

System Architect Essentials I

STUDENT GUIDE (v. 7.1)

Pega®
ACADEMY ▶

© Copyright 2015
Pegasystems Inc., Cambridge, MA
All rights reserved.

This document describes products and services of Pegasystems Inc. It may contain trade secrets and proprietary information. The document and product are protected by copyright and distributed under licenses restricting their use, copying, distribution, or transmittal in any form without prior written authorization of Pegasystems Inc.

This document is current as of the date of publication only. Changes in the document may be made from time to time at the discretion of Pegasystems. This document remains the property of Pegasystems and must be returned to it upon request. This document does not imply any commitment to offer or deliver the products or services provided.

This document may include references to Pegasystems product features that have not been licensed by your company. If you have questions about whether a particular capability is included in your installation, please consult your Pegasystems service consultant.

PegaRULES, Process Commander, SmartBPM® and the Pegasystems logo are trademarks or registered trademarks of Pegasystems Inc. All other product names, logos and symbols may be registered trademarks of their respective owners.

Although Pegasystems Inc. strives for accuracy in its publications, any publication may contain inaccuracies or typographical errors. This document or Help System could contain technical inaccuracies or typographical errors. Changes are periodically added to the information herein. Pegasystems Inc. may make improvements and/or changes in the information described herein at any time.

This document is the property of:

Pegasystems Inc.
1 Rogers Street
Cambridge, MA 02142
Phone: (617) 374-9600
Fax: (617) 374-9620
www.pega.com

Document Name: SAEI_71ML_6_StudentGuide_20141218.pdf

Date: 20141120

Table of Contents

Orientation.....	1
Course Overview.....	2
BPM and Case Management Overview.....	13
Introduction to BPM and Case Management.....	14
Accessing a PRPC Application.....	24
Tour of the Designer Studio.....	34
Start Building an Application.....	40
Introduction to the HR Services Application.....	41
Creating a New Application.....	53
Identifying Case Stages.....	65
Defining the Process.....	78
Adding Processes to Stages.....	79
Modeling the Process.....	90
Adding Business Use Cases To A Process.....	104
Review Of Defining The Process.....	117
Defining the Data Elements.....	123
Introduction to Data Elements.....	124
Introduction to Data Modeling.....	138
Setting Values of Data Elements.....	147
Inspecting Data Elements.....	157
Review of Defining the Data Elements.....	164
Enhancing the User Interface.....	172
Introduction to User Interface.....	173
Designing Layouts.....	201
Creating Repeating Layouts.....	215
Formatting Data Elements.....	226
Validating Data Elements.....	250
Review of Enhancing the User Interface.....	273
Accessing Data in the Application.....	302

Introduction to Data Tables.....	303
Accessing Data in the User Interface.....	314
Creating a Dynamic User Interface.....	337
Accessing External Systems.....	355
Review of Accessing Data in Application.....	384
Automating Business Policies.....	413
Introduction to Declarative Processing.....	414
Conditional Processing.....	429
Automating Business Processes using Decision Rules.....	437
Review of Automating Business Processes.....	452
Advanced Case Processing.....	459
Routing Cases.....	460
Case Stage Configuration.....	486
Creating Optional Process.....	499
Introduction to Task based Smart Shapes.....	506
Creating a Case Level SLA.....	517
Review of Advanced Case Processing.....	525
Documenting the Application.....	542
Documenting an Application.....	543

Module 01: Documenting the Application

This lesson group includes the following lessons:

- Course Overview

Welcome to PRPC: System Architect Essentials I



Welcome to PRPC: System Architect Essentials I, the essential, foundational course for application designers who want to learn the fundamentals of creating and maintaining Pega 7 solutions that address complex business needs.

In this course, students focus on the core element of the Pega 7 platform – PegaRULES Process Commander, or PRPC – to become competent in the technology and capabilities available in Pega 7.

Students use PRPC tools and technology to rapidly prototype a simple case-management application. Using an iterative development process, students create case stages and steps, implement processes, define properties, create a user interface, and create business rules and processes. Students can build upon these core solution development concepts in more advanced and/or specialized training.

Course Overview

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

- Understand the intent of System Architect Essentials I
- Communicate the course objectives and describe the intended audience

Getting to Know You

Who are you?

Name

Company

Role

What is your prior experience?

Pegasystems

General programming

Relational databases

How do you expect to benefit from this course?





Course Pre-requisites

Intended for System Architect candidates who want to create Pega 7 case-management solutions



Understand business processes, policies, and best practices



Some application development experience



Familiar with at least one development methodology

System Architect Essentials I is intended for System Architect candidates who want to create Pega 7 case-management solutions, and is not recommended for business users. To succeed in this course, students should have:

- Knowledge of the business processes, best practices, and policies in use at their company.
- Some application development experience.
- And familiarity with one or more project methodologies, such as Scrum, RUP, or Waterfall.

You will learn how to?



Add properties to a data model



Create /extend a process model



Configure UI controls



Implement business policies



Upon completion of System Architect Essentials I, students should understand and be able to use core PRPC features – flows, data elements, UI, decision rules – as members of a solution implementation team, performing tasks such as:

- Adding properties to an existing data model.
- Creating and extending a process definition.
- Modifying a user interface by adding and configuring basic UI controls.
- And implementing business policies with decision rules.

Course Agenda

Introduction to BPM and Case Management

Accessing a PRPC Application

Tour of the Designer Studio

Introduction to the HR Services Application

Creating a New Application

Case Stages, Processes, Modeling the Process

Adding Business Use Cases to a Process

Introduction to Data Elements

Data Modeling, Data Elements

Introduction to User Interface

Layouts, Formatting Data Elements, Validating Data Elements

Introduction to Data Tables

Accessing Data in the User Interface

Dynamic User Interface

Integration

Accessing External Systems, Accessing Data in Application

Course Agenda (continued)

Automating Business Policies

Introduction to Declarative Processing,
Conditional Processing, Decision Rules
Routing Cases

Case Stage Configuration

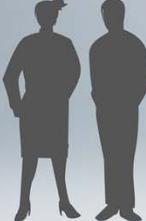
Introduction to Task Based Smart Shapes
Creating a Case Level SLA
Review of Advanced Case Processing

Documenting an Application

Certification



Completion of System Architect Essentials I satisfies the first eligibility requirement for the Certified System Architect (CSA) exam



SAE I **SAE II**

Completion of System Architect Essentials I satisfies the first eligibility requirement for the Certified System Architect (CSA) exam. Students can continue their learning with PRPC: System Architect Essentials II, which focuses on how to design Pega 7 solutions that enable users to manage complex cases with PRPC's Build for Change® technology.

Certified System Architect (CSA)



Benefits

- Industry credibility and recognition
- Demonstrates that you have acquired the knowledge and skills required to be an effective contributor to the design and development of Pega 7 applications

CSA Exam

- The Certified System Architect (CSA) Professional Certification Path is for developers and technical staff members who want to learn how to develop Pega applications. This certification provides a baseline measurement on your knowledge of PRPCReview of

Recommended path to success

- Attend PRPC: System Architect Essentials I and II
- Access Study materials and notes, Developer Help and PDN articles related to the course topics

CSA Certification



Administered by Pearson VUE (Virtual University Enterprises)

- World's leading test center network, with over 5,000 test centers in 175 countries
- Register online at <http://www.vue.com/Pegasystems>

Exam Format and Details

- Answer 73 Questions in 90 minutes
- Passing score of 70% (49 correct answers)
- Results are available immediately
- May be taken 3 times per year (three day wait period between exams)

Recommended enablement path to success

- Attend PRPC: System Architect Essentials I and II
- Study materials, notes, Developer Help and PDN articles related to the course topics

Completing the Exercises

To learn software, we must use the software. Most lessons include a hands-on exercise that allows us to practice what we just learned. By completing the guided exercises in this course we create an application in PRPC.

The lessons include an exercise that provide a scenario which describes the business problem that needs to be solved.

You will login into the Exercise System as a System Architect.



Module 02: BPM and Case Management Overview

This lesson group includes the following lessons:

- Introduction to BPM and Case Management
- Accessing a PRPC Application
- Tour of the Designer Studio

Introduction to BPM and Case Management

To understand the value in Pega solutions, we must first understand the benefits of – and differences between – Business Process Management and Case Management

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

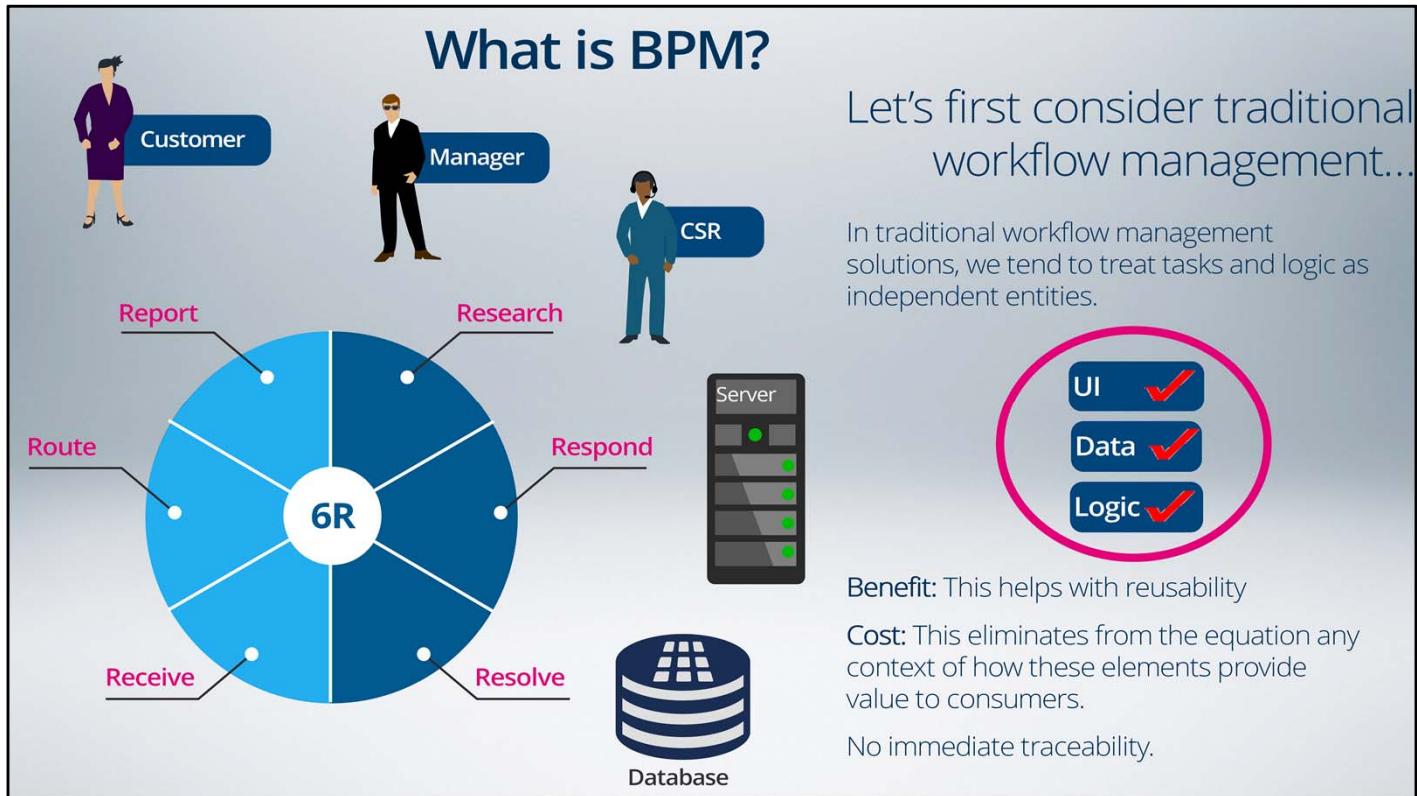
- Define the terms Business Process Management (BPM) and Case Management
- Identify key BPM and Case Management concepts
- Understand how combining elements of BPM and Case Management can yield solutions to address complex, evolving business needs

Business Process Management (BPM)

- Manage Processes – creation and resolution of work activities
- Improve Business Performance – increase net productivity and ROI
- Improve Operational Agility – highly responsive to changing conditions

The discipline of **managing processes**, rather than tasks, **to improve business performance outcomes and operational agility**

To build a Pega solution, we must first understand how PRPC works. To understand how PRPC works, we start with the principles of Business Process Management or BPM.



All business processes are a combination of tasks and logic needed to perform work.

Tasks generally involve the gathering and reviewing of data – either by individuals working through a user interface or an automated exchange with an external data store – and logic represents the selection of the optimal outcome for a particular task, based upon the information – our data – at hand.

Taken together, tasks and logic allow us to process work – a transaction we need to complete – through all of the phases of its lifecycle: receiving it, routing it, researching it, responding to it, reporting on it, and, ultimately, resolving it.

When developing a traditional workflow management solution, we tend to treat our tasks and logic as independent entities. This helps to make these elements reusable, but eliminates any context – how they provide value to consumers – from the equation.

What is BPM?

Customer **Manager** **CSR**

Processes span organizational boundaries, linking together people, information flows, systems and other assets to create and deliver value to customers and constituents.¹

Database **Server**

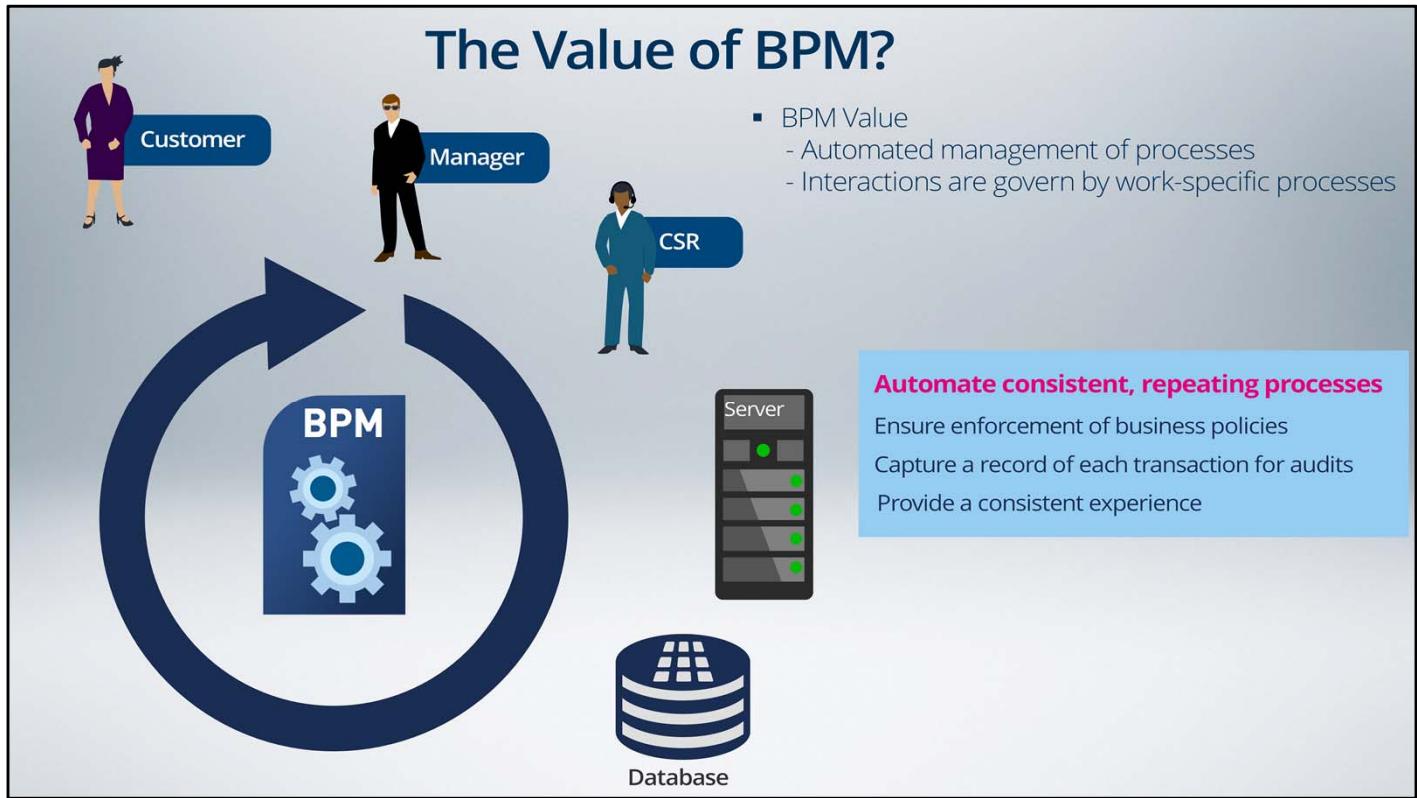
Process

- BPM is a structured approach
 - employs methods, policies and metrics
- Pega 7 Process
 - links together UI, data and logic
 - delivers value, improved capabilities and greater efficiencies

¹ Gartner IT Glossary, retrieved 6/18/2013

The discipline of BPM involves managing the process, or workflow, as a whole, rather than managing the individual components separately. Processes span the boundaries within an organization to link its assets, which in turn creates and delivers value to the consumer of that organization's services. BPM is a structured approach – one that employs methods, policies, metrics, management practices, and software tools to manage and continuously optimize processes within an organization.

When implementing a Pega solution, we start with the process, and use it as a tool to link together our UI, logic, and data. We no longer focus solely on developing capabilities to create and process work. Now, we also need to focus on how these elements come together to deliver value – new or improved capabilities, or greater efficiency for consumers.



So, with BPM, an organization establishes automated management of its processes. All interactions of a specific type – for example, updating a mailing address on a bank account or changing coverage options on an insurance policy – are governed by the same work-specific process.

Let's say that a customer wants to change an address on their account.

They call the bank's customer service department or log in to their account on-line and the system – with or without the assistance of a representative – guides the customer through a specific process with a defined start and end.

Because this process is so deterministic – known, quantifiable inputs lead to a defined set of outputs, according to known processing paths – the business can codify and automate the process to ensure a consistent experience from one customer to the next.

And since each transaction has been recorded, it's easy to determine which participant performed what action, and when.

This ultimately improves the outcome for consumers, by allowing the bank to analyze the process to improve its overall efficiency.

What is Case Management?

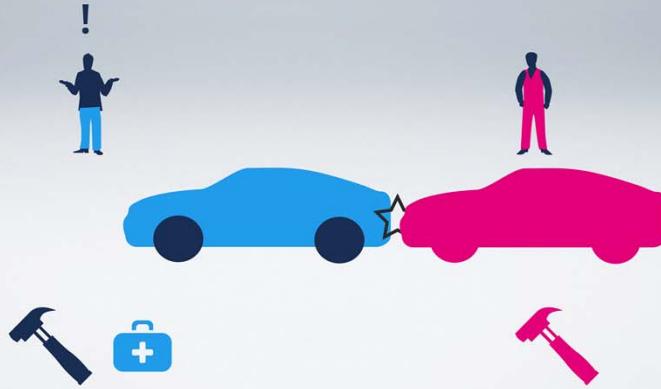
- Asynchronous Management
 - Processes create and resolve work
 - Ad-hoc content requires research/respond
- Supports specific business objectives – research/respond/resolve

Asynchronously manage a collection of tasks, actions, **processes** - even ad-hoc content - **to support a specific business objective**

Enterprise workflows are more complex than a simple process, however. When organizations interact with a customer or client, they often need to coordinate different types of work performed by multiple parties to resolve the transaction. To manage these complex workflows, Pega solutions incorporate case management principles as well.

Case Management

- Case Type (claim work item)
 - a series of process (assess physical damages and medical injuries)
- Processes overlap (in sequence or parallel)
- Involves different actors to resolve the claim



Let's look at an insurance claim.

When we file a claim, it launches a series of processes to assess damages, such as physical damage to vehicles and other property, as well as medical injuries sustained by any of the parties.

All of these processes overlap and involve different actors working together – some in sequence, others in parallel – on several work items to resolve the claim.

Case Management

Cases are:

- Projects, transactions, services or responses
- Opened and closed to resolve a problem, claim, request, or complex activity



Cases involve:

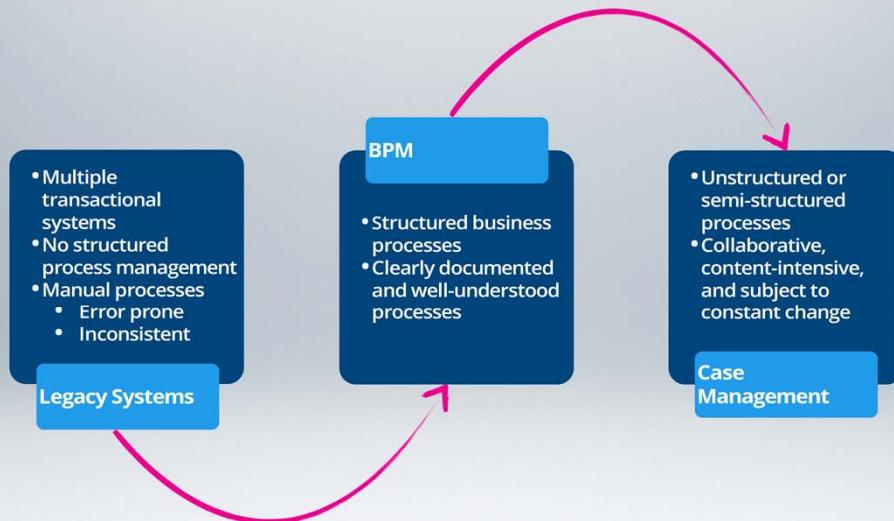
- Multiple persons, inside and outside of the organization
- Multiple processes that include documents, messages, and other exchanges - even other cases



By incorporating a more-flexible Case Management framework, our solution allows us to manage work holistically, rather than as a set of unrelated processes performed by independent parties in an uncoordinated manner.

