tab on the properties panel, enter numeric values in the Min/Max characters field as shown in the following example. If necessary, you can enter a value in only one field.



Edit options: Auto
Read-only value: Property value
EDITABLE SETTINGS
Specify width: [✓ Auto] Custom
Specify height: [✓ Auto] Size to Content Custom
Min/Max characters: 20 500

4. In the properties panel, click **Submit**.

When a user enters fewer than 20 characters and submits the form, the following error message appears.


Reason for Rejecting Request

Doc not attached

The field pyTemplateTextArea should be at least 20 characters long

Dynamic lists of data entry items

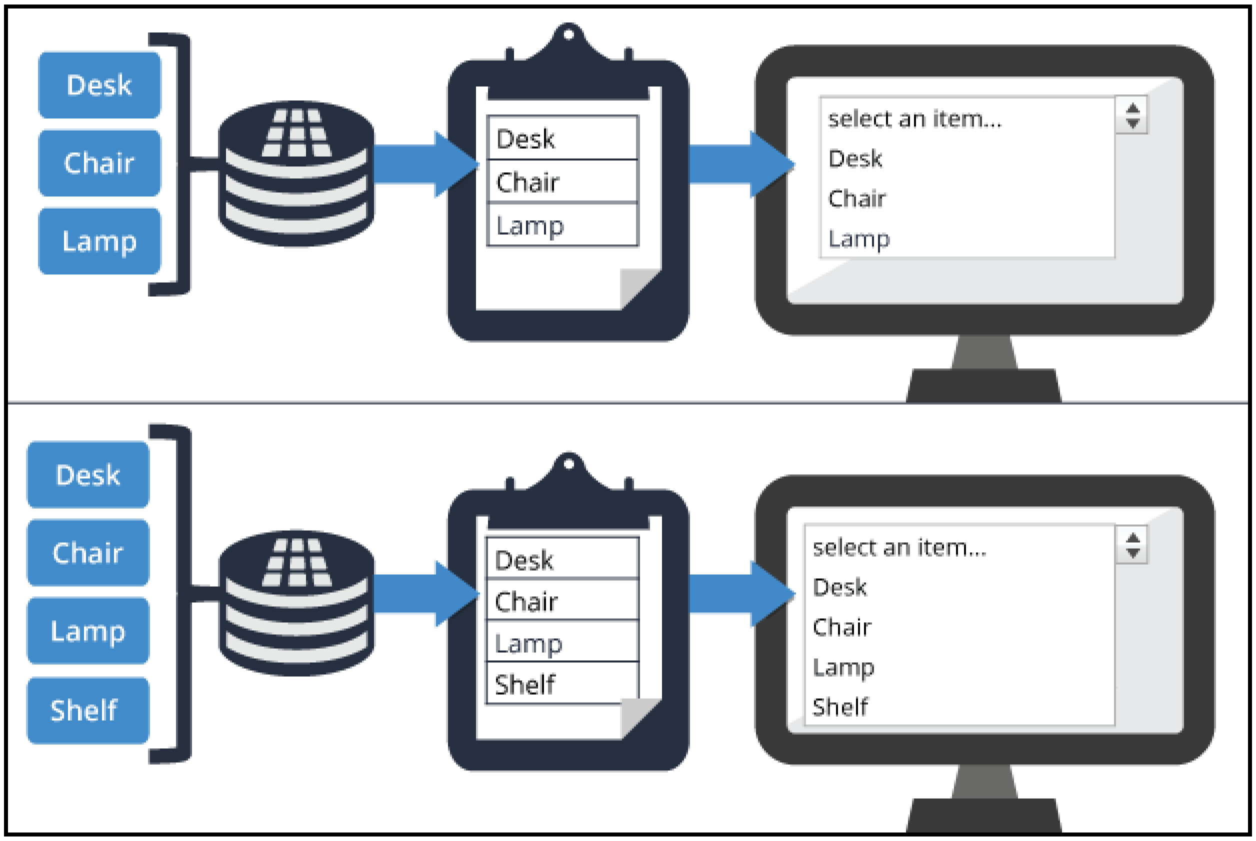
Autocomplete or drop-down controls display lists of items so users can make valid selections. In many situations, the items are listed in the control's property. However, if items on the list change frequently, the list may not display the most current information. For instance, assume your organization lists office furniture items in a drop-down control. The items may change every few days. When a user displays the list, that list may not display the most current items.

However, using **dynamic lists** helps ensure that the items users see on the list reflects the most recent information. Dynamic lists reference data on the clipboard. Using a data page is the most common method of sourcing the data. A data page receives the data from a database, and then populates the clipboard. When the clipboard is refreshed, the data page dynamically updates the list.

Note: A **data page** is a persistent page on the clipboard, used to cache data for use in an application. In a future lesson, you learn how to create and populate a data page. In this lesson, you focus on how to retrieve information from an existing data page.

Using the previous example, assume your organization maintains data for office furniture in a database. The data includes an item ID and an item name. If you use a data page to refer the data to the clipboard, a drop-down list can reference the data dynamically and display the items by the item name. When items are added or removed from the database, the drop-down list displays the currently available items.

As shown in the following example, when the Shelf item is added to the database, the drop-down list references the new item from the clipboard.

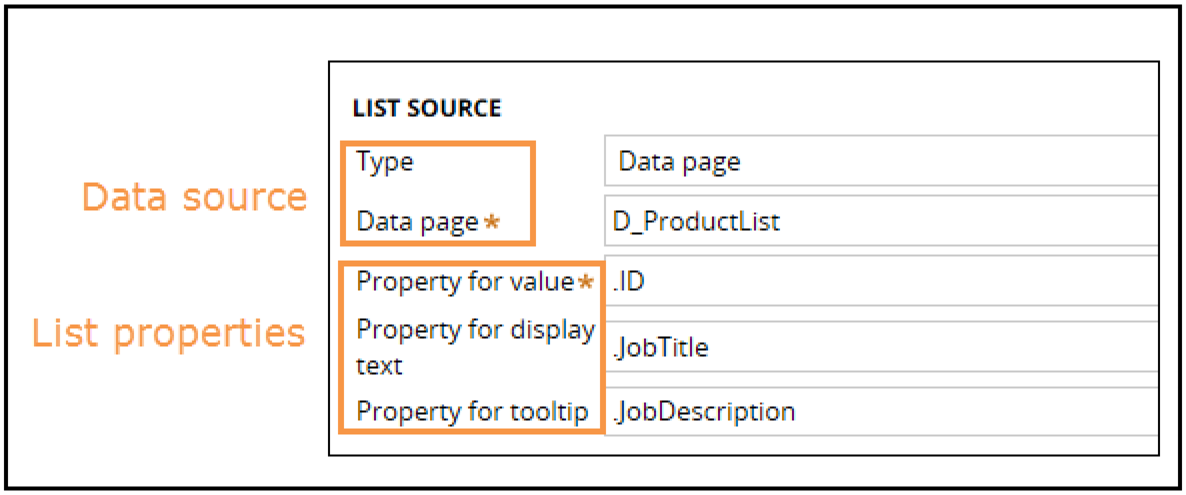


You can use data page or report definition parameters to filter the clipboard data you want to display. For example, assume the clipboard contains records for automobile makes and models. When a user selects a specific automobile make — such as Ford — in the first drop-down list, the next drop-down list uses a parameter to display only Ford models.

How to create a dynamic list

You dynamically list data values in drop-down and autocomplete controls. The dynamic list uses data on the clipboard that is sourced from a data page, a report definition, or a clipboard page.

There are two major steps to creating a dynamic list. First, specify and identify the source. Then, define the properties you want to include in your list. To begin the procedure, open the control's properties panel. The settings are located in the List Source section on the panel.



LIST SOURCE

Type	Data page
Data page *	D_ProductList
Property for value *	.ID
Property for display text	.JobTitle
Property for tooltip	.JobDescription

First, specify the type of list source and identify it. In the Type field, select the type of source. Then, define the source. The options depend upon the source type. For example, if you select a data page, enter the name of the data page.

Then, specify the following properties used in the list:

- A property from the source data that identifies the property used in the control
- A property to display the names of the items that appear on the list
- Optionally, a property to use in a tooltip

For example, assume the source data contains information about office furniture products. Each item is identified by a product ID property, product name property, and product description property. You can use the properties as follows:

- Product ID — Identify the property that identifies the items in the list (list of office products).
- Product name —Display the names of the items (desk, chair, and lamp).
- Product description — Display tooltip text for each item (for example, a desk item might display "The desk is solid oak and has three drawers").

For more information

For more information about creating dynamic lists in an autocomplete control, see the help topic Autocomplete Properties — General tab.

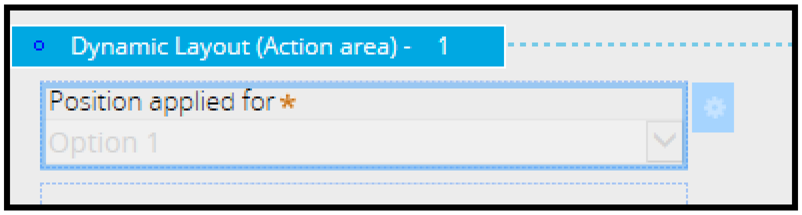
For more information about creating dynamic list in a drop-down control, see the help topic Dropdown control Properties General tab.

Creating a dynamic list

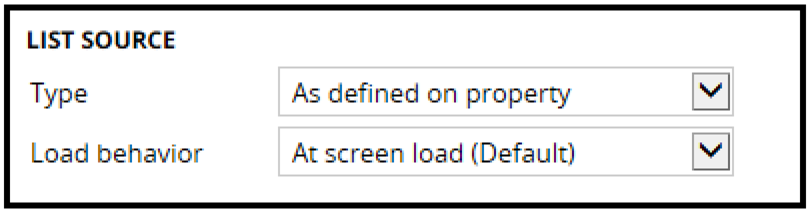
You can use a data page, a report definition, or a clipboard page as a source for a drop-down or autocomplete control. The source provides the data used in the control. You specify the control's source on the control's property panel. .

Follow these steps to create a dynamic list for a drop-down control:

1. Open the section that contains the control you want to update.
2. In the section layout, select the drop-down control and click the **gear** icon to open the control's property panel.



3. Scroll to the List Source section.

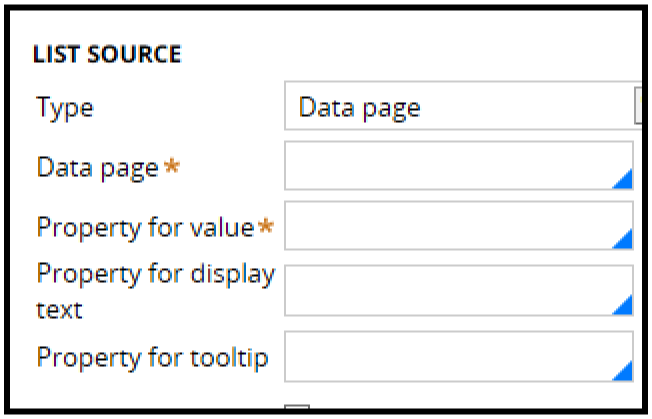


LIST SOURCE

Type	As defined on property
Load behavior	At screen load (Default)

4. In the **Type** field, select a list source type. In the following example, the selected source type is Data page. The List Source form then displays **Data page**.

Note: The field(s) that appear after the Type field depend upon the value you selected.



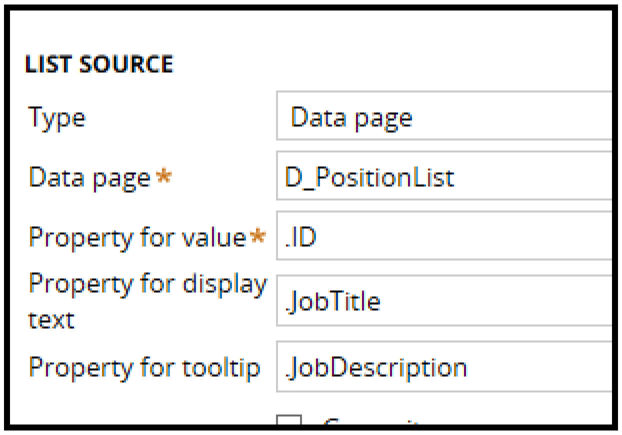
LIST SOURCE

Type	Data page
Data page *	
Property for value *	
Property for display text	
Property for tooltip	

5. In the **Data page** field, select a data page in this field.
6. In the **Property for value** field, enter the property you want to use to identify the items on the list. This is the value you use to set the property specified on the form.
7. Optional: In the **Property for display text** field, select the property you want to display in the list.

8. Optional: In the **Property for tooltip** field, select the property that contains text you want to display when a user hovers the mouse pointer over an item in the list.

   The following image displays a completed List Source section.

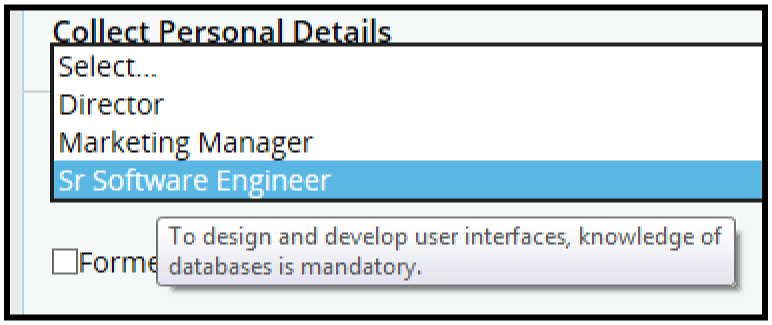
   

   LIST SOURCE

Type	Data page
Data page*	D_PositionList
Property for value*	.ID
Property for display text	.JobTitle
Property for tooltip	.JobDescription

9. Click **Submit**in the properties panel.
10. Click **Save** in the section.

When users select the Position Applied for drop-down control (beneath the Collect Personal Details label), the system displays the following items and tooltip.



Collect Personal Details

Select...
Director
Marketing Manager
Sr Software Engineer

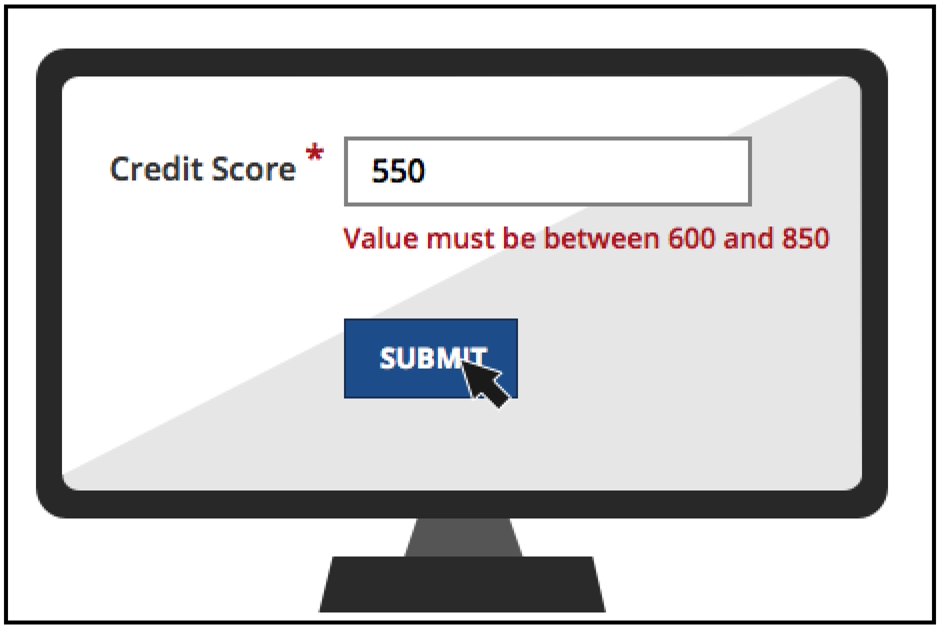
Forme

To design and develop user interfaces, knowledge of databases is mandatory.

Validate rules

Controls and properties help control the type of data users enter. However, these validation methods cannot ensure that the data is correct in specific business logic. Use validate rules to make sure that the data conforms to the logic.

For example, you can use a validate rule so users must enter a credit score that is between 600 and 850. An error message is displayed if the value is outside this range.

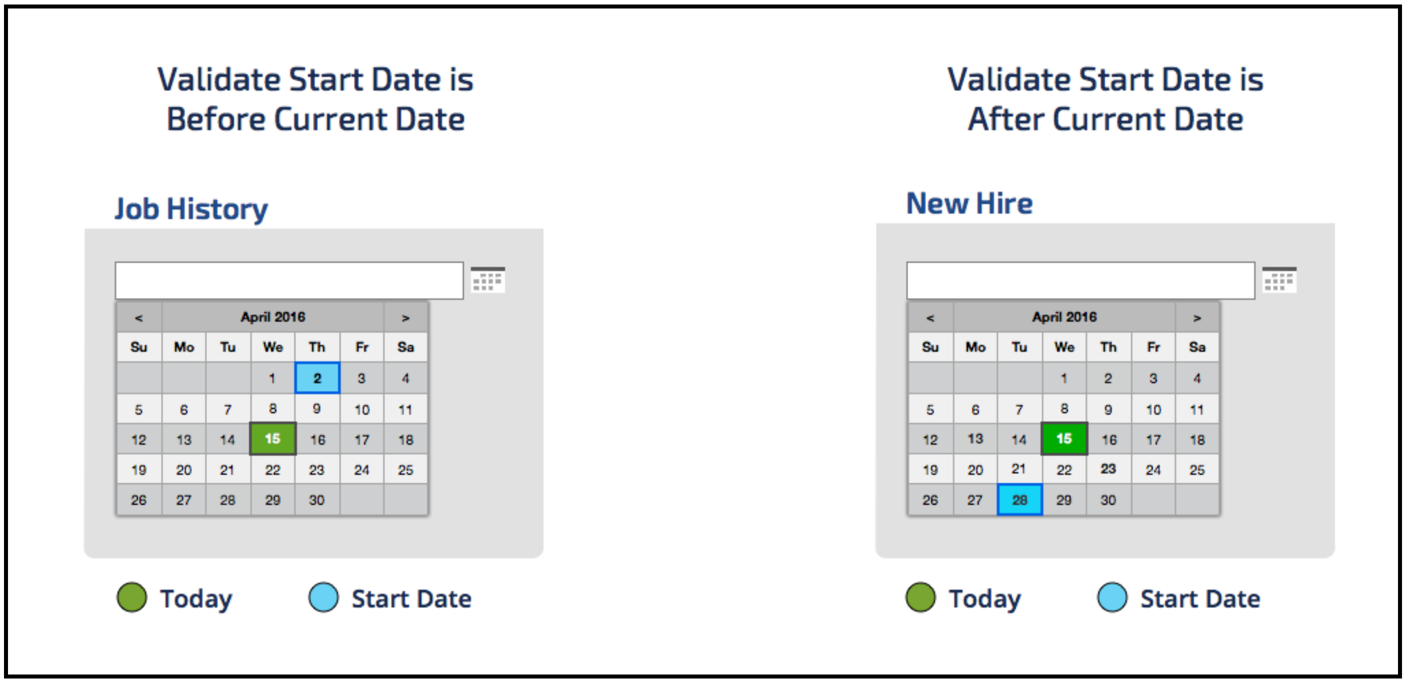


Validate rules are associated with processes such as flow actions. The association between validate rules and processes allows you to validate the property based on specific business needs.

For example, two forms may contain the same employee start date property.

- In a job history form, a user enters the start date of an employee who already works at the company. The user must enter a date before the current date.
- In a new hire form, a user enters a start date for an employee who has not started work. The user must enter a start date after the current date.

Using two validate rules, you can ensure that the dates conform to the date values on each form. You do not need to use two versions of the start date property.


Validate Start Date is Before Current Date
Job History
April 2016
Today
Start Date
Validate Start Date is After Current Date
New Hire
April 2016
Today
Start Date

In addition to flow actions, you can use validate rules in other areas of the business process. For instance, you can use a validate rule in a stage in the case life cycle. A validate rule can ensure that users have entered the correct data or performed all the processes before the case can enters the next stage. For example, in an Onboarding application, a resume must be collected in a Collect Information stage before the case can enter the Interview stage.

KNOWLEDGE CHECK

ANSWER

To ensure that a user selects a value from a drop-down list, you should use a validate rule.

No. You would configure the drop-down list as a required field.

How to use validate rules

A validate rule tests an input property against one or more conditions that you specify in the rule. You use functions to define conditions. Each condition consists of a value to compare, a value to compare against, and a function that describes the comparison. You associate validate rules with the business process. Typically, validate rules are associated with a flow action or a stage.

There are two major steps to using validate rules. First, define the conditions. Second, associate the validate rule with the business process.

Define the conditions

After you have created a validate rule in your ruleset, open the **Validate** tab. Then, in the **Property** field, enter the property that will be compared in a validation condition.

In the Conditions area to the right of the property, use the fields and controls to define a property and function that defines the comparison condition. You can select from a large set of a standard functions for you conditions. For example, you can use a standard function to make sure a date is either before or after the current date.

You can add one or more conditions using the **add row** icon. Use the AND/OR drop-down field to indicate the logical relationship between the functions. For instance, you can validate that a user must have a user score greater than 650 and have a savings account balance greater than USD5,000 in order to qualify for a promotional credit card interest rate.

You can validate more than one property in the validate rule. Under the a Property field, click the **add row** icon to add a property.

Optionally, in the message field, enter a specific text description of the issue so users know how to correct the issue and successfully submit the form. For example, enter a message that reads, "Value must be greater than 200," rather than ">200".

Associate the validate rule with the business process

To use a validate rule with a flow action, open your process model. Open the flow action you want to update. Then, on the Validate tab of the flow action form, select your flow action in the **Validate** field.

To use a validate with a stage, select the stage in the case life cycle. In the Stage panel on the right side of the case life cycle, open the Validation tab. Then, select your validation rule in the **Set entry validation** field.

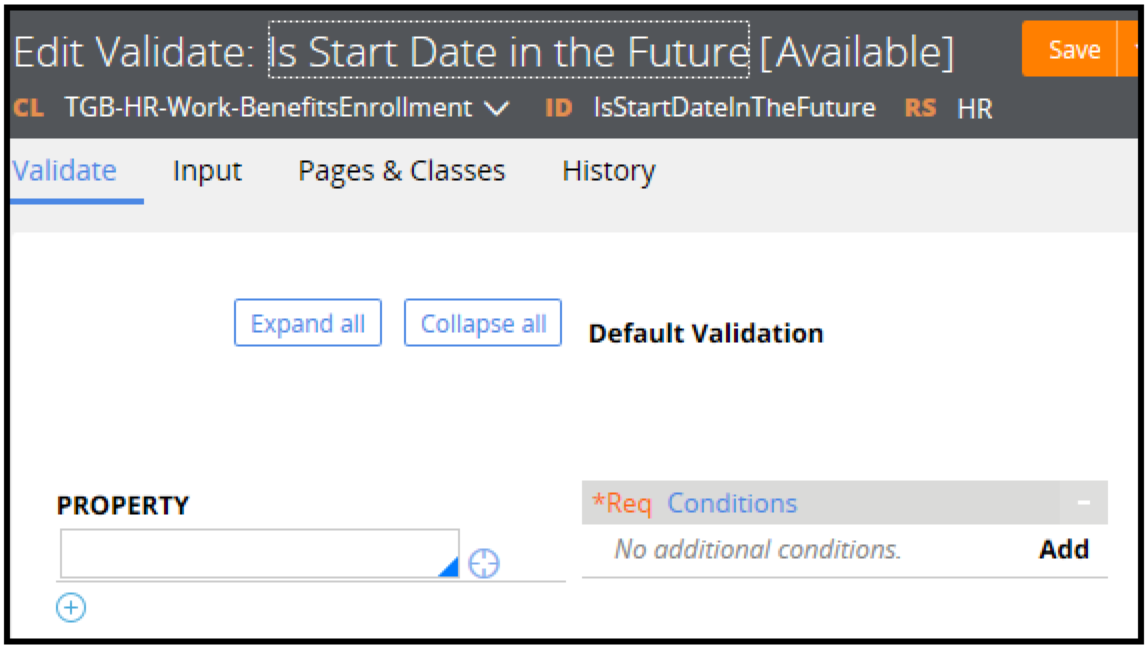
- For more information about configuring validate rules, see the help topic Validate form — Completing the Validate tab
- For more information about using the Validation tab on the flow action form, see the help topic Flow action form — Completing the Validation tab.

Validating a flow action with a validate rule

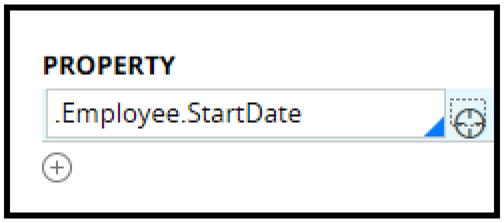
To validate an input field with a validate rule is a two-step process. First, create and define a validate rule. Then, update the flow action properties panel to specify the validation rule you want to use.

Create the validate rule

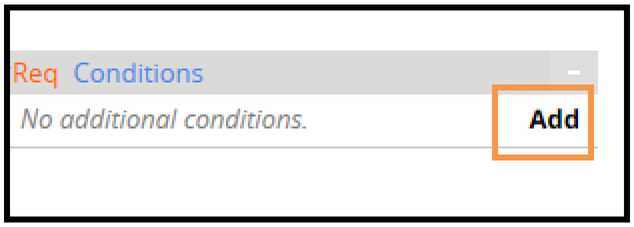
1. In the Application Explorer, right-click the case type in which you want to create the validate rule.
2. Select **Create > Process > Validate** to open a Create Validate form.
3. Enter a specific name in the **Label** field. For example, enter "Is Start Date in the Future" for a rule that validates whether the start date for a new employee is in the future.
4. In the Context area, add the Apply to class and ruleset. Then, select **Create and Open.** The Application Explorer displays the Validate form.


Edit Validate: Is Start Date in the Future [Available]
Save
CL TGB-HR-Work-BenefitsEnrollment
ID IsStartDateInTheFuture
RS HR
Validate
Input
Pages & Classes
History
Expand all
Collapse all
Default Validation
PROPERTY
*Req Conditions
No additional conditions.
Add

5. In the Validate form, enter the property you want to test in the Property field. In the following example, the value for .Employee.StartDate is tested when a user enters a date and submits the form.



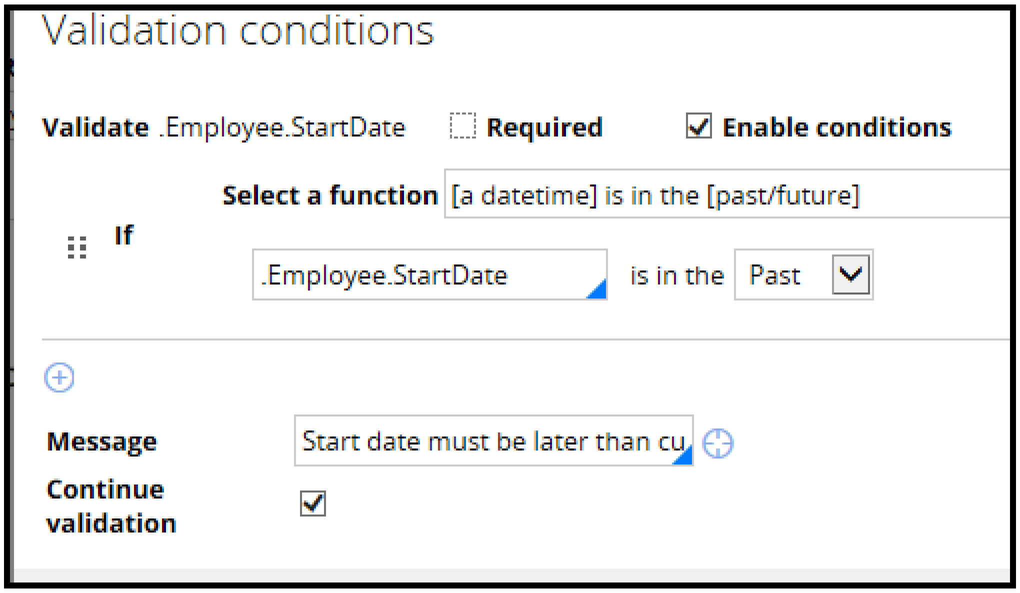
6. In the Conditions header, click **Add**.



7. In the Validation conditions dialog, enter the following information.

Field	**Information**
Select a function	[a datetime] is in the [past/future]
If	.Employee.StartDate is in the Future
Message	Start date must be later than the current date

The following image displays the completed dialog.



Validation conditions

Validate .Employee.StartDate ☐ **Required** ☑ **Enable conditions**

Select a function [a datetime] is in the [past/future]

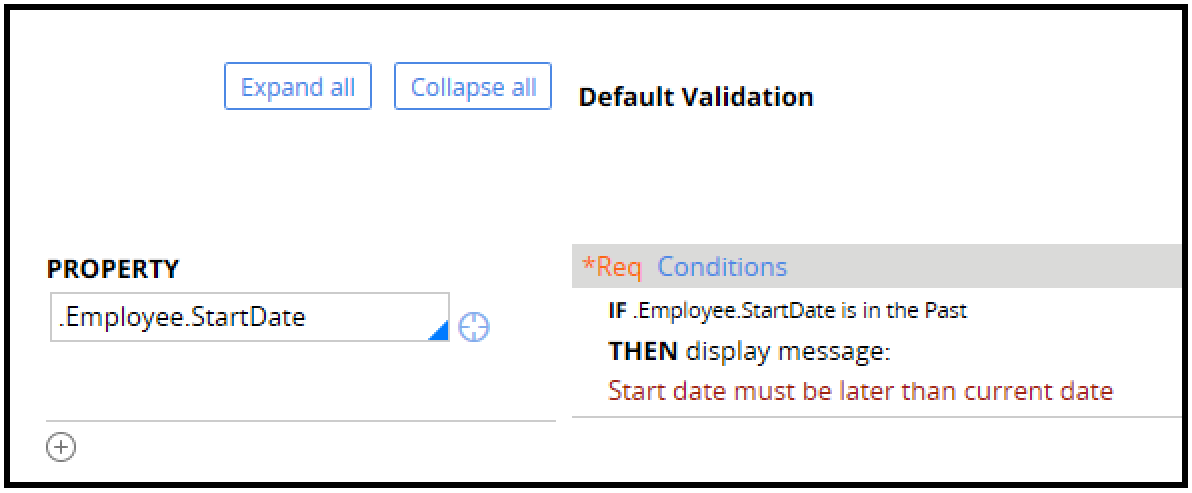
If .Employee.StartDate is in the Past

Message Start date must be later than cu

Continue validation ☑

8. In the dialog, click **Submit**.

The Validation form displays your updates as shown in the following example.



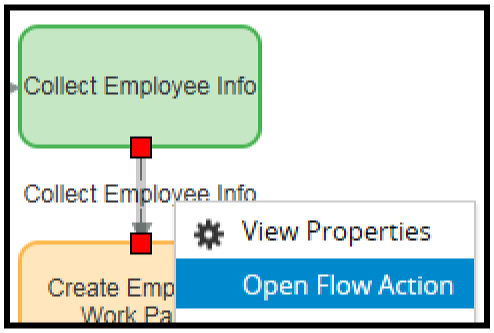
.Employee.StartDate

IF .Employee.StartDate is in the Past
THEN display message:
Start date must be later than current date

9. In the validate form, click **Save.**

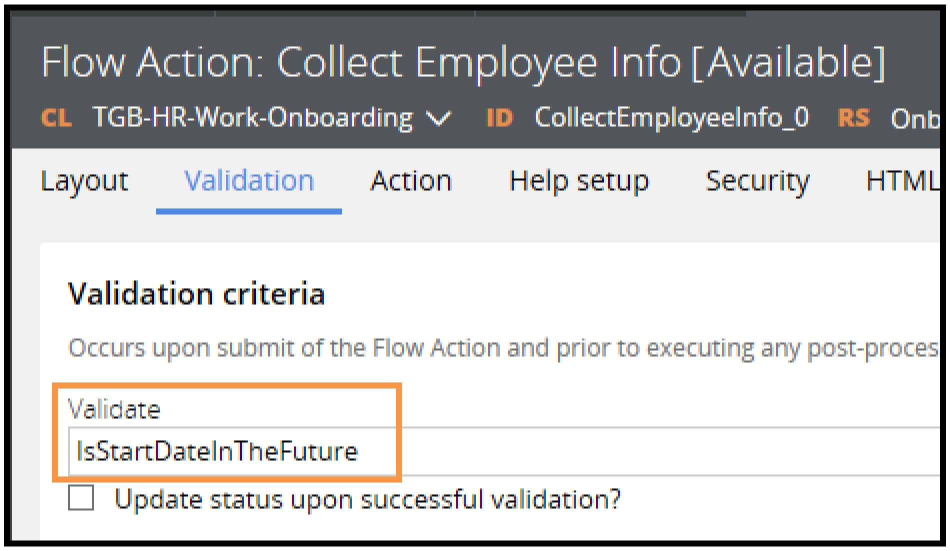
Associating the validate rule with a flow action

1. Open the flow diagram for the process you are updating.
2. Right-click the connector that contains the flow action you want to update and select Open Flow Action. The flow action form opens.



3. On the flow action form, open the Validation tab.

4. In the **Validate** field, select your validation rule IsStartDateInTheFuture.


Flow Action: Collect Employee Info [Available]

CL TGB-HR-Work-Onboarding ID CollectEmployeeInfo_0 RS Onb

Validation criteria

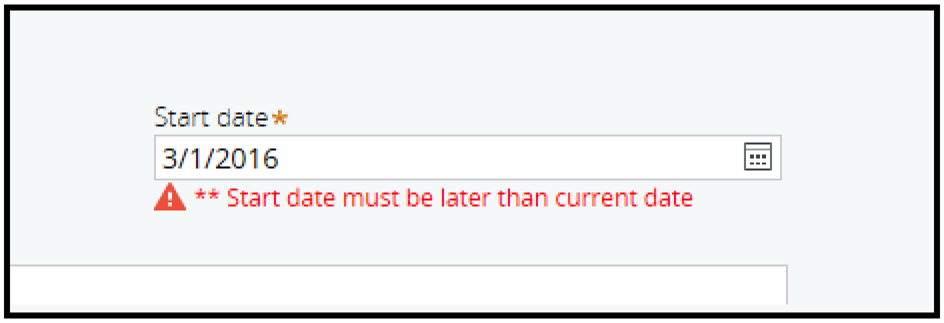
Occurs upon submit of the Flow Action and prior to executing any post-proces

Validate: IsStartDateInTheFuture

☐ Update status upon successful validation?

5. Click **Save**.

   When a user enters a date in the **Start date** field that is not a future date and submits the form, your error message appears as shown in the following example.


Start date*: 3/1/2016

** Start date must be later than current date

Demo: Validating a flow action with a validate rule

This video shows you how to configure a validate rule that ensures the date entered by the user on a form is later than the current date.

Select the Validate rule type in the Application Explorer.

Right-click and select the +Create option.

Enter a name for the validate rule.

Select the property that you want to validate.

Select **Add** to add a validate condition.

Select the function for the validate condition.

Select the property you want to include in the evaluation.

Add a message that appears if the entered value fails validation.

Open the flow rule.

Select the flow action and open the flow action rule.

On the **Validation** tab, select the validate rule.

Create a case to test the validate rule.

In the **Start Date** field, select a date earlier than the current date.

The validation error appears beneath the **Start Date** field.

How to use edit validate rules

Edit validate rules validate character patterns. For example, assume you want users to enter a valid email address in a field using a decimal property type. However, the field cannot verify that the input value contains an "at" @ symbol. You can use an edit validate rule to ensure that the field contains the symbol. If it does not, an error message appears when the form is submitted.

e-mail*
JohnSmith_Pega.com
Enter a valid email address
SUBMIT

When the email address contains the symbol, the error message does not appear.

e-mail*
JohnSmith@Pega.com
SUBMIT

The logic in edit validate rules is written as a Java procedure. Pega provides a large set of the standard edit validate rules so you do not have to create your own rules.

You can associate a single edit validate rule with a property. You can also reference edit validate rules in a validate rule. This approach enables you to apply multiple edit rules to a single property.

- To associate an edit validate rule with a property, open the property rule. The property must be a single value, value list, or value group. Open the Advanced tab and select an edit validate rule in the **Use validate** field.
- To use an edit validate rule with a validate rule, open the validate rule. In the **Select a function** field, select the function Validation of [Property Name] using [Edit Validate Name]. In the **Validation of** field, enter the property you want to validate. Then in the **using** field, select an edit validate rule.

For more information about standard edit validate rules, see the help topic Standard Edit Validate rules.

KNOWLEDGE CHECK

ANSWER You have added a field for entering a U.S. phone number. Do you use a integer data type or an edit validate rule to validate that the phone number is in the correct format?

An edit validate rule ensures that the phone number contains the correct number of digits. The integer data type only ensures that the user enters numbers in the field.

REPORT DESIGN

Creating reports

Introduction to Creating Reports

Good reports provide the information that business users need to make good decisions. Well-organized and comprehensive reports help users take advantage of process information stored in a database. In this lesson, you learn how to use Pega's reporting features to create effective reports that can support almost any business requirement.

After this lesson, you should be able to:

- Identify the types of reports in applications.
- Explain how columns affect the results of reports.
- Explain how filters affect the results of a report.
- Describe the symbolic options for report filtering.
- Explain how summary reports organize columns.
- Create a report to query and present data.

Reports

Pega's reporting capabilities allow you to create reports that provide real-time information and analysis in your application. Business analysts and work managers use reports to assess the performance of the application. Report information can also be used in the application to allow users to review or select items from a list or table while working in an assignment.

When you design reports, you must know specifically what information the user or application needs. You must also know how the information will be used. Business analysts can provide you with specifications and requirements that were created in project meetings.

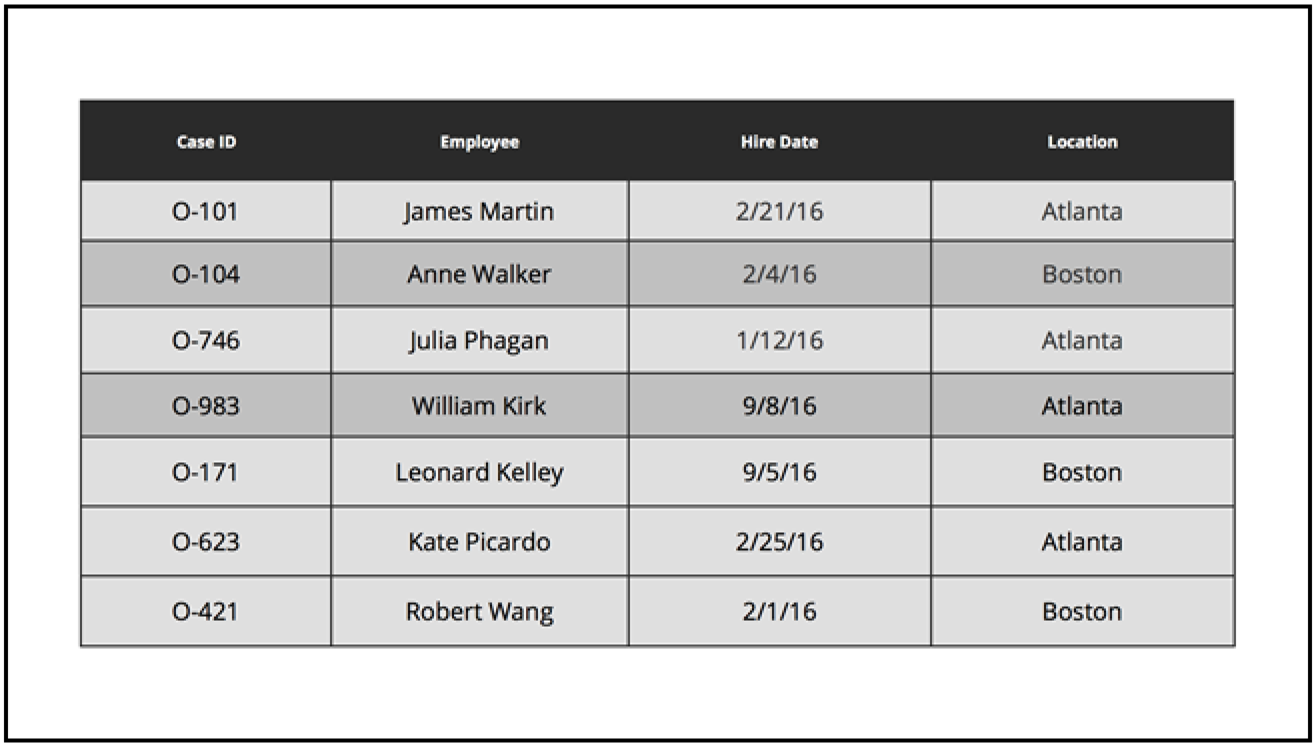
For example, the analysts may want a report that shows the number and dates of resolved cases so managers can monitor process performance. Users who need the reports can get them in their user portal using the Report Browser feature.

In another example, the analysts may need a report that contains a list of customers and their purchase orders. The requirement states that when a customer calls a customer service representative (CSR) to file a complaint, the CSR can view the customer's previous orders on their user forms. You create a report that populates a grid on the user form that displays the purchase history for that customer.

When you have your design requirements, you use a **report definition** to build the report.

Report definitions

Report definitions retrieve records from a database. You use the report definition to specify the data from each record that you want to include in the report. The report definition retrieves the data from a database and returns the results in a table of columns and rows. The rows represent records retrieved from the database. The columns contain the data values in each record that you want users to see. In the following example, a report definition returns the case ID, employee name, employee hire date, and office location for onboarding cases.



Case ID	Employee	Hire Date	Location
O-101	James Martin	2/21/16	Atlanta
O-104	Anne Walker	2/4/16	Boston
O-746	Julia Phagan	1/12/16	Atlanta
O-983	William Kirk	9/8/16	Atlanta
O-171	Leonard Kelley	9/5/16	Boston
O-623	Kate Picardo	2/25/16	Atlanta
O-421	Robert Wang	2/1/16	Boston

Report columns

Report columns define the report's contents. Each column corresponds to a single data element. The value in the column can be a value property such as a case ID, last modified date, or work status. You can format the page in various ways, such as text, currency, or date. For example, you can format currency properties to include a currency symbol.

The following example demonstrates how you would design report definition to support a report request.

Transcript

For example, a human resources application processes cases for onboarding newly hired employees. The organization has three office locations. The facilities manager wants a report to monitor new hires. The manager wants to see information about new employees so that office space can be prepared for them. You design a report in the onboarding class that has four columns. Each column includes a property — case ID, new hire name, date of hire, office location, and salary. When the manager generates the report, the report populates a list of rows with values for each column.

Defining columns in report definitions is critical to designing reports users need.

Functions

You can use functions in columns to make reports more useful. Functions allow you to calculate results derived from data in the database. For example, every new hire is evaluated 30, 60, and 90 days from the start date. A manager wants to see the number of days remaining until each evaluation. The function calculates the difference between the new hire date and the evaluation date.

You cannot display a page in a column. For instance, if you have an employee data object, you can return specific properties on the page, but not an entire page.

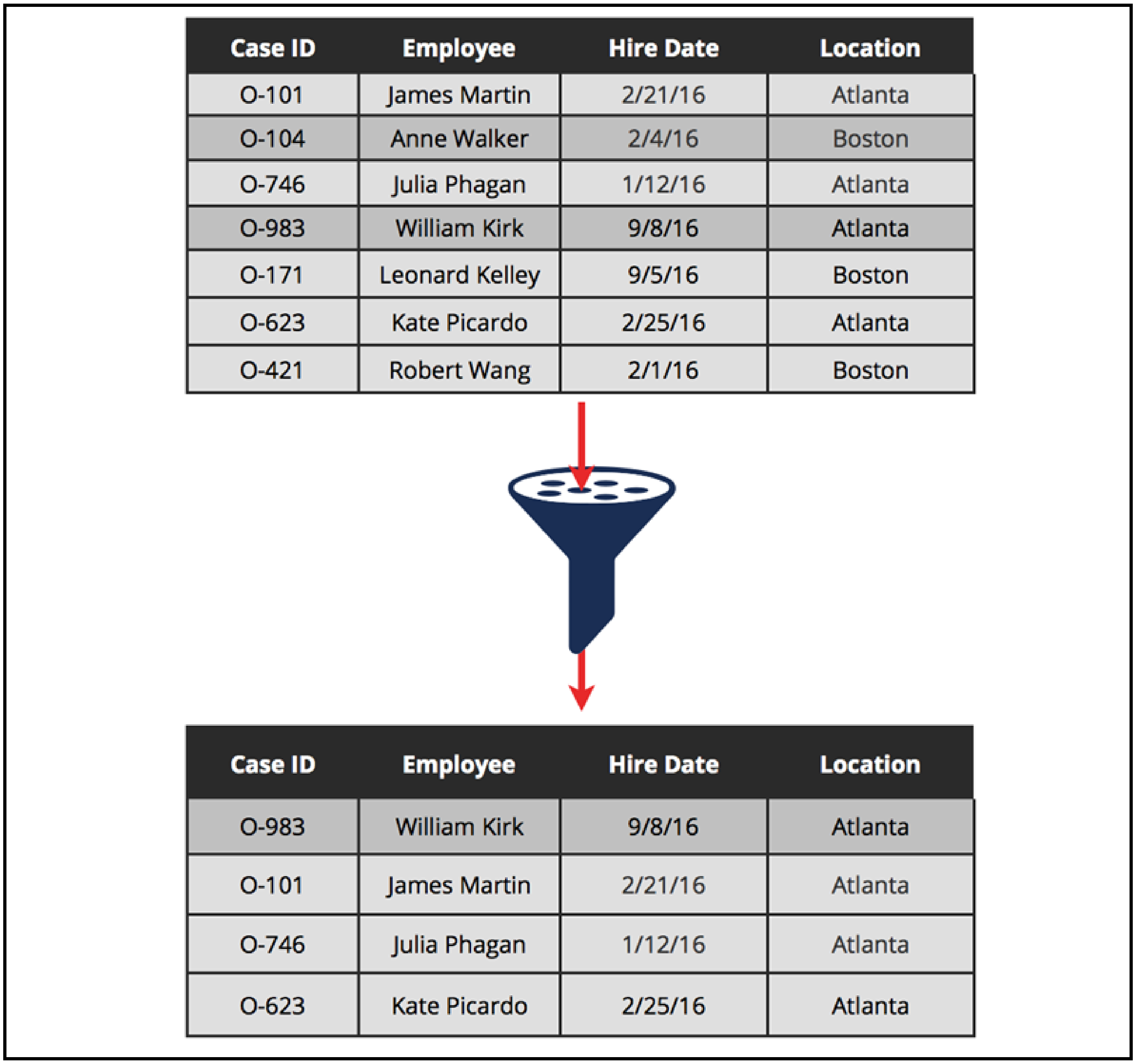
Pega provides many standard functions you can use without having to create or customize functions. The available functions appear in a drop-down list when you open the function option in the report definition.

Report filters

By default, report queries return all the records that contain data from all the columns. You may want to only show records that are relevant to your design requirement. For example, your onboarding application collects information on all new hires. The facilities manager at each location needs a report that shows when the new hire needs a work space. You use report filters to show only the records your users need. In this case, the user needs new-hire start dates in the coming month.

A filter compares a data value in the record against a defined condition. If the comparison result is true, the report includes the record. If the comparison fails the filter tests, the record is not included.

Assume you work for a company with two locations, Atlanta and Boston. You want to create a report of onboarding cases only for the Atlanta office location. You create a filter in the report that tests whether the office location for each onboarding case is Atlanta. When you run the report, the filter returns only cases for the Atlanta office. If the office is in Boston, that office is excluded.



Case ID	Employee	Hire Date	Location
O-101	James Martin	2/21/16	Atlanta
O-104	Anne Walker	2/4/16	Boston
O-746	Julia Phagan	1/12/16	Atlanta
O-983	William Kirk	9/8/16	Atlanta
O-171	Leonard Kelley	9/5/16	Boston
O-623	Kate Picardo	2/25/16	Atlanta
O-421	Robert Wang	2/1/16	Boston

Case ID	Employee	Hire Date	Location
O-983	William Kirk	9/8/16	Atlanta
O-101	James Martin	2/21/16	Atlanta
O-746	Julia Phagan	1/12/16	Atlanta
O-623	Kate Picardo	2/25/16	Atlanta

In the previous example, you use a filter to determine an office location. To create the filter, you define a filter condition in the report definition. A filtering condition is a logical expression that determines whether a record is included in the report.

The comparison can be an explicit value or the value of a property. For instance, if you want to create a report that returns open orders for a customer, you can use the .CustomerName property as the comparison in the filter condition. The returned records show the open orders for each customer. In the previous example, the filter condition uses a comparison that states "office location equals

Atlanta." Therefore, only records that contain the data value Atlanta are included.

You can also use more complex conditions such as testing values that are greater than a specified threshold, like a date. When you use date or date time column data in your filter, you can select time periods using **symbolic dates**. Symbolic dates let you select time periods or dates without having to build functions. For example, you may want to filter all cases created in the previous month. You can select the Previous Month symbolic option rather than write a function to define the period.

Sometimes you may need to create a more complex filter to capture multiple filtering conditions. You can use multiple filters by adding AND/OR conditions. For example, assume you want to filter out cases with a status of Pending and for the manager Anne Walker. You create two filter conditions. One filter states the cases equal status of Pending. The other filter states that the manager equals Anne Walker. Use an AND condition so a record must pass both filters in order to be included in the report.

Report results for business reports or processes

You can integrate report results in two ways. You can allow users to generate business reports for monitoring process performance. You can also integrate the results into business processes so users can generate report information that appears in their user forms.

Processes

You can integrate report results into a business process. For example, you want customers to be able to review their orders from the past six months. You might add a button to the form that allows users to review the order history. When the user clicks the button, the application runs the report and displays a list of previous orders in the form.

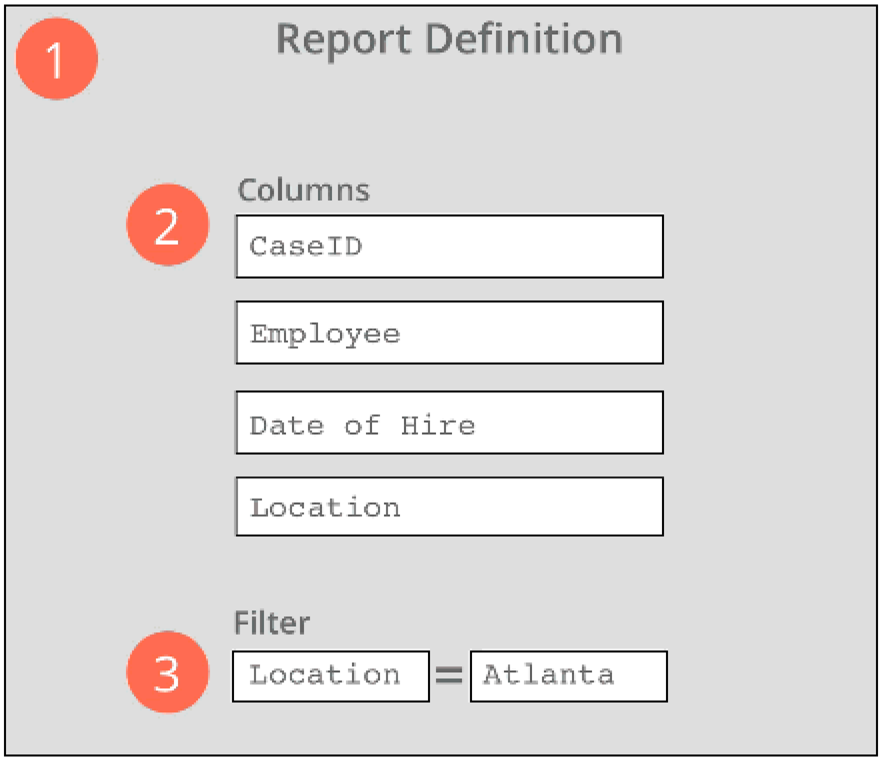
Business reports

You can use the results retrieved by a report definition to build business reports. People use reports to view the status of ongoing or completed work, or to gauge the efficiency of a business process. For instance, managers use reports to monitor how long cases have been in process or to see which assignments have taken the longest to complete.

You can use a setting in the report definition that allows users to generate business reports from their user portal. The portal provides a catalog of reports in a feature called the Report Browser. When you use the setting, the report is available in the Report Browser. The Report Browser organizes reports by category so users can quickly identify and run the report they need. Pega provides standard categories such as Monitor Assignments and Analyze Performances. You can create other categories to help find reports you have designed to support application-specific requirements.

How to create a report

Creating a report is a simple process. First, create the report definition rule. Then, add columns in the report definition form. Finally, add a filter to get the records you want to display.



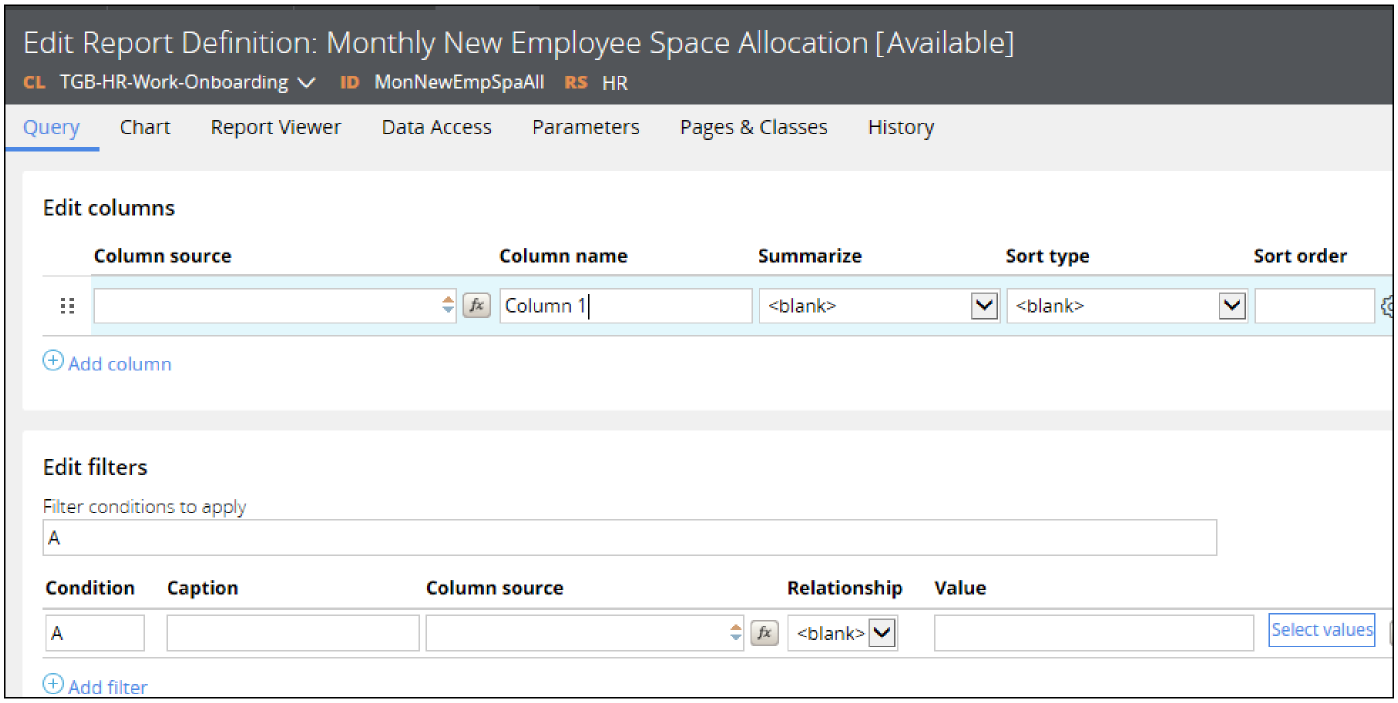
Optionally, you can make the report available to users in the Report Browser.

Step 1: Create a report definition rule

Create the report definition rule in the same class that contains the records you want to report on. If you create the report in the wrong class, you do not get results. For example, if you want to report on cases produced by your application, create the report definition in the application's work class. You can create report definition rules from the Application Explorer by selecting the specific class and using the right-click menu **Create > Reports > Report Definition**.

Give the report a name that clearly describes the report's purpose. For example, use "Monthly New Employee Space Allocation," not "Facilities." This name enables users to find and identify it when they search for the name in the Report Browser.

After you create the report definition, the system displays the rule form as shown in the following example. The Query tab contains sections for specifying columns, creating filters, and creating summary reports.



Edit Report Definition: Monthly New Employee Space Allocation [Available]

CL TGB-HR-Work-Onboarding ID MonNewEmpSpaAll RS HR

Edit columns

Column source	Column name	Summarize	Sort type	Sort order
	Column 1	<blank>	<blank>	

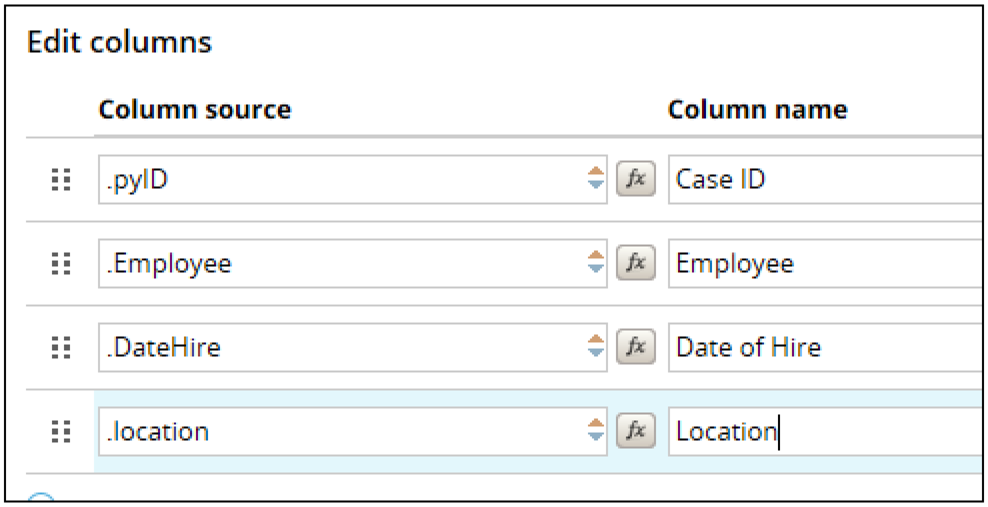
Edit filters

Filter conditions to apply: A

Condition	Caption	Column source	Relationship	Value
A			<blank>	

Step 2: Add columns

In the Edit Columns section of the report definition, add a column for each data element you want to include and select a data element as shown in the following example. The report retrieves from the database values for each of these data elements. Optionally, use functions if you want to calculate values not found in the database. Order the columns in the same order you want them to appear in the report.