And this allows us to track and manage these different types of work, or subcases – regardless of whether the work is performed by one person or several, either sequentially or in parallel – according to our needs.

Case Management with BPM



When businesses combine the standardization of BPM with the flexibility of Case Management, they build solutions that return greater value than if they rely on either approach alone.

Kitson, Ravisanskar, and Soudamini. "Case Management – Managing chaos: unstructured processes and dynamic BPM" CapGemini. 19 Jun. 2012
<http://www.capgemini.com/resources/managing-chaos-unstructured-processes-and-dynamic-bpm>

So, what happens when we put BPM and Case Management together?

Without *any* structured approach, business processes are manual, inconsistent, error-prone, and potentially spread out over multiple systems.

BPM works best when all of the elements of the process can be accurately modeled. It allows the business to create structured, repeatable processes that are clearly documented and well-understood by the parties that need to execute them. This allows PRPC to “understand” the process, and even perform those steps that lend themselves well to automation – such as enforcing business policies, assigning tasks, and sending correspondence.

Case Management adds value by filling in the gaps when asynchronous, unanticipated, or unpredictable events affect the processing of work. This allows the business to create a more-flexible solution that processes work better than a BPM system.

When businesses combine the standardization of BPM with the flexibility of Case Management, they build solutions that return greater value than if they rely on either approach alone.

BPM, Case Management, and PRPC



Case Management with BPM in PRPC

- Automate system actions and responses
- Provide flexibility for manual, optional, or conditional processes
- Create subcases as needed and manage them to resolution
- Report on all elements of a case
- Organize processes to provide a “big picture view” of a workflow



So, now that we have a better understanding of BPM and Case Management, what do we need to know about how PRPC solutions can leverage both?

By developing processes in accordance with BPM principles, we create repeatable, automatable processes that drive operational efficiency.

By implementing a solution as a case, we provide a framework to manage its workflow – not just the automatic processes, but also manual, optional, and conditional processes.

Since our solutions are implemented as cases, we can more easily spawn them from other cases, making it easier to manage tasks across the entire workflow.

We can report on the progress of each case – including subcases – separately, improving visibility into the flow of work.

Altogether, these capabilities provide a big picture view of the case – including its components – which we can't get from just BPM alone.

Accessing a PRPC Application

Users must be created and provisioned with appropriate roles so they can perform their tasks.

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

- Define the terms OperatorID, Access groups and Portals
- Understand how they are related and how they are configured
- List standard Access groups and Portals

Users - Roles

The diagram illustrates six user roles represented by stylized human figures:

- System Architect:** A woman in a blue dress and a man in a blue suit.
- Senior System Architect:** A woman in a blue top and dark skirt, and a man in a blue suit.
- Lead System Architect:** A woman in a dark blue blazer and a man in a dark blue suit, enclosed in a blue rounded rectangle.
- Business Architect:** A woman in a red dress and a man in a red suit.
- Lead Business Architect:** A woman in a red top and dark skirt, and a man in a red top and dark pants, enclosed in a blue rounded rectangle.
- System Admin:** A woman in a teal dress and a man in a teal suit, enclosed in a blue rounded rectangle.

Note: Creation and configuration of these records are not covered in this course since they are typically maintained by Lead System Architects or System Administrators

Welcome to the lesson on Accessing a PRPC Application. In this lesson, we'll discuss how users access an application built in Pega 7. Like any enterprise application it requires that we set up users, their roles and their privileges (or access rights). This lesson provides a brief introduction on the terminology used in Pega 7 for users, roles and the privileges. We'll look at some examples to understand the relationship of these concepts. Creation and configuration of these records are not covered in this course since these are usually done by Lead System Architects or System Administrators.

Users - Roles



A user logging into a Pega 7 system is called an Operator and their information is saved as “Operator IDs”. An Operator ID can be created manually by administrators or automatically. Each user logging in to the application must have a corresponding Operator ID record.

Demo

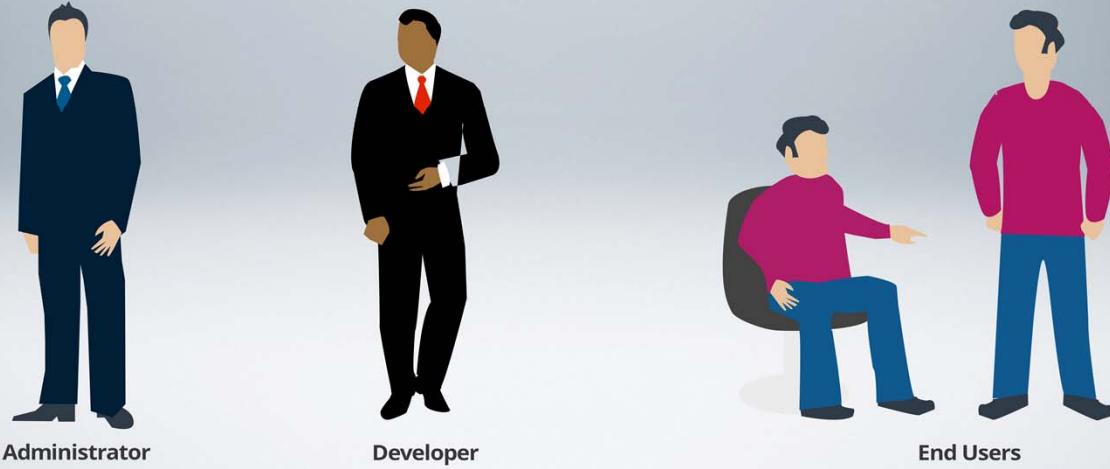


Operator ID

This is a sample operator ID record that is in our application. The Operator ID stores the contact information and other personal details of the user, the organizational setting for that user to determine what cases they can work on, their managers and lastly the Security settings in terms of the passwords and so on.

Other Users

- Pega 7 installation comes with a default administrator operator ID that can be used to setup other users.



A standard Pega 7 installation comes with a default administrator operator ID that can be used to setup other users. There are different sets of users accessing the application – an administrator who is involved in setting up users and their roles, developers who are creating applications and end users who use the application created by the developers.

Other Users

1 Access Group → 1 Application

1 Access Group → 1...* Role(s)

- Access group is assigned to each operator

- Provides access to the application and defines the roles



System Architect

1 Operator ID → 1...* Access

Group(s)

Access Group Record

An Access group is a record assigned to each operator. Access groups provide two types of accesses for the operator – the application the operator can access and the roles that an operator can perform in that role.

Portals

- Defines the workspace of an operator
- Determines what is displayed to the operator



Lastly the Portal defines the workspace of an operator. The Portal determines what is being displayed to the operator after they login to the system.

Demo



Access Group

Now that we have defined the terminology let's see how they are related. . In an operator ID record, the access groups are identified. An Operator can belong to multiple access groups. When the operator logs in to the system it uses the default access group. Users have the option to switch to a different access group if there is more than one access group.

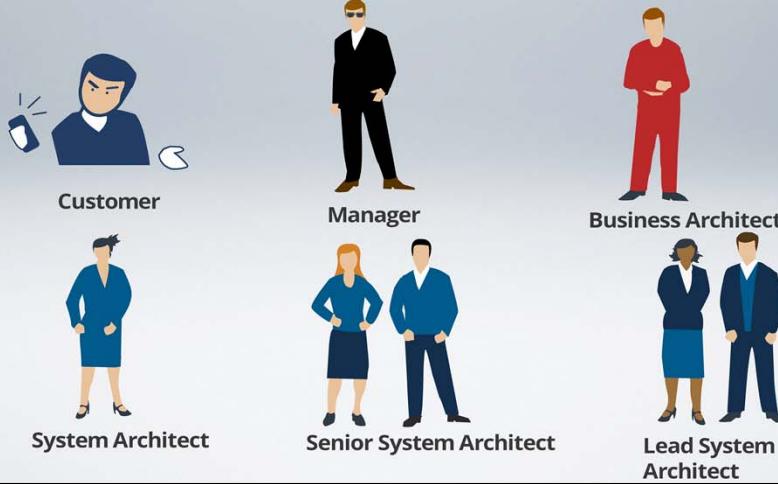
The Access group references a record named Application which lists the components of the application. We'll learn more about what represents an application in the lessons that follow.

The Access group record lists the portal rules that the user can view. The selected portal is the primary portal and the secondary portals provide the ability for users to switch to a different workspace if required. In this case all user portals are added to the developer operator so they can verify how it looks during development.

Access groups also reference Access roles, Pega 7 comes with a set of standard Access roles for system administrators, system architects, end users and managers. In most cases, the application also has specific roles built by lead system architects.

Users

- Work users are the end users of our applications
- Work managers are the people who manage the work users
- Business Architects are the users who capture requirements and define processes
- System Architects are the users who build the applications



Work users are the end users of our applications. They create, update and resolve cases. For example, a work user can be a customer who is entering a purchase request or a Customer care representative who is entering the purchase request for a customer on the phone.

Work managers are the people who manage the work users. They manage the workloads, monitor the activity of the users reporting to them and they can also run reports which help them in managing the work users.

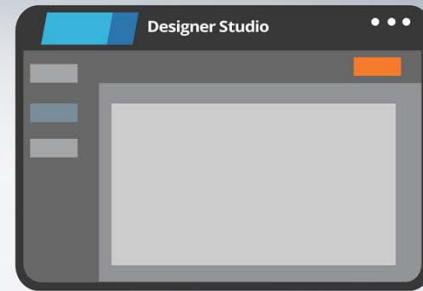
Business Architects are the users who capture application requirements and define business processes. They primarily use the application to capture business requirements, model processes and design case stages and so on.

System Architects are responsible for building the application, they are also referred to as application developers or application architects.

These roles are shipped as a point of reference and in most cases project teams create additional roles that are unique for the company.

Standard Portals

- Pega 7 comes with a standard portal rule for Case Managers
- Pega 7 comes with a portal (workspace) for developers called Designer Studio



Pega 7 comes with a standard portal rule for Case Managers and provides a point of reference for developers to create customized versions for end user portals. Creation of custom end user Portal is done by a senior system architect.

Pega7 comes with a portal (workspace) for developers, this is called the Designer Studio. Typically this portal is used as-is and very little customization is done. If customization is required, it is done by project teams. Designer Studio has lot of features. We'll primarily use this portal for doing exercises in this course. Please take a look at the Tour of the Designer Studio lesson to learn more about the Designer Studio.

In this lesson, we learned that: Operator IDs store information about the users accessing the application, Access groups are the groups to which users belong, the system comes with Standard Project roles and User Portals, the Case manager portal provides a starting point for developers to create customized portals end users, and we also learned about the developer portal (Designer studio)

Tour of the Designer Studio

Designer is the portal shipped with developers using Pega 7

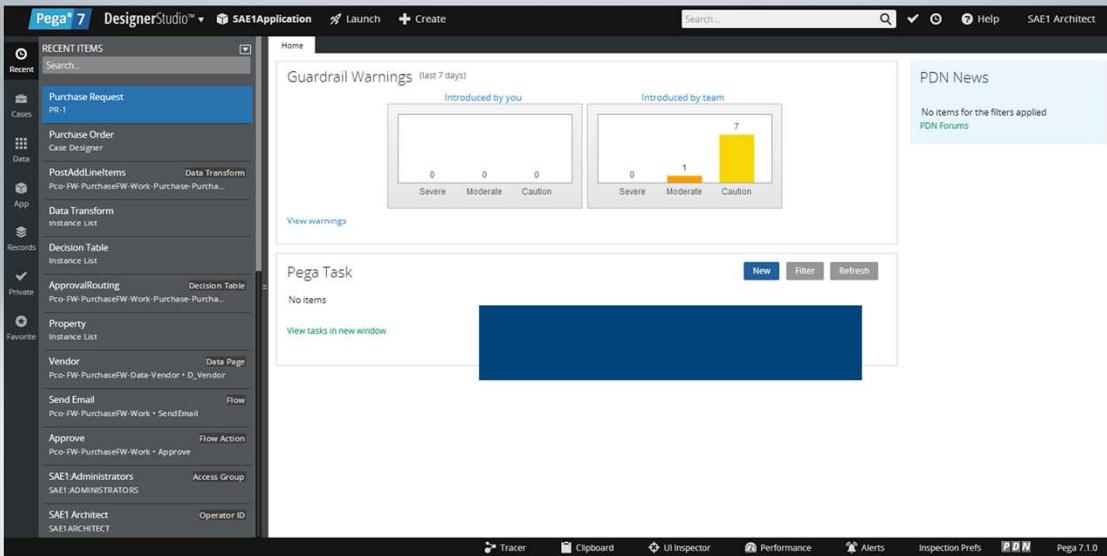
At the end of this lesson, you should be able to:

- Navigate the Designer studio
- Use different explorers
- Access help and other tools

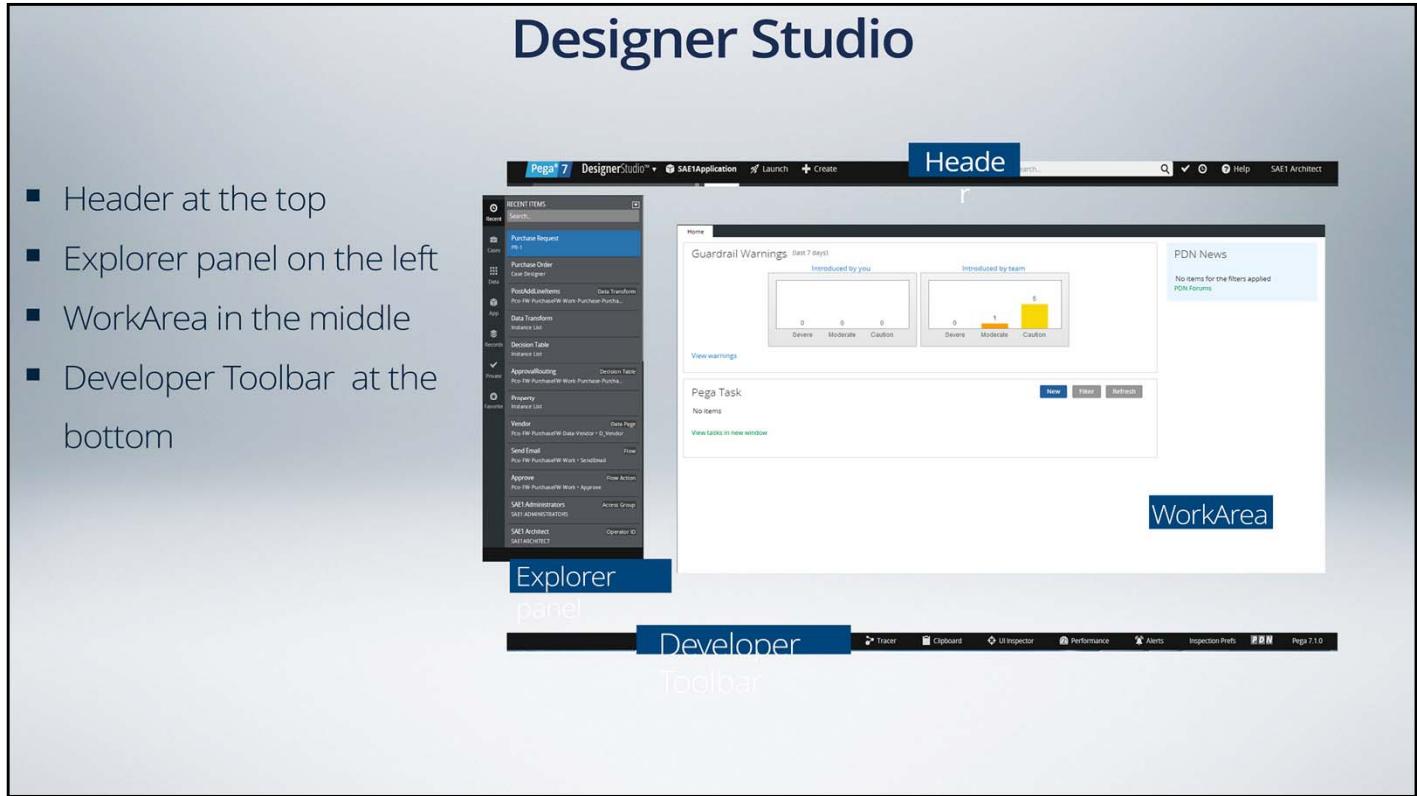
Designer Studio is the portal shipped for developers using Pega 7. We can access the designer studio from any web browser like chrome or IE. When users log into a Pega 7 system the system verifies their identity and then displays the designer studio if they are configured as a developer. Let's take a closer look at the designer studio to see what it offers for developers.

Designer Studio

- Designer Studio has four components



Designer Studio has four components –



Header at the top, explorer panel on the left, WorkArea in the middle, and the Developer Toolbar that displays as a footer.

These four components offer assistance to developers to them navigate through the designer studio. Let's look at each of these areas individually.

The screenshot shows the Pega Designer Studio interface. On the left is a sidebar with sections for Recent, Cases, Data, App, Records, and Favorites. The Favorites section is expanded, showing items like ApprovalRouting, Property, Vendor, Send Email, Approve, SAE1 Administrators, SAE1 ARCHITECT, and SAE1 ARCHITECT. The main area displays 'Guardrail Warnings (last 7 days)' with two charts: one for 'Introduced by you' (0 Severe, 0 Moderate, 0 Caution) and one for 'Introduced by team' (0 Severe, 1 Moderate, 7 Caution). A blue callout box in the center says 'Now let's take a closer look...'. The top navigation bar includes tabs for DesignerStudio™, SAE1Application, Launch, Create, and Help, along with a search bar and user account information.

Layout 2 - Sub Header

Demo



Navigate the Designer Studio
Accessing the Explorer pane
Accessing Help
Using the tools menu

Exercise: Use Designer Studio to Create and Process a Case



Module 03: Start Building an Application

This lesson group includes the following lessons:

- Introduction to HR Services
- Creating a New Application
- Identifying Case Stages

Introduction to the HR Services Application

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

- Understand the HR solutions to be built for SAE Corp in two releases:
 - SAE I (Candidate)
 - SAE II (New Hire Onboarding)
- Review the objectives of the Candidate case



Candidate

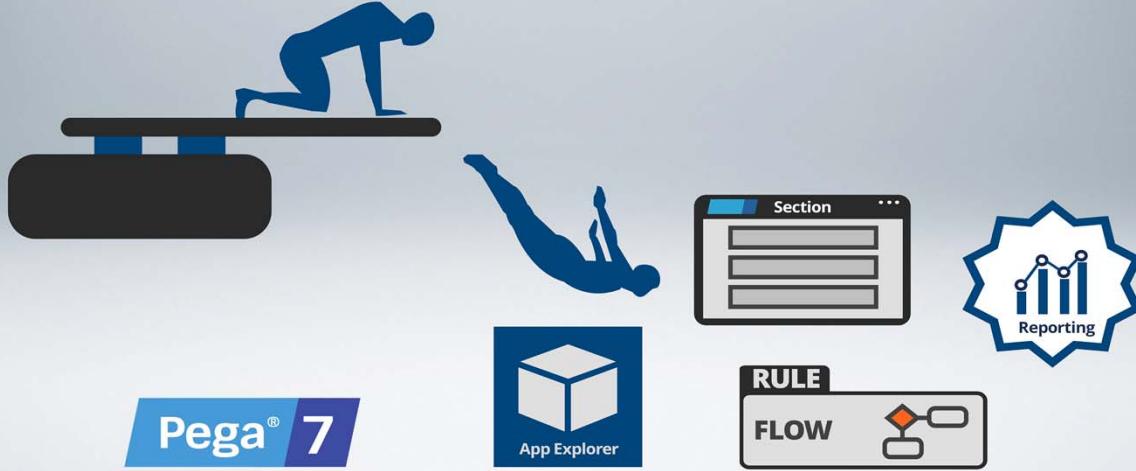


Onboarding



Introduction to the HR Services Application

- The focus is on how to solve business problems and to become familiar with the capabilities as we develop a solution
- SAE Corporation – we will manage the job application and onboarding process



To learn how to use software, the best approach is to jump in and start using the software. This allows us to become more familiar with its capabilities and limitations, and helps us to understand how to combine the features and tools that software provides to create valuable solutions that solve business problems.

Up to this point, we've focused on a basic conceptual understanding of the Pega platform – explaining how its process management and case management capabilities allow businesses to develop solutions that solve tough business problems.

The remaining lessons in this course – and those in the System Architect Essentials II course – focus on how to use these capabilities to solve business problems. To help us become more familiar with these capabilities, we develop solutions to solve two business problems for a fictitious corporation, SAE Corporation: managing their job application and onboarding processes.



To start, let's discuss the solutions that we'll build out. These are simplified versions of solutions that might be developed to address similar needs on a real-world implementation.

In this course SAE I – we develop a proof-of-concept solution to manage applicants for open positions at SAE Corp.

For this solution, we implement a single case type, designed to provide experience with the basic capabilities of the Pega platform and reinforce its foundational concepts.

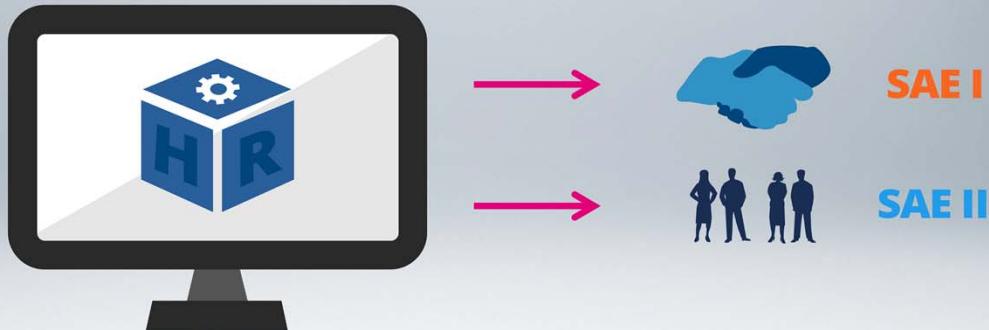
These concepts – tools; rule types such as flows, user interfaces, properties, and decisions; and navigating the Designer Studio – are ones that System Architects must understand to become productive members of an implementation team.

In System Architect Essentials II, we extend our application by implementing a solution to manage the on-boarding process for new hires at SAE Corp.

The solution built in System Architect Essentials II is more sophisticated than the Candidate solution, involving multiple case types.

We build upon the basics covered in System Architect Essentials I by learning design principles and best practices to create a more robust solution. We also reuse elements of the Candidate solution, to leverage existing functionality rather than re-implement it.

The Complete HRServices Application



These two solutions comprise the HRServices application. When employees of SAE Corp log in, they can access both solutions and create the case that matches their need. (In a real-world implementation, our solutions would create cases automatically – either when a candidate expresses interest in a position, or when they accept a job offer. These options are more complex, and beyond the scope of these introductory courses.)

HRServices Application



The Candidate Case



Collect Candidate Information

Now that we understand the two solutions we need to build – and how they relate to each other – let's focus on the Candidate case that we implement in this course.

Our solution should solve a simple business problem: how can I – as a recruiter, hiring manager, or HR partner – manage the processing applicants for an open position at SAE, which involves the following tasks:

- Collecting information about the candidate,

HRServices Application



The Candidate Case



Match candidate to position:

 Senior Accountant

Project Manager

Account Executive



- Matching the candidate to an open position,

HRServices Application



The Candidate Case



Schedule interview:



- Scheduling an interview with a hiring manager,

HRServices Application

The Candidate Case

Collect and process feedback:



1

2

3

4

No

Yes

- Collecting and processing the feedback upon which a hiring decision is made, and

HRServices Application

The Candidate Case



Extend offer



- Extending an offer to the candidate.

HRServices Application

The Candidate Case



Hiring policies to be considered:



Tech positions require samples of past work



All candidates pass background check



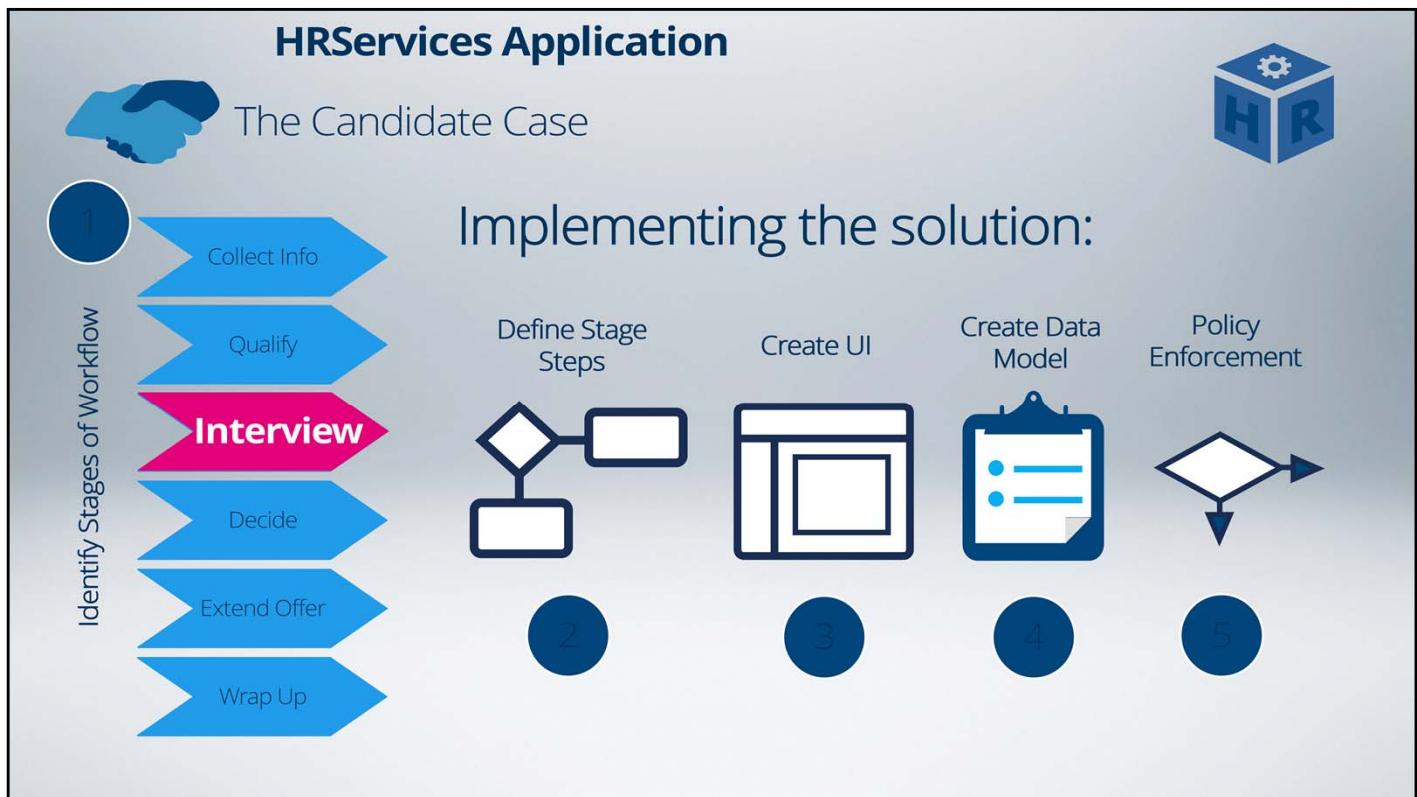
Compensation offer is consistent with position guidelines



Candidates informed of their status via email

Our solution should also reflect the hiring policies currently in effect at SAE Corp:

- Candidates for technical positions are required to submit a sample of their work,
- The hiring decision for a candidate is subject to their passing a background check,
- Compensation offered to a candidate must be consistent with the guidelines established for the position,
- Candidates are informed of their status – either rejection or approval – by email.



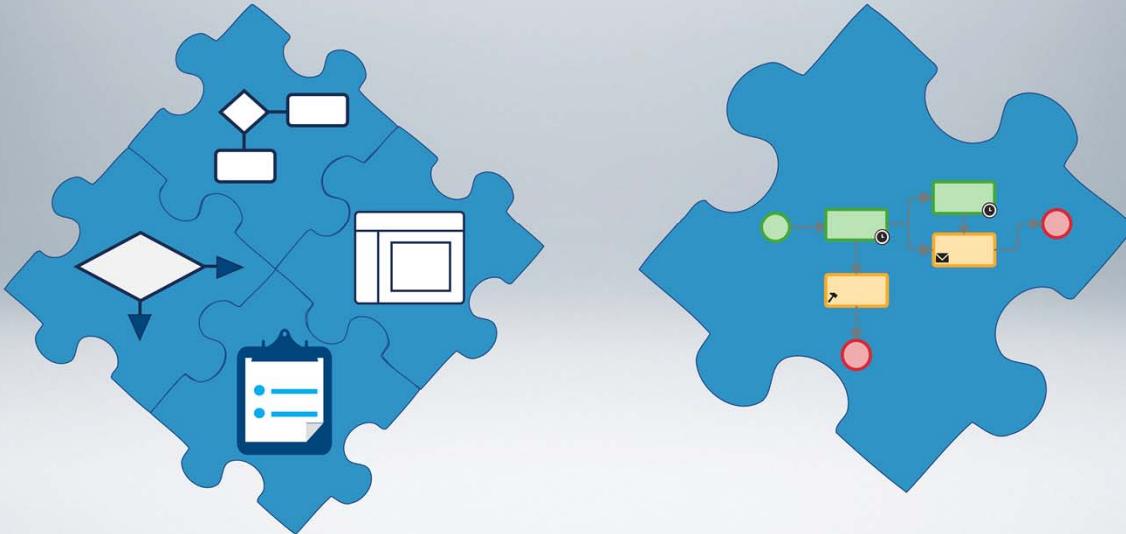
To implement this solution, we must first identify the stages of the workflow.

We then need to define the steps of each stage – including conditional or alternate steps – create the user interface for those steps, and create a data model to store and present the correct information at each step.

We also need to implement the policies that govern the process, such as automated decisions and assignment routing.

HRServices Application

Our lesson plan will cover this one piece at a time...



In the following lessons, we create a solution that manages these tasks and enforces SAE's hiring policies, one piece at a time.

Each lesson presents a different capability, one which is reinforced in exercises that guide us through creating and updating elements of the Candidate case.

Every few lessons, we analyze our progress by reviewing recently completed lessons and exercises. These reviews help us to further reinforce the concepts taught in the corresponding lessons by focusing on the overarching themes that tie the lessons together, such as process definition, data modeling, and policy enforcement.

Create a New Application

The New Application wizard – intended for and used by LSAs – allows us to create a starting point for application development.

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

- Describe how the New Application wizard can be used to create an application
- Review the application created by the New Application wizard

Traditional Application Development

- Traditional Application Developers write a lot of code

```

-----  

---- SETCLASS CLONES THE BASIC OBJECT CLASS TO CREATE NEW CLASSES  

-----  

-- Supports INHERITANCE  

-----  

-- EVERYTHING INHERITS FROM THIS BASIC OBJECT CLASS  

BaseObject = {  

    super  = nil,  

    name   = "Object",  

    new    =  

        function(class)  

            local obj = {class = class}  

            local meta = {  

                __index = function(self,key) return class.methods[key] end  

            }  

            setmetatable(obj,meta)  

            return obj  

        end,  

        methods = {classname = function(self) return(self.class.name) end},  

        data   = {}  

    }  

function setclass(name, super)  

    if (super == nil) then  

        super = BaseObject  

    end  

    local class = {  

        super = super;  

        name  = name;  

        new   =  

            function(self, ...)  

                local obj = super.new(self, "__CREATE_ONLY__");  

                -- check if calling function init  

                -- pass arguments into init function  

                if (super.methods.init) then  

                    obj.init_super = super.methods.init  

                end  

                if (self.methods.init) then  

                    if (tostring(arg[1]) ~= "__CREATE_ONLY__") then  

                        obj.init = self.methods.init  

                        if obj.init then  

                            obj:init(unpack(arg))  

                        end  

                    end  

                    return obj  

                end,  

                methods = {}  

            }  

            -- if class slot unavailable, check super class  

            -- if applied to argument, pass it to the class method new  

            setmetatable(class, {  

                __index = function(self,key) return self.super[key] end,  

                __call = function(self,...) return self.new(unpack(arg)) end  

            })  

            -- if instance method unavailable, check method slot in super class  

            setmetatable(class.methods, {  

                __index = function(self,key) return class.super.methods[key] end  

            })  

            return class  

    end
}

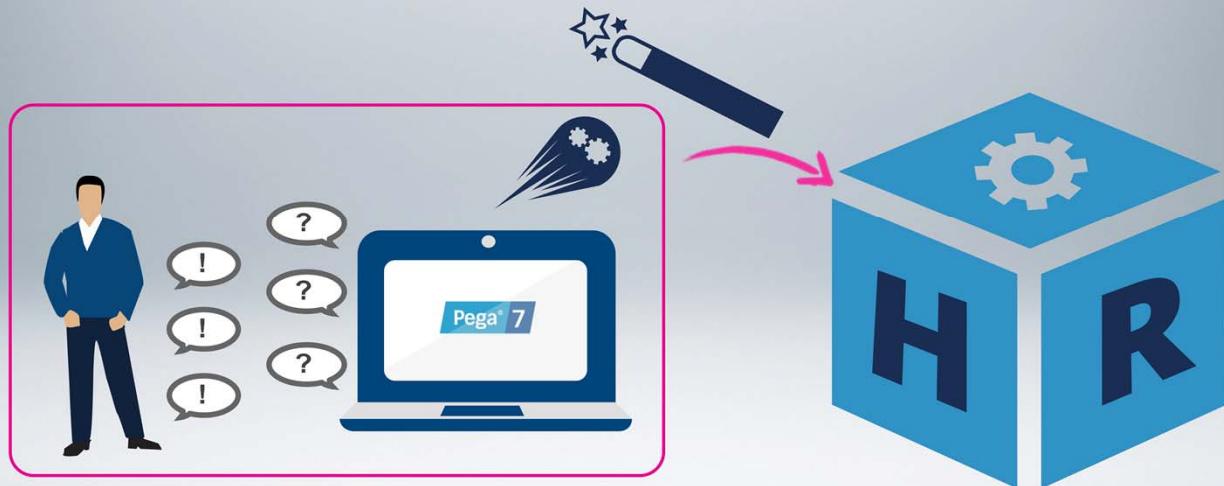
```

Now that we have learned the basics of Pega 7 and explored the Designer Studio, we can begin to develop an application.

Application developers using traditional programming tools and methods need to perform a lot of configuration before they can write even a single line of code. And they generally need to write a **lot of code**.

Application Development

- Pega Application Development...is different

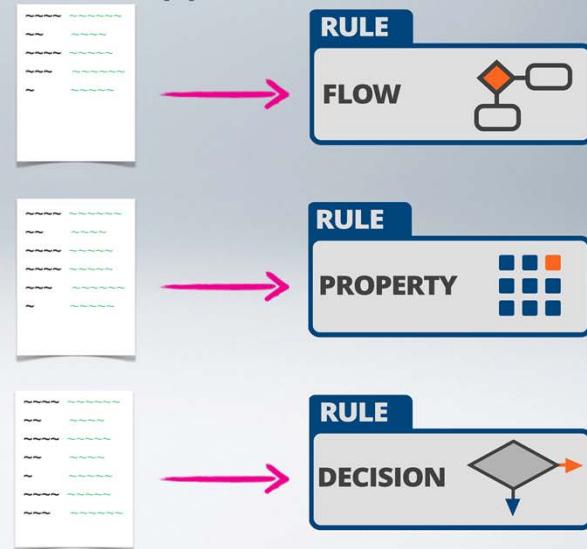


In Pega 7, application development is ... different.

First, we have tools available that help us configure things when we start – such as the New Application wizard, which creates a structure for our application. We provide some basic information – answers to a few questions, like we would in an interview – and the wizard creates the core of our application.

Application Development

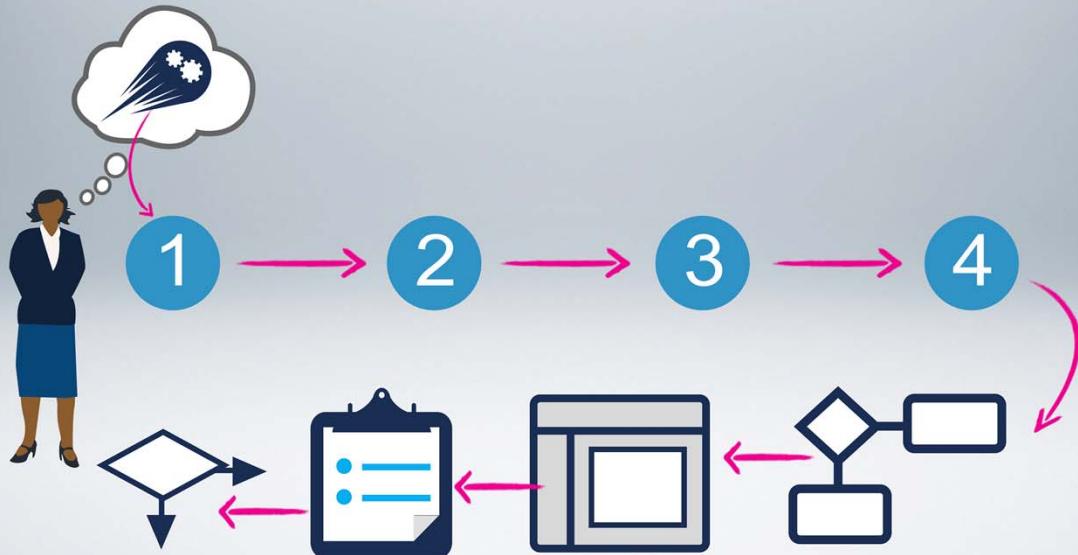
- Configure the elements of an application with rules
- Rules make up the model of an application



From this starting point, we can configure the elements of our application. Rather than writing lines upon lines of code, Pega 7 application design involves completing forms to create elements – called rules – that make up a model of our application. And, like a model, we can explore it, experiment with changes, and review the results. When we complete our modifications and tests, we can release our application to end users.

Creating an Application

- The New Application wizard coordinates a four-step process



Our first step is to create our application. To do that, we use the New Application wizard.

Normally, creating an application – a full-scale, reusable, enterprise application – is the responsibility of a Lead System Architect, or LSA. But since our project lacks an LSA to run the wizard – and we want to build a proof-of-concept application, anyway – we can create our own application. We need to make some assumptions and accept some guidance, but for the purposes of this course, we can run the wizard ourselves.

The New Application wizard coordinates a four-step process that creates the basic structure of our application. Once we complete it, we can start to define processes, user interface features, data elements, and business logic – the instructions that control how our application works.

Creating an Application

**1**

Basic information: application name, description, organization, etc.

**2**

Business objectives: what do we hope to accomplish?

**3**

Define cases: describe the transaction to model.

**4**

Define data: describe the information we need to collect.

First, we need to provide some basic information about our application, such as its name and a brief description of its purpose, and the name of our organization. The wizard uses this information to create the basic frame of our application, much in the same way that we would build a house by starting with a foundation and a frame.

Next, we can capture the business objectives for our application – these represent solutions to the problems we want to solve. For example, if management feels that employees waste money by not using preferred vendors during the purchase requisition process – vendors who agree to provide the company with a discount on pricing – we might define an objective such as “Cut costs by 15% by use of preferred vendors”. We do *not* need to provide all of our objectives at this point in time – we can still add them later – but doing so provides a basic understanding for everyone involved in application design.

Once we complete our objectives, we can define our cases. These are the business transactions we want to perform, such as purchase requests and purchase orders.

And last, we can define data objects. These data structures represent the data needed to process our case.

Once we perform those four steps, we complete the wizard and are ready to create our application.

Demo



Describe the New Application wizard

Now that we understand the wizard, we can use it to create our proof-of-concept application. So, first, we want to start the wizard. We can access the wizard from the Application menu by clicking **New Application**.

For the purposes of this demonstration, let's focus on one particular case – the Purchase Request – even though our demonstration application includes other cases. We also focus only on the minimum of information needed to perform the accompanying exercise. Remember that the New Application wizard is a tool intended for LSAs, and not system architects. For us, the wizard is a means to an end – an application that we can configure throughout the remainder of this course. For now, we want to focus more on the output of the wizard, rather than the steps we take through it.

The first step of the wizard prompts for basic information about our application. Here, we provide an application name and description, along with the name of our organization.

We also need to provide two other pieces of information – information that would normally be determined by an LSA. For our application, we'll use **PegaRULES** and **Implementation only**. The explanation for these selections is outside the scope of this course, but we will cover them in System Architect Essentials II.

The second step prompts us to provide business objectives. A business objective represents the value that a solution provides, and is determined by business analysts. We can enter two: one to decrease processing time by 25%, and another to cut costs by 15%.

The third step prompts us to specify the case we want to implement, along with a brief description. In our example, we want to implement a purchase request.

Enter **Purchase Request** and a description: Request for H/w, S/w, Telecom, etc. that goes through

approvals based on authority matrix. Once approved will generate a Purchase Order for fulfillment.

The fourth and last step prompts us to specify the persistent data objects we anticipate a need to implement.

Once we finish, we can create our application. The wizard creates a basic framework for us which we can configure into a functional solution.

Pega 7.1.6 Update Notes

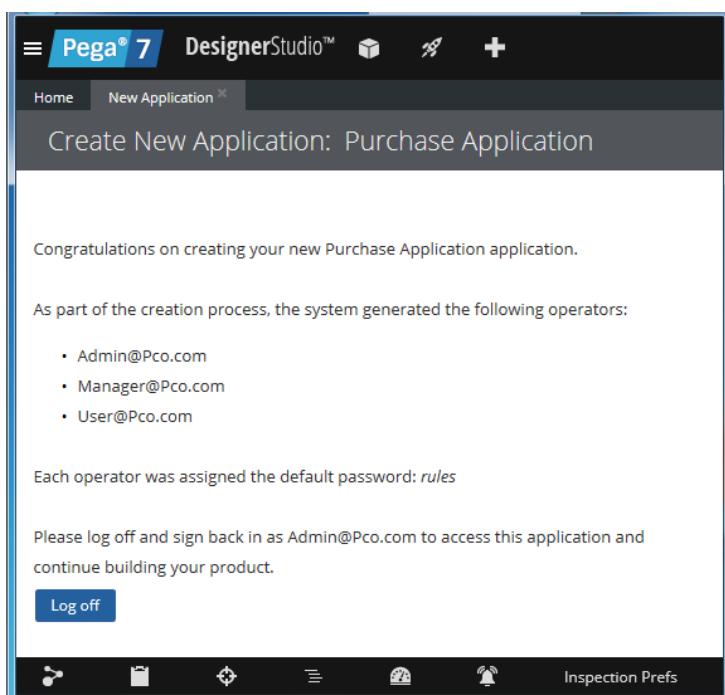
Exercises will use Operators created by App Express

From the Pega 7.1.3 release notes:

The Organization Setup Wizard has been retired in PRPC 7.1.3. This feature was previously launched from the Designer Studio > Org & Security > Organization > Organization Setup menu. Users are encouraged to run the New Application Wizard (also known as Application Express) instead.