Edit columns

Column source	Column name
.pyID	Case ID
.Employee	Employee
.DateHire	Date of Hire
.location	Location

When you generate the report, the results show values for each of the column data elements you defined.

Step 3: Add a filter

You may want to filter the results so the report definition retrieves the information you need. For example, you want to display resolved cases instead of all cases. You create filter conditions in the Edit filters section.

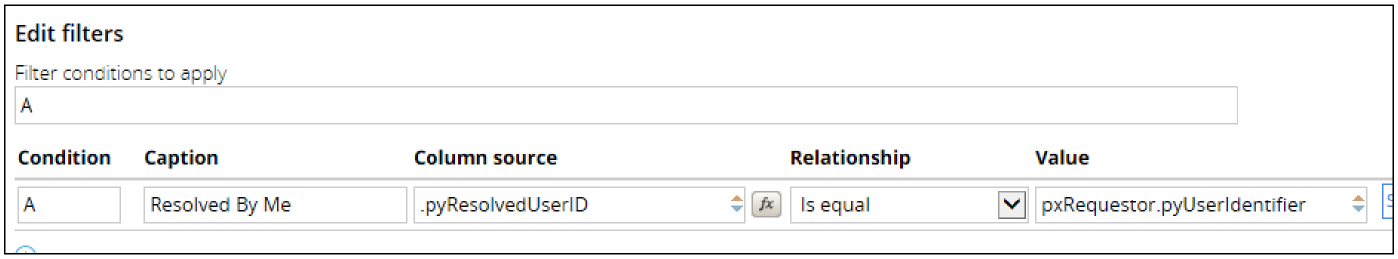
In the Condition field, enter a capital letter to identity the filter. By default, the first filter condition field is "A." If you add filters, give each filter a unique identifier. When you use more than one filter, you can specify AND/OR conditions in the filter conditions field.

Name the filter in the Caption field. This name appears in the report header so users know that the results are filtered.

In the Column source field, select a property reference for the condition.

Identify the data element you want to compare against a defined condition. In the Relationship field, specify a relationship such as equals or greater than. In the Value field, specify the value you want to use in the comparison test.

The following example shows a filter for showing only cases resolved by the user.

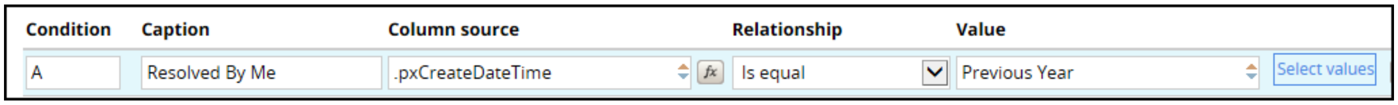

Edit filters

Filter conditions to apply

A

Condition	Caption	Column source	Relationship	Value
A	Resolved By Me	.pyResolvedUserID	Is equal	pxRequestor.pyUserIdentifier

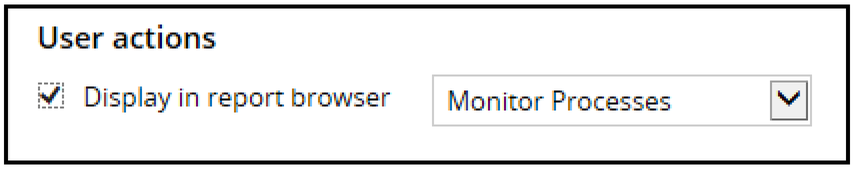
If you use a date time property in the Column source field, you can use the symbolic date feature to select a value. The system calculates specific time periods and identifies them by name. This makes selecting time periods easy. You can also use the feature to select specific dates from a calendar. To use the feature, click **Select values**.


Condition	Caption	Column source	Relationship	Value
A	Resolved By Me	.pxCreateDateTime	Is equal	Previous Year

Make the report available in the Report Browser (optional)

Use a setting in the Report Viewer tab if you want to make the report available in the Report Browser. This setting allows users to generate business reports they can use to monitor application performance.

In the User actions section on the tab, select **Display in report browser**. In the drop-down list next to the setting, select the category the report will be located. Categories group reports according to the type of information the reports contain. Categories enable users to find reports in a user portal Report Browser.


User actions

☑ Display in report browser: Monitor Processes

Creating a report

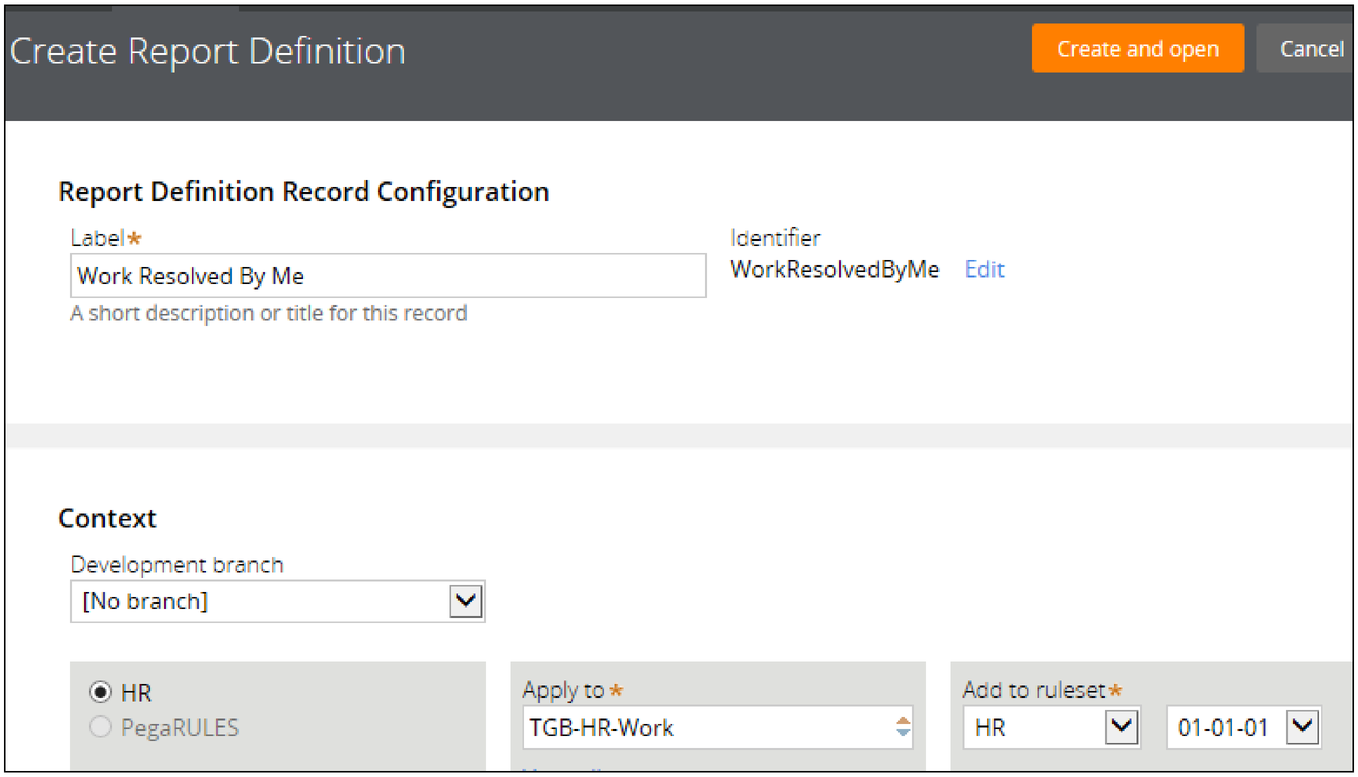
When you create a report, you design the report definition in three major steps: create the report definition rule, add columns, and add a filter. Optionally, you can make the report available in the Report Browser.

Create a report definition rule

Do the following:

1. In the Application Explorer, navigate to the class in which you want to create the report. The class determines the cases the report returns. For example, enter TGB-HR-Work to return any of the cases created in the HR application.
2. Right-click and select **Create > Reports > Report Definition**. The Create Record form is displayed.
3. In the **Label** field, enter the name of the report. For example, to create a report that returns cases resolved the operator running the report, enter Work Resolved by Me.
4. Optional: In the **Add to Ruleset** drop-down list, select the ruleset in which to save the report. By default, Pega selects the highest open ruleset and version for the application.

   The following image shows an example of a completed Create Report Definition form.



Report Definition Record Configuration

Label* Work Resolved By Me
A short description or title for this record

Identifier WorkResolvedByMe Edit

Context

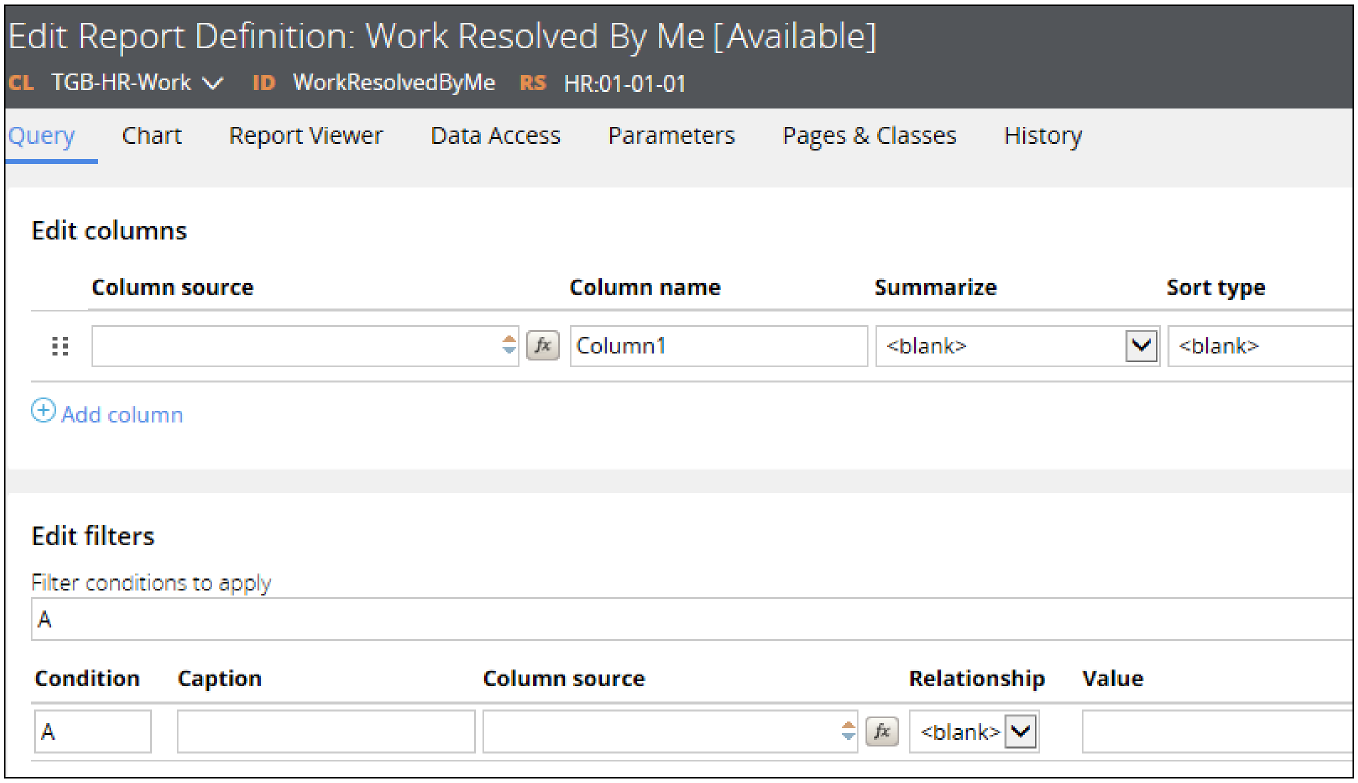
Development branch [No branch]

(●) HR
( ) PegaRULES

Apply to* TGB-HR-Work

Add to ruleset* HR 01-01-01

5. Click **Create and open**. The Report Definition form is displayed as shown in the following image.


Edit Report Definition: Work Resolved By Me [Available]

CL TGB-HR-Work ID WorkResolvedByMe RS HR:01-01-01

Edit columns

Column source	Column name	Summarize	Sort type
	Column1	<blank>	<blank>

Edit filters

Filter conditions to apply

A

Condition	Caption	Column source	Relationship	Value
A			<blank>	

Add columns

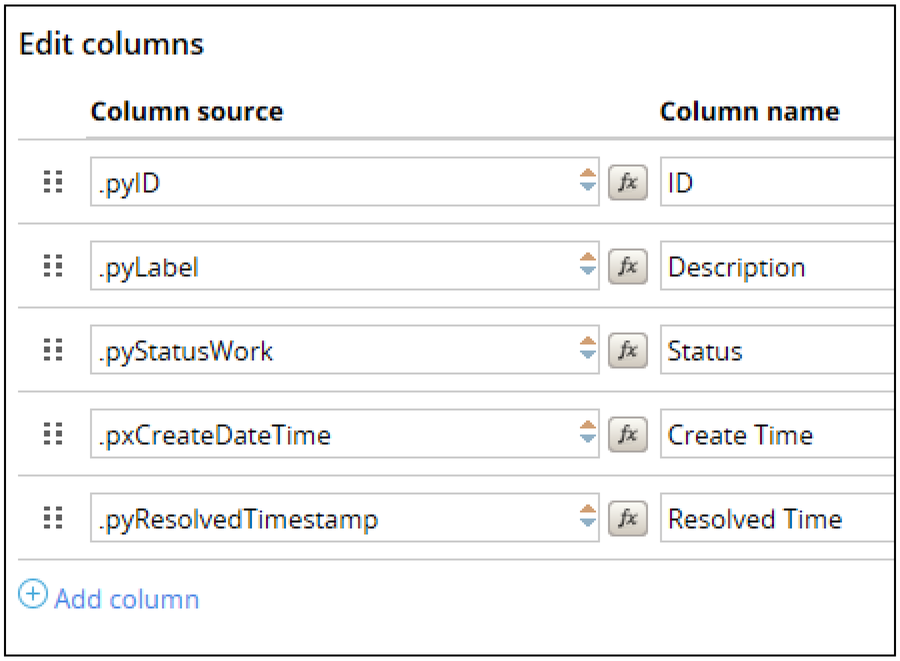
In the **Edit Columns** section on the Query tab, do the following:

1. In the **Column source** field, select the property to add to the report. For example, to display the case ID, enter .pyID. This property identifies cases by their ID values. The property values appear in the first column on the report.

   Note: You can also enter the property label to find the rule you are looking for. For example, you can enter ID to find .pyID.

   Optional: In the **Column name** field, you can keep the default property name or update it. The property name you select in the Column source field appears by default. This name appears as a column header on the report.

2. Click **Add column** to add columns as needed. For example, add columns as shown in the following example.

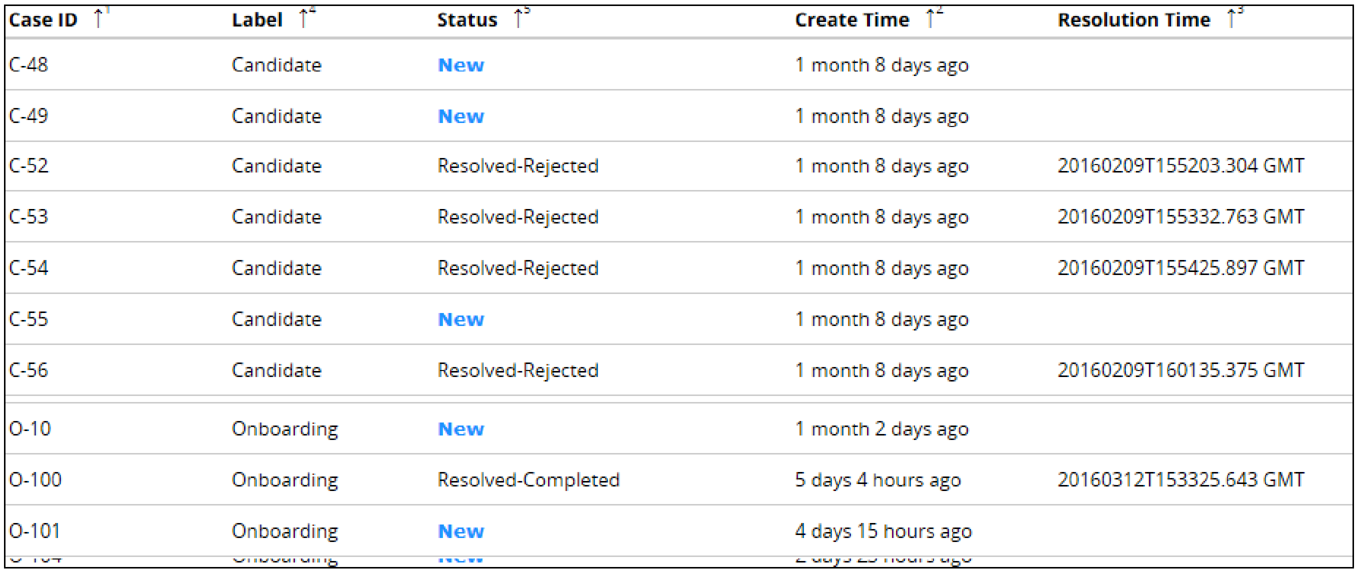


Edit columns

Column source	Column name
.pyID	ID
.pyLabel	Description
.pyStatusWork	Status
.pxCreateDateTime	Create Time
.pyResolvedTimestamp	Resolved Time

3. Click **Actions > Run** in the form header to test the report.

   The following image shows the generated report.



Case ID	Label	Status	Create Time	Resolution Time
C-48	Candidate	New	1 month 8 days ago	
C-49	Candidate	New	1 month 8 days ago	
C-52	Candidate	Resolved-Rejected	1 month 8 days ago	20160209T155203.304 GMT
C-53	Candidate	Resolved-Rejected	1 month 8 days ago	20160209T155332.763 GMT
C-54	Candidate	Resolved-Rejected	1 month 8 days ago	20160209T155425.897 GMT
C-55	Candidate	New	1 month 8 days ago	
C-56	Candidate	Resolved-Rejected	1 month 8 days ago	20160209T160135.375 GMT
O-10	Onboarding	New	1 month 2 days ago	
O-100	Onboarding	Resolved-Completed	5 days 4 hours ago	20160312T153325.643 GMT
O-101	Onboarding	New	4 days 15 hours ago	

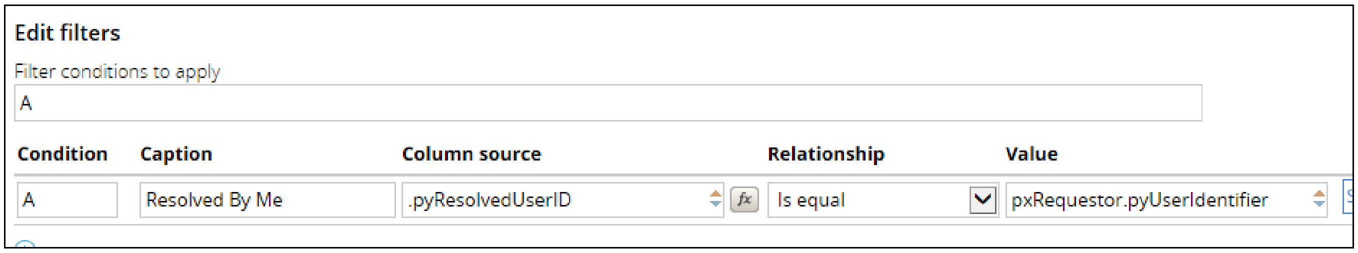
Add a report filter

In the **Edit Filter** section on the Query tab, do the following:

1. In the **Column source** field, select the property to compare in the filter conditions. For example, enter .pyResolvedUserIDto filter cases by the operator who resolved them.
2. In the **Relationship** field, specify the relationship between the source and the comparison value. For example, enter Is equal to only return cases where the source and the comparison match.

3. In the **Value** field, enter the value to compare against. For example, to compare against the operator who runs the report, enter pxRequestor.pyUserIdentifier. This value identifies the user running the report from session data on the Clipboard.

   The following image shows a completed filter.

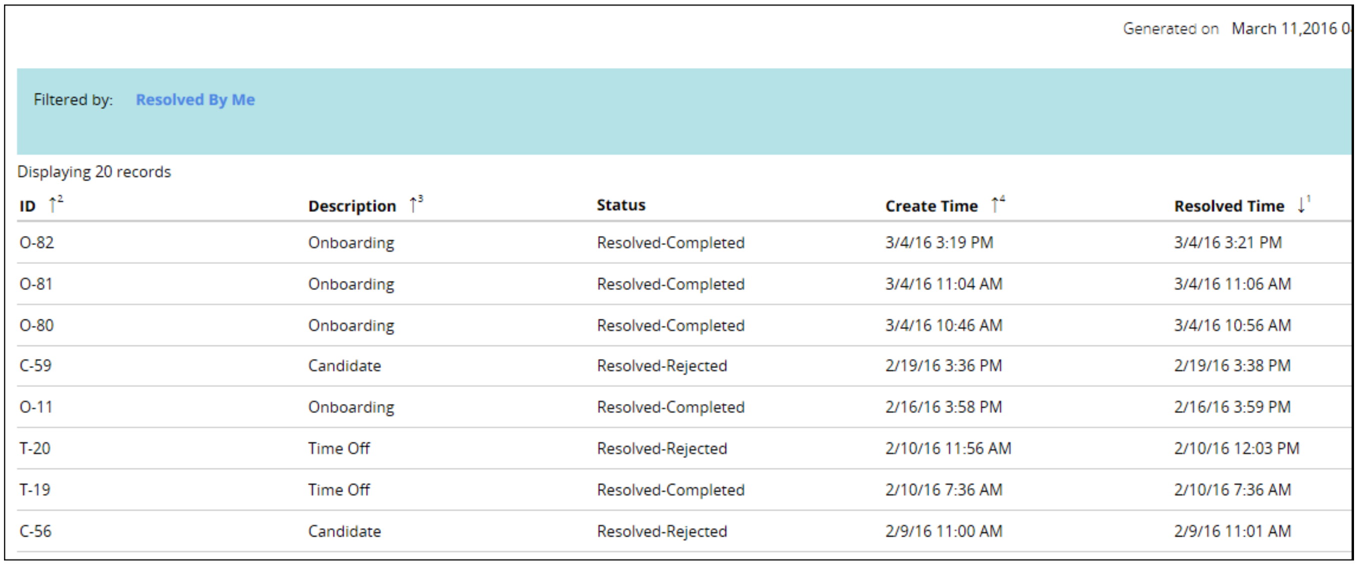


Edit filters

Filter conditions to apply: A

Condition	Caption	Column source	Relationship	Value
A	Resolved By Me	.pyResolvedUserID	Is equal	pxRequestor.pyUserIdentifier

4. Click **Save**.
5. Click **Actions > Run** in the form header to test the report. The following example shows how the report is displayed. Note that all of the results have a status of Resolved. The following image shows the filter name Resolved By Me appears in the report header.



Generated on March 11,2016 0

Filtered by: Resolved By Me

Displaying 20 records

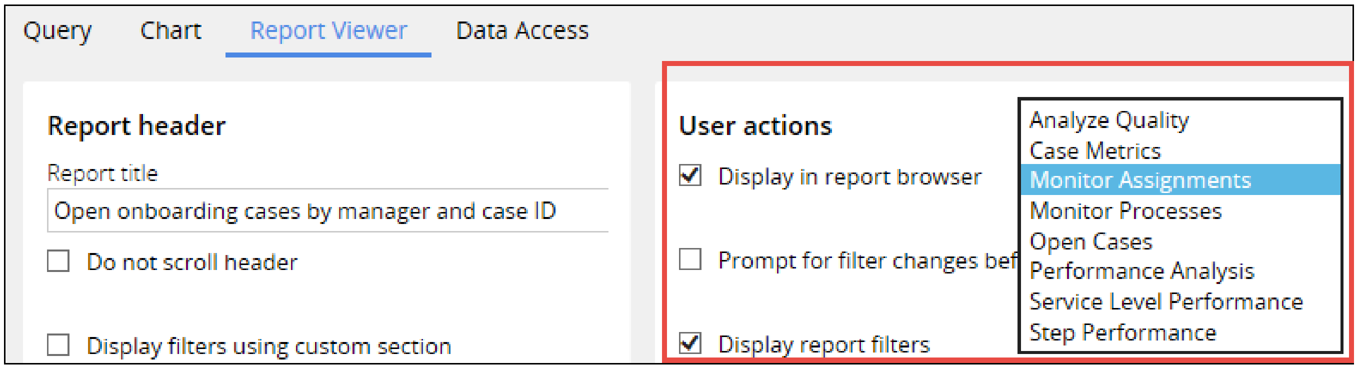
ID	Description	Status	Create Time	Resolved Time
O-82	Onboarding	Resolved-Completed	3/4/16 3:19 PM	3/4/16 3:21 PM
O-81	Onboarding	Resolved-Completed	3/4/16 11:04 AM	3/4/16 11:06 AM
O-80	Onboarding	Resolved-Completed	3/4/16 10:46 AM	3/4/16 10:56 AM
C-59	Candidate	Resolved-Rejected	2/19/16 3:36 PM	2/19/16 3:38 PM
O-11	Onboarding	Resolved-Completed	2/16/16 3:58 PM	2/16/16 3:59 PM
T-20	Time Off	Resolved-Rejected	2/10/16 11:56 AM	2/10/16 12:03 PM
T-19	Time Off	Resolved-Completed	2/10/16 7:36 AM	2/10/16 7:36 AM
C-56	Candidate	Resolved-Rejected	2/9/16 11:00 AM	2/9/16 11:01 AM

Make reports available in the Report Browser (Optional)

In the **User actions** section on the Report Viewer tab, do the following:

1. Select the **Display in report browser** check box. When you select the check box, a drop-down field appears next to the check box.

2. In the drop-down field, select a report category as shown in the following image.



Report header

Report title: Open onboarding cases by manager and case ID

☐ Do not scroll header

☐ Display filters using custom section

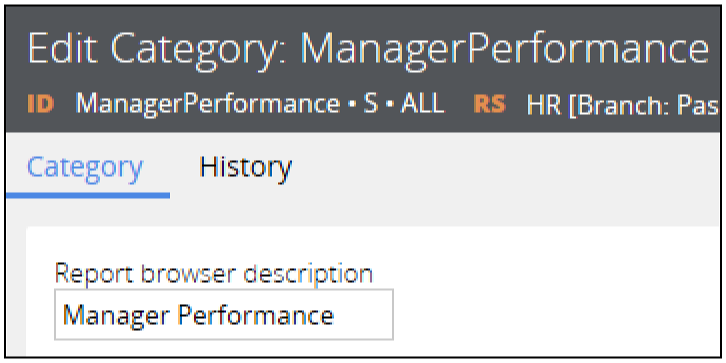
User actions

☑ Display in report browser

☐ Prompt for filter changes bef

☑ Display report filters

You can add your own categories by creating a Category rule as shown in the following example.



Edit Category: ManagerPerformance

ID ManagerPerformance • S • ALL RS HR [Branch: Pas

Report browser description: Manager Performance

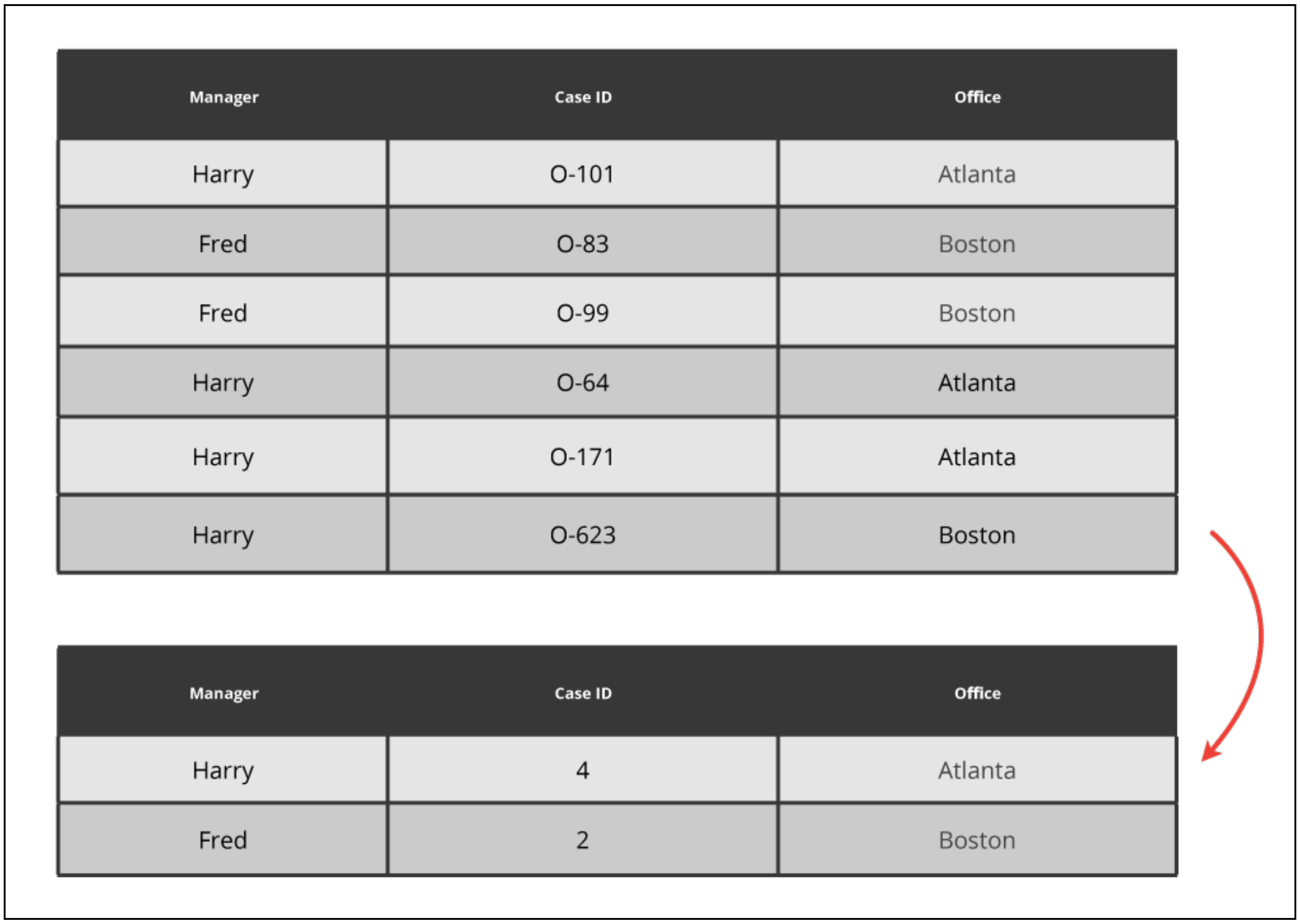
Report results organization

The way you format report results helps users easily find and analyze specific information. You can group and summarize the information for effective business presentations. Additionally, you can display reports as lists, charts, or graphs to create impact.

Summarizing results

Summarizing reports is useful when users must analyze a large amount of data. While list reports contain the detailed information a user needs, summary reports allow users to quickly identify the key statistics they are looking for.

When you generate a report, Pega returns the results as a list of records. You can convert list reports to summary reports, which summarize one or more columns to calculate counts, totals, or averages. For example, a user may ask for a count of cases handled by the manager who created them. A list report could show the same data but would not provide the summary counts that the user is looking for.

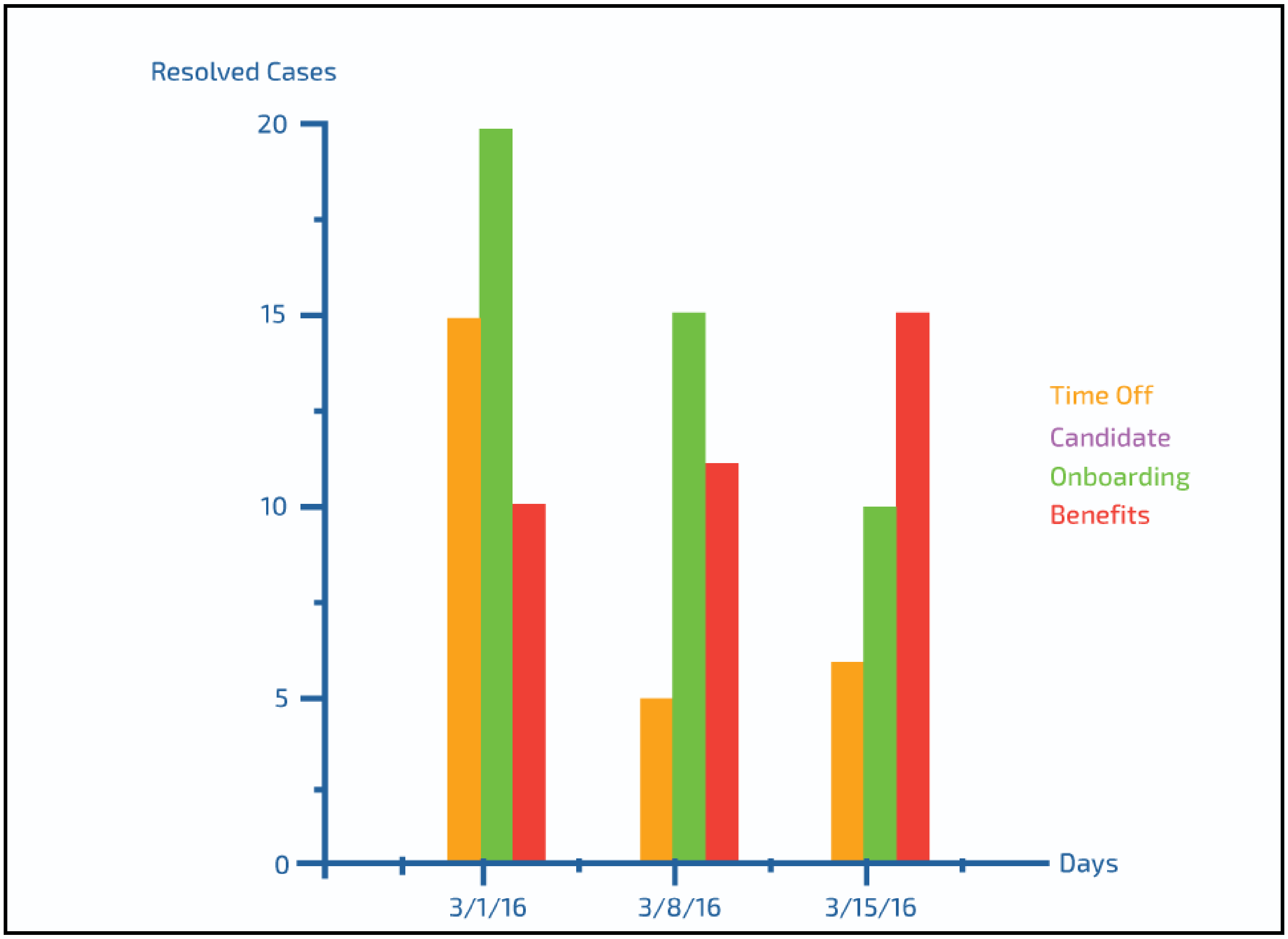


Manager	Case ID	Office
Harry	O-101	Atlanta
Fred	O-83	Boston
Fred	O-99	Boston
Harry	O-64	Atlanta
Harry	O-171	Atlanta
Harry	O-623	Boston

Manager	Case ID	Office
Harry	4	Atlanta
Fred	2	Boston

Visualizing summary results

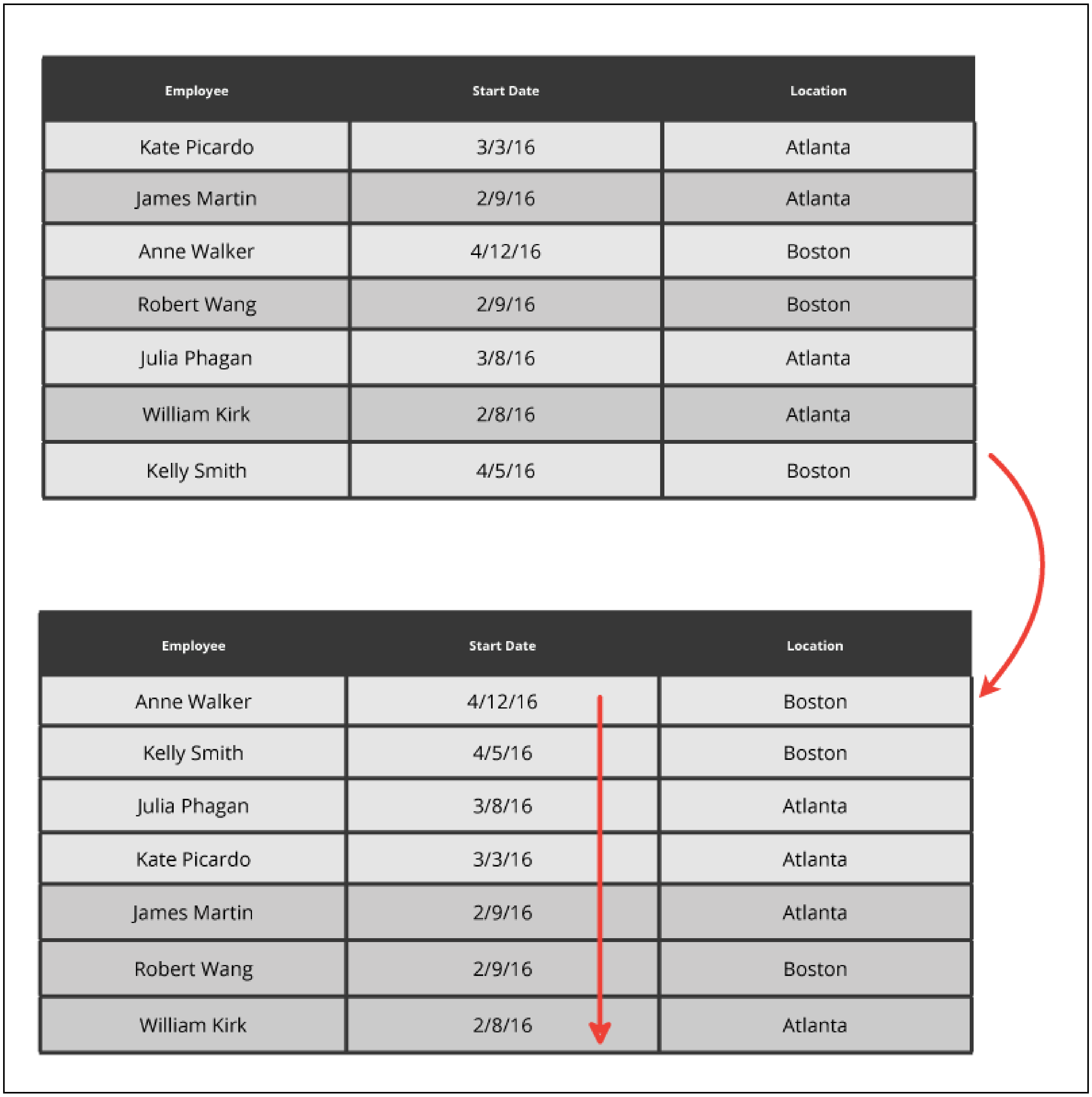
Visual presentation of summary data can be effective for business analysts who want to quickly review and analyze the data. You can display summary report data in charts or graphs. Using charts and graphs help users identify trends or statistics quicker than sorting through lists of data. For example, you can create a line graph that shows the number of cases resolved for each case type over a number of weeks.



Sorting values in the columns

Users who want to see a sequential ordering of data can sort values in ascending or descending order. Text characters are sorted alphabetically, and numbers are sorted numerically. You can control which column is sorted first by specifying the sort order.

For example, the facilities manager wants sort the new hires by start date so that the manager can prepare the office space. Assume the report orders the columns as Employee, Start Date, and Location. You select the start date as the column you want to sort first. You then specify the column as the first one in the sort order. When you generate the report, the employee with the most current date appears at the top of the list. The other employees are listed in descending order according their start dates.

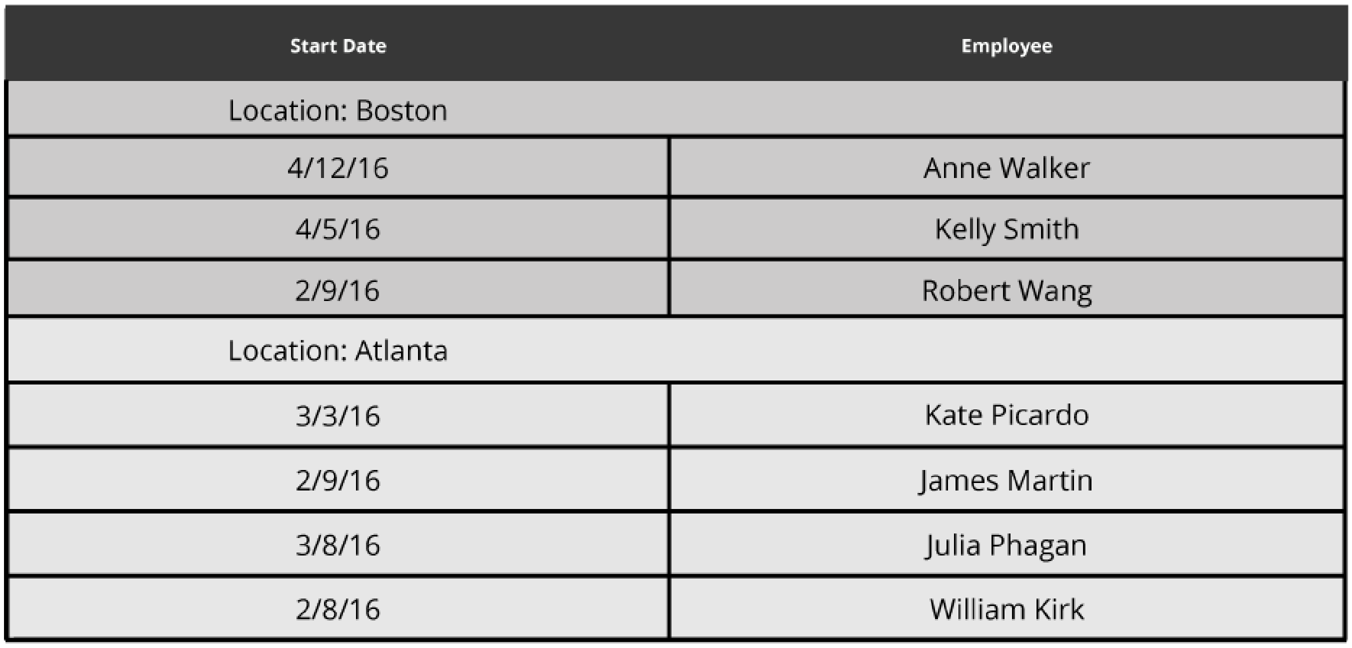


Employee	Start Date	Location
Kate Picardo	3/3/16	Atlanta
James Martin	2/9/16	Atlanta
Anne Walker	4/12/16	Boston
Robert Wang	2/9/16	Boston
Julia Phagan	3/8/16	Atlanta
William Kirk	2/8/16	Atlanta
Kelly Smith	4/5/16	Boston

Employee	Start Date	Location
Anne Walker	4/12/16	Boston
Kelly Smith	4/5/16	Boston
Julia Phagan	3/8/16	Atlanta
Kate Picardo	3/3/16	Atlanta
James Martin	2/9/16	Atlanta
Robert Wang	2/9/16	Boston
William Kirk	2/8/16	Atlanta

Grouping results

Grouping results helps users easily analyze trends or statistics found on large reports. You can group report results so that records are grouped in a column you specify. When you group results, the grouped column values appear once for each group. The rows that contains the column value are listed under the group by value. For example, you want to show onboarding cases for each office location. You specify office location as the one you want to group. When you run the report, each location name appears once in the left column. The cases are grouped in rows next to the location name.



Start Date	Employee
Location: Boston	
4/12/16	Anne Walker
4/5/16	Kelly Smith
2/9/16	Robert Wang
Location: Atlanta	
3/3/16	Kate Picardo
2/9/16	James Martin
3/8/16	Julia Phagan
2/8/16	William Kirk

Organizing report results

You built your basic report by adding columns and filtering the results. You can organize the results so that users can easily find and analyze the information in the report.

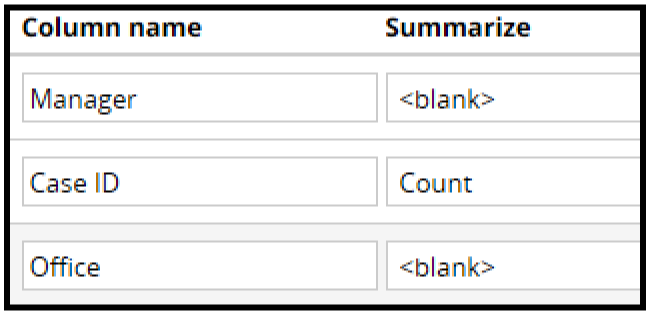
You can organize report results in three ways:

- Summarize the values in one or more columns. When you summarize report results, you can also graphically display the summary results.
- Group the results under a column that you specify.
- Sort values in columns.

Summarize the results

To create a summary report, do the following:

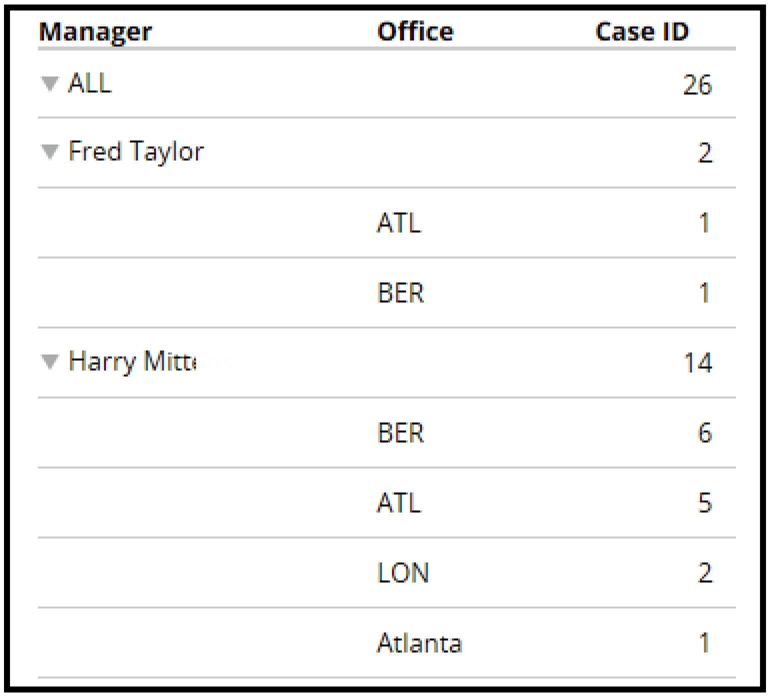
1. In the Edit Columns section, identify the data element column you want to summarize.
2. In the Summarize column, select how you want the to summarize the column. If the column contains text values, you can summarize by count or sum. If the column contains numeric values, you can summarize them by count, minimum, maximum, average, or sum. In the following example, you are summarizing the Case ID by count.



Column name	Summarize
Manager	<blank>
Case ID	Count
Office	<blank>

3. In the form header, click **Actions > Run** to test the report.

The results show the count of cases for each of the managers.

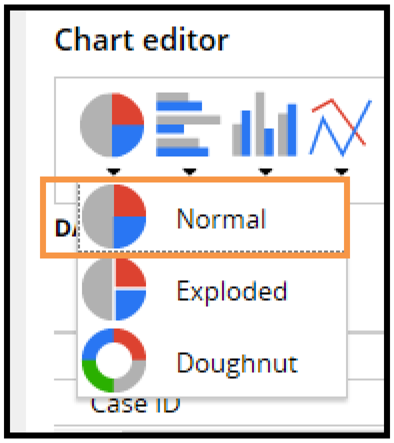


Manager	Office	Case ID
ALL		26
Fred Taylor		2
	ATL	1
	BER	1
Harry Mitt		14
	BER	6
	ATL	5
	LON	2
	Atlanta	1

Graphically display summary results

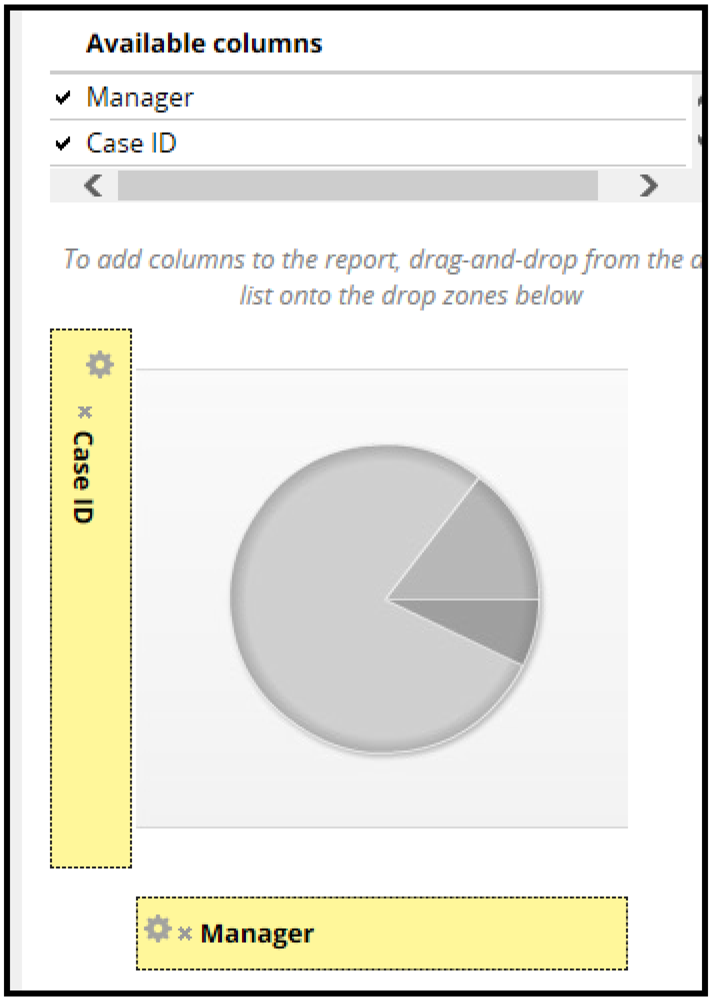
To display a summary report in a chart or graph, do the following:

1. Open the **Chart** tab.
2. In the Chart editor section, select **Include Chart.**
3. In the Chart editor, select the type of chart you want. In the following example, you choose a pie chart.



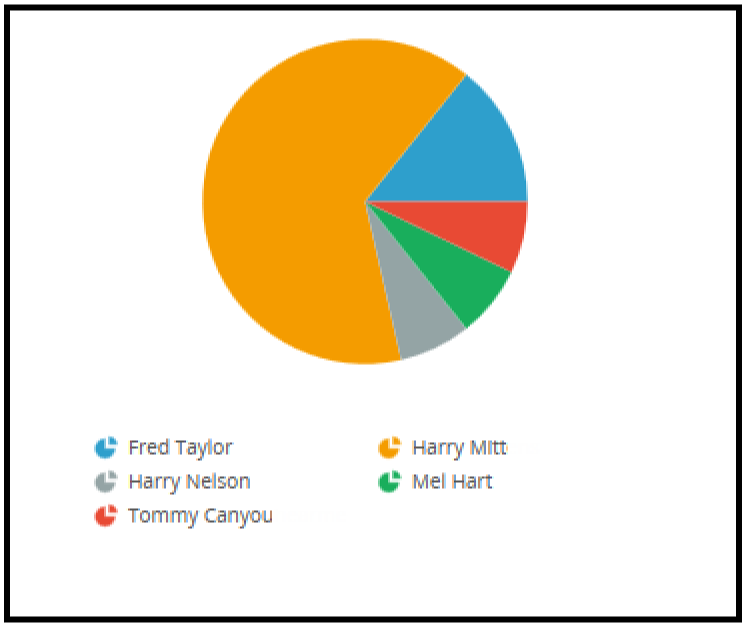
4. From the Available columns list:
   a. Select the column you summarized and drag it to the vertical drop-zone.
   b. Select the column you want to measure and drag it to the horizontal drop-zone.

      In the following example, you design a pie chart that groups in each section cases by manager.


Available columns
✓ Manager
✓ Case ID
To add columns to the report, drag-and-drop from the a
list onto the drop zones below
× Case ID
× Manager

5. In the form header, click **Actions > Run** to test the report.

   The chart looks like the following when you run the report.

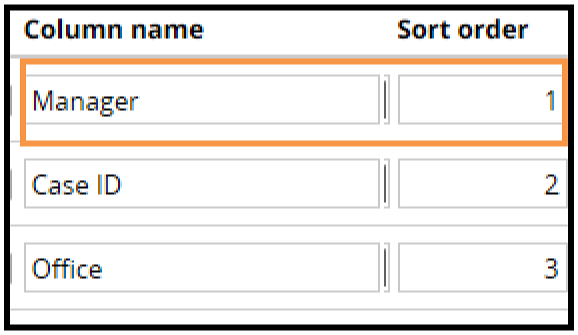


Grouping the results

Note: You cannot group summary report results.

To group report results, do the following:

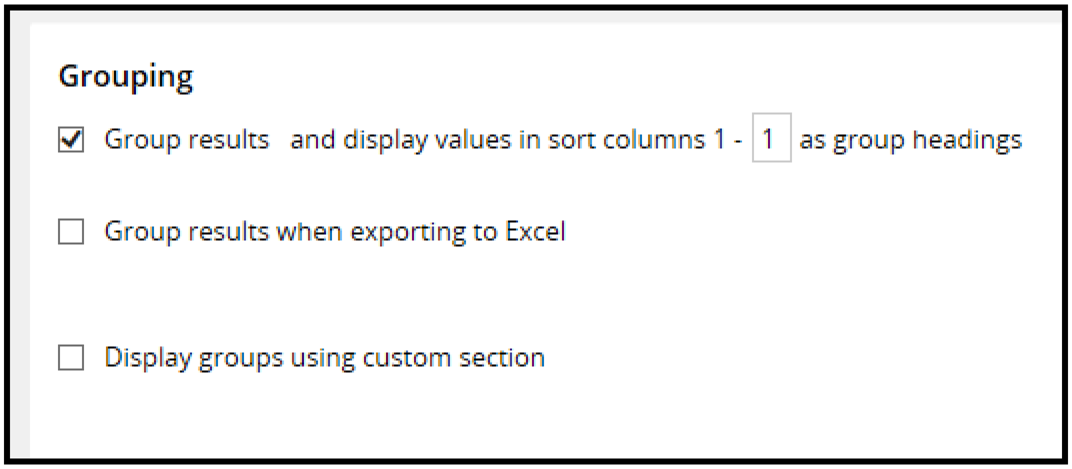
1. On the Query tab, identify the column you want to group the results under and enter 1 in the Sort order column.