If the organization that the operator enters in the first step of App Express doesn't already exist, it will be created as part of the new application. The exercises now use the operator IDs created by the wizard.

If the Administrator operator is used to create the new application, you will see the following screen when App Express has completed.



Demo

- **Describe the New Application wizard**
- **Review the application created by the wizard**

Once we complete the New Application wizard, PRPC creates our application and refreshes to display the Application Overview. This landing page presents basic information about our application, which we may need to review on occasion. If we do, we can open the Overview from the Application menu.

This is where we see the application name and description we entered in step one of the wizard, and where we can find the business objectives we entered in step two of the wizard.

The Application Overview also tells us how many requirements and specifications have been created for our application. Right now, we have none for our application, but for a real-world solution, business analysts would provide this information and we would see larger numbers as we begin adding features and functionality.

Finally, the case we created appears at the top of the Application Overview, along with the description we provided.

We can also locate our case by using the Case Explorer – which we will see in a minute. But before we do that, we can switch to the Data Explorer, where we see the data object we created in step four of the wizard. The Data Explorer lists all of the data objects and data pages created as part of our application. For now, we don't need to worry about data pages. We will come back to them later in the course.

And if we switch to the Case Explorer, we can see the case we created in step three. The Case Explorer displays a hierarchy of the cases in our application, which allows us to identify independent cases and dependent subcases.

When we click on a case, we open it in the Case Designer. Here, we can see a set of default stages and steps. And that brings us to our next task: to add stages and steps to our case to model its lifecycle.

Exercise: Create the HR Services Application



Identifying Case Stages

The first step in building a solution is to deconstruct a case into stages, which we can further deconstruct into processes and assignments

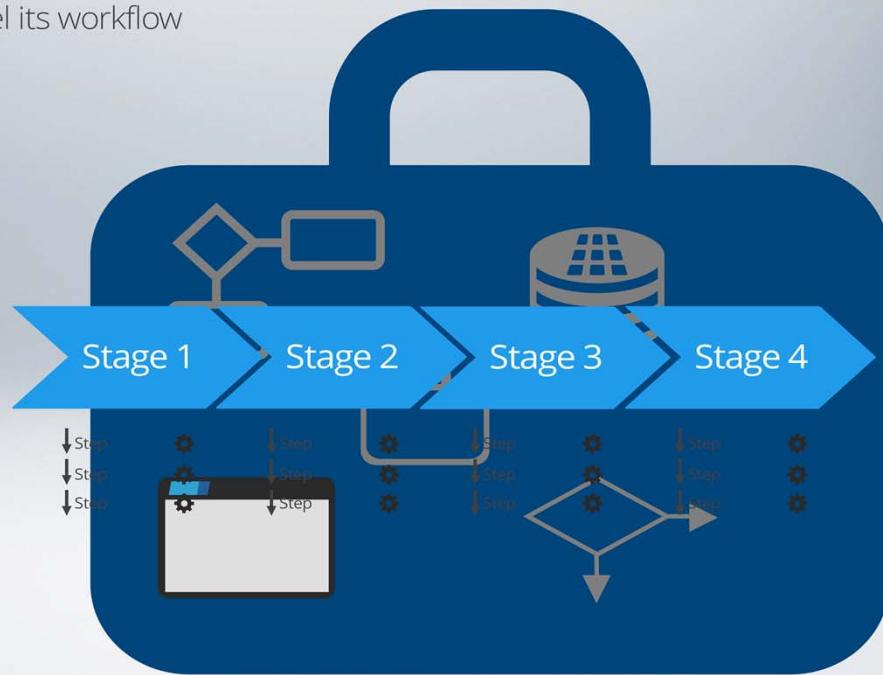
At the end of this lesson, you should be able to:

- Divide a single case into a series of stages
- Navigate the Case Explorer
- Create case stages using the Case Designer

Now that we have our application...
we can begin to model its workflow



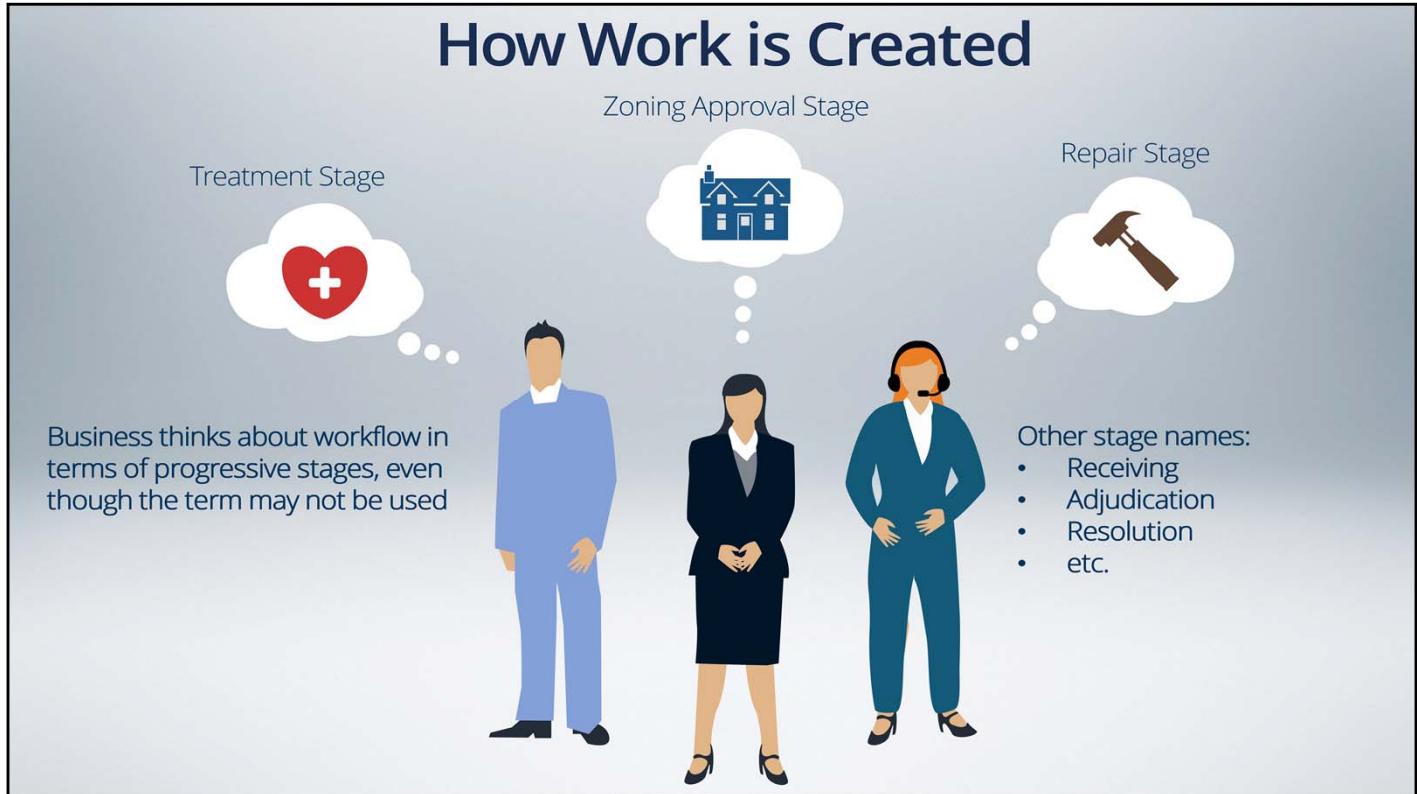
Model the Work FLow



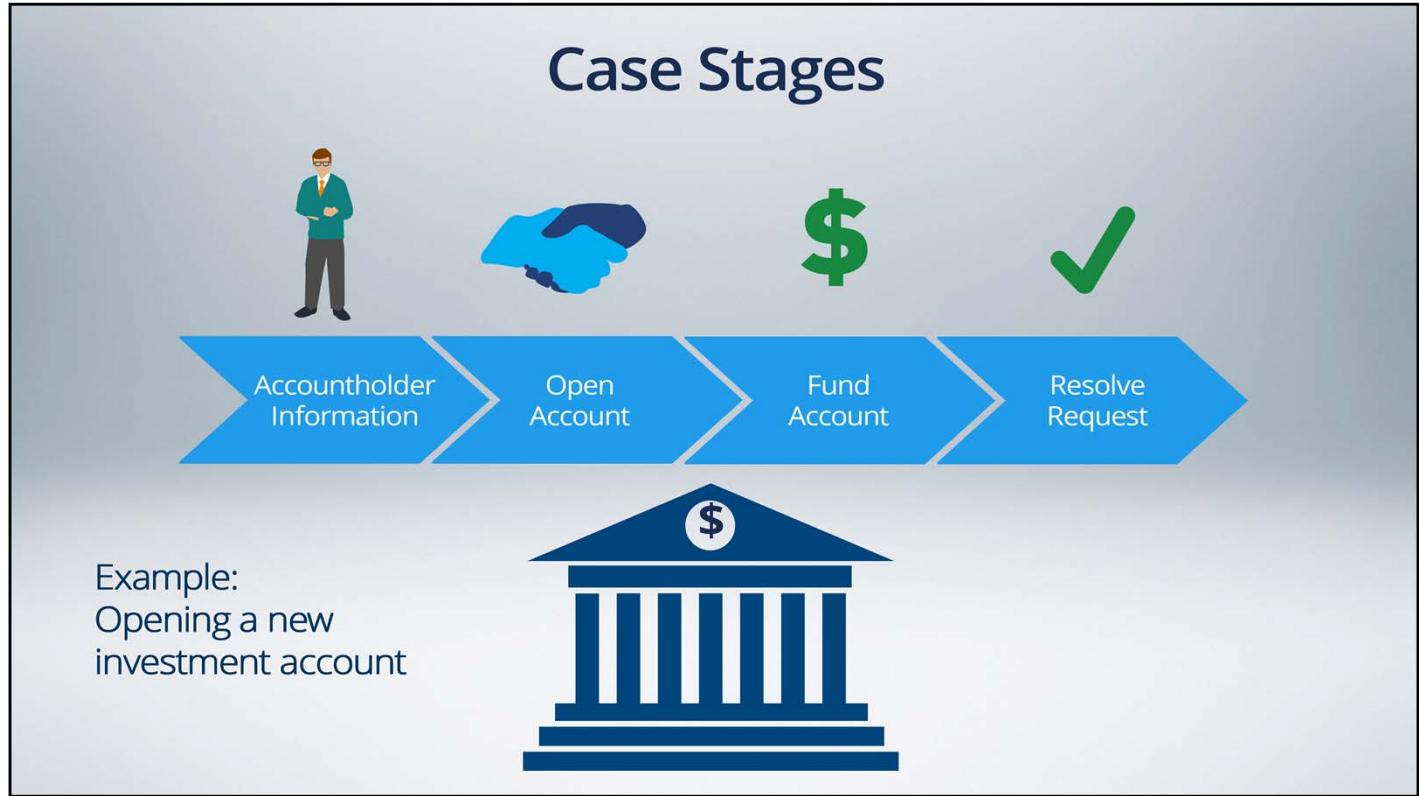
Now that we've created our application, we can begin to model its workflow.

The workflow for a case can be very simple – one actor performing one or two steps. Or, it can be much more complex – several actors performing in excess of 20 steps, in addition to automated actions such as decisions and integration points with other systems.

To help us model this workflow, we can divide our case into stages. By dividing a case into stages, we create a framework that allows us to group related processes and subcases. This framework provides basic context for the events, assignments, decisions, and automated actions – that occur as the case is processed.



Ask a business person how work is created, processed, and resolved, and you often hear them refer to the workflow in stages. For example, in healthcare, you might hear a reference to the “treatment” stage. For an insurance claim, it might be the “repair” stage. On a construction project, it might be the “zoning approval” stage.



These stages represent portions of an overall workflow. Each stage represents a distinct phase of the overall progression.

For example, if we wanted to model the workflow for opening a new account for an investor, we might divide the case – the account opening – into the following stages:

- First, collect the accountholder's information.
- Next, open the account.
- Then, fund the account.
- Finally, resolve the request. Note that resolution may involve creating new cases for follow-up work, such as enrolling the customer in a training course, or signing them up for an advisor, or adding them to an interest-based mailing list.

Case Stages

- Each stage has one or more steps to complete
- Each step represents a process
 - Single step process
 - Multi-step process
 - The creation of a subcase



Each of these stages has its own processes to run. If the case is complex enough, we may also need to create subcases to track related work.

By grouping these processes and subcases into a specific stage, we can provide some control over exactly when – and under what circumstances – those processes can be run and those subcases are created.

We can also provide a clearer picture to an end user that shows how far their case has progressed toward resolution.

Case Stages

What constitutes a stage?

Actually, there are no definitive 'rules', but...



Change in case ownership

Change in case status

Steps that "belong together"

Identifying stages for complex solutions may be a lengthy process involving interviews with multiple stakeholders

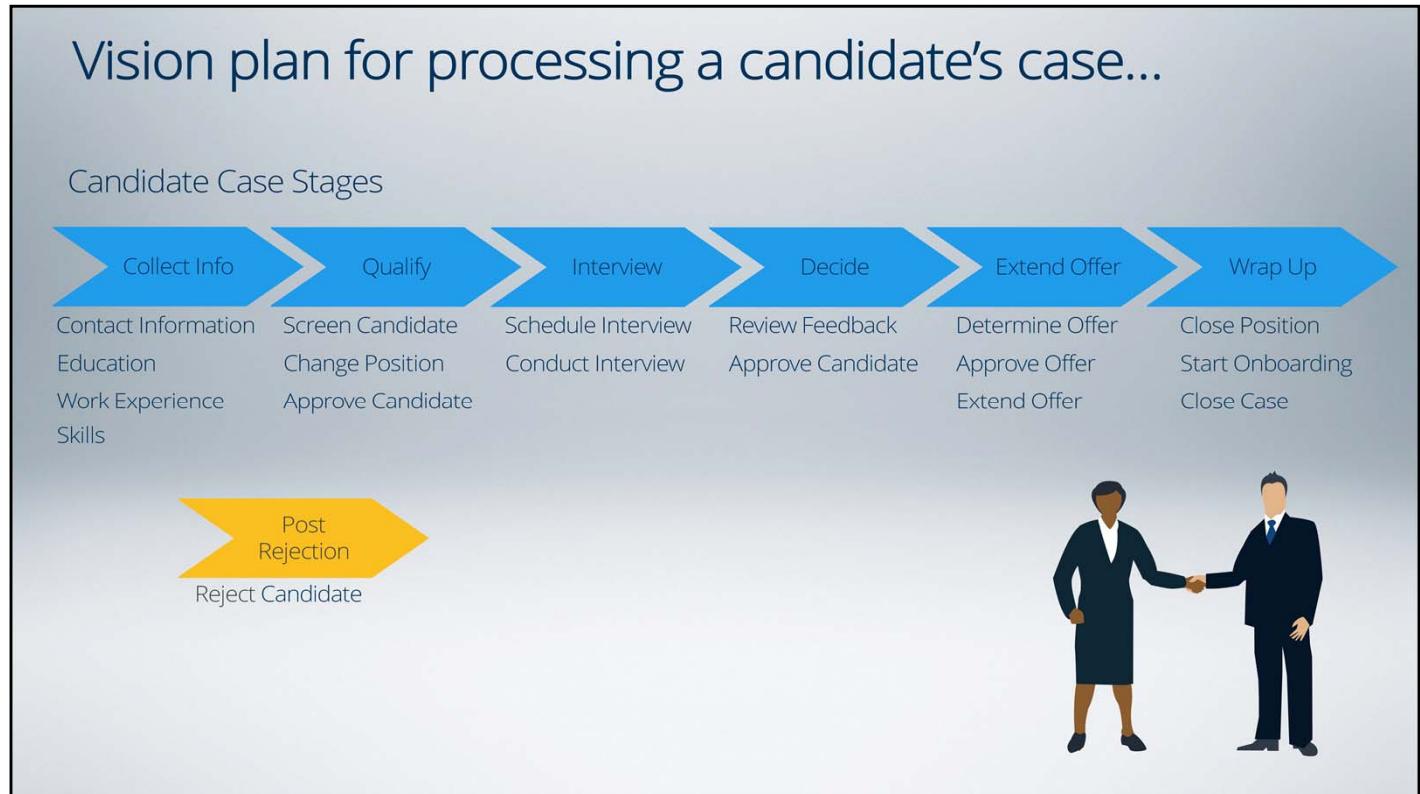


So ... how do we determine what constitutes a stage?

Truthfully, there are no definitive rules for identifying a stage. Changes in ownership or status can provide clues that guide us, but the easiest way is to simply determine which steps in the work flow belong together.

In a complex, real-world solution, identifying case stages can be a lengthy process involving discussions with multiple stakeholders.

In System Architect Essentials II, we'll explore stages in greater detail, but for our simple solution, let's just walk through the process and see if we can't identify our stages.



First, a candidate needs to apply for an open position. To do this, they must provide some basic contact information, along with their educational background and work experience, and perhaps highlight some relevant skills that support their qualifications. We seem to be collecting a lot of information here, so perhaps that's a good name for our first stage: **Collect Information**.

Once the candidate submits their information, a recruiter needs to review it, before scheduling an interview. The recruiter should screen the candidate to better understand their experience and qualifications. If the candidate isn't a good fit, they can qualify them for a different position, or elect not to pursue the candidate. It sounds as though the recruiter is attempting to qualify the candidate for a position, so maybe that should be the name of our second stage: **Qualify**.

Now that the recruiter has qualified the candidate, they can set up an interview with the hiring manager. Once the hiring manager completes the interview, they need to provide their feedback and recommendation, which will be used to make a hiring decision. Since we're primarily concerned with the interview at this point in the process, we can name this stage **Interview**.

Once the interview is complete, we can assess the candidate, make a hiring decision, and extend an offer of employment to the best candidate. Since the approvals process can be time-consuming – and we may need to repeat it if the candidate negotiates their compensation – we can divide this portion into two stages: **Decision** and **Offer**.

Once the candidate accepts the offer, we're just about done. We need to do a few finishing tasks – update the open position so that someone else doesn't apply for it, close our candidate case, and perhaps start the onboarding process. Since we're wrapping up the case at this point, we can use that as the name of our final stage: **Wrap Up**.

We also need to create a stage to reflect the other task we've hinted at: rejecting a candidate. This stage isn't part of the primary path of our case. It's an exception, so we'll ignore it for now, though we still need to represent it before we can consider our application complete.

Demo



Navigate the Case Explorer

So, we've identified seven stages in all: six stages that form our regular work flow, and one other stage to process deviations. Now that we have identified our stages, let's learn how to implement them in an application.

To help us access our cases, PRPC provides the Case Explorer. The Case Explorer – one of the seven explorers for visualizing the elements of a solution that are available in Pega 7 – lists all of the cases and subcases available in an application.

We can use the Case Explorer to add a new case – either as a top-level case or a subcase – open an existing case, and rename an existing case. We can also add cases to – or remove them from – the New Work menu. Cases listed in the New Work menu can be created as a top-level case. If our case is not listed in the menu, we can only create it as a subcase of another case.

Clicking on a case opens that case in the Case Designer, where we can specify the stages, processes, and subcases that our case entails. Since the New Application wizard has already created our cases, we don't need to create one. So, let's open the existing Purchase Request case.

The Case Designer contains three tabs that provide important information about our case type. On the Stages & Processes tab, we can use the Stage Designer to configure case stages as part of our process decomposition. In these stages, we can define the specific steps – akin to items on a checklist – users must perform to process and resolve a case.

The Details tab contains settings that affect how Pega 7 creates cases, such as copying data from the case to any subcases when they're created. For now, we don't need to worry about the Details tab.

The Specifications tab lists all of the specifications we create for our solution. Since we haven't yet created any, the tab is empty. We discuss specifications more in a later lesson.



Now that we understand how to use the Case Designer, we can add stages to our case. We can use our sample application – which processes purchase requests – to explore the Stage Designer. Before we do, though, let's review the stages that make up our sample application.

First, we have the **Request Entry** stage, where employees create a purchase request against a specific program.

Next, in the **Vendor Addition** stage, employees can select a vendor to fulfill the request.

In the **Quote** stage, employees can obtain a price quote, and either submit the request or withdraw it.

In the **Approval** stage, a submitted request is routed through the approval chain in accordance with company policy. Each approver has the option to approve the request or reject it.

In the **Procurement** stage, we create and place the necessary purchase orders, and the requestor waits for their items to be delivered.

In the **Closure** stage, the requestor's items have been delivered, so we can resolve the purchase request.

We also discussed two other options for resolving the purchase request: the requestor can withdraw the request, and the request can be rejected during the approvals process. This means we identified two other stages: **Rejection** and **Withdrawal**. Since these stages are exceptions – lower-likelihood outcomes – they aren't part of our primary path, so we'll set them aside for now.

Demo



Create Case Stages Using the Case Designer

Now that we have identified the stages in our application, we can create them with the Stage Designer.

The New Application wizard starts us with three default stages. The appearance of each stage shape provides a clue about how PRPC processes the stage. For example, some shapes have vertical edges, while others have angled edges. To help us decipher these meanings, let's review the shape legend.

Here, we can see that the angled edge on both Stage 1 and Stage 2 means that each of these stages transition to the next one automatically, while Stage 3 transitions manually, since we don't currently have a stage to which it can transition.

We can also see that we have two other types of stages: resolution stages, and alternate stages. We'll create these stages in a later lesson, so for now we can focus solely on the primary stages.

So, the three stages that were provided for us are primary stages. We've already identified a primary processing path for our case – an employee enters their purchase request, selects a vendor, and obtains a quote. Once they have a quote, they submit the request for approval, the items are procured, and the request is resolved. We'll model this path – the path that represents the desired outcome – with primary stages. Let's start by renaming the three existing stages.

To rename a stage, we can click the Options menu and open a dialog to configure the stage. In the Stage Configuration dialog, we can provide a new name for our stage. There are several other options available in this dialog – options that allow us to control when, or if, our case will enter the stage, and what other actions might be available to the end user – but for now, we're just concerned with naming the stage. We'll come back to configure some of these options later.

We still have three primary stages to create, and our two alternate stages as well. Let's first add the remaining three primary stages to our case. We can add the Closure stage by clicking the icon after the Procurement stage. The Stage Designer automatically creates a fourth stage for us – and, more importantly, the Procurement stage automatically transitions to this new stage. We'll rename this new stage Closure.

Since we skipped a couple of stages – Vendor Addition and Approval – and we need to insert them in the correct order – we can use the Edit Primary Stages dialog to add them.

Here, we can add our two remaining stages: Vendor Addition and Approval. To add a stage, click Add Stage.

Now, we can reorder our stages to match the behavior dictated by the business, placing the Vendor Addition and Approval stages into the correct sequence.

And when we're done, we can close the dialog and the Stage Designer updates with our changes.

We also identified two other processing paths. These are paths that some – but not most – of our cases may follow. We can model these stages as alternate stages, which we'll cover in a later lesson.

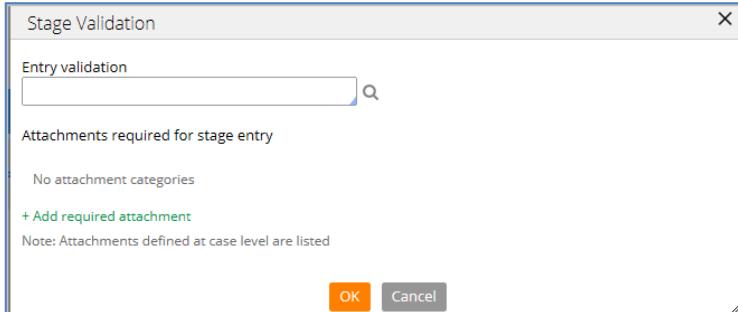
Pega 7.1.6 Update Notes

Configure entry validation has been added to the Stage options menu

A Configure entry validation option has been added to the Stage options menu and removed from the Stage Configuration form.

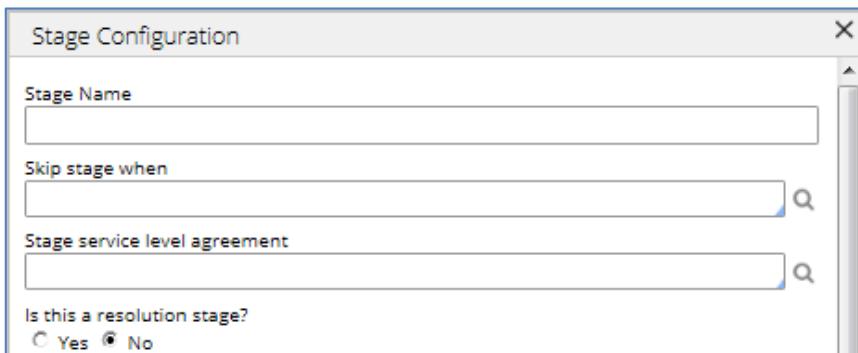


When the user selects Configure entry validation from the Stage options menu, a new entry validation form displays. This form allows us to specify that an attachment is required for stage entry.



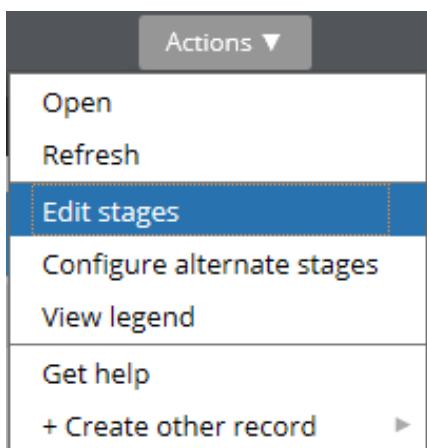
Stage Configuration Form has been updated

Entry validation has been removed from the Stage Configuration form and a Stage service level agreement field has been added.



Edit Stage option moved from the toolbar to the Actions menu

Edit stages used to be a button on the toolbar. It is now an option in the Actions menu.



Exercise: Add Stages to a Case



Module 04: Defining the Process

This lesson group includes the following lessons:

- Adding Processes to Stages
- Modeling the Processes
- Adding Business Use Cases to Process
- Review of Defining the Process

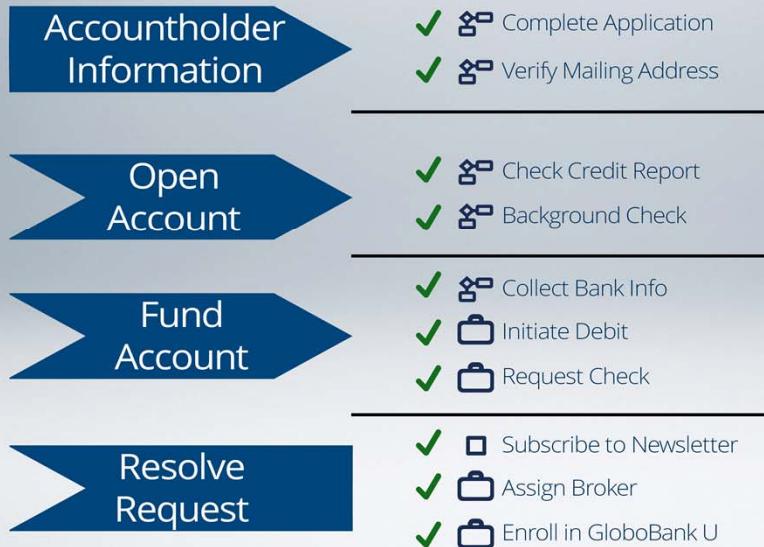
Adding Processes to Stages

Before we can begin to draft processes and user interfaces, we need to complete our case decomposition by adding steps to each stage

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

- Add steps to case stages
- Differentiate between assignments, processes, and cases in the Stage Designer
- Configure parallel and sequential steps within a stage

Reviewing the New Account Application Case



Now that we have created our stages, we can begin adding processing steps to each stage.

A step represents an action that may be taken to process our case. It can represent a simple task, such as subscribing to a newsletter. It can also represent a complex process that involves multiple parties, coordinated with some processing logic. It can even represent another case – one that can be processed in parallel.

Think of each step as an item on a checklist – an item to cross off as the case is processed to completion.

What each step represents – a task, a process, or a case – is important, but we need not decide on those implementations *right now*. For now, we want to understand that series of steps.

When we finish this part of the process, our case will consist of a series of steps, divided into stages – one that models the lifecycle of our case. More importantly, these steps provide us a framework, one which we can expand to implement functionality as we complete our solution.

Demo



Add steps to case stages

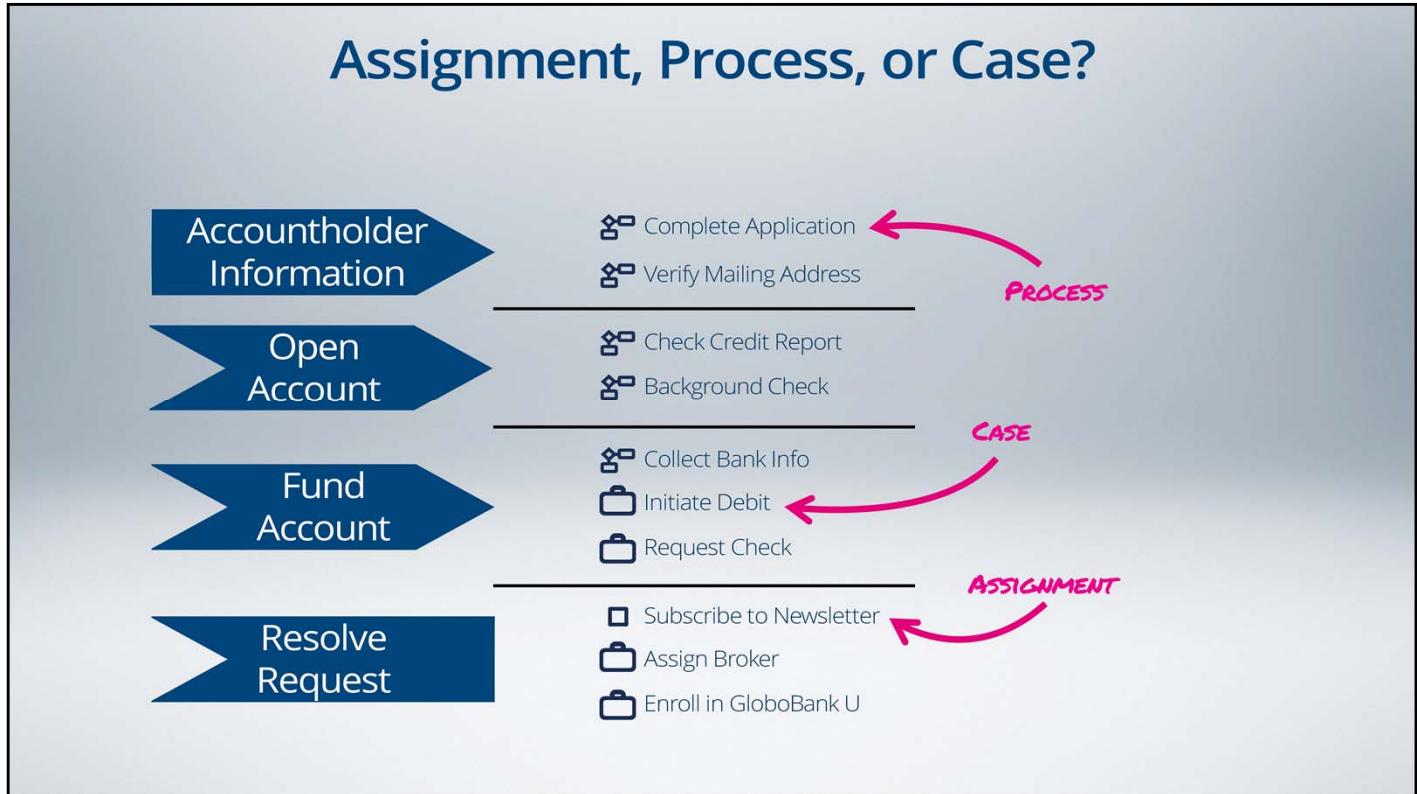
Our first step is to learn how to add a step to a case stage.

Under each stage is a default step. We can rename this placeholder step to something more appropriate.

We can also add additional steps to a stage by clicking Add step and entering the name of the step.

We can also rearrange the step order within a stage by dragging steps into the desired order.

When we add a step to a stage, we have three options as to what the step represents. A step can represent a single **assignment**, a **process**, or a **case** to be created. There are some differences between each option, but those differences affect the implementation – which we aren't worried about yet. For now, each step is still just a checklist item to be completed.



When we add a step to a stage, we have three options as to what the step represents. A step can represent a single **assignment**, a **process**, or a **case** to be created. There are some differences between each option, but those differences affect the implementation – which we aren't worried about yet. For now, each step is still just a checklist item to be completed.

Demo



Differentiate between types of steps

By default, each step is configured as a single assignment. We can easily change its type to either a process or a case by using the Step Configuration dialog.

Assignments are simple – individual tasks performed by an end user. They represent one discreet act, performed by one end user. When the user performs that act, the step is complete. By default, each step is an assignment, unless we change it.

Processes are really just collections of actions. Those actions may involve multiple users, PRPC itself, or other systems – perhaps an integration with a vendor to place an order, or a credit check performed by a credit bureau, or a background check performed by another party. We don't know at this point whether we need processes or not – and we won't know until business analysts decompose each step. But for now, we can make some educated guesses.

For example, if we anticipate that a step requires the system to perform some action – such as integrating to an external system, or managing a series of approvals – we should configure the step to represent a process.

Cases are automated actions that – as you might guess – create another case. For example, if we wanted to model an insurance claim, our solution might create cases for damage to any other vehicle involved in the accident; damage to property such as fences, lawns, and telephone poles; and medical care – because each of these other cases will likely be resolved by other parties, most likely at the same time.

Since we only cover a single case in this course, we can skip this change. We will cover creating other cases in the System Architect Essentials II course.

With updates to Pega 7.1, two new step types were added:

Approval — A single-step flow that contains the Approval Flow subprocess. You can configure this step for cascading or single approval.

and **Attachment** — A single-step flow that contains the single-assignment flow named Attach File.

One warning about changing the step type: once we switch from a single-step assignment to either a

process or a case, we can't switch back to a single-step assignment. However, that is not necessarily a problem, as we'll learn in the following lesson.

At this point, we can also start to make some decisions about when each step occurs within the stage.

By default, each step occurs in sequence, starting with the top-most step and working down through the list of steps. That's great – if we want a sequential order. If that's not desirable, however, how can we change the order? Or is a sequential order our only option?

The good news is that we do have other options for ordering our steps. Two such options are available within the Stage Designer.

Our first option is to make a step available to an end user when our case enters the stage. By default, the top-most step is available, but we can enable this for any step, as we will see momentarily.

Our second option is to make a step *conditionally* available to the end user. This option is helpful when we want a step to be available at some later point during the stage. This option requires a feature that we are not yet ready to learn, so we discuss this second option briefly and return to it later in the course.

Let's go back to our example. What if we want the end user to perform either the Initiate Debit or the Request Check step? Our process involves one funding request, so we don't need to do both, though we do need the final step to occur after either funding request.

First, we can make the Request Check step available at the start of the case. That establishes two possible processing paths for our case – and also establishes that the Initiate Debit step is dependent upon the applicant first providing information about their bank account, but that the Request Check step is not. We can denote the two paths with a line, for clarity.

Write Bank Info, Debit, and Check steps.

That's better – but it still doesn't completely solve our problem. We don't want end users to prompt applicants for both a check and a debit. We also don't want to confirm either funding request until after it has been initiated. We can resolve the second issue easily – all we need to do is add the Confirm Funding Minimum step to each path.

A resolution for the first issue is a little more involved. We can configure a step to be available to the end user only when a specific condition has been met. We can do this for all of the steps – in both paths – to allow or deny access as warranted, and prevent end users from both initiating a debit and requesting a check.

We learn how to create such a condition later in this course. For now, we only need to know that the option is available.

First sentence in the this section is "At this point, we can also start to make some decision about when each step occurs within the stage." Decision should be plural, not singular. Affects both video and transcript.

Pega 7.1.6 Update Notes

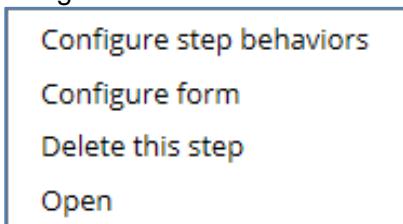
Two additional step types

Two additional step types – Approval and Attachment – have been added. Steps type can be:

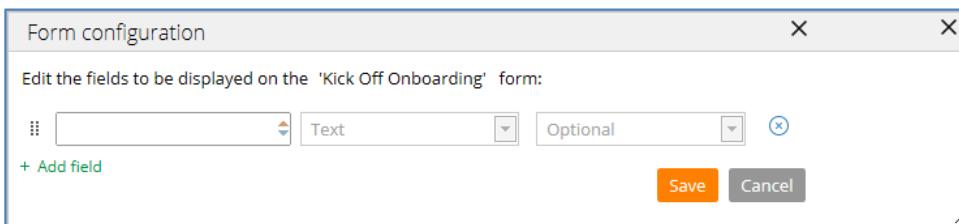
- Single-step assignment — Flow contains one Assignment shape
- Multi-step process — Flow contains two or more shapes
- Case — Flow contains a Create a Case Smart Shape
- Approval — Flow contains the Approval Flow subprocess (new)
- Attachment — Flow contains the Attach File assignment (new)

Step Options Menu Updated

Configure form has been added to the Step options menu.

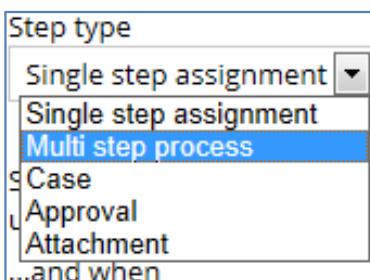


When we select Configure form from the Step Options menu, the Form Configuration rule form displays. Form Configuration replaces Edit Fields.



Step Type Choices Updated

Approval and Attachment choices have been added to the Step Type menu.



Step Sequence: Control Step Order and Availability



At this point, we can also start to make some decisions about when each step occurs within the stage.

By default, each step occurs in sequence, starting with the top-most step and working down through the list of steps. That's great – if we want a sequential order. If that's not desirable, however, how can we change the order? Or is a sequential order our only option?

The good news is that we do have other options for ordering our steps. Two such options are available within the Stage Designer.

Our first option is to make a step available to an end user when our case enters the stage. By default, the top-most step is available, but we can enable this for any step, as we will see momentarily.

Our second option is to make a step *conditionally* available to the end user. This option is helpful when we want a step to be available at some later point during the stage. This option requires a feature that we are not yet ready to learn, so we discuss this second option briefly and return to it later in the course.

Step Sequence: Control Step Order and Availability



Let's go back to our example. What if we want the end user to perform either the Initiate Debit or the Request Check step? Our process involves one funding request, so we don't need to do both, though we do need the final step to occur after either funding request.

First, we can make the Request Check step available at the start of the case. That establishes two possible processing paths for our case – and also establishes that the Initiate Debit step is dependent upon the applicant first providing information about their bank account, but that the Request Check step is not. We can denote the two paths with a line, for clarity.

That's better – but it still doesn't completely solve our problem. We don't want end users to prompt applicants for both a check and a debit. We also don't want to confirm either funding request until after it has been initiated. We can resolve the second issue easily – all we need to do is add the Confirm Funding Minimum step to each path.

A resolution for the first issue is a little more involved. We can configure a step to be available to the end user only when a specific condition has been met. We can do this for all of the steps – in both paths – to allow or deny access as warranted, and prevent end users from both initiating a debit and requesting a check.

We learn how to create such a condition later in this course. For now, we only need to know that the option is available.

Demo



Configure step order within stage

Now, let's try this in an application. In our purchase request application, we need to allow two courses of action once a case reaches the Procurement stage. We want to send an email to the requestor that their request was approved, and at the same time, forward the request to the Purchasing department so they can complete the request and generate the necessary purchase orders.

The Send Mail step is already configured to start when our case enters the Procurement stage, so we only need to configure the Enter Accounting Info step to start upon stage entry as well. By default, the remaining actions automatically occur in sequence, after the Enter Accounting Info step.

We also want all of our approvals to occur in parallel. To do so, we can make each step in the Approval stage start upon stage entry.

Now, when an end user creates a purchase request, the Program approval, Manager approval, and IT approval can occur in any order once the request reaches the Approval stage. Also, when the request enters the Procurement stage, the requestor receives an email that their request has been approved at the same time that the Purchasing department receives the purchase request for processing.