Column name	Sort order
Manager	1
Case ID	2
Office	3

2. Open the **Report Viewer** tab.

3. In the Grouping section, select **Group results** .



Grouping

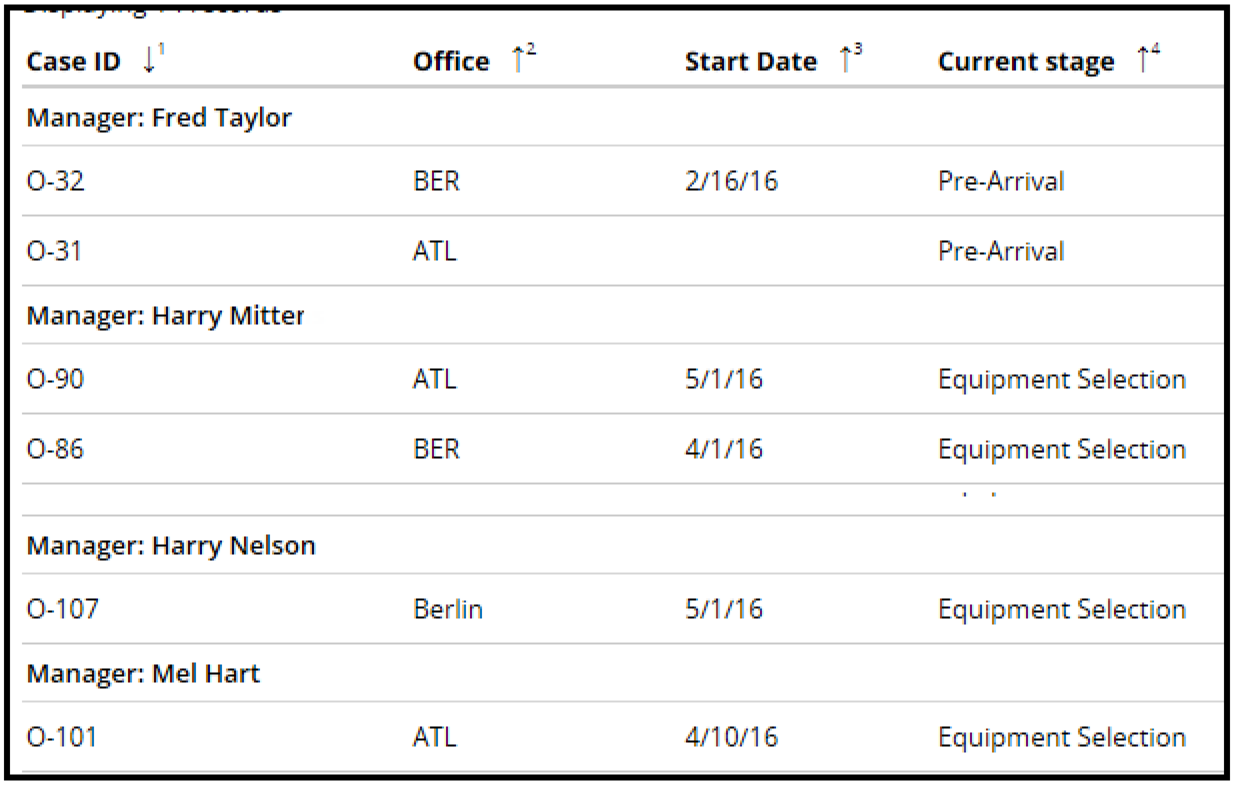
☑ Group results and display values in sort columns 1 - 1 as group headings

☐ Group results when exporting to Excel

☐ Display groups using custom section

4. In the form header, click **Actions > Run** to test the report.

   The following image shows the results.

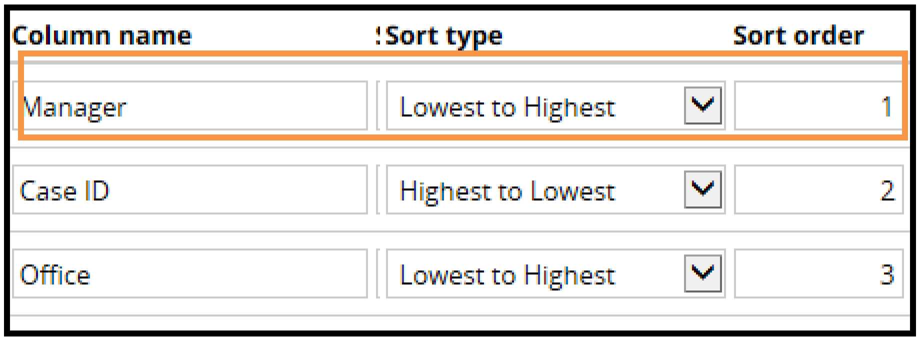


Case ID ↓1	Office ↑2	Start Date ↑3	Current stage ↑4
Manager: Fred Taylor			
O-32	BER	2/16/16	Pre-Arrival
O-31	ATL		Pre-Arrival
Manager: Harry Mitter			
O-90	ATL	5/1/16	Equipment Selection
O-86	BER	4/1/16	Equipment Selection
Manager: Harry Nelson			
O-107	Berlin	5/1/16	Equipment Selection
Manager: Mel Hart			
O-101	ATL	4/10/16	Equipment Selection

Sort values in a column

To sort values in a column, do the following:

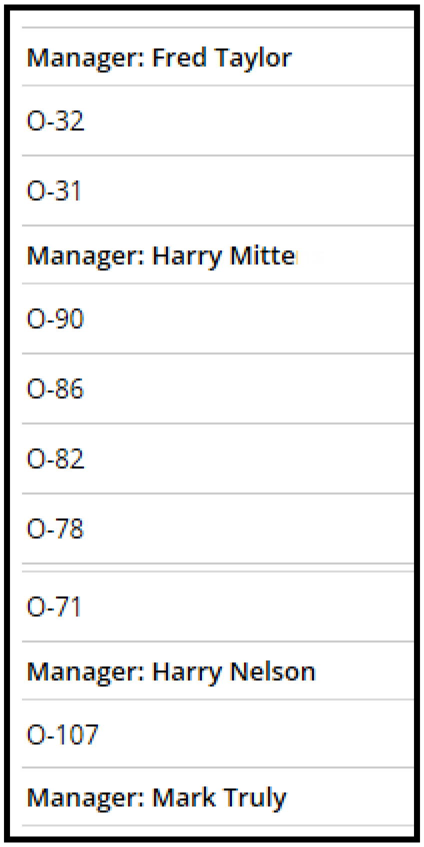
1. Open the **Query** tab.
2. In the Sort order column, enter **1** in the first column you want sorted. Enter **2**in the next column you want sorted, and so on. After column 1 is sorted, then column 2 is sorted.
3. In the Sort type column, select the order to sort the values. The following example sorts the manager name alphabetically starting with the letter A first. Then, the Case ID values are sorted, starting with the highest value.



Column name	Sort type	Sort order
Manager	Lowest to Highest	1
Case ID	Highest to Lowest	2
Office	Lowest to Highest	3

4. In the form header, click **Actions > Run** to test the report.

   The following example shows the results. In the rows under the first column, Manager, the first letter in the first name begins with "H" and ends with "M" — lowest to highest. The rows under the second column, Case ID, starts at the highest number for each manager.



Manager: Fred Taylor
O-32
O-31
Manager: Harry Mitte
O-90
O-86
O-82
O-78
O-71
Manager: Harry Nelson
O-107
Manager: Mark Truly

Optimizing report data

Introduction to Optimizing Data

Pega applications allow system architects to improve report performance through a process called optimization. Optimizing properties allows Pega to extract report data without the need to open each case.

In this lesson, you learn how Pega stores case data and how data storage affects report performance. You also learn how to optimize case data to improve report performance for users.

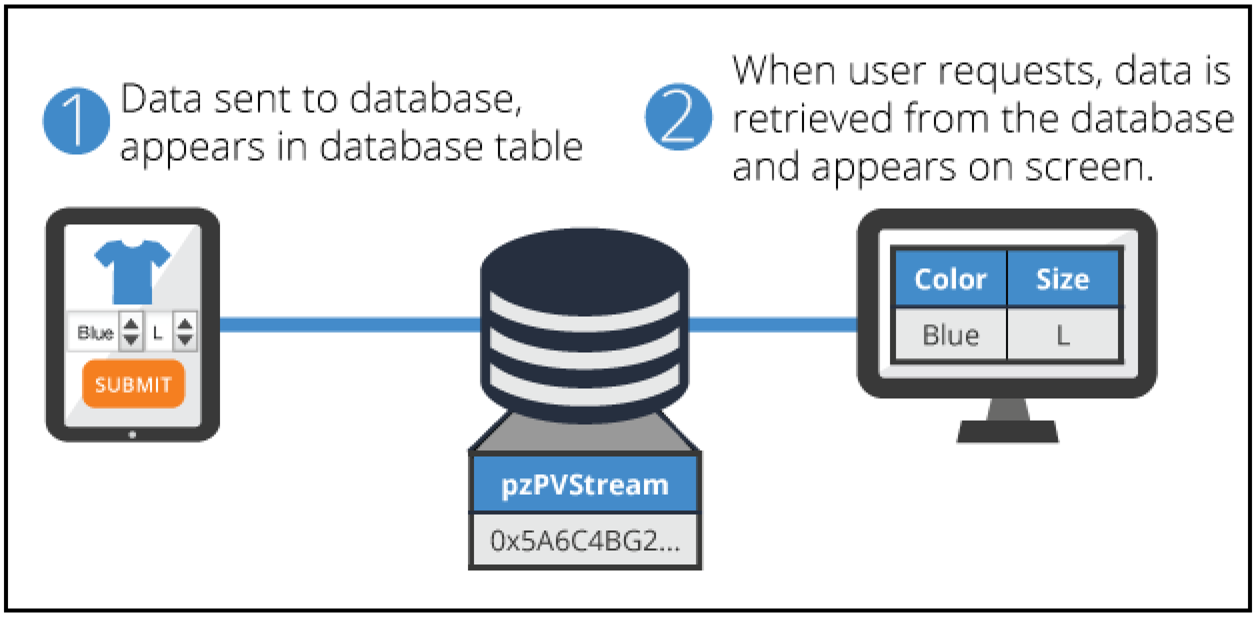
After this lesson, you should be able to:

- Explain the impact of property optimization on report performance.
- Describe how Pega stores case data.
- Explain how property optimization affects properties.
- Optimize case data for reports.

Data Storage in Pega applications

Pega applications store each case as a unique record in a relational database. Within each record, Pega stores case data in a **binary large object** (BLOB) field. Each time an end user completes an action by clicking OK or Submit, Pega writes the case data to the BLOB field.

Within the database, each record is a row in a database table. Each column in the table displays the contents of a field from the case record, including the BLOB field. When an end user opens a case, Pega locates the record in the correct table, then reads the contents of the BLOB column to extract the case data.


1 Data sent to database, appears in database table
2 When user requests, data is retrieved from the database and appears on screen.
Blue
L
SUBMIT
pzPVStream
0x5A6C4BG2...
Color
Size
Blue
L

A BLOB field offers three advantages for storing case data:

- Unlimited storage size — BLOB fields are not constrained by size, so a BLOB field can contain any amount of information.
- Flexibility — Pega writes all case data to the BLOB, so changes to the data model of a case are contained within the BLOB. Using the BLOB field avoids the need to update the database structure, or schema, as the data model changes.
- High performance — Since the BLOB field stores all case data, Pega reads and writes the entire case at one time. This optimizes application performance for end users as they create and process cases.

Using the BLOB penalizes performance for reporting. When an end user runs a report, an application must decompress the BLOB to extract the required data. For a large number of cases, this process significantly increases the time needed to run the report.

KNOWLEDGE CHECK

ANSWER

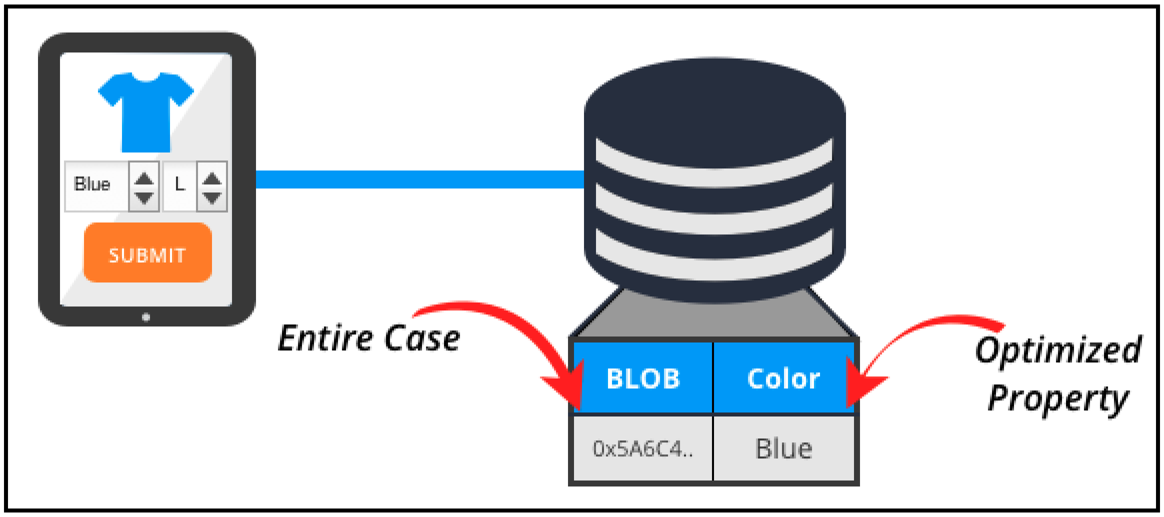
Why is case data stored in a BLOB column?

The BLOB field provides greater flexibility and performance for case data. The BLOB field is capable of storing an unlimited amount of data. The BLOB field also allows the data model of a case to change without impacting database storage. And the BLOB field allows an application to read or write the entire case at once.

Property optimization

End users rarely need to retrieve an entire case when running a report. Rather, end users only need the data elements required by the report. Extracting data from a BLOB impacts performance compared to reading property values from a database table. This impact is most pronounced when extracting data for report filters and sorting or grouping the content of a column.

To improve report performance, Pega offers a hybrid data storage model. This model allows applications to store data both in dedicated data fields and in a BLOB field. To store property values in a data field, you must **optimize** the property for reporting.



When you optimize a property, Pega creates a column for the property in a database table. Because the value of the property is then visible in the table, or exposed, optimizing a property for reporting is also called "exposing" the property.

When a case uses an optimized property, Pega writes data to both the property field and the BLOB field. When a report uses optimized data, Pega reads from the property column, rather than the BLOB. By not decompressing the BLOB field to read case data, optimization reduces the time and memory needed to run the report.

By default, Pega optimizes properties that store process data such as:

- The creation date of a case
- The status of a case
- The case ID

Properties that store process data begin with the letters px, py, or pz.

Properties created by system architects to store business data are not optimized by default. Reports that use an unoptimized property display a warning that states the potential impact on performance. Performance warnings due to an unoptimized property are resolved by running the Property Optimization tool.

KNOWLEDGE CHECK

ANSWER Why are properties exposed, or optimized, for reporting?

Exposing properties allows Pega to read the property value without decompressing the BLOB to extract the property value.

Optimizing properties for reporting

Pega provides the Property Optimization tool to optimize properties for reporting.

When you use the Property Optimization tool on a property, Pega exposes the property as a database column, and populates the new column by extracting values from the BLOB column. For an embedded property in a page group or page list, the tool automatically creates a new database table for the property, and the appropriate Index- class and Declare Index rule to update the new table.

To optimize a property:

1. Using the Application Explorer, expand the Data Model node.
2. Right-click a property name and click **Optimize for reporting**.
3. Select the tables in which you want to create a dedicated database column for this property. You must select at least one table.
4. If the property is embedded, specify the ruleset and version that is to contain the new Index- class, the properties in that class, and the Declare Index rule.
5. Select whether to optimize the property now or later. If you select later, click the calendar icon to select a date within seven days of the current date to optimize the property.
6. Click **Next**.
7. Review the tables to which the property column will be added and the classes that will be mapped to each table.
8. Click **Next**.
9. Click **Finish**.

Note: Background processing may take minutes or longer, depending on volume. Computations in your applications involving the property value may fail or produce incorrect results until all background processing is complete.

You can view the status of your background job by clicking **Designer Studio > System > Database > Column Population Jobs**.

You can view the classes for which a property has been optimized on the **Advanced** tab of the Property record.

DATA MANAGEMENT

Caching data with data pages

Introduction to caching data with a data page

A data page loads data into memory and stores it on the clipboard, making the data accessible to an application. By storing the information in a data page, you improve application performance since the data remains on the clipboard for subsequent access. In addition to improving performance, data pages also increase maintainability of an application by abstracting the data source.

After this lesson, you should be able to:

- Identify the role of data pages in managing data.
- Explain how data pages abstract data from the source.
- Explain the life cycle of a data page.
- Explain the affects of the scope setting of a data page.
- List the sourcing options for a data page.
- List the options for refreshing a data page.
- Configure a data page.

Data pages

When you create and process a case, you need data. Some data is collected from the user as part of the case process, while other data is retrieved from the application or from external systems. For example, if you want to see the claim history for a customer in a claims application, you retrieve application data. If you need to display customer data held in a system of record, you retrieve data from an external system.

You use a **data page** to retrieve data for your application, regardless of the source. Data pages cache data on demand to a clipboard page and have a scope. The **scope** defines who can access the data page. You can make the data retrieved accessible for all applications, or limited to the logged in user or specific case only.

Every data page defines a **refresh strategy**. The refresh strategy defines when the data page is considered stale and needs to be reloaded. Data pages are created and updated on demand. Even if a data page is considered stale, the page is never reloaded until it is referenced. The available refresh strategy options depend on the scope of the page.

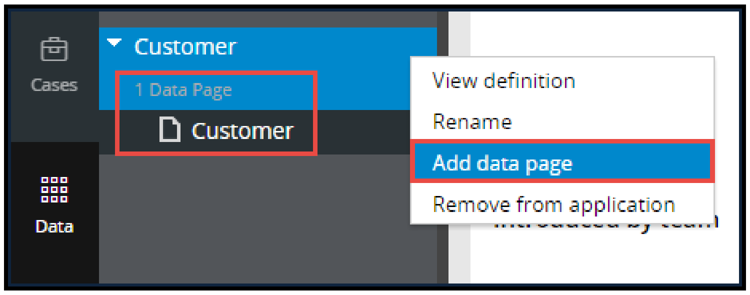
Data pages can use a variety of **sources** to load data. A data page provides an abstraction between the application and data layers. This means that you can use the data page in your application without knowing the data source. Your application configuration does not need to change if the source does. For example, you may configure a data page to look up customer data from a database table. If the interface to the customer data changes to a SOAP web service, you only need to change the data page source and not the application code.

How to configure a data page

Before you create a data page, remember to answer the following questions:

- What is the structure of the data page?
- What is the scope of the data page?
- How is the data page sourced?
- When does the data become stale?

You create a data page for a data type from the Data Explorer. The Data Explorer also shows data pages already defined for the data type.



A **data page rule** defines the structure, scope, source, and refresh strategy of the cached data.

First, you need to specify the structure of your data page. The structure is either page or list. Choose page if you want to load a single record (such as a single customer). Choose list if you want to load multiple records (such as a list of insurance claims filed by a customer). The Object type defines the class of the page or pages in the list.

A data page is typically read-only, but you can set the Edit mode to editable if you need to update the data page after it has been loaded. Updates to the data page are only local and not propagated back to the source.

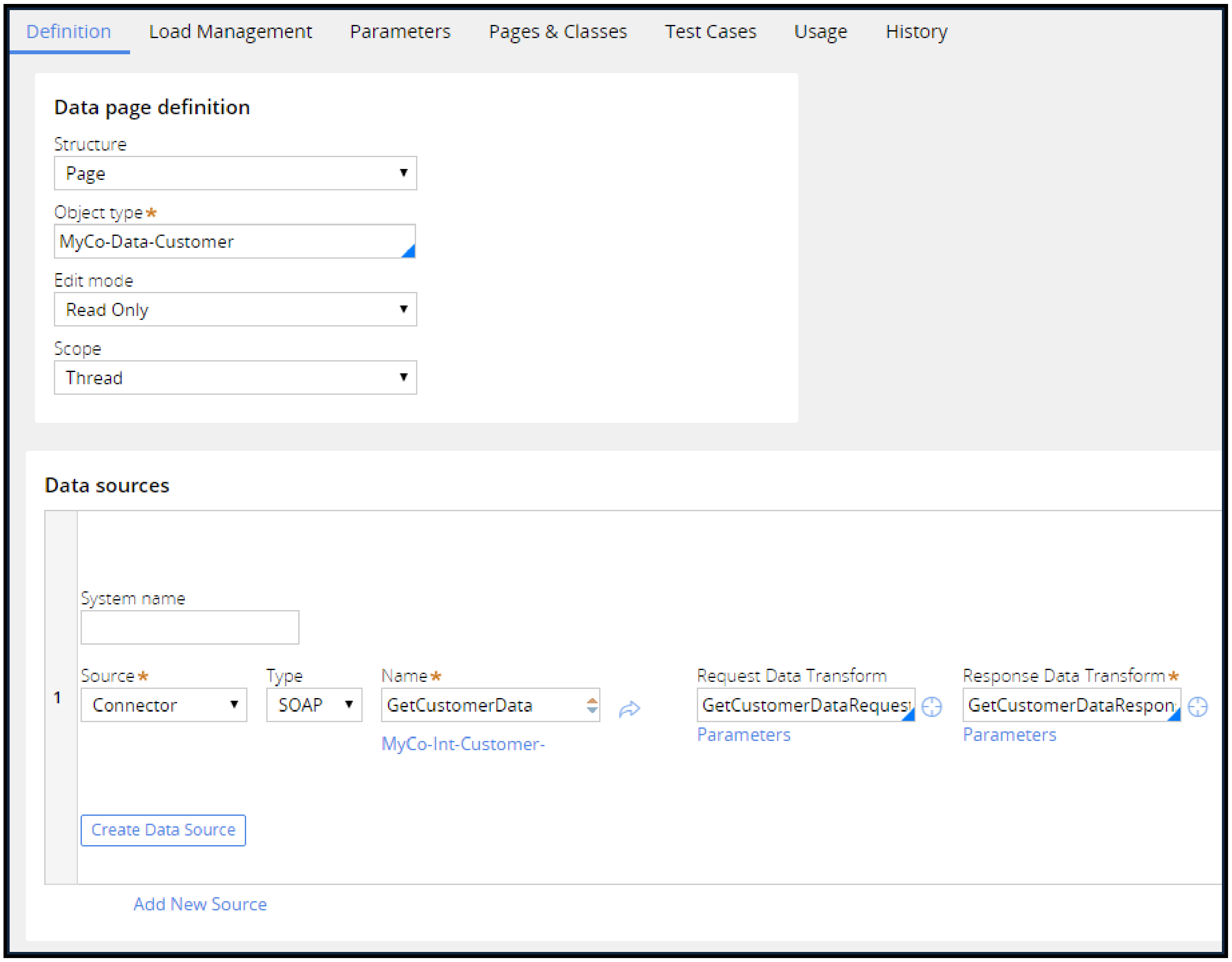
Next, you need to decide on the scope. You have three options for the scope: thread, requestor, and node.

1. The **thread** level scope is useful when the data page is context-sensitive to a particular case. For example, if a customer service representative (CSR) is simultaneously working on several cases belonging to different customers, the data page must be defined as thread scope since the customer data should be limited to a specific case. Setting the scope to thread ensures that the CSR sees only the data relevant to an individual customer's case. If the data can be shared across cases, then a broader scope, such as requester or node, should be chosen.
2. The **requestor** scope allows you to share data pages for a given user session and is often used when the data page contains data associated with the logged in operator. The work list or local weather information are both examples of data associated with the logged in operator.
3. The **node** option makes a single data page instance accessible by all users of the application and other applications running on a given node. On a multinode system, each Java Virtual Machine instance has one copy of the node level data page. Node level pages reduce the memory footprint by storing only a single copy of the data available to all users. Node level pages are an excellent option for storing common reference data such as currency exchange rates.

Then, you need to configure the data sources. For list structures the sourcing options include: connector, data transform, report definition, or a load activity. For page structures, the look-up data source replaces report definition as an option.

Source	Description
Connector	Use a connector to obtain data from an external data source as specified by the connector type.
Data transform	Use the data transform option to populate a data page using a data transform.
Report definition	Use a report definition to return a list of data objects mapped in the application.
Look-up	Use the look-up to return a specific data object mapped in the application.
Load activity	The activity can be used for special situations where none of the other options are suitable.

The **Request Data Transform** and **Response Data Transform** allow you to map the outgoing and incoming data to the application data structure.



Data page definition

Structure: Page

Object type*: MyCo-Data-Customer

Edit mode: Read Only

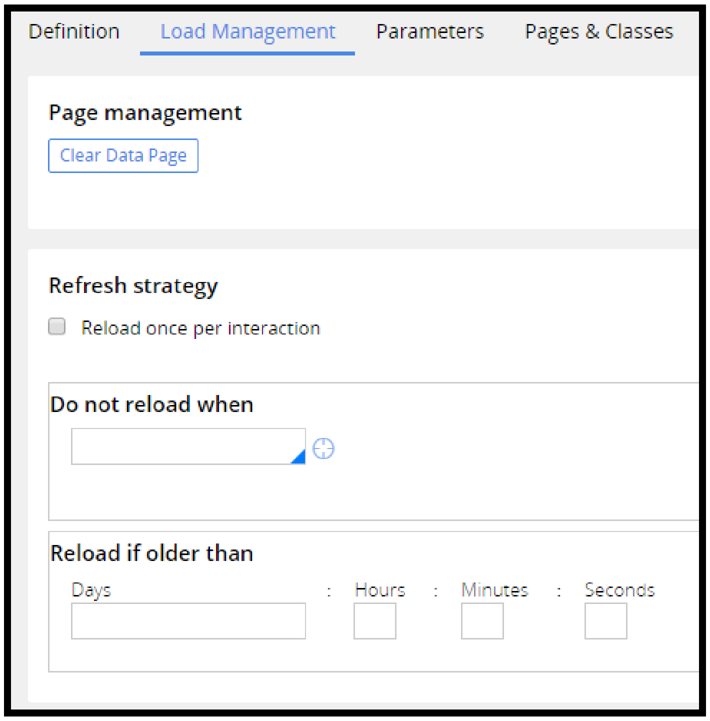
Scope: Thread

Data sources

System name:

	Source*	Type	Name*	Request Data Transform	Response Data Transform*
1	Connector	SOAP	GetCustomerData MyCo-Int-Customer-	GetCustomerDataReques Parameters	GetCustomerDataRespon Parameters

You configure the **Refresh strategy** on the **Load Management** tab. The available reload options depend on the scope of the page. For requestor or thread pages, you can reload the data for each interaction or upon a when rule evaluating to false. The data page can also be marked for refresh based on an elapsed time interval calculated from the last load time. If you combine the **Do not reload when** and **Reload if older than** options, the data page refreshes as soon as either condition is met.


Definition | Load Management | Parameters | Pages & Classes

Page management

Clear Data Page

Refresh strategy

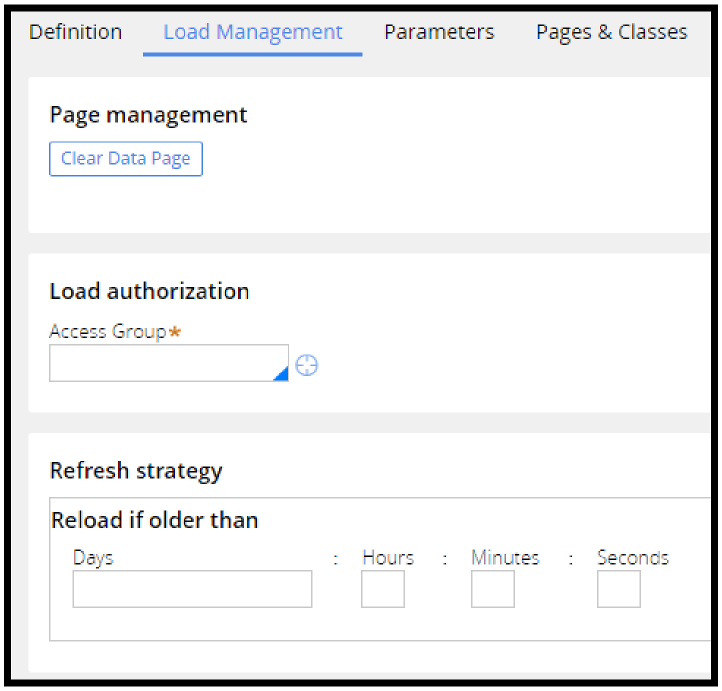
☐ Reload once per interaction

Do not reload when

Reload if older than

Days : Hours : Minutes : Seconds

The refresh strategy for a node level page reloads per interaction and does not reload when options are not available. Notice the addition of the Load Authorization section prompting you for an Access Group.


Definition | Load Management | Parameters | Pages & Classes

Page management

Clear Data Page

Load authorization

Access Group*

Refresh strategy

Reload if older than

Days : Hours : Minutes : Seconds

Node level data pages are not executed in the context of a logged in operator since they are available for all applications on the node. Instead an access group is specified to provide the requestor context used by the system when loading the node level data page.

Data pages load to memory on demand. A data page remains in memory, on the clipboard, to serve requests without reloading the data until marked to be reloaded. If a data page is configured to reload if older than one hour, then after one hour the page is marked for reload but is not reloaded until the next request for the page occurs following the one hour mark.

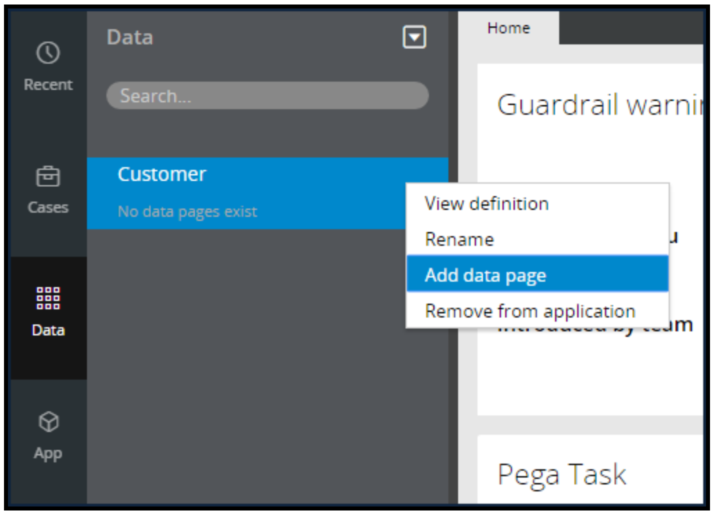
Configuring a data page

You create and configure a data page by creating the data page, configuring the data page definition, and configuring the data source. The configuration varies depending on the requirements for the application. In this example, you configure a data page that holds customer data for a claim case handled by a customer service representative. The customer data is retrieved from a SOAP connector.