Exercise: Add Steps to a Case Stage



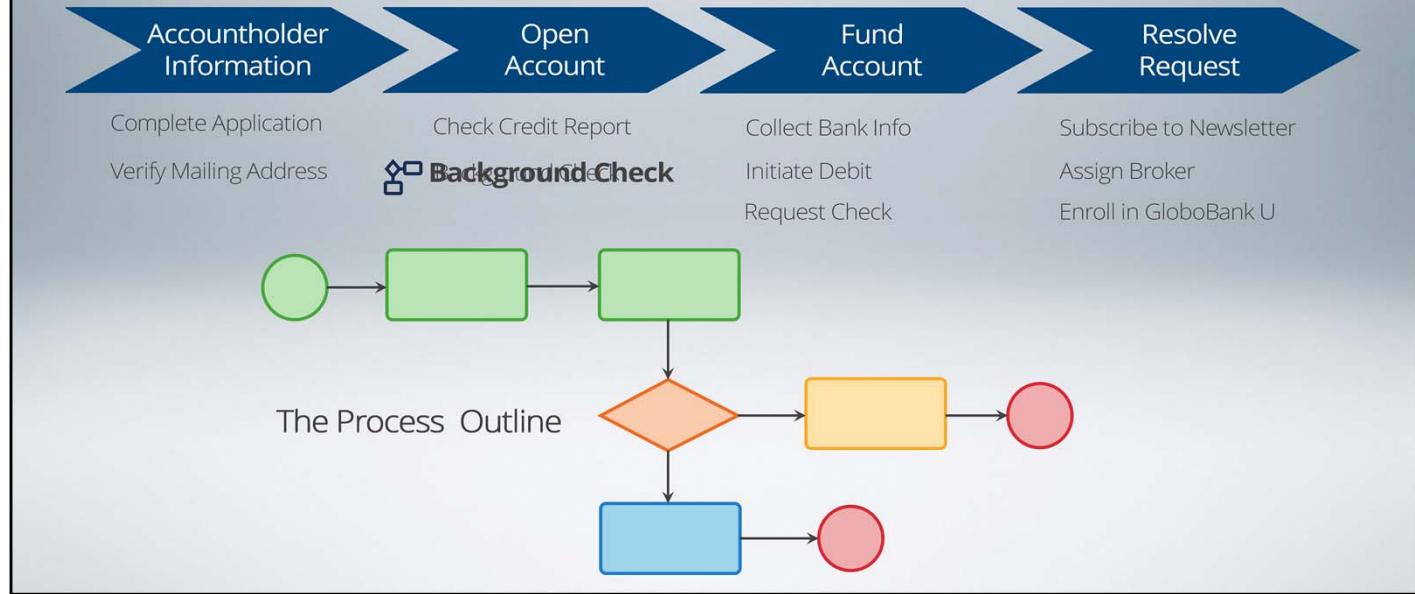
Modeling the Process

Modeling our processes allows us to review flow logic to identify and correct issues before we create dependent rules for our UI, data model, and business policies

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

- Navigate the Process Outline
- Describe the basic flow shapes
- Configure process flows

Modeling Processes: Describing Step Logic



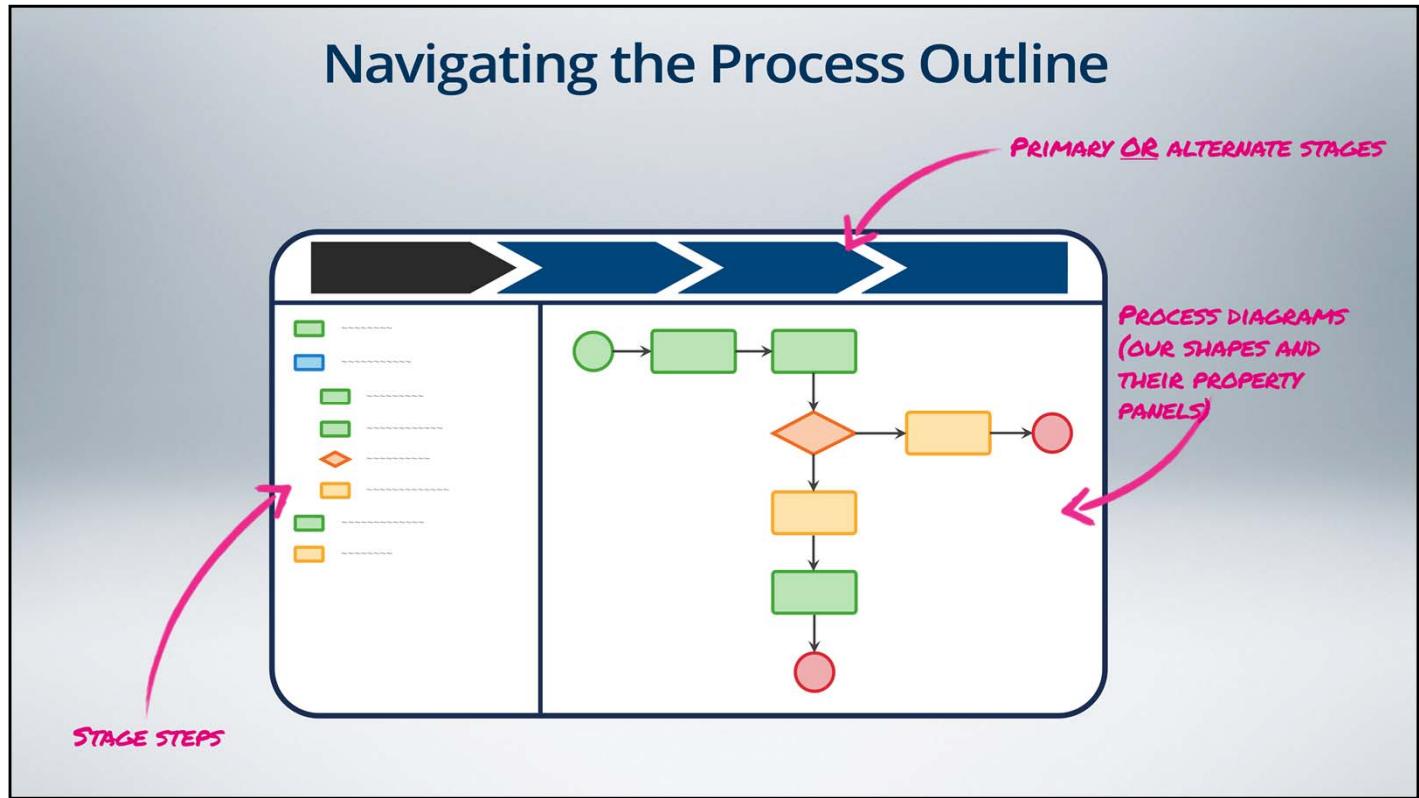
Now that our case has been divided into stages and steps, we can begin to configure the actual work flow – the part that tells PRPC how to ultimately resolve a case.

To do that, we need to understand how to use the Process Outline to build and configure our assignments and processes. And, as we cover the Process Outline, we'll also learn why we shouldn't worry too much about the difference between single-step assignments and multi-step processes.

Let's start with our case stages.

So far, we've created our case, divided it into stages, and elaborated a set of steps for each stage. We know what we want PRPC to do to process our case – our steps – but we haven't yet learned *how* to make PRPC perform and complete those steps. To do *that*, we can use the Process Outline, which allows us to design and configure processes from the Stage Designer.

Each process is represented by a series of shapes that models a specific portion of our work flow. These shapes – also referred to as flow shapes – indicate specific processing actions that we can configure to perform a specific action, such as determining the path a case takes through its work flow, indicating an interaction with an end user, or indicating an automated action performed by PRPC. We'll learn more about these shapes in a moment.



The Process Outline is divided into three main areas:

- Across the top, the Process Outline displays all of our stages, which allows us to quickly navigate between stages without needing to return to the Stage Designer. The Process Outline works for either primary stages or alternate stages, but not both. If we start with a primary stage, we only see the primary stages, and if we start with an alternate stage, we only see the alternate stages. Since our case consists only of primary stages at this point, we don't need to worry about any alternate stages.
- On the left, we have a tree view of the steps in the active stage. Each step in the stage appears in this view, and multi-step processes automatically expand to allow quick access to each shape within the process.
- And on the right, we have our working area, which contains the Process Modeler. This tool allows us to configure a process diagram, or flow, using a set of flow shapes. Each shape represents a task, which must be completed to advance the case toward resolution. Each shape performs a specific type of action, and we can fine-tune the behavior of a shape through its properties panel.

If we implement a step in the Stage Designer as a multi-step process, the Process Outline presents us with a process diagram. For single-step assignments and cases, we see the appropriate properties panel. Let's focus on the process diagram for now, and come back to the properties panel in a few minutes.

Flow Modeling: Basic flow shapes



A point of human interaction. The user selects the desired action to advance in the flow.



An automated decision. The system chooses the correct path and advances to the next shape.



A subprocess. The flows pauses until the subprocess completes, then resumes processing.



An automated system action. The system performs the action and proceeds to the next shape.



Start and end points of the process.

With the Process Outline, we can construct a process visually, with shapes that represent various types of processing actions. Each shape represents a unique type of action, and there are a lot of shapes available to us.

First, we have six basic shapes. These are the most-common shapes encountered in flows.

- An **assignment** indicates a point at which we require that an action be performed by a person. Assignments provide one or more possible outcomes, from which end users select the most appropriate one.
- A **decision** represents a choice between outcomes. Unlike an assignment, where the end user chooses the processing path, a decision is evaluated automatically, by our solution, in accordance with instructions we provide in a decision rule.
- A **subprocess** references another process flow. With a subprocess, we can divide our process into more granular flows. This allows us to collect several flow shapes into one subprocess and reduce the complexity of our process diagram
- A **utility** represents an automated action, performed by PRPC itself. These actions – also known as activities – are similar to functions in a programming language. They represent a series of steps, executed in a prescribed order, that return a result. We can use a utility to create a case automatically, and to copy data between cases.
- The **start** and **end** shapes indicate the beginning and ending of the process diagram. Each flow should have one starting point, but can have multiple end points.

Flow Modeling: Smart Shapes & Advanced Shapes

Smart Shapes – flow shapes preconfigured to perform specific tasks



Advanced shapes – flow shapes that perform more complex tasks



The Process Modeler provides us with two additional palettes: one for Smart Shapes, and one for advanced flow shapes. Smart Shapes are pre-configured utilities or subprocesses that facilitate specific actions – such as creating a PDF, changing to a different case stage, or attaching a file – similar to the macros or scripts found in various software packages.

Advanced flow shapes provide more complex functionality, and are used less frequently than the basic shapes. In an upcoming lesson, we use the Integrator shape to obtain data from a Google search. Other advanced shapes are covered in more advanced courses.

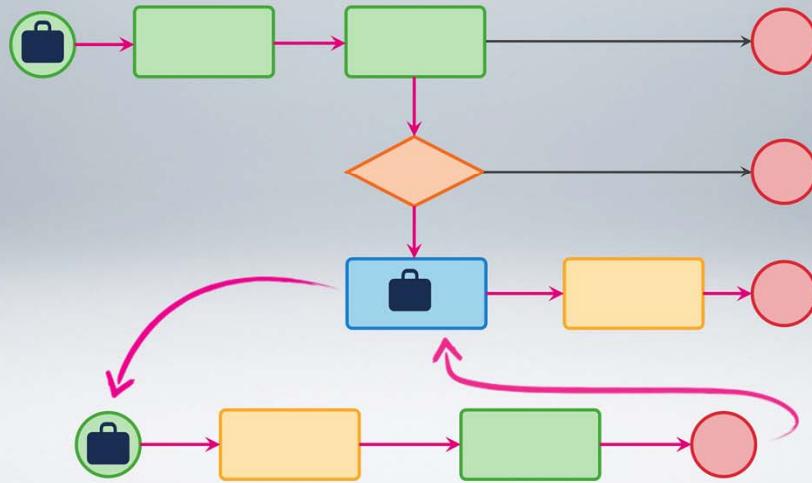
Flow Modeling

Connectors: One Shape, Many Uses



We also use connectors – represented as arrows – to indicate the path between flow shapes. Depending on the shape from which the connector emanates, the connector may represent a user interface, a decision result, a condition to evaluate, or nothing at all.

Flow Modeling: Understanding Flow Logic



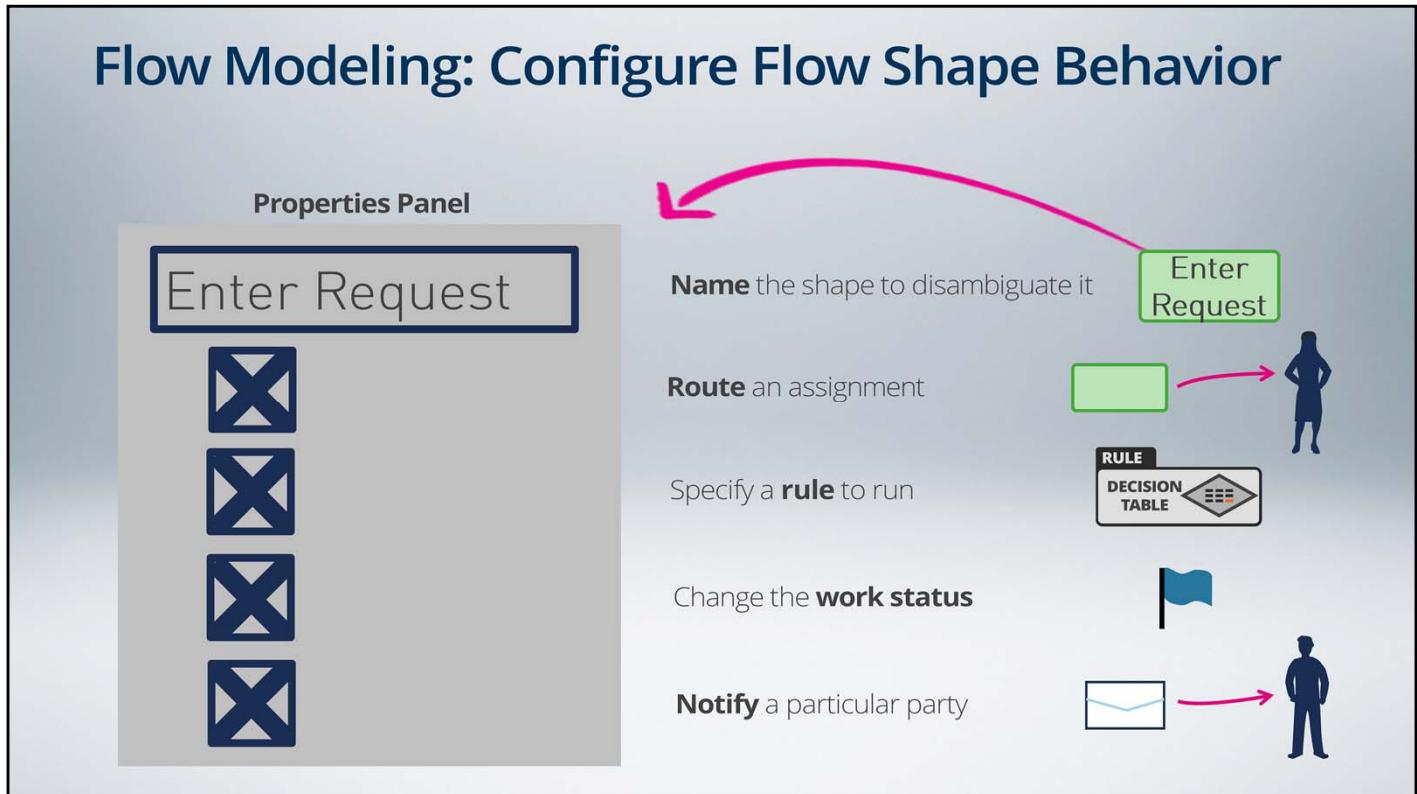
Let's look at a fairly simple flow diagram. When we create our case, it automatically proceeds to the first step, an assignment. The user selects the one available action, and advances the case to a second assignment, perhaps for a different user.

This user must select from two options – one ends the process, and the other continues it. If the user continues the process, they advance the case to a decision, where PRPC evaluates a set of conditions to determine the path of the case. One path leads to a subprocess, while the other ends the process.

PRPC selects the appropriate path, and automatically advances the case along it. Our case advances to the subprocess, at which point control transfers to the subprocess. Our process pauses until the subprocess completes, at which point control returns to our process, and our case advances to a utility.

Here, PRPC performs a specified action for us, without user intervention, and automatically advances the case to the next shape – an End shape, which stops processing.

Flow Modeling: Configure Flow Shape Behavior

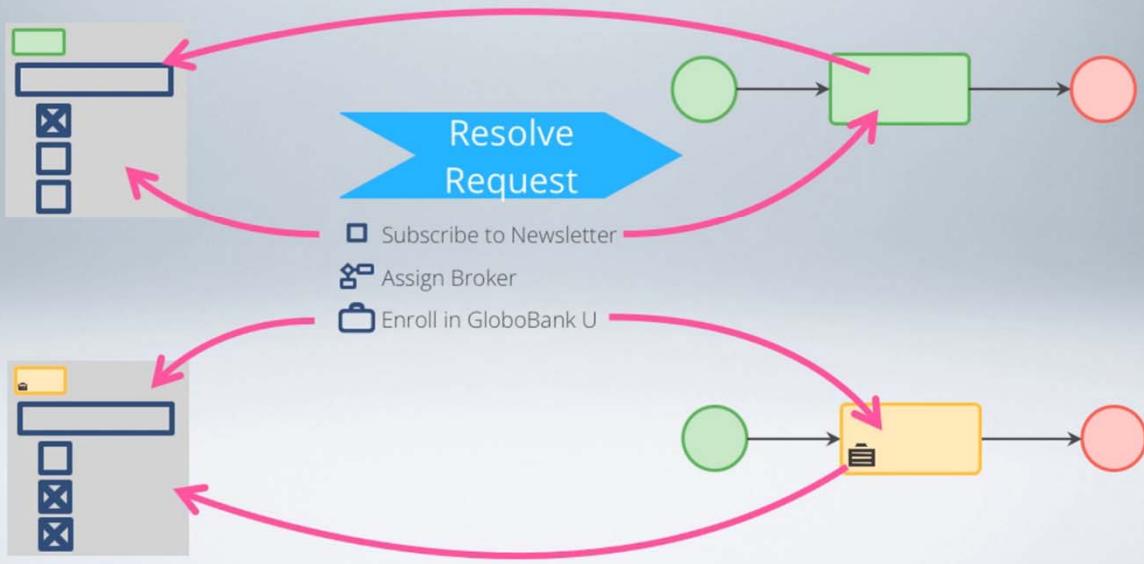


We can configure each shape to customize its behavior, using the properties panel provided for the shape. With the properties panel, we can:

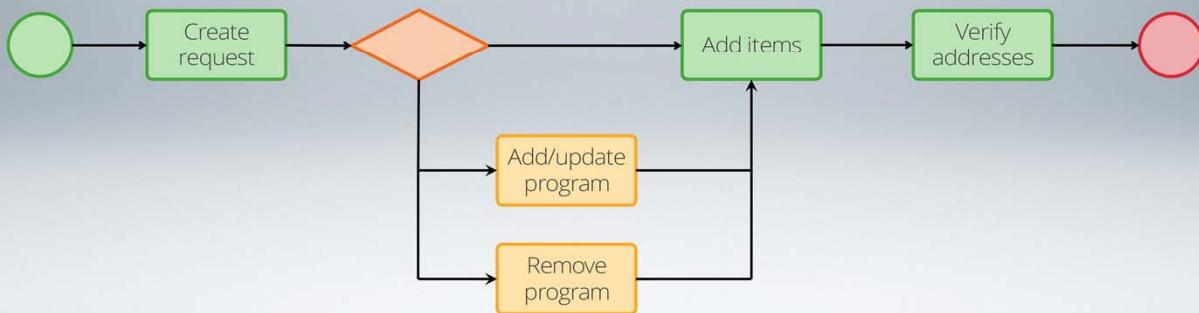
- Name the shape to differentiate it from other shapes of the same type. Naming a shape allows us to communicate our intent on the flow diagram itself, without needing to check the configuration of the shape.
- Route an assignment to a specific person or work group. Routing allows us to transfer ownership of an assignment. We'll discuss routing in more detail later in the course.
- Specify a rule to run. Shapes that perform automated actions – such as decisions, subprocesses, and utilities – reference specific rules. This allows the system to determine the behavior intended for the shape. Some rules allow us to specify parameters, so we can further customize the behavior of the shape at run-time by reflecting case-specific data.
- Change the work status of the case. The work status indicates the progress of our case towards a resolution, and can be set on any shape. Cases typically use the default status values of "New", "Open", "Pending", or "Resolved". The Pending and Resolved statuses are extended to provide greater detail, such as Pending-ThirdParty, or Resolved-Rejected.
- Or one or more parties that the case has reached a certain point. These notifications – made by email, fax, text message, or mail – allow us to communicate the status of the case, and prompt a responsive action.

When we configure a case step to represent a single-step assignment or create a new case, we see the same properties panel that we would see in a process diagram, either for an Assignment shape or a Create Case shape. Each of these steps corresponds to a flow rule that contains three shapes: a start shape, an end shape, and either an assignment or utility between them. Since the flow rule contains only one step of importance – the assignment or utility – the Process Outline bypasses the flow diagram and drills down to the flow shape and its property panel.

Flow Modeling: Configure Flow Shape Behavior



Flow Modeling: Submitting a Purchase Request



Now that we know about the flow shapes we need to use, we can model our first process – a fairly simple flow to submit a purchase request.

This flow should take the user through three assignments: entering basic information about the request, adding items to the request, and verifying the requestor's billing and delivery addresses.

We also want to determine if the request must be logged against a specific program budget. The system may need to add, update, or remove program information for the request. If so, the system should do that work for the requestor, automatically, before prompting them to enter the items for their request.

Demo



Configure Process Flows in Draft Mode

First, we switch to the Process Outline. When we enter the Process Outline for the first time, our process diagram is in Draft mode. Normally, the Process Modeler requires us to provide certain information for each shape, such as a rule to reference. If this information is absent, we receive an error that prevents us from saving the flow. If we're just starting to configure our flow, however, we may not yet know this information, but we still want to diagram our process. To work around this requirement, we save the flow in Draft mode. Draft mode allows us to save our flow even if it references rules that don't exist. Once we implement these rules, we can return to the flow and disable Draft mode.

Since we already configured this step as a multi-step process, our flow contains two assignments. The first assignment inherited its name from the step itself, and the second assignment is generically named Step2.

Let's start by renaming the existing assignments. We can name the first one Enter Purchase Request, and the second one Add Line Items.

Now, we add a third assignment from the Flow Shapes palette. We can name this assignment Verify Addresses. And we can add it to the process immediately before the End shape by pointing an existing connector to our new assignment, and creating a connector to point from the new assignment to the End shape. Creating a connector is easy – we click a connection point on our new assignment, and drag the cursor to the End shape.

We also need to configure these connectors. Connectors that lead from an assignment reference a particular type of rule called a flow action. A flow action describes the action that users can perform – such as “Approve” or “Reject” – at an assignment. Flow Actions reference other rules – called sections – that describe the form users see, but for now we only need to provide the name of a flow action. Since each assignment provides only one option for advancing the case, we can name our flow actions after the corresponding assignment.

We can also name the flow action from the properties panel for the assignment. This allows us to specify a flow action for a single-step assignment, when the Process Outline bypasses the flow rule and drills down directly to the properties panel.

Since we're in draft mode, the Process Modeler ignores the error regarding the flow actions when we save our process diagram. We will correct complete this error in an upcoming lesson.

Now, we need to add our decision. Let's drag a decision shape from the palette onto the diagram, and reconfigure our process by reconnecting the EnterPurchaseRequest connector to the decision.

We also need to add two utilities, for our two automated actions – one to add a program or update an existing one, and one to remove an existing program. Again, we don't need to know anything about the implementation at this point, only that we need the system to perform these actions automatically.

It appears that we need some more room for our process. We can shift the drawing area left or right, and up or down by clicking on an empty spot and dragging the mouse. We can reposition shapes in a similar fashion, by clicking the shape – or shapes – and dragging the mouse.

Now, we can configure our decision shape, add the desired outcomes, and connect each to the correct shape. First, we configure the decision as a fork. Unlike many of the other choices for decision type – such as decision table, decision tree, or map value – a fork requires no additional configuration.

And we can add three connectors, one for each utility and one for the Add Line Items assignment.

For the two connectors that lead to the utilities, we need to provide the conditions under which the system will choose each path. Had we chosen a different type of decision for our decision shape – such as a decision table – our decision logic would be associated with the decision shape. With a fork, however, the logic is associated with the connector. We'll eventually create the rules that describe the conditions under which the system follows each path, but for now, let's configure each connector to evaluate a when rule, and name the rules that we'll create in an upcoming lesson.

For the remaining connector emanating from the fork, we want to take a different approach. We want the system to choose this connector whenever neither of the other two connectors is appropriate. To do this, we can configure the connector with an “else” condition.

Now, let's direct the utility output back into the flow.

At this point, we're nearly done. Before we go any farther, though, let's zoom out to see all of the flow.

Our last tasks to complete for our draft flow are adding instructions to our assignments, likelihoods to our connectors, and audit notes to shapes that call rules, and changing the case status when we complete the flow.

Let's start by adding instructions to our assignments, so that when an end user sees a pending assignment in their worklist, they'll know what to do with the request. We can skip the first assignment, since it won't appear in a worklist, but we should add instructions to the remaining assignments.

Next, we can add likelihoods to each connector. The likelihood indicates the probability of a user or the system selecting that option when processing the case. Flow action connectors default to a likelihood of 100% – we don't need to change that in this process, as each flow action is the only outcome for the corresponding assignment. However we need to determine the likelihood of each outcome for our decision. We don't need to add one for the Else connector – by definition, its likelihood is determined by totaling the likelihood of every other option and subtracting that from 100%.

Now, let's add audit notes to each shape – including connectors – that represents a rule. When the system performs the action described by the rule – evaluate a condition to make a decision, update case information, or present a UI to the user – it adds the audit note to the case history, or audit trail. Audit notes can confirm compliance with internal policies and industry and government regulations.

Finally, we can change the case status when we reach the end shape. Since the request has been submitted at this point, we can use the Open status.

Now that we've modeled our flow, we can test it by creating a new case. When we run a flow in draft mode, PRPC provides a basic UI for any missing rules – flow actions, activities, or decisions – allowing us to manually advance through the process by selecting outcomes that would otherwise be selected automatically. This helps us to test the logic of our process – the sequence of shapes in the flow – before we create the rules that would be needed on a production system. And, if we mock up UI rules, we can use our draft flow to confirm the contents of each form, before we start to construct our data model. If we need to change our flow logic or UI, we can make the necessary changes before they begin to involve editing properties, decisions, and other rules as well.

**Exercise 1: Configure Assignments with Process Outline
Exercise 2: Model Processes with Process Outline**



Pega 7.1.6 Update Notes

Rule Form Changes

Many of the Rule Forms have been updated to clarify instructions and simplify use. The Step Name field has been customized for the rule (i.e. Assignment in place of Step Name or Connector in place of Step Name). Tabs have been removed to place all configuration information on the main page and instructions have been clarified. Some examples are listed below.

Assignment Properties Rule Form

The Assignment Properties rule form is now divided into Routing (Who should complete this step?), Service Level Agreement (How long should it take to complete this step?) and Specification (How should this assignment behave?) sections.

The screenshot shows the Assignment Properties rule form with three main sections:

- Assignment:** [Assignment] - A step in your process that requires human judgement and input.
- Routing:** Who should complete this step? - Route to: Current operator
- Service Level Agreement:** How long should it take to complete this step? - Service Level: Default
- Specification:** How should this assignment behave?

Connector Properties Rule Form

The Connector Properties form is now divided into Connector (What is it doing?) Set Properties (What does it need?) and Specification (How should this connector behave?) sections.

The screenshot shows the Connector Properties rule form with three main sections:

- Connector:** Do Something
- Connector:** What is it doing?
 - Flow action: DoSomething
 - Likelihood: 100
 - Audit note: (empty)
- Set Properties:** What does it need?
 - Set Properties (radio button selected)
 - Add Item
 - Delete

NAME	VALUE
(empty)	(empty)
- Specification:** How should this connector behave?

Adding Business Use Cases to a Process

Flow modeling is an important part of Pega's Direct Capture of Objectives strategy, designed to help us accurately and efficiently implement solutions that solve business needs.

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

- Describe the purpose of a business use case and a specification
- Locate specifications within the Designer Studio
- Describe how specifications relate to flows

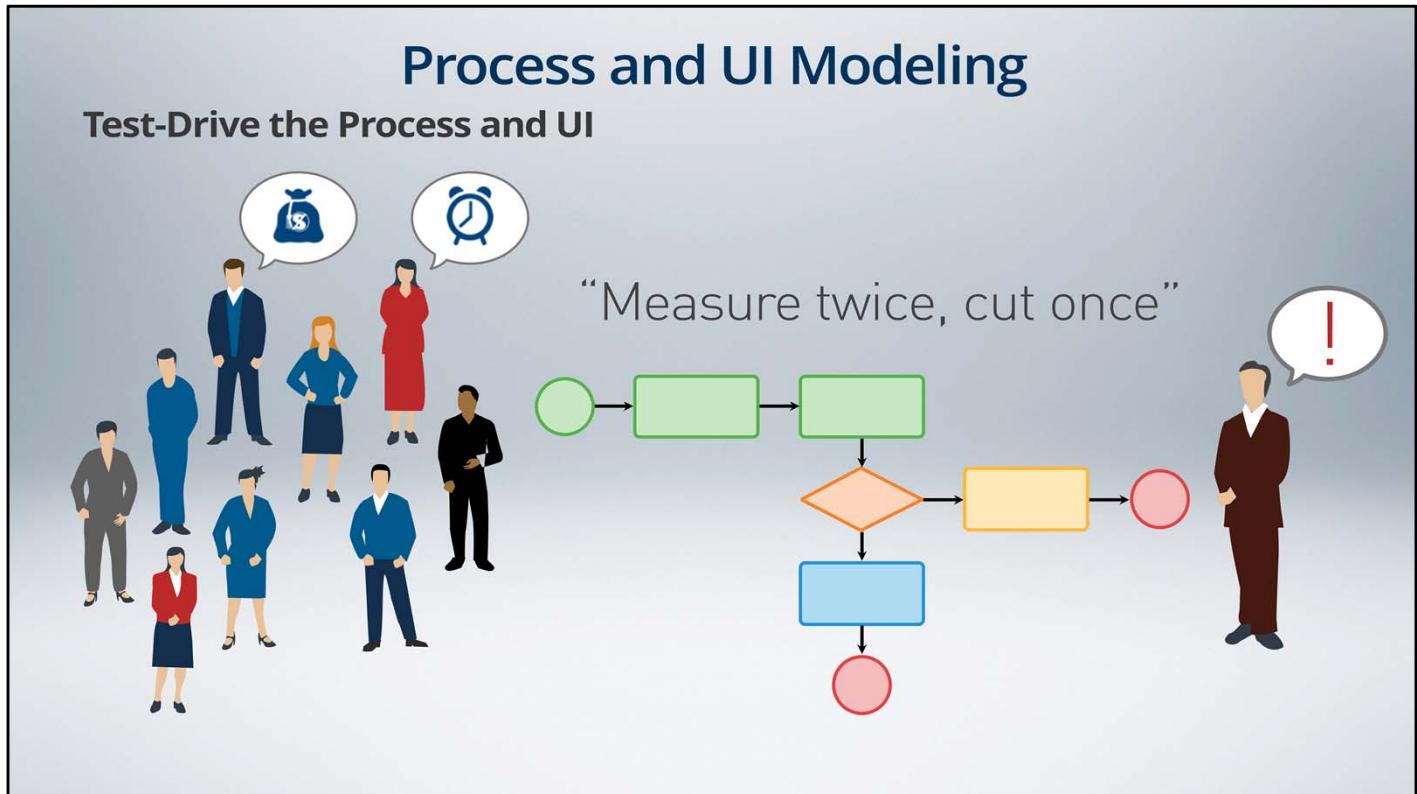
Process and UI Modeling

Test-Drive the Process and UI

The diagram shows two carpenters standing next to long blue bars representing pieces of wood. The top carpenter, wearing a yellow shirt and dark pants, has a green checkmark in a speech bubble above his head. A dimension line below him indicates a length of 1.5m. The bottom carpenter, also in similar attire, has a red exclamation mark in a speech bubble above his head. A dimension line below him indicates a length of 1.4m. The text "Measure twice, cut once" is centered between the two carpenters.

Now that we've built a couple of process models and performed some basic configuration of our processes and assignments, let's step back for a moment to better understand what we just did, and why it was important.

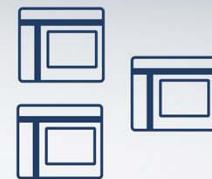
Tradespersons such as carpenters frequently use the phrase "Measure twice, cut once" to emphasize the importance of proper planning. Before they cut a piece of wood, or pipe, or wire, they want to double-check their measurements, to ensure that their cut yields a piece of wood, pipe, or wire of the correct length. Otherwise, they may need to start over again, wasting both time and money.



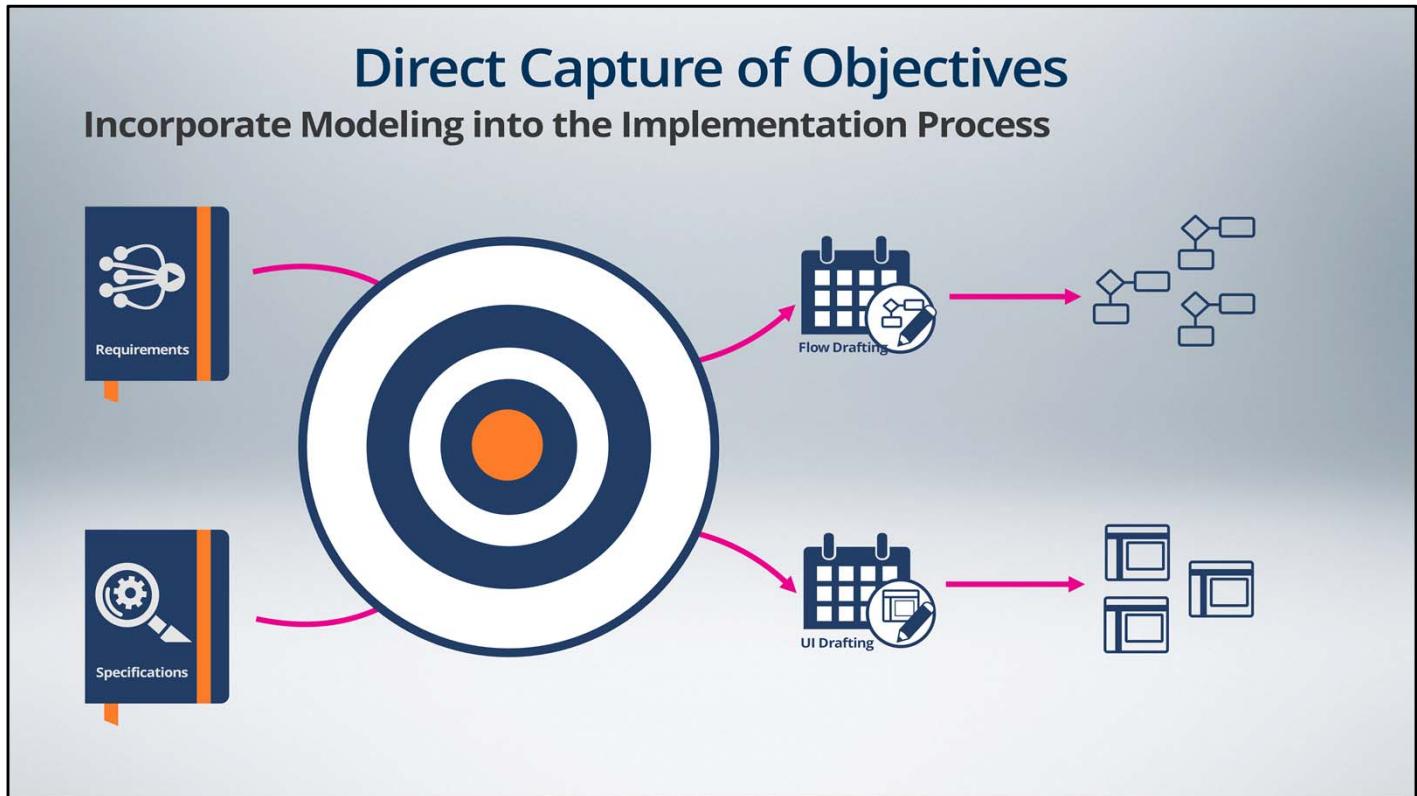
Pega solution development is similar. If we build something the business doesn't want – rather than what it does want – we waste time and money fixing it. The solution development process is never perfect, so we'll always need to spend time and money fixing things, but if we can eliminate some of those errors with better planning, everyone wins: the business saves money, and we save time.

Process and UI Modeling

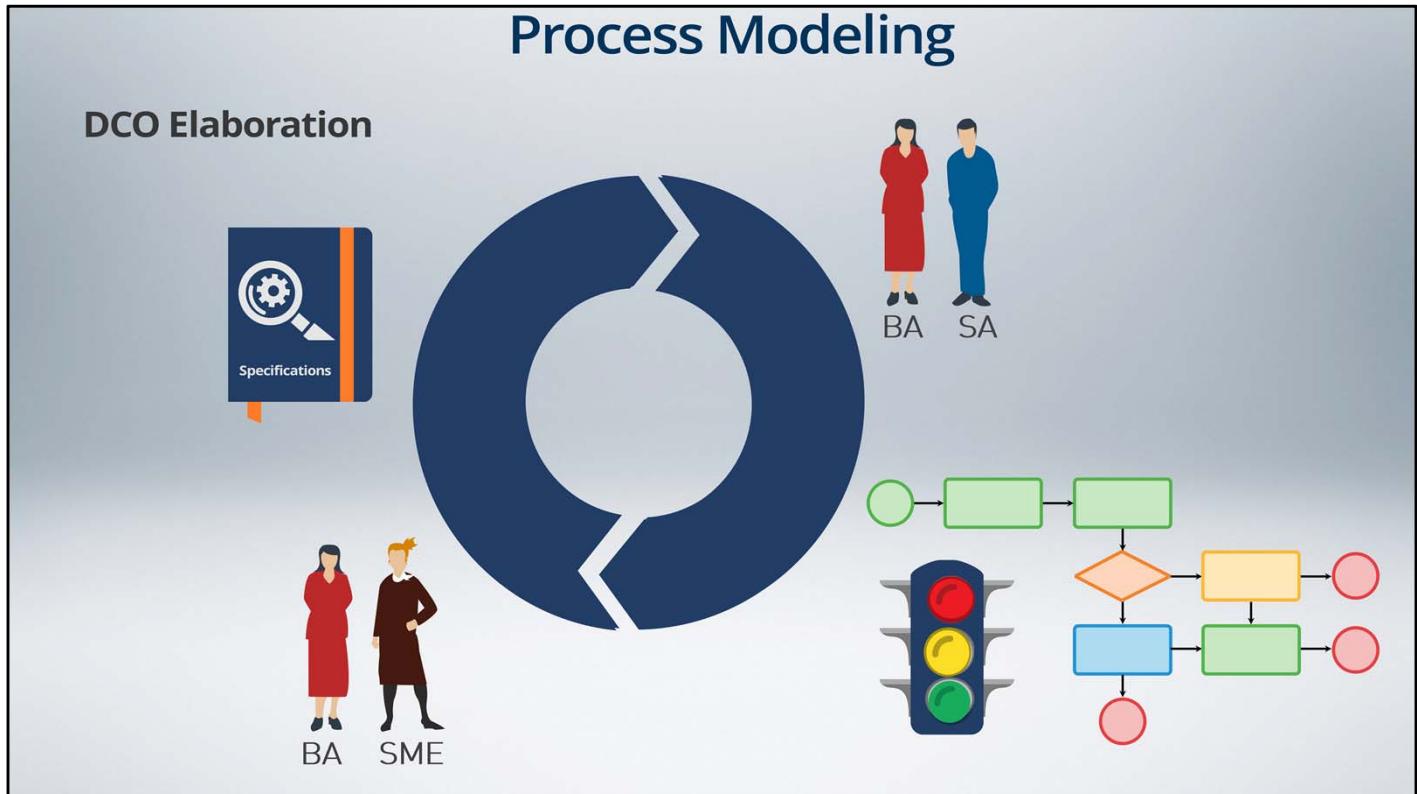
Test-Drive the Process and UI



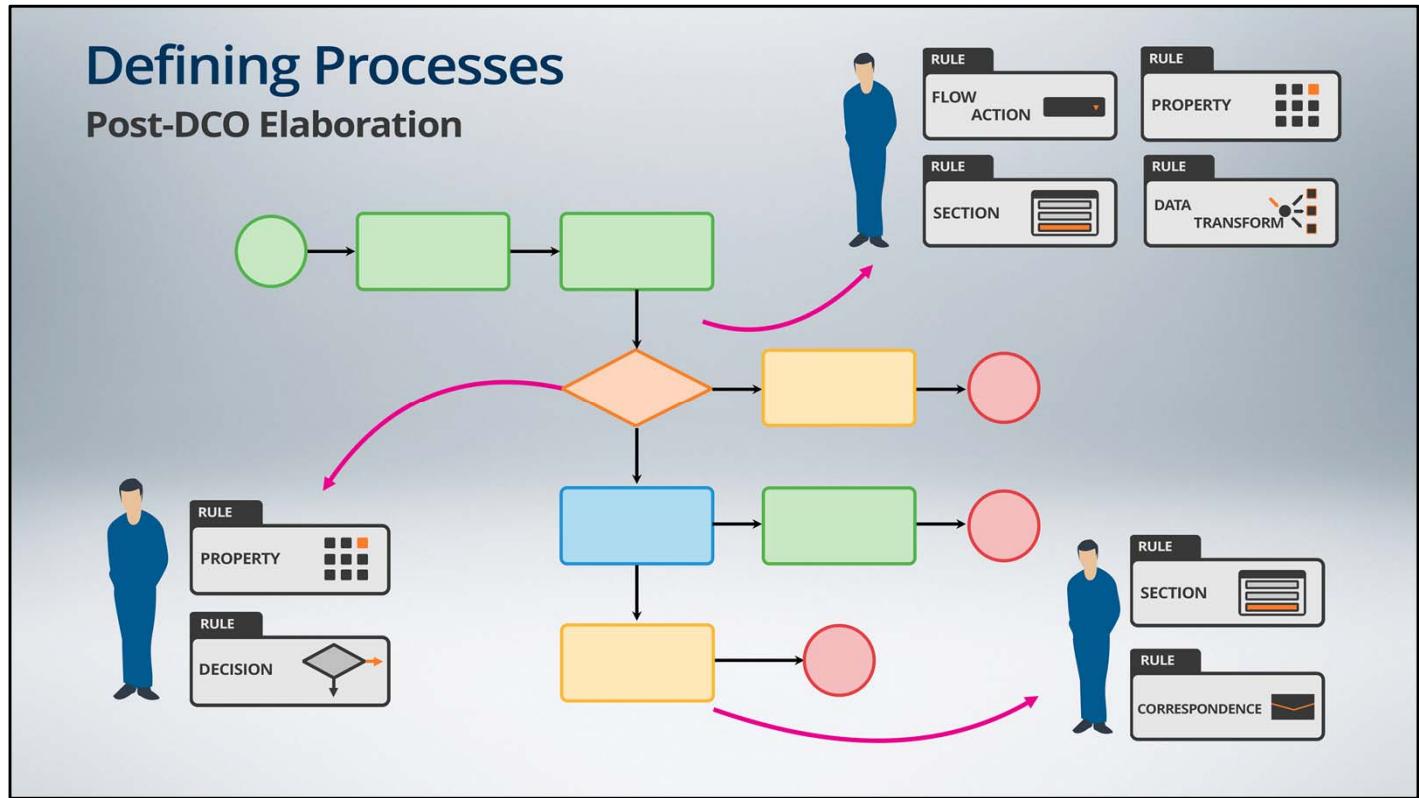
Process modeling and UI modeling help us eliminate errors by letting us represent processes and user interfaces without needing to create and configure additional rules, such as properties and decisions. This helps us to determine that flow rules model processes correctly, and that UI rules request and present the correct data at each step.