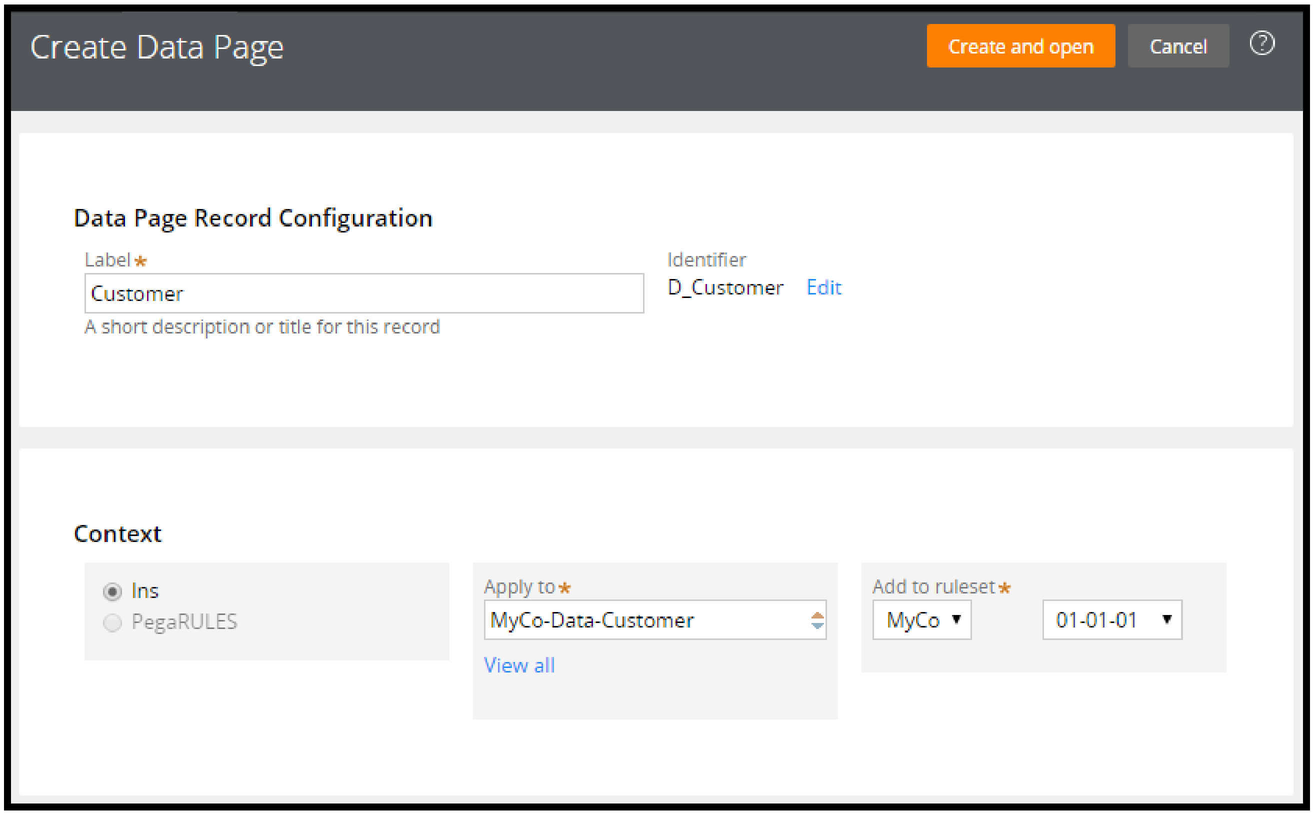
Create data page

Follow these steps to create a new data page for the data type:

1. In the Data Explorer, locate the data type for which you want to create a data page.
2. Right-click the data type to select **Add data page**. In this example, you add a data page for Customer data.



3. Enter a Label that is descriptive of the data. The system populates the Identifier based on the Label. You can leave the default Identifier or change this value.


Create Data Page

Data Page Record Configuration

Label: Customer

A short description or title for this record

Identifier: D_Customer

Context

[✓ Ins] PegaRULES

Apply to: MyCo-Data-Customer

Add to ruleset: MyCo 01-01-01

4. Select a Context. The system defaults values in the Apply to and Add to ruleset fields.
5. Click **Create and open**.

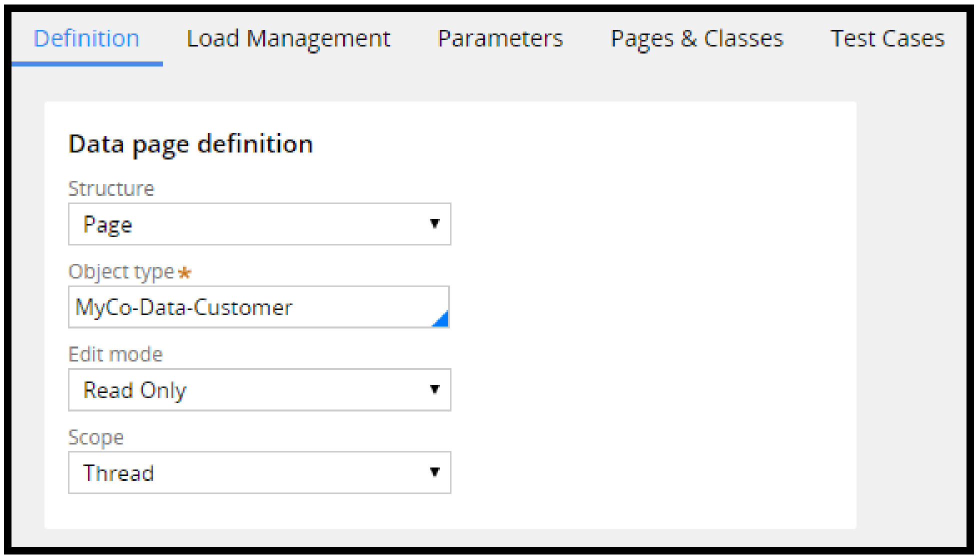
Configure data page definition

The data page holds the data for a specific customer record.

Follow these steps to configure the data page definition:

1. In the **Structure** drop-down, select **Page** since you want to retrieve data for a single customer.
2. In the **Object Type** field, enter the class of the page or pages in the list.
3. Leave the Edit mode set to Read Only since the data page will not be manipulated.

4. Set the Scope to Thread since the customer data is specific to a case.



Data page definition

Structure: Page

Object type*: MyCo-Data-Customer

Edit mode: Read Only

Scope: Thread

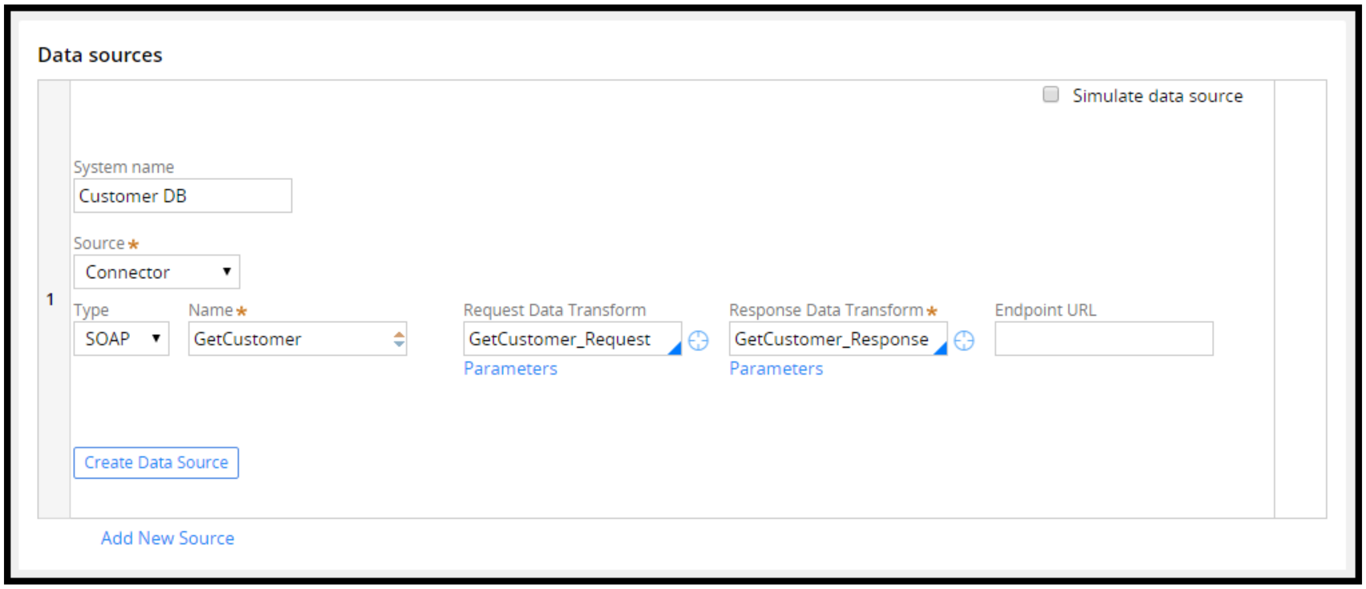
Configure data source

The data in this example is retrieved from a SOAP service.

Follow these steps to configure the data source for the data page:

1. Provide a System name for your data source.
2. In the **Source** drop-down, select **Connector**.
3. In the **Type** drop-down, select **SOAP**.
4. Specify the Name of the connector — GetCustomer, in this case.
5. Optionally, specify a Request Data Transform that maps application data to the request.
6. Specify a Response Data Transform to map the data returned by the connector to the application

data structure.



Data sources

Simulate data source

1

System name: Customer DB

Source: Connector

Type	Name	Request Data Transform	Response Data Transform	Endpoint URL
SOAP	GetCustomer	GetCustomer_Request	GetCustomer_Response	

Managing reference data

Introduction to managing reference data

Applications often require access to reference data. Reference data is used in the case processing, but is not directly part of the application. Reference data is often used in the user interface to provide options for the user. For example, a drop-down can contain a list of store branches for the user to select from.

Reference data is sometimes retrieved from external systems using connectors. However, in some cases, the data needs to be stored and distributed as part of the application.

After this lesson, you should be able to:

- Identify the need for reference data in an application.
- Explain the concept of reference data.
- Explain the benefits of incorporating reference data into an application.
- Explain how local data storage manages reference data in an application.
- Incorporate reference data in an application with local data storage.

Reference data

Every application collects data. Sometimes the values for an input field are limited to a set of values. For example, a shirt might only be available in the colors white, black, and gray. Similarly, a customer can typically select standard, express, or next-day shipping options. However, for a specific product the shipping option might be limited to standard. Reference data defines permissible values for data fields. Limiting the input values to valid options reduces errors and allows for automation. Reference data gains in value when it is widely reused and referenced.

Reference data should be distinguished from master data. Master data represents key business entities, such as customers. Master data contains all the necessary detail — for example, an identifier, name, address, and date of account creation for a customer. Reference data consists of a list of permissible options with relevant metadata.

Key	Label	Description	Cost
FREE	Free delivery	Order arrives 3 to 5 business days after dispatch.	0
STD	Standard delivery	Order arrives 1 to 2 business days after dispatch.	10
EXP	Express delivery	Orders placed before the Express deadline arrive by 1:00 P.M. the following day.	25

A change to the reference data values may need an associated change in the business process to support the change. A change in master data is always as part of existing business processes. For example, adding a new customer is part of the standard business process. Adding a new customer level — for example, platinum — results in a modification to the business processes to manage platinum customers.

Reference data is sometimes retrieved from external systems using connectors. In some cases, the data needs to be stored as part of the application. A **local data storage** allows you to store reference data as part of the application. Reference data stored in a local data storage can be packaged and distributed as part of an application.

How to use local data storage

Local data storage lets you store data records for a data type without having to manually create or maintain database tables. The Designer Studio provides a user interface for managing the data. The reference data in the local data storage can be accessed using a data page in the application.

You can create a local data storage for any data type. To create a local data storage, you first need to add or identify an existing data type for the reference data. Then, you create a local storage for the data type. When creating the local storage, you need to specify which property or properties serve as the unique key for the record. The properties selected as keys form the key in the database table. After you create the local data source, you can add records to the data source using Designer Studio. The reference data can be packaged and distributed with the application.

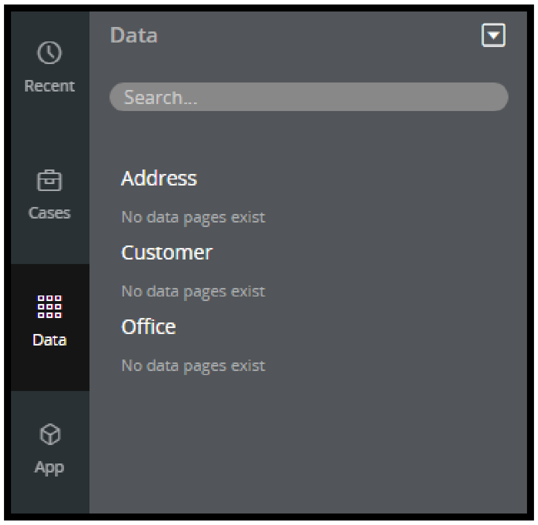
Defining reference data for an application

To define reference data for an application, you first need to create a local storage for the data type. Then, add records to the local data storage.

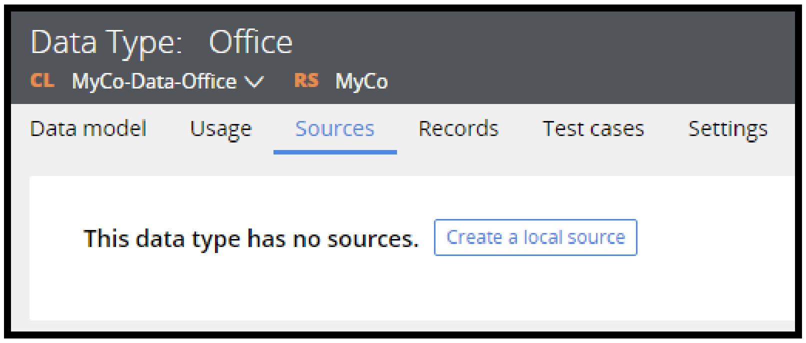
Create a local data source

Follow these steps to create a local data source for a data type:

1. Select the data type for which you want to create a local source in the data explorer. Here you create a local data source for office branches.

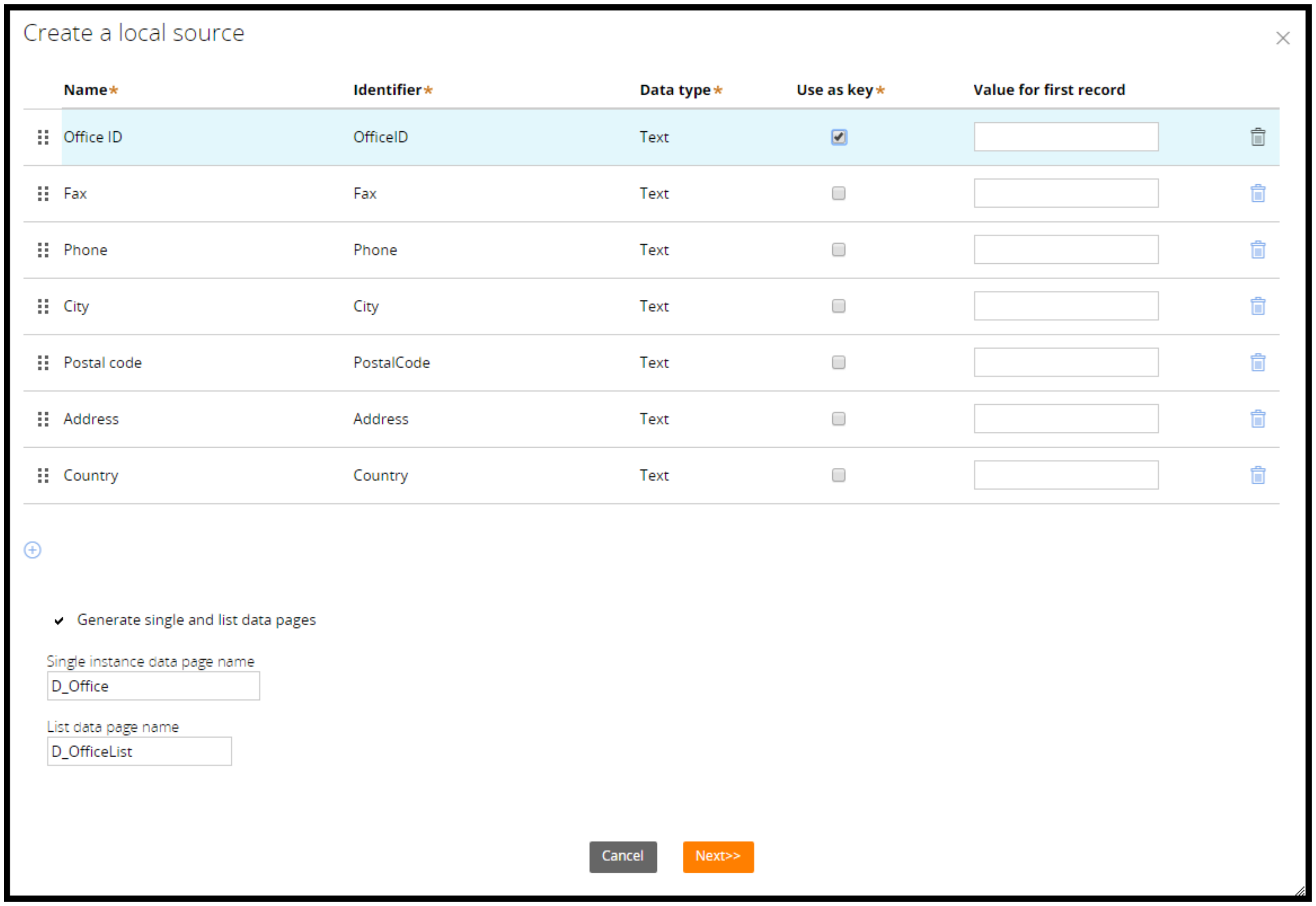


2. Select the **Sources** tab and click the **Create a local source** button.



This data type has no sources.

3. The first screen displays the properties that are already available in the selected data type.
   a. Click the **Plus sign** icon to add properties.
   b. Click the **Trash icon** to delete properties.
   c. Click and drag the **Row** icon to the left of the property row, and reposition the property, to change the order of the properties.
4. Select **Use as key** on the properties you want to define the key in the database table.
5. You have the option to edit the data page names.


Create a local source

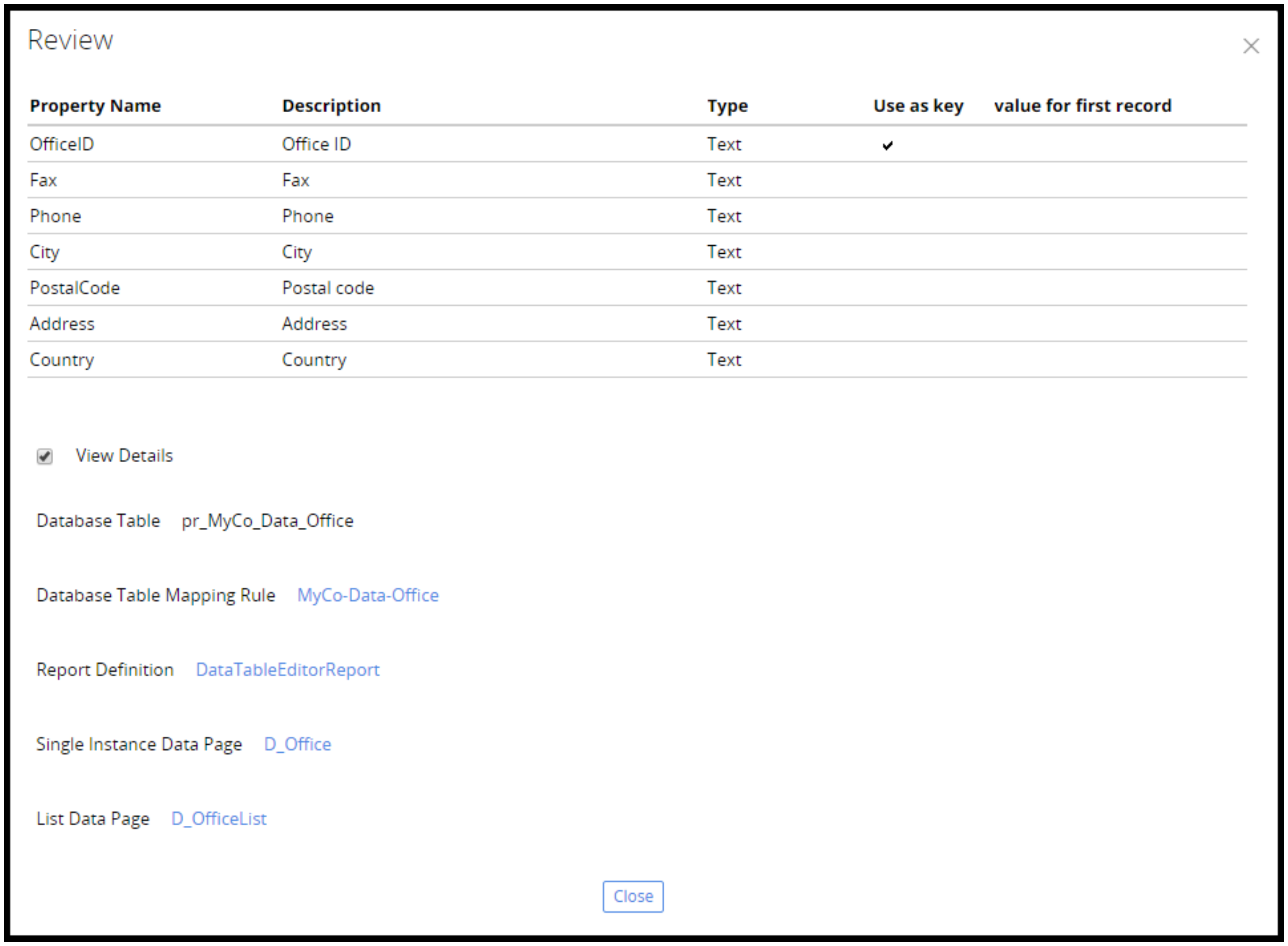
	Name*	Identifier*	Data type*	Use as key*	Value for first record
	Office ID	OfficeID	Text	✓	
	Fax	Fax	Text	☐	
	Phone	Phone	Text	☐	
	City	City	Text	☐	
	Postal code	PostalCode	Text	☐	
	Address	Address	Text	☐	
	Country	Country	Text	☐	

✓ Generate single and list data pages

Single instance data page name
D_Office

List data page name
D_OfficeList

6. The final screen displays a summary of the created records.



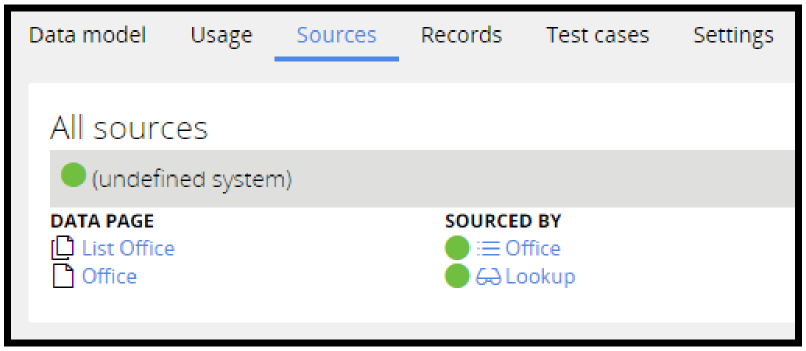
Review

Property Name	Description	Type	Use as key	value for first record
OfficeID	Office ID	Text	✓	
Fax	Fax	Text		
Phone	Phone	Text		
City	City	Text		
PostalCode	Postal code	Text		
Address	Address	Text		
Country	Country	Text		

☑ View Details

Database Table	pr_MyCo_Data_Office
Database Table Mapping Rule	MyCo-Data-Office
Report Definition	DataTableEditorReport
Single Instance Data Page	D_Office
List Data Page	D_OfficeList

7. Click **Close**.



All sources

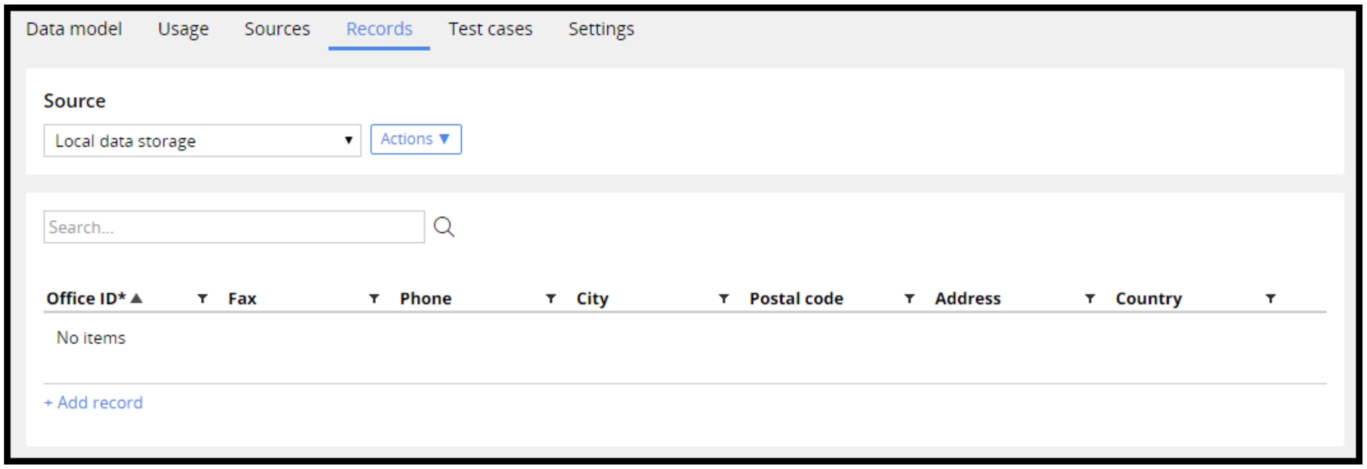
● (undefined system)

DATA PAGE	SOURCED BY
List Office	● Office
Office	● Lookup

Manage records in the local data source

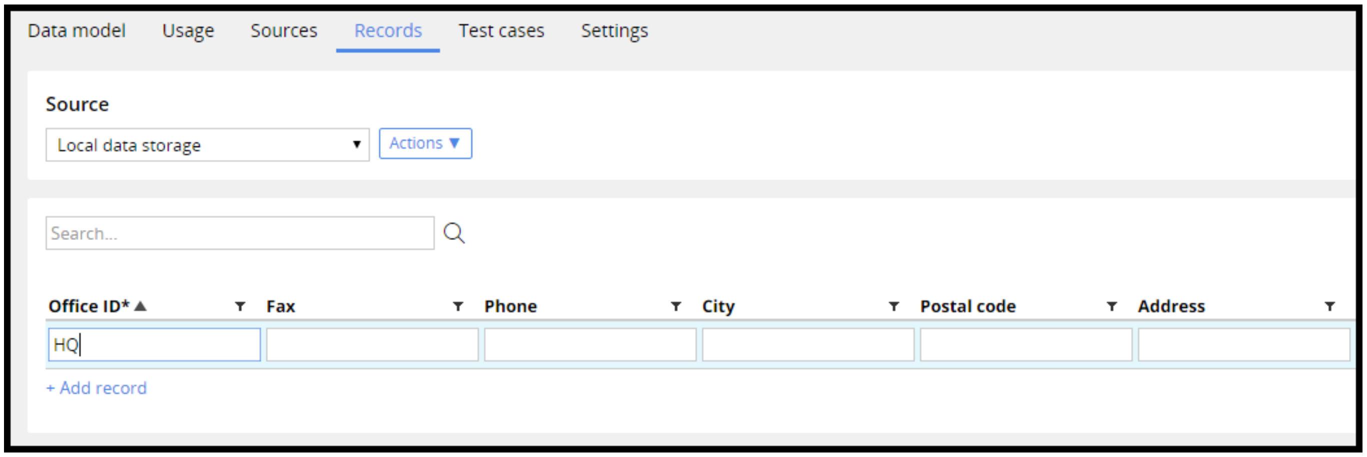
Follow these steps to add a record to a local data source:

1. Select the **Records** tab on the data type.



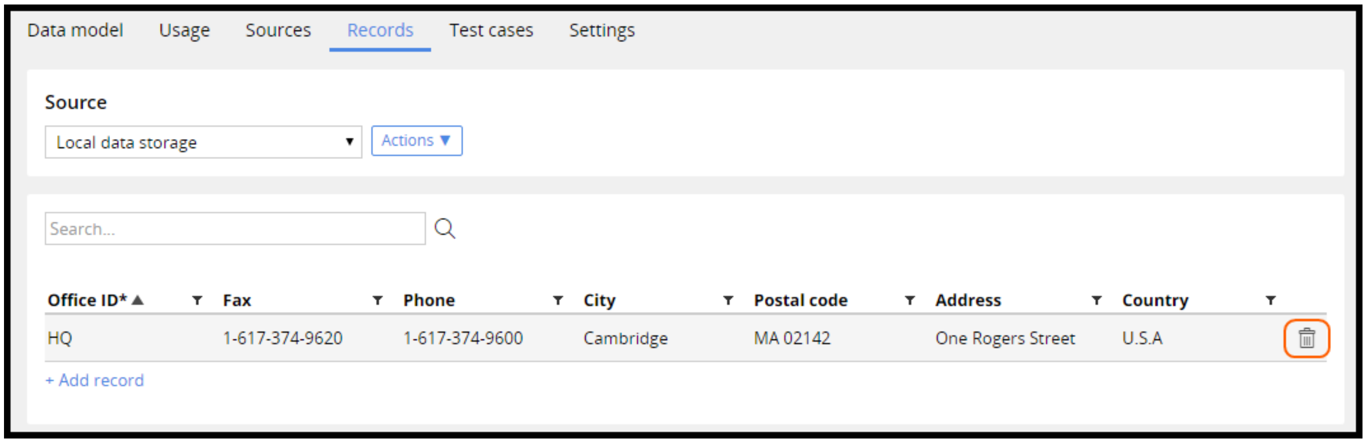
Office ID*	Fax	Phone	City	Postal code	Address	Country
No items						

2. Click the **Add record link**.



Office ID*	Fax	Phone	City	Postal code	Address
HQ					

3. Enter the data for the properties. The data is persisted when you leave the field.



Office ID*	Fax	Phone	City	Postal code	Address	Country
HQ	1-617-374-9620	1-617-374-9600	Cambridge	MA 02142	One Rogers Street	U.S.A

4. Click the **Trash can** icon to remove a record.

Integration in Pega applications

Introduction to Integration in Pega Applications

Most businesses have applications that handle a wide variety of business needs. Many of these applications need to exchange data with each other as well as with external systems. For example, an enterprise's order management system handles employee purchases. However, the product system of record and inventory is maintained in another system. After placing and approving an order, the order management system needs to communicate the update with the system of record.

Integration enables your application to exchange data with other systems. For example, your application might need to access data or computations provided by an external system, or respond to requests from external systems.

After this lesson, you should be able to:

- Explain how connectors exchange data with other systems.
- List the types of connectors available in Pega.
- Explain how services process requests from other systems.
- List the types of services available in Pega.
- Connect to database data using database table class mapping.

Connectors

Imagine the process of applying for a loan. First, a customer completes a loan application and submits the application to the bank. The bank enters the customer's information into the application system. The bank then verifies the customer's credit score with a credit agency. The bank application must integrate with the credit agency system to verify the credit score.

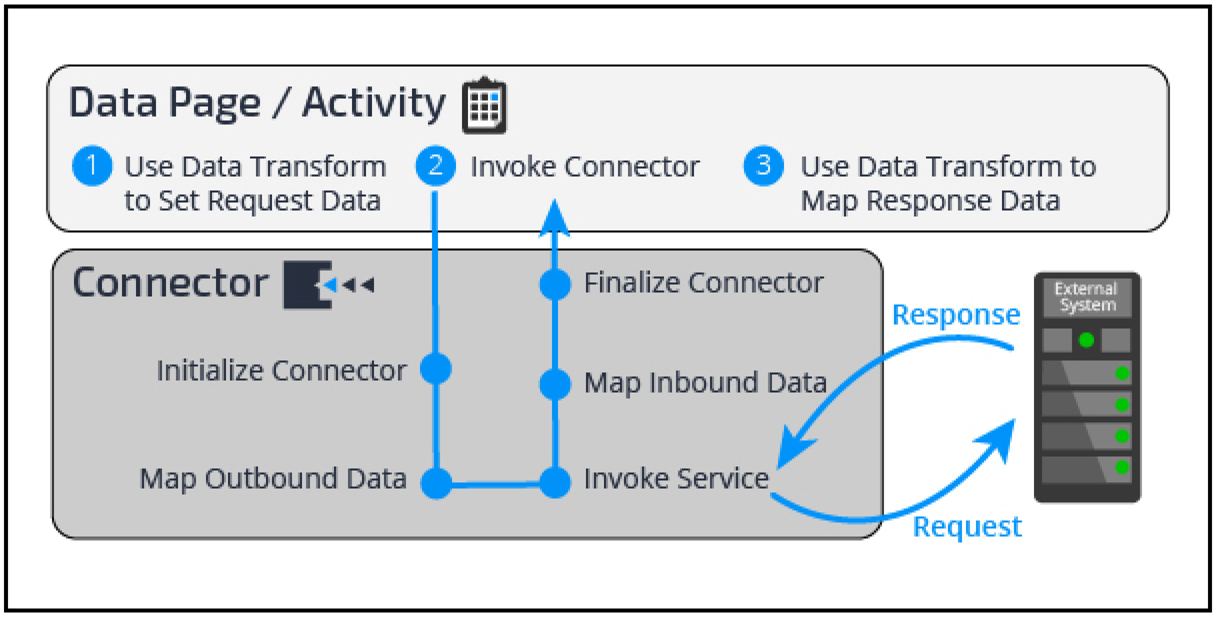
In this example, the application requests data from another system. Pega 7 uses Connectors to facilitate this type of integration. Connectors are protocol specific, and they establish the link to the external system. Connectors implement the interface of the service running on the external system. Connectors also map the data structure of the application to the data structure used by the service called. You can parse, convert, and map data in either direction to or from the clipboard. For example, you can map data to and from XML, fixed record structure, or a record structure separated by a delimiter character.

You can invoke Connectors from data pages and activities. Use data pages to read, or pull, data from the external system. Use activities to write, or push, data to the external system.

How connectors exchange data

The invocation of a connector involves five components:

- Data page or activity — Specifies the connector to use and data transforms for request and response mapping.
- Data transforms — Maps the data structure of your application to the integration clipboard pages, which correspond to the format expected by the service.
- Connector rule — Uses the integration clipboard pages to build the request according to the protocol and service definition, invokes the service, and parses and places the response on the integration clipboard pages.
- Mapping rules — For most connectors, mapping rules are used to build outgoing and parsing incoming messages.
- External system — Exposes the service called.



Look at the steps executed when invoking a connector:

1. The data page or activity executes a data transform to map the data from your application to the integration clipboard pages.
2. The data page or activity invokes the connector:
   a. The connector is initialized based on its type. The type is the protocol the connector supports.
   b. The connector maps the request data to the protocol-specific format using the mapping rules specified for the connector. Do not confuse this mapping with data transforms. This mapping is between the clipboard and the format required by the protocol.
   c. The application sends the request to the external system.
   d. The application receives the protocol-specific response. The response data is parsed using the mapping rules specified in the connector rule and placed on the integration clipboard pages.
   e. The connector is finalized and returns control to the data page or activity.
3. Finally, a data transform maps the response data from the integration clipboard data structure to your application.

Supported connectors

Pega 7 provides connectors for a wide range of industry-standard protocols and standards. Standard connectors include SOAP, REST, SAP, EJB, JMS, MQ, File, and CMIS.

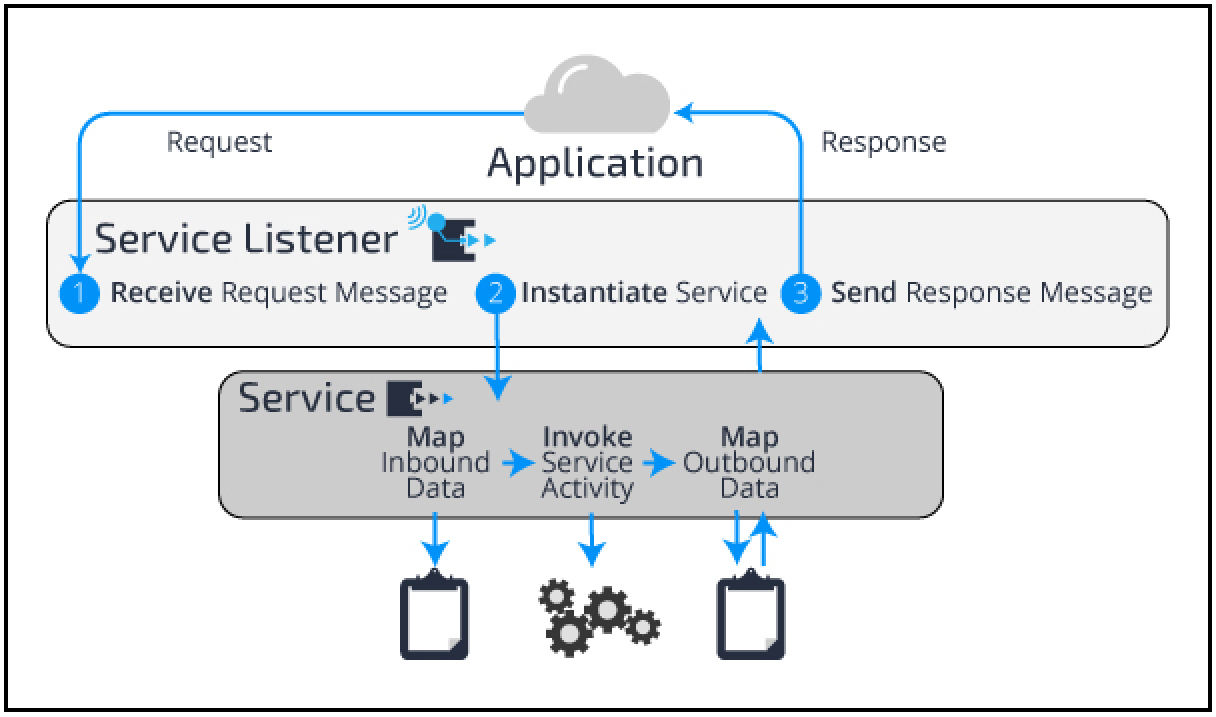
Services

To get a 360-degree view of a customer, you must gather information for that customer from several different systems. One of those systems could be a Pega 7 Platform. For example, if an insurance claims application is implemented on the Pega 7 Platform, you may need to expose a service that returns the claims for a specific customer. In this example, an external system requests data from your application. Pega 7 uses services to facilitate this integration.

Services allow you to expose the data and functionality of your application to external systems. Services implement the interface of a specific protocol and provide data mapping for outbound and inbound content. You can parse, convert, and map data in either direction to or from the clipboard. Data can be in XML, fixed record structure, or separated by a delimiter character format.

How services exchange data

The service listener senses for incoming requests. Service listeners provide the Pega 7 Platform with information the platform needs to route incoming messages to a specific service. The service listener establishes a requestor. A requestor is the processing and data associated with the incoming request initiated by the external system. This functionality is sometimes provided by the underlying Web or Application Server, and sometimes provided by a Pega 7 listener.



Review the procedure for processing a request from an external system:

1. The service listener instantiates the protocol-specific Service to provide communication with Pega 7 and establish a requestor. The service listener optionally performs authentication.
2. The service parses the incoming request and maps the request onto the clipboard. The service then invokes the service activity. The service activity provides the logic for the service.

3. When your service activity is complete, control is returned to the service. The service builds the response using data on the clipboard, and the service listener sends the response to the external system.

Supported services

Pega 7 provides services for a wide range of industry-standard protocols and standards, including SOAP, REST, EJB, JMS, MQ, and File.

Connecting to an external database

Sometimes your application requires access to reference data in an external database. For example, a purchase application needs access to currency exchange rates. Exchange rate information is stored in a table in an external database.

You have two options when integrating with an external database in Pega 7:

- Database Table Class Mapping tool
- SQL connector

The Database Table Class Mapping tool provides a wizard to generate all the artifacts needed to interact with reference data in an external database. These artifacts include a data class, a database table instance, and a link between those two artifacts.

The data mapping creates a pass-through from your application to the table in the external database. When enabled, you can access data in the external database as if the data were within your application. Using the Database Table Class Mapping tools is preferred when integrating with external databases.

The SQL connector requires you to write SQL queries to interact with the data in the external database. Use a SQL connector when you need to perform advanced queries (such as advanced joins), or when you need to use vendor-specific SQL syntax. Setting up a SQL connector is a Senior System Architect or a Lead System Architect task. In this course, you use the Database Table Class Mapping tool to set up your integration.

Database Table Class Mapping tool

Take a look at how you can use the Database Table Class Mapping tool. The tool creates a data class with the data mapping and a database table instance that references the external table.

Before you begin

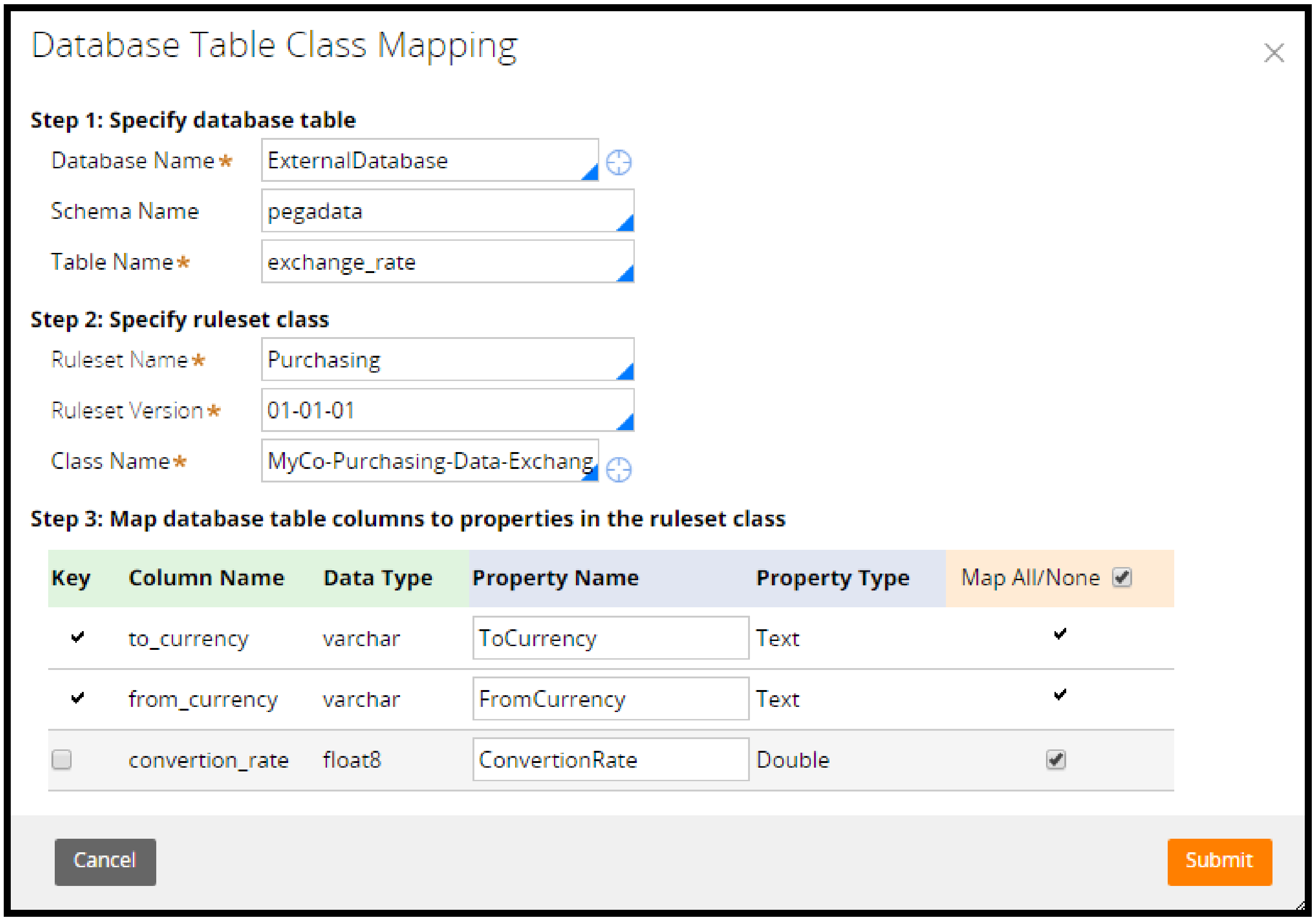
Pega 7 requires database record for each database that your application connects to. When the Database instance has been created and a suitable JDBC library has been installed on the server, you can connect to and interact with that database.

Run the tool

The Database Class Mappings landing page (Designer Studio > Data Model > Classes & Properties > Database Class Mappings) shows you all of the existing database table mappings.

1. Click **New External Database Table Class Mapping** to launch the Database Table Class Mapping tool.
2. Choose the database, schema, and the name of the table you want to create the mapping for.
3. Enter a ruleset name and version.
4. Enter the name of the data class you want to create. The class cannot exist as the Database Table Class Mapping tool generates a new class.

5. Select the columns you want to access and specify the name of the properties you want to map to in the data class.
6. Click **Submit**.



Database Table Class Mapping

Step 1: Specify database table

Database Name*	ExternalDatabase
Schema Name	pegadata
Table Name*	exchange_rate

Step 2: Specify ruleset class

Ruleset Name*	Purchasing
Ruleset Version*	01-01-01
Class Name*	MyCo-Purchasing-Data-Exchang

Step 3: Map database table columns to properties in the ruleset class

Key	Column Name	Data Type	Property Name	Property Type	Map All/None ✓
✓	to_currency	varchar	ToCurrency	Text	✓
✓	from_currency	varchar	FromCurrency	Text	✓
☐	convertion_rate	float8	ConvertionRate	Double	✓

The Database Table Class Mapping tool creates a class and properties. The External Mapping tab on the class record contains the mapping details.

You can use the class created with a data page and the lookup feature to fetch an instance, or with a report definition to fetch a list of instances. Alternatively, you can use the class created with the Obj-methods to open, save, and remove rows from the table.

Creating a connector

Introduction to Creating a Connector

Create and configure a Connector when you want to call a service exposed by an external system. The connector wizards guide you though the process of setting up the Connector. The wizards generate the records required to integrate with the service based on the service definition.

After this lesson, you should be able to:

- Explain the role of the connector wizard.
- Explain how connector wizards configure connectors.
- List the connector wizards available in Pega 7.
- Configure a connector to source data for a data page.
- Configure a connector to update a data source with an activity.

Creating a connector

Pega 7 includes several connector wizards. Using connector wizards speeds up the integration development. The wizards use metadata, such as a WSDL file or EJB class, to create the connector records. The created records include classes, properties, mapping rules, data transforms, and an authentication profile.

All connector wizards follow the same process:

- Upload service metadata
- Configure integration specifics
- Select methods
- Generate records

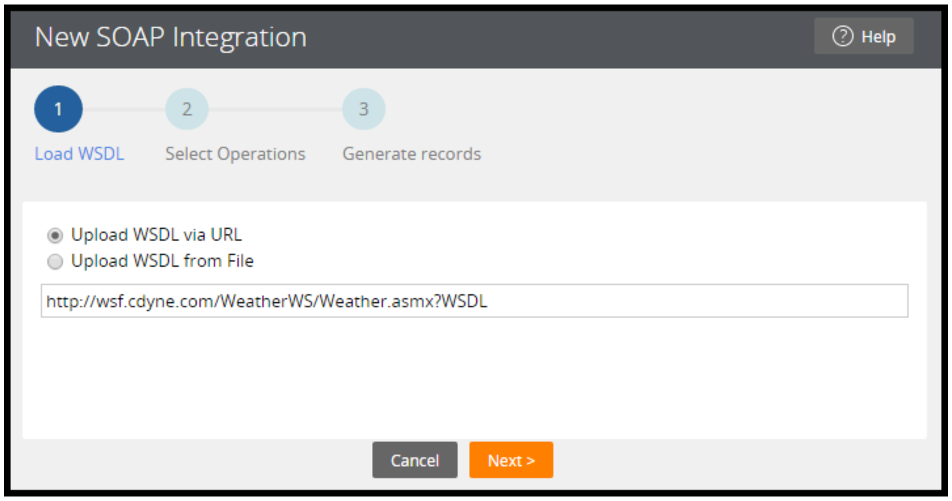
You can find the connector wizards under **Designer Studio > Integration > Connectors**:

- Create SOAP Integration — Creates data sources from the operations in external SOAP services defined in a WSDL file
- Create REST Integration — Creates data sources to obtain data from external REST services
- Create SAP Integration — Creates a data sources from IDoc XSD documents or operations in external SAP services
- Connector and Metadata Wizard — Creates connectors for EJB, Java, and SQL

Creating a new SOAP integration

The connector wizards follow the same steps when creating an integration. Look at the SOAP integration wizard as an example. Here, you want to integrate with a weather service.

1. Start the wizard by selecting **Designer Studio > Integration > Connectors > Create SOAP Integration**.
2. First, upload the service definition metadata (for SOAP it is the WSDL file). Select **Upload WSDL via URL** or **Upload WSDL from File** and click **Next**. The wizards analyzes the WSDL and displays the operations.


New SOAP Integration

Help

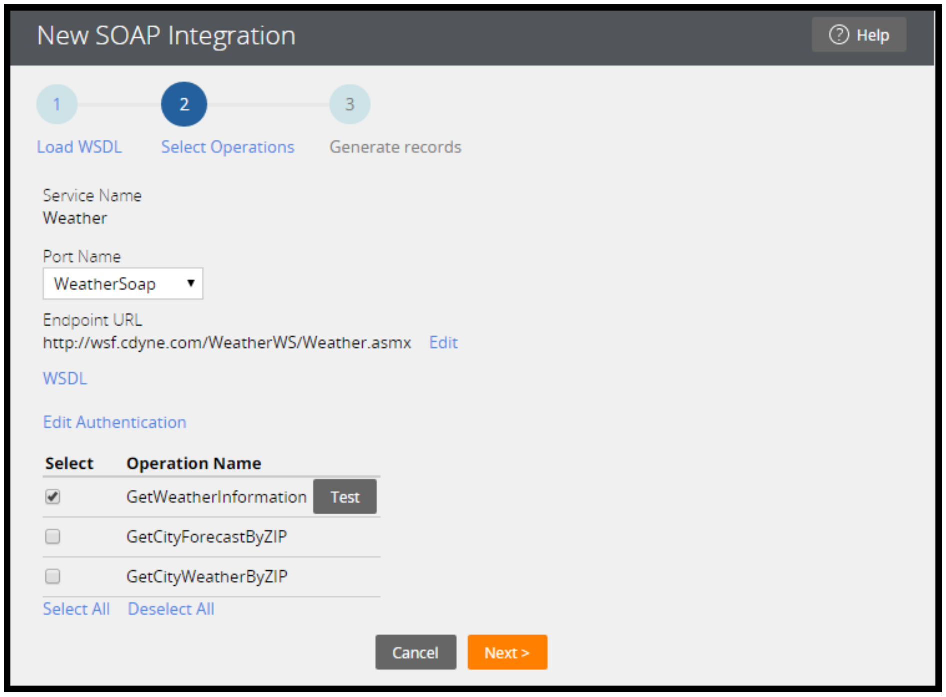
1 Load WSDL — 2 Select Operations — 3 Generate records

(●) Upload WSDL via URL
( ) Upload WSDL from File

http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL

Cancel | Next >

3. Select the operations you want to use in your application. Click the **Edit Authentication** link to specify credentials if the service requires authentication.


New SOAP Integration

Help

1 Load WSDL — 2 Select Operations — 3 Generate records

Service Name
Weather

Port Name
WeatherSoap ▾

Endpoint URL
http://wsf.cdyne.com/WeatherWS/Weather.asmx Edit

WSDL

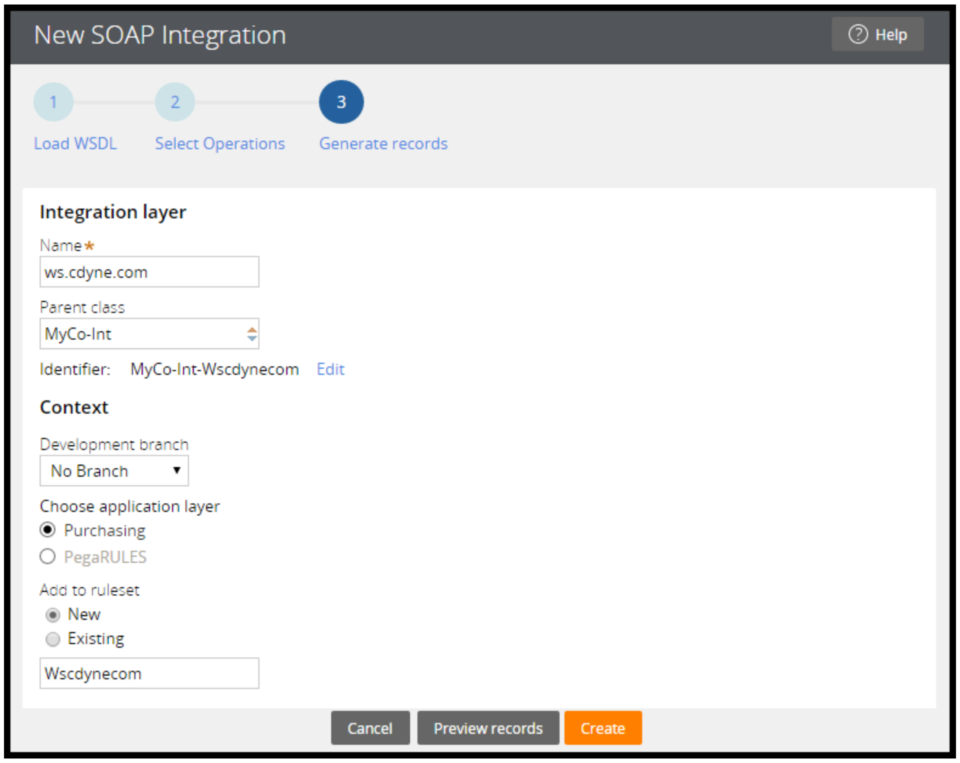
Edit Authentication

Select	Operation Name
✓	GetWeatherInformation Test
	GetCityForecastByZIP
	GetCityWeatherByZIP

Select All   Deselect All

Cancel | Next >

4. Select a name, parent class, ruleset, and application layer for the integration records. The wizard adds the new ruleset to your application. Click **Create** to generate the records.


New SOAP Integration

Help

1 Load WSDL — 2 Select Operations — 3 Generate records

Integration layer

Name*
ws.cdyne.com

Parent class
MyCo-Int

Identifier: MyCo-Int-Wscdynecom Edit

Context

Development branch
No Branch

Choose application layer
● Purchasing
○ PegaRULES

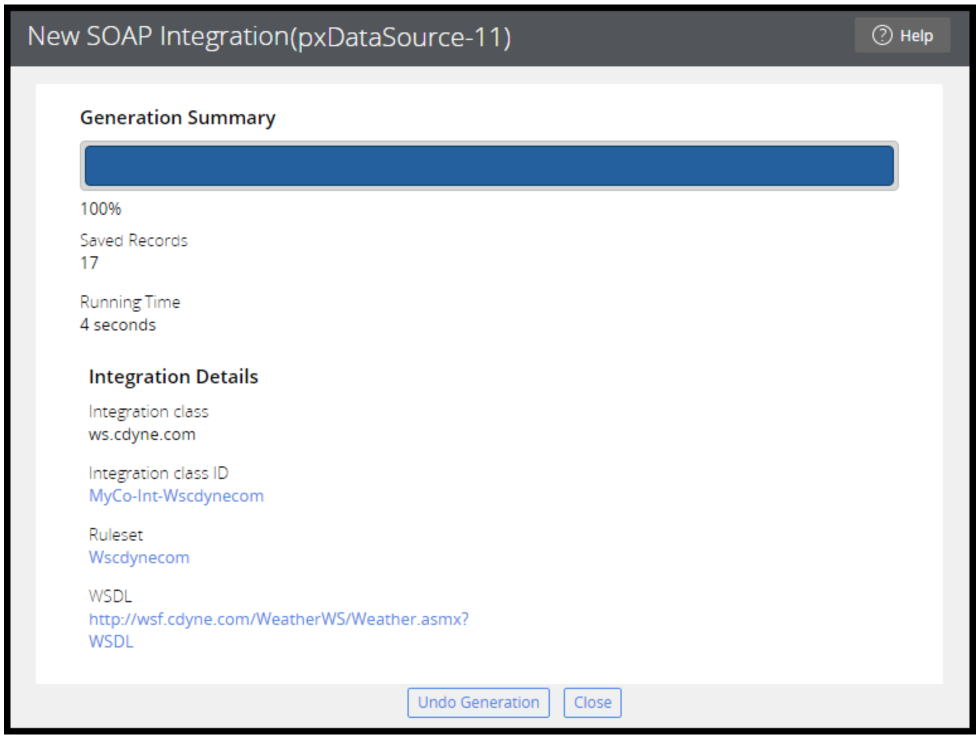
Add to ruleset
● New
○ Existing

Wscdynecom

Cancel | Preview records | Create

5. The last screen displays a summary. Use the **Undo Generation** button to remove the generated records.

   You can access the wizards used in an application at any time by selecting **Designer Studio > Application > Tools > All Wizards**.


New SOAP Integration(pxDataSource-11)

Help

Generation Summary

100%

Saved Records
17

Running Time
4 seconds

Integration Details

Integration class
ws.cdyne.com

Integration class ID
MyCo-Int-Wscdynecom

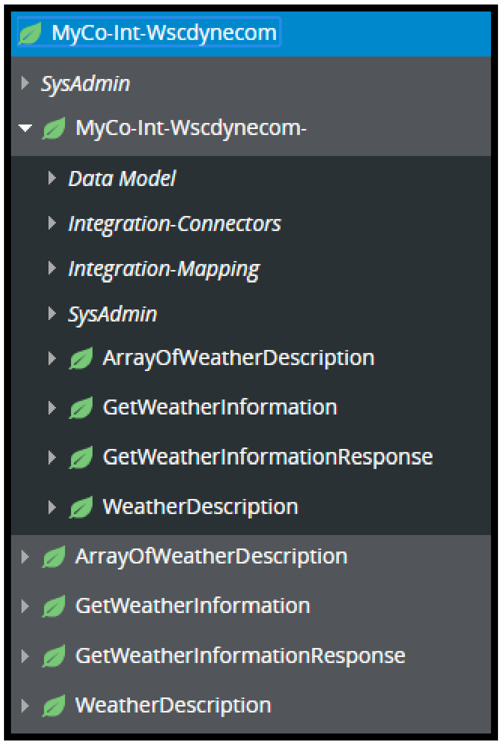
Ruleset
Wscdynecom

WSDL
http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL

Undo Generation | Close

6. The wizard creates a base class for the integration. The base class contains the rest of the records.

In this particular case, you see classes defining the request and response as well as mapping rules.

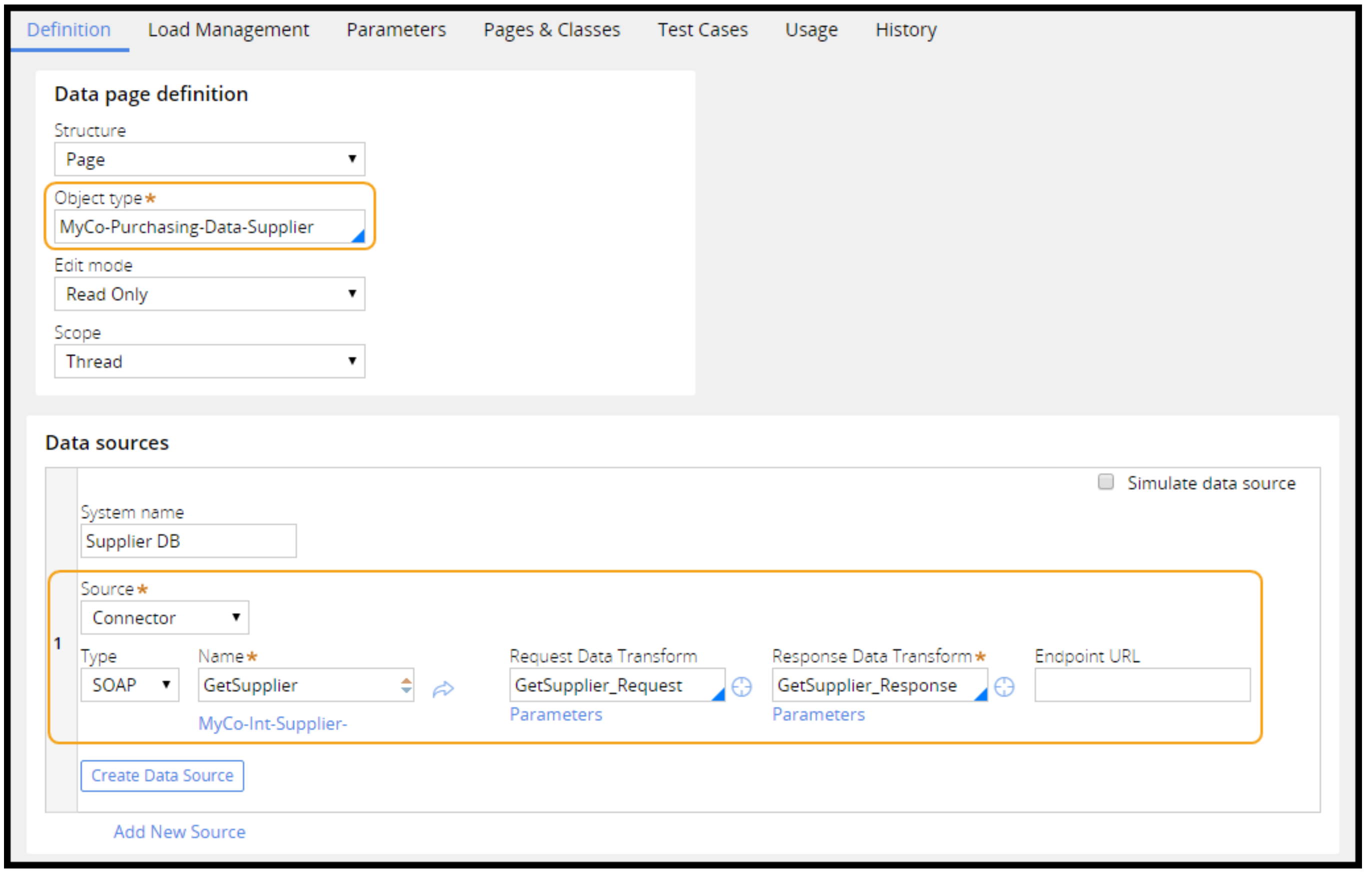


You can now use the connector in a data page or an activity.

Reading data from an external system

Use the connector with a data page if you want to read, or pull, data from a service. The data page invokes the connector and makes the data available in your application.

1. Create a data page in your data type. The data returned by the service is mapped to this data type. Here, the class MyCo-Purchasing-Data-Supplier is used. Select the Connector in the source drop-down. Then specify the type and name.


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

Data page definition

Structure: Page

Object type*: MyCo-Purchasing-Data-Supplier

Edit mode: Read Only

Scope: Thread

Data sources

Simulate data source

System name: Supplier DB

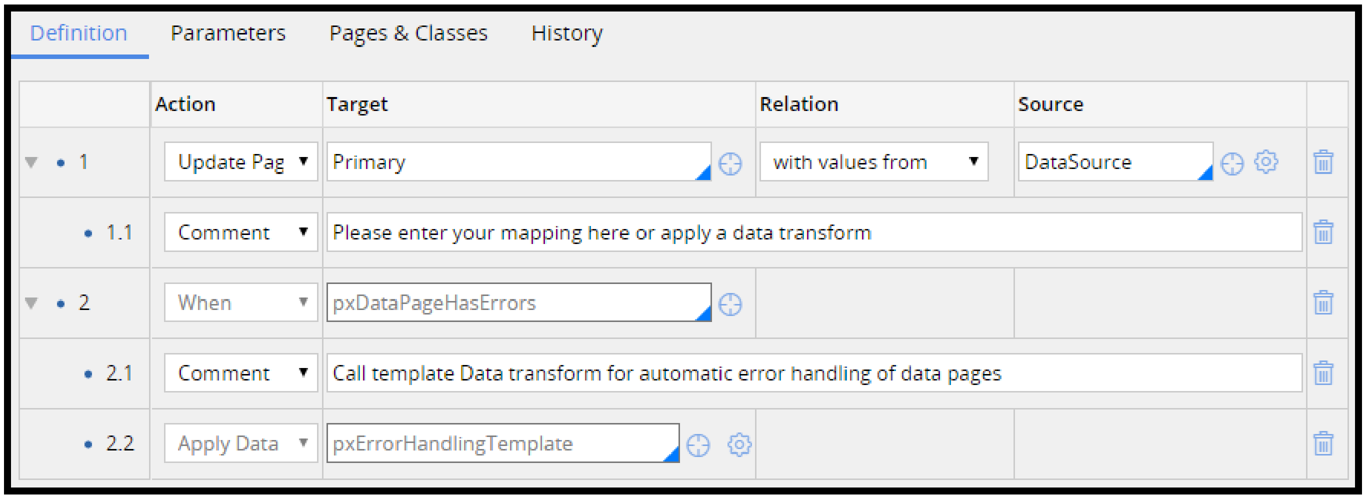
Source*: Connector

Type	Name*	Request Data Transform	Response Data Transform*	Endpoint URL
SOAP	GetSupplier MyCo-Int-Supplier-	GetSupplier_Request Parameters	GetSupplier_Response Parameters	

Create Data Source

Add New Source

2. The pages are set up in the response data transform if you create the data transform from the data page form. The DataSource page in the integration clipboard page with the response data.

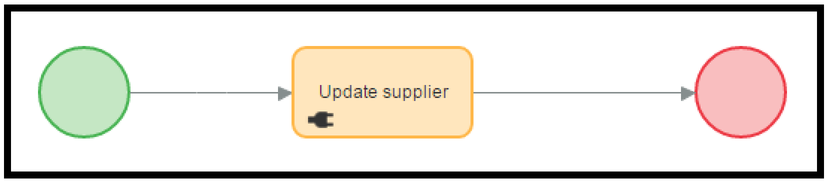

Definition | Parameters | Pages & Classes | History

	Action	Target	Relation	Source
1	Update Pag	Primary	with values from	DataSource
1.1	Comment	Please enter your mapping here or apply a data transform		
2	When	pxDataPageHasErrors		
2.1	Comment	Call template Data transform for automatic error handling of data pages		
2.2	Apply Data	pxErrorHandlingTemplate		

Your data page is now ready for use in the application.

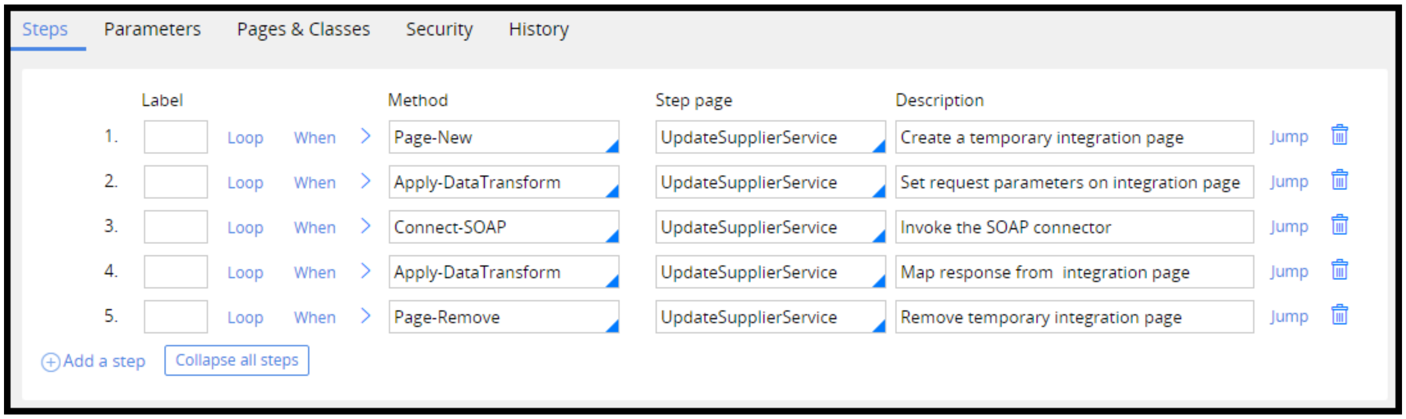
Writing data to an external system

Use the connector with an activity if you want to write, or push, data to the service. An activity allows you to call the connector at a specific point in the process. Use an Integrator shape to call the activity in a flow.



Assume you have an application in which you want to update a supplier record held in an external system. Create an activity to map the request data. Invoke the connector and map the response by following these steps:

1. Set up the integration clipboard page on the Pages & Classes tab. The page should be set to the class with the integration assets as created by the wizard.
2. Create a new integration clipboard page.
3. Use data transforms to set request data on the integration clipboard page.
4. Invoke the connector. The methods are connector-type specific and start with Connect- (for example, Connect-SOAP).



	Label				Method	Step page	Description		
1.		Loop	When	>	Page-New	UpdateSupplierService	Create a temporary integration page	Jump	🗑
2.		Loop	When	>	Apply-DataTransform	UpdateSupplierService	Set request parameters on integration page	Jump	🗑
3.		Loop	When	>	Connect-SOAP	UpdateSupplierService	Invoke the SOAP connector	Jump	🗑
4.		Loop	When	>	Apply-DataTransform	UpdateSupplierService	Map response from integration page	Jump	🗑
5.		Loop	When	>	Page-Remove	UpdateSupplierService	Remove temporary integration page	Jump	🗑

See Pega 7 help for details on how to configure each individual step.

APPLICATION DEBUGGING

Debugging applications with the Tracer

Introduction to Debugging Pega Applications

Software applications are rarely — if ever — written free of errors. Finding problems early on and giving system architects the tools to find problems is instrumental to a successful development cycle.

Some errors in an application are easy to diagnose. Other errors can prove difficult to identify and resolve. To help you identify and resolve errors in your applications, Pega provides the Tracer to allow you to review application execution and identify the root cause of errors.

After this lesson, you should be able to:

- Identify the role of the Tracer in debugging applications.
- Use the Tracer to investigate application errors.

The Tracer

When an error occurs in an application, you need to identify the cause error so you can correct the application behavior. For example, a declare expression may return an unexpected value. If you forget to add one of the input properties to a UI form, users cannot provide a value and the declare expression returns an incorrect result. If you forget to make entry required the field for the input property, users can submit a form without providing a needed value.

Or, perhaps you use a data page to populate a drop-down list. If the contents of the drop-down list are incorrect, you need to determine whether the control or the connection to the data source was configured improperly.

Identifying the root cause of an error is critical to correcting application behavior. You view the events that lead to the error and determine which behavior to address to fix the issue.

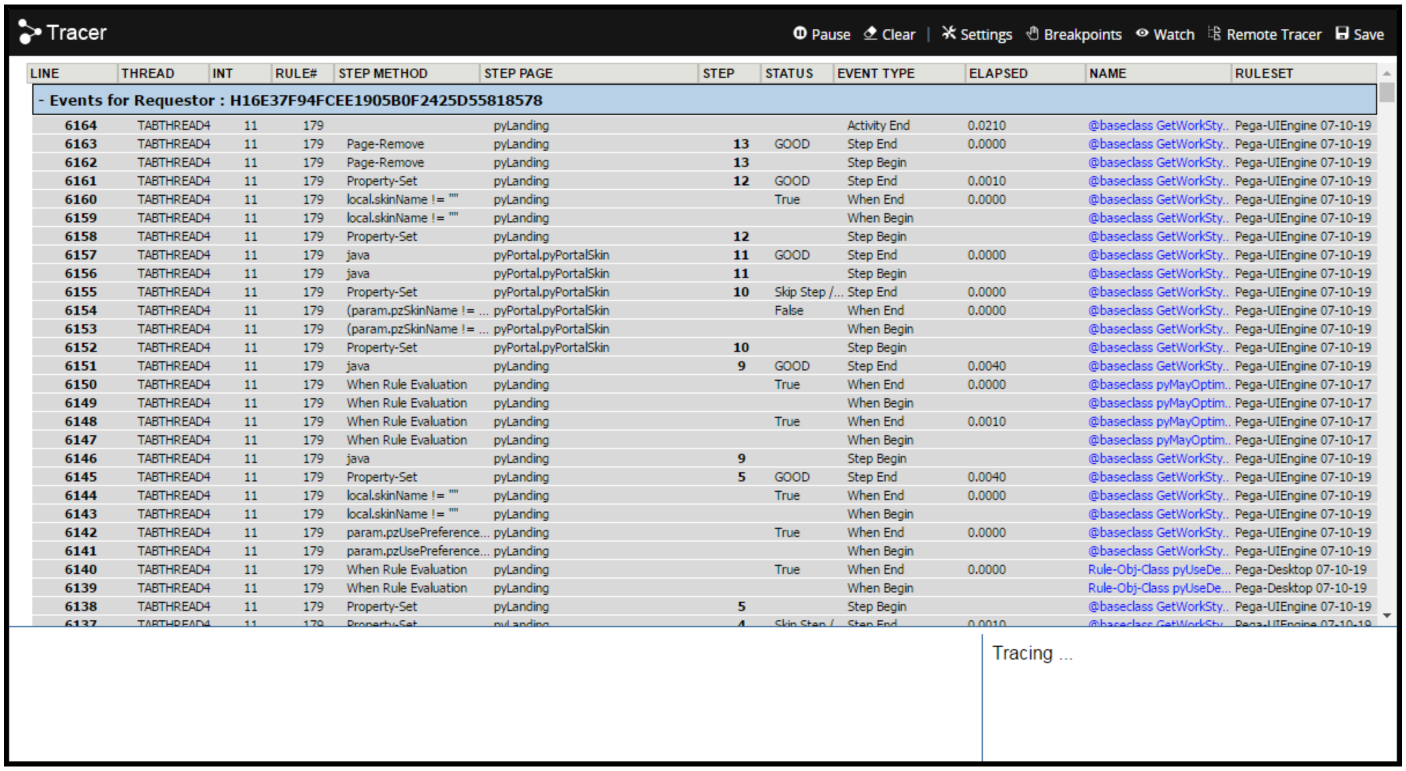
To view events such as these that occur when a case is processed, you use the **Tracer**. In Pega, the Tracer allows you to capture and view the events that occur during case processing. Unlike the Clipboard tool, which presents the current value of properties in memory, the Tracer presents a complete log of the events that occur during case processing. This allows you to identify the cause of execution errors, such as Java exceptions or incorrect property values.

To help identify errors in case processing, the Tracer identifies processing steps that lead to an error. In the Tracer log, most steps return a status of **Good**, indicating that the step completed successfully. If a step returns a status of **Fail**, an error occurred and the step completed unsuccessfully. An error in an application may indicate only the last step in a sequence of failed steps. Reviewing the sequence of events in the Tracer helps to identify the root cause that leads to the error seen by users.

How to investigate application errors with the Tracer

To investigate an issue with the Tracer, you configure the Tracer to monitor application execution. As your application executes, Pega logs all the processing events that result from application execution. You then view the events logged by the Tracer to analyze processing errors and identify their cause.

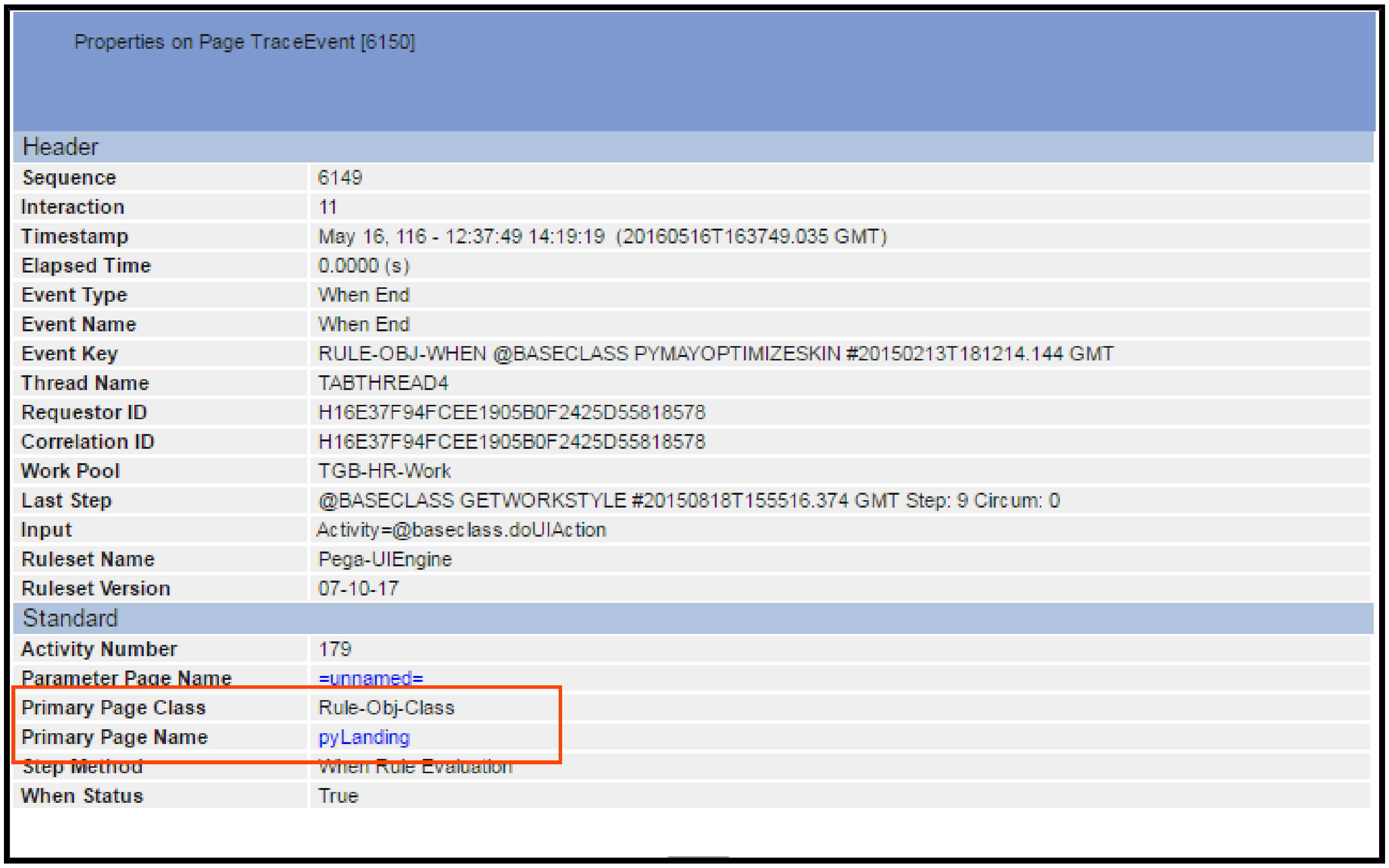
To open the Tracer, click **Tracer** on the **Developer** toolbar. The Tracer logs all of the actions and events that occur in a requestor session in Designer Studio. Each event is logged in order of occurrence, and is identified by thread, event type, and status.



- Events for Requestor : H16E37F94FCEE1905B0F2425D55818578

LINE	THREAD	INT	RULE#	STEP METHOD	STEP PAGE	STEP	STATUS	EVENT TYPE	ELAPSED	NAME	RULESET
6164	TABTHREAD4	11	179		pyLanding			Activity End	0.0210	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6163	TABTHREAD4	11	179	Page-Remove	pyLanding	13	GOOD	Step End	0.0000	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6162	TABTHREAD4	11	179	Page-Remove	pyLanding	13		Step Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6161	TABTHREAD4	11	179	Property-Set	pyLanding	12	GOOD	Step End	0.0010	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6160	TABTHREAD4	11	179	local.skinName != ""	pyLanding		True	When End	0.0000	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6159	TABTHREAD4	11	179	local.skinName != ""	pyLanding			When Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6158	TABTHREAD4	11	179	Property-Set	pyLanding	12		Step Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6157	TABTHREAD4	11	179	java	pyPortal.pyPortalSkin	11	GOOD	Step End	0.0000	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6156	TABTHREAD4	11	179	java	pyPortal.pyPortalSkin	11		Step Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6155	TABTHREAD4	11	179	Property-Set	pyPortal.pyPortalSkin	10	Skip Step /...	Step End	0.0000	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6154	TABTHREAD4	11	179	(param.pzSkinName != ...	pyPortal.pyPortalSkin		False	When End	0.0000	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6153	TABTHREAD4	11	179	(param.pzSkinName != ...	pyPortal.pyPortalSkin			When Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6152	TABTHREAD4	11	179	Property-Set	pyPortal.pyPortalSkin	10		Step Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6151	TABTHREAD4	11	179	java	pyLanding	9	GOOD	Step End	0.0040	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6150	TABTHREAD4	11	179	When Rule Evaluation	pyLanding		True	When End	0.0000	@baseclass pyMayOptim..	Pega-UIEngine 07-10-17
6149	TABTHREAD4	11	179	When Rule Evaluation	pyLanding			When Begin		@baseclass pyMayOptim..	Pega-UIEngine 07-10-17
6148	TABTHREAD4	11	179	When Rule Evaluation	pyLanding		True	When End	0.0010	@baseclass pyMayOptim..	Pega-UIEngine 07-10-17
6147	TABTHREAD4	11	179	When Rule Evaluation	pyLanding			When Begin		@baseclass pyMayOptim..	Pega-UIEngine 07-10-17
6146	TABTHREAD4	11	179	java	pyLanding	9		Step Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6145	TABTHREAD4	11	179	Property-Set	pyLanding	5	GOOD	Step End	0.0040	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6144	TABTHREAD4	11	179	local.skinName != ""	pyLanding		True	When End	0.0000	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6143	TABTHREAD4	11	179	local.skinName != ""	pyLanding			When Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6142	TABTHREAD4	11	179	param.pzUsePreference...	pyLanding		True	When End	0.0000	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6141	TABTHREAD4	11	179	param.pzUsePreference...	pyLanding			When Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6140	TABTHREAD4	11	179	When Rule Evaluation	pyLanding		True	When End	0.0000	Rule-Obj-Class pyUseDe...	Pega-Desktop 07-10-19
6139	TABTHREAD4	11	179	When Rule Evaluation	pyLanding			When Begin		Rule-Obj-Class pyUseDe...	Pega-Desktop 07-10-19
6138	TABTHREAD4	11	179	Property-Set	pyLanding	5		Step Begin		@baseclass GetWorkSty..	Pega-UIEngine 07-10-19
6137	TABTHREAD4	11	179	Property-Set	pyLanding	4	Skip Step /	Step End	0.0010	@baseclass GetWorkSty..	Pega-UIEngine 07-10-19

Click a line in the Tracer to view details about the event. The details for the step are presented in a new window. From this window, you can view the contents in memory at the time the event occurred. When you finish reviewing the event properties, close the window to return to the Tracer.



Properties on Page TraceEvent [6150]

Header	
Sequence	6149
Interaction	11
Timestamp	May 16, 116 - 12:37:49 14:19:19 (20160516T163749.035 GMT)
Elapsed Time	0.0000 (s)
Event Type	When End
Event Name	When End
Event Key	RULE-OBJ-WHEN @BASECLASS PYMAYOPTIMIZESKIN #20150213T181214.144 GMT
Thread Name	TABTHREAD4
Requestor ID	H16E37F94FCEE1905B0F2425D55818578
Correlation ID	H16E37F94FCEE1905B0F2425D55818578
Work Pool	TGB-HR-Work
Last Step	@BASECLASS GETWORKSTYLE #20150818T155516.374 GMT Step: 9 Circum: 0
Input	Activity=@baseclass.doUIAction
Ruleset Name	Pega-UIEngine
Ruleset Version	07-10-17
Standard	
Activity Number	179
Parameter Page Name	=unnamed=
Primary Page Class	Rule-Obj-Class
Primary Page Name	pyLanding
Step Method	When Rule Evaluation
When Status	True

Even the events that occur as you use the Designer Studio are logged and displayed in the Tracer. To suspend the logging of events in the Tracer, click **Pause**. While the Tracer is paused, the Pause button is replaced by a **Play** button. To resume the logging of events in the Tracer, click **Play**.

When you use the Tracer, you may want to focus on specific parts of your application. The Tracer provides several options to focus event logging available on the toolbar that runs along the top of the Tracer window.

Button	Function
Settings	Select the rule types, rulesets, and events to trace. Also identify when a step results in a Java exception or a status of Fail or Warn.
Breakpoints	Identify when the application reaches a specific step in an activity
Watch	Monitor a variable to detect when its value changes.

To trace events in another requestor session, click **Remote Tracer** to connect to the session.

COURSE SUMMARY

Next steps for system architects

System Architect Essentials 7.2 Summary

Now that you have completed this course, you should be able to:

- Apply Pega's principles of application design and development to deliver business applications that are Built for Change™.
- Use Pega Express to model the life cycle of a case that mirrors the way business people think about how work is completed.
- Directly capture business objectives to help ensure that business requirements are accurately captured, and that business and IT stakeholders share a common understanding.
- Use Designer Studio to refine and enhance the case life cycle design.
- Identify the tasks and responsibilities of the system architect on a Pega Implementation.
- Configure a case and case processing behavior.
- Create data classes and properties for use in a Pega application.
- Automate decision-making throughout an application to improve process efficiency.
- Design responsive user forms for use on any platform or browser.
- Design reports to deliver key insights to business users.
- Incorporate and manage reference data to allow applications to adapt to changing business conditions.
- Test your application design to analyze rule behavior and identify configuration errors.

Next Steps

Completion of Pega System Architect 7.2 helps prepare students for the Certified System Architect exam. To help you study for the exam, enroll in the CSA Practice Exam course in Pega Academy. Register for the exam.