Process modeling and UI modeling are key activities in a Pega-specific development strategy known as Direct Capture of Objectives, or DCO. With DCO, the business needs are captured within PRPC as requirements, and linked to the specifications that address them. These specifications – also known as use cases – represent proposed solutions to the business' needs. We tend to use the term “business use case” to describe an entire process, and the term “use case” to describe a specific user interface or flow shape, although you may hear the two terms used interchangeably.



To ensure that each use case adequately addresses the corresponding business need, we build our process models and UI models in a process known as DCO Elaboration. In this process, we work with a business architect to build a model that matches the specification. The business architect then reviews the model with a subject matter expert from the business and updates the specification with any feedback captured during the review. We then refine the model with the feedback from each review until the model matches the business need.



Once the specification has been aligned with the business' expectations, we can implement the specification by creating properties, decisions, and other policy rules. Since the chance of a misunderstanding over the use case has been minimized by the use of DCO, the likelihood of wasted effort is reduced, and along with it the cost incurred by that wasted effort.

Good Specifications

What to Look For in a Specification



Description:

An explanation of functionality to implement



Subject Matter Expert:

Business who understands the process/case

Status:
Progress towards completion of the implementation



Pre - and - Post - Conditions:

Entry conditions and results for the feature



A good specification provides us with all of the information we need to implement it correctly. When we review a specification, we should look for the following information:

- A description of the specification. This describes the functionality to implement, such as UI features or process steps.
- The status of the specification. The status indicates the progress of our implementation. All specifications start with a status of new. As we implement them, we can update this status, which is reflected in reports run by business architects and project managers.
- A subject matter expert. The subject matter expert is the business representative who understands the process and is responsible for ensuring that our solution meets the business' needs. If we need additional information about the specification, this is the person with whom to discuss.
- Pre-conditions and post-conditions to describe how this specific specification fits into the overall set of specifications. A pre-condition describes the criteria that must be satisfied before we start to implement the specification, while a post-condition indicates the result of a user completing the use case successfully. If our specification is in the form of a Scrum user story, rather than a traditional use case, the pre- and post-conditions are replaced with the acceptance criteria for the specification, which outlines the criteria by which the specification can be considered "complete".



As part of the DCO strategy, we capture requirements and specifications within the solution itself – hence the term “Direct Capture”. This ensures that the knowledge captured in our requirements and specifications is always available to the system architects and business architects who need it to implement the corresponding rules. Now that we understand what specifications tell us, let's see how to find them and use them.

Demo



Locating Specifications with Designer Studio

To view all of the specifications for our solution, we can return to the Application Overview. When we last reviewed the Application Overview – immediately after running the New Application wizard – we had no specifications to review. Once a Lead System Architect (LSA) runs the New Application wizard, business architects can add specifications ahead of the DCO Elaboration process.

So, let's return to the overview, and see what specifications have been added. We now have over a dozen specifications. As we can see, some of these specifications describe steps performed by operators. Other specifications describe automated actions, such as sending correspondence and switching to alternate stages. And a few specifications describe entire sub-processes.

One specification – Purchase Request Entry – is an example of a business use case. This specification describes the entire subprocess of entering a purchase request: creating the request, adding a program fund if necessary, adding the line items, and finally verifying the requestor's billing and shipping addresses. The icons to the right of the specification indicate the presence of additional information on the specification.

Let's look at the specification for entering purchase requests. This specification should describe the user interface for the Enter Purchase Request form. It's empty, so let's add some information to see how the process works. Remember, the entry and updating of specifications is generally the responsibility of a business architect, but knowing how specifications work helps us to understand how to use them during solution development.

First, let's enter a description to describe the UI we want to present to the end user. We don't have a mockup of our UI, but if we did, we could add it as an attachment. This allows everyone assigned to the implementation of the specification to review the proposed UI and comment on it.

And, speaking of comments, let's add one to indicate that we've added a basic description of the UI we want to display. Specifications should always be associated with business objectives, requirements, and actors. Our business architect has added all three to the specification, which helps to ensure that the business objectives and requirements are satisfied by our solution. Our specification lacks a subject matter expert and any pre- and post-conditions. We should follow up with the business architect to ensure that a subject matter expert is assigned to the specification, and verify that there are no pre- and post-conditions that might affect our implementation.

Most of the remaining information in the Advanced section provides some background on the specification. The Type tells us kind of behavior it describes, such as a decision, correspondence, user action, or integration. Most types correspond to specific flow shapes, such as assignments, sub processes, and utilities.

The Complexity is used by business architects and project managers to estimate the time and effort needed to implement the specification. The Trigger describes how the resulting feature is accessed once our solution is in use, and the Format drop-down allows us to switch between a traditional use case and a user story.

Finally, since we're ready to start working on the implementation, let's update the status to In Progress. This status is reflected in reports, such as the ones available on the Analysis tab of the Application Profile.

The Implementations tab of a specification tells us where in our solution the specification has been implemented. This traceability keeps the specification linked to its implementation, and allows us to quickly review the specification as we're implementing it. We don't have an implementation yet, so let's see how that works.

Demo



How Specifications Relate to Flows

We can associate specifications with the shapes in our process flows.

Aside from start and end shapes, each shape in our flows – as well as the flow itself – can be associated with a specification. This association links the specification to its implementation, allowing us to trace the implementation back to the appropriate specification.

For example, we can add a specification to both an assignment and the connector that leads away from the assignment. The specification on the assignment should describe the position of the assignment in the process, and include any options for advancing the case. The assignment for the connector is associated with the flow action, and should describe the corresponding UI form.

Here, we're using a special type of flow called a screen flow. In a screen flow – unlike the type of flows we just created in the last lesson – flow actions are associated with an assignment, rather than a connector. This means that each assignment allows only one flow action to advance the case. Since the flow action is associated with the assignment, we only need one specification, but it should describe both the UI and the position of the assignment in the flow sequence. We'll learn about screen flows in System Architect Essentials II.

Once we add a specification to a flow shape, we can review the link from the specification. The Implementations tab lists every rule that contains a reference to the specification, allowing business architects to review the progress of the implementation against the specification.

Exercise: Add Specifications to a Process



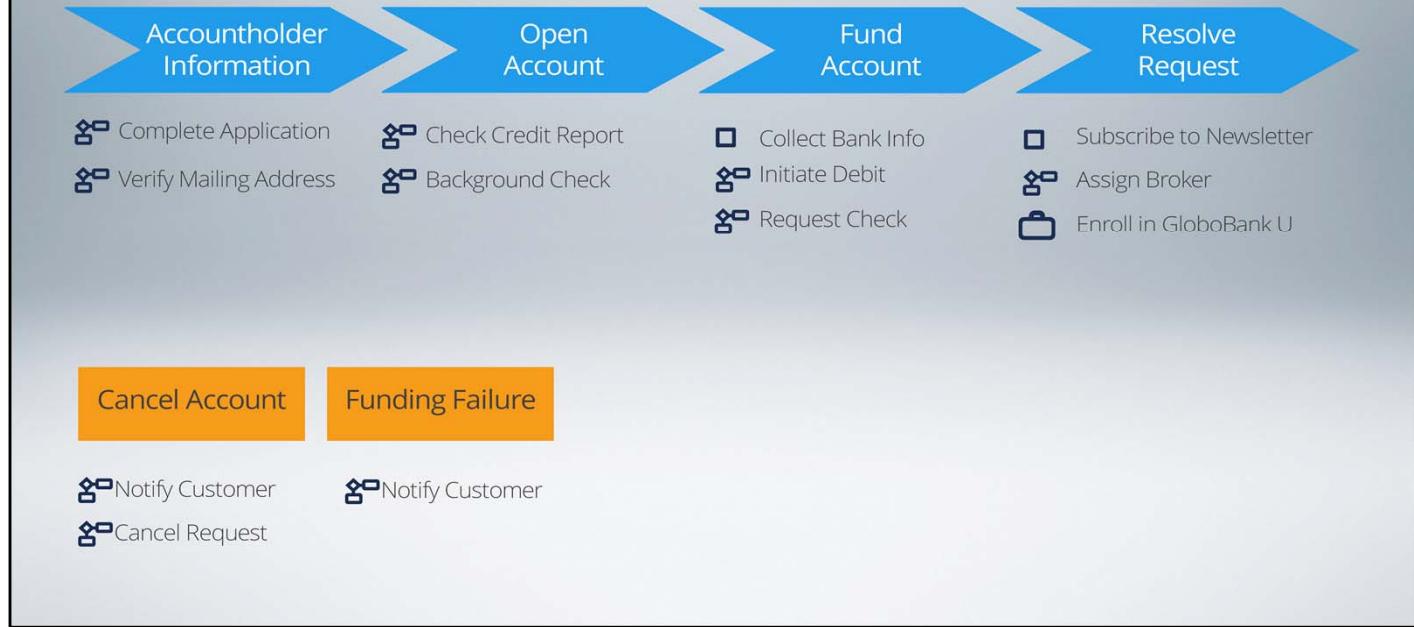
Review of Defining the Process

System architects work with business architects and subject matter experts from the business to turn specifications into process models that can be reviewed, tested, and approved

In the previous lessons, you learned how to:

- Create and configure case stages and steps
- Define process diagrams using basic flow shapes

Create and configure cases and steps



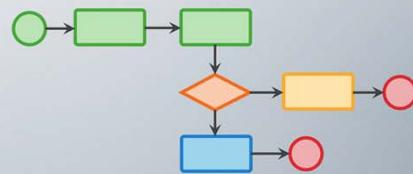
In the previous three lessons, we discussed the concept of process modeling. A process model allows us to establish the flow of a case from creation to resolution, before we start to create rules to store data, describe user interfaces, or codify business logic.

We started with a set of stages and steps for our case.

Stages are the least granular level of case decomposition, and represent the processing phases for the case that we want to model. Primary stages represent the preferred progression of our case, while alternate stages represent exceptions to the preferred progression. Within each stage, the steps represent actions that must be performed to process and ultimately resolve the case.

Once we establish the stages and steps for our case, we can focus on describing those actions. Each step represents either an assignment for an end user, a subcase to be created by the system, or a process. Assignments and cases are singular events – an end user or the system performs one act to complete the step – but processes generally represent multiple acts performed either by end users, the system, or a combination of the two.

Define (draft) process diagrams

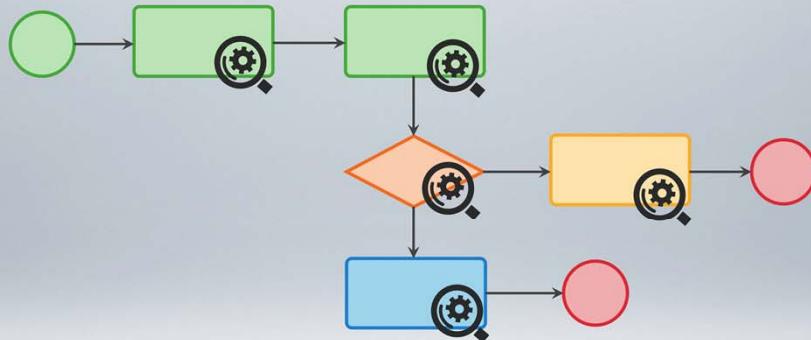


Assignment		A point of human interaction. The user selects the desired action to advance in the flow.
Decision		An automated decision. The system chooses the correct path and advances to the next shape.
Subprocess		A subprocess. The flows pauses until the subprocess completes, then resumes processing.
Utility		An automated system action. The system performs the action and proceeds to the next shape.
Start & End		Start and end points of the process.

Processes – also known as flows – consist of shapes that represent specific processing actions. These shapes – assignments, decisions, subprocesses, and utilities – allow us to model user and system actions. Flow shapes represent rules that PRPC uses to process a case as it advances through the process.

We also discussed briefly some more advanced flow shapes. We'll learn to use some of these shapes later in this course or in the System Architect Essentials II course. Other shapes are more advanced, and fall to Senior or Lead System Architects to configure. Even with just these 6 basic shapes, however, we can model fairly complex processes involving user actions, system actions, and policy decisions.

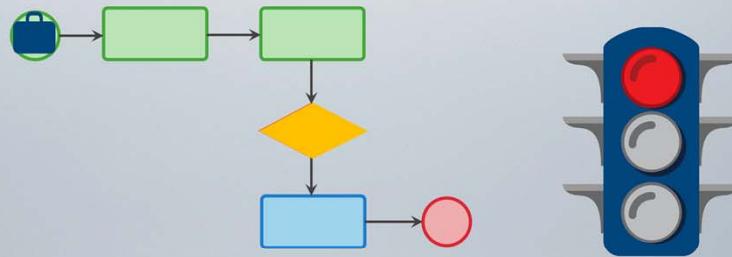
Specification and flow shape relationships



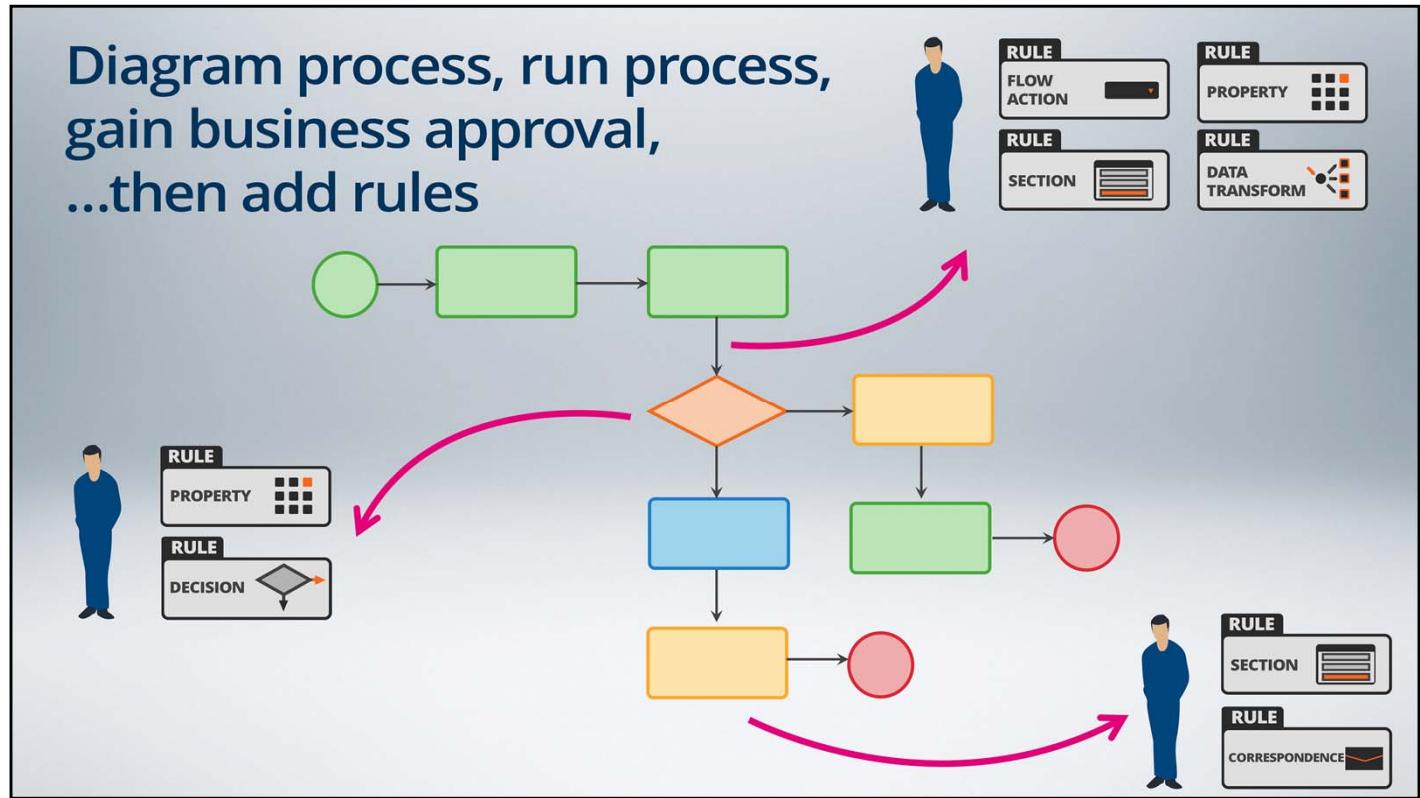
A use case that describes a proposal to address a business need

To understand how to build each process – and configure each shape – we rely on specifications. Each specification – a use case that describes a proposal to address a business need – describes the behavior of a user interface, decision, expression, or other component of our solution.

Specifications and process diagram drafting



Business analysts work with subject matter experts from the business to develop these specifications, which we can then read and implement. We start by creating process diagrams to draft a model of the desired process logic. We can then run through this draft process with business analysts and subject matter experts to confirm our understanding of the solution with the business. We can refine the model as needed until the subject matter experts are satisfied with the proposed implementation.



As system architects, once we provide the business with a model that adequately represents the flow of the process, we can create rules to represent user interfaces, our data model, and policy logic.

Module 05: Defining the Data Elements

This lesson group includes the following lessons:

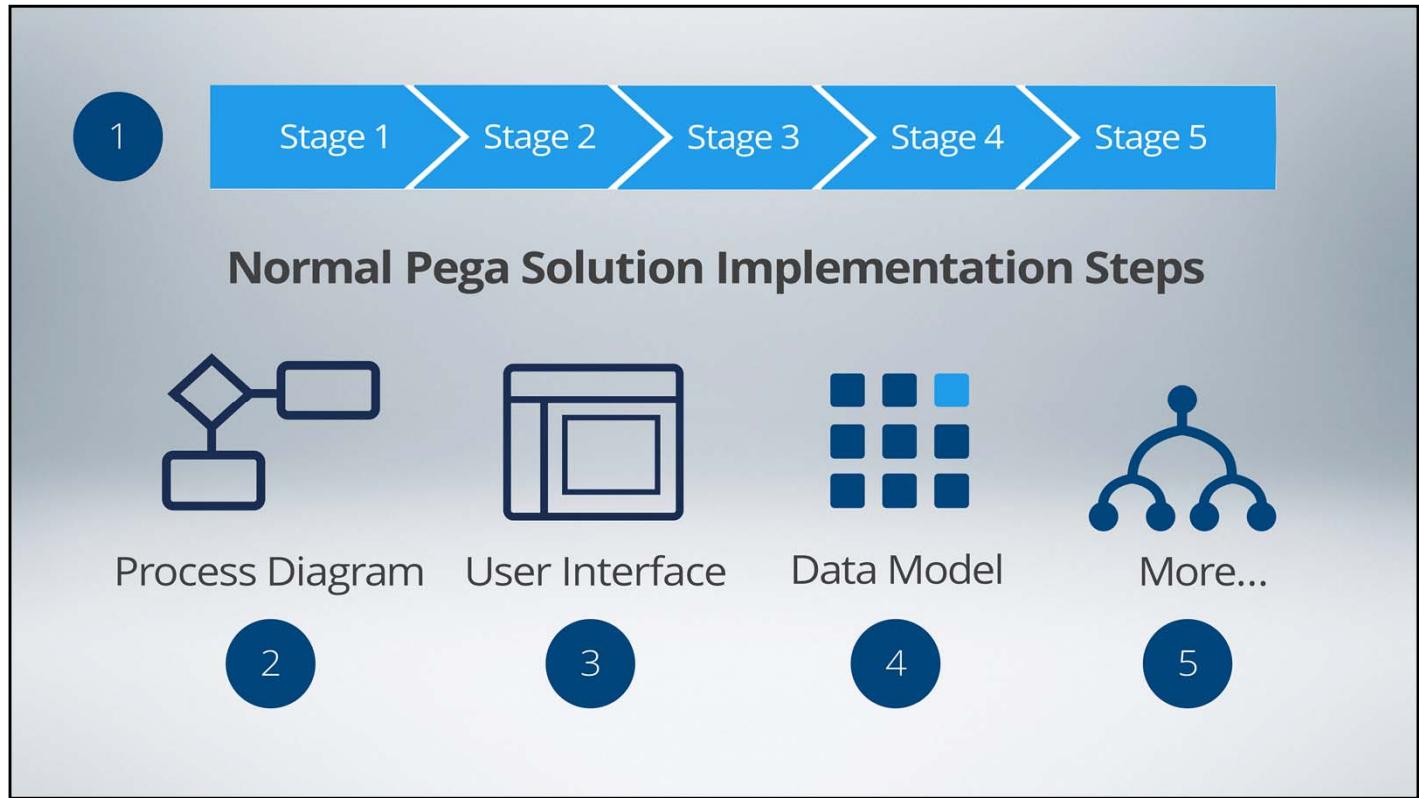
- Introduction to Data Elements
- Introduction to Data Modeling
- Setting Values of Data Elements
- Inspecting Data Elements
- Review of Defining the Data Elements

Introduction to Data Elements

To process cases, we must understand how to create and use properties to store data in PRPC

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

- Understand property modes
- Differentiate between the different types of single-value properties
- Create single-value properties to store data



Now that we have elaborated our case into steps and stages, we can start to expand upon this framework and create functional elements that end users can use to process and resolve cases. To do this, we need to create user interfaces to present and collect information, data elements to store that information, and business policies to act upon it.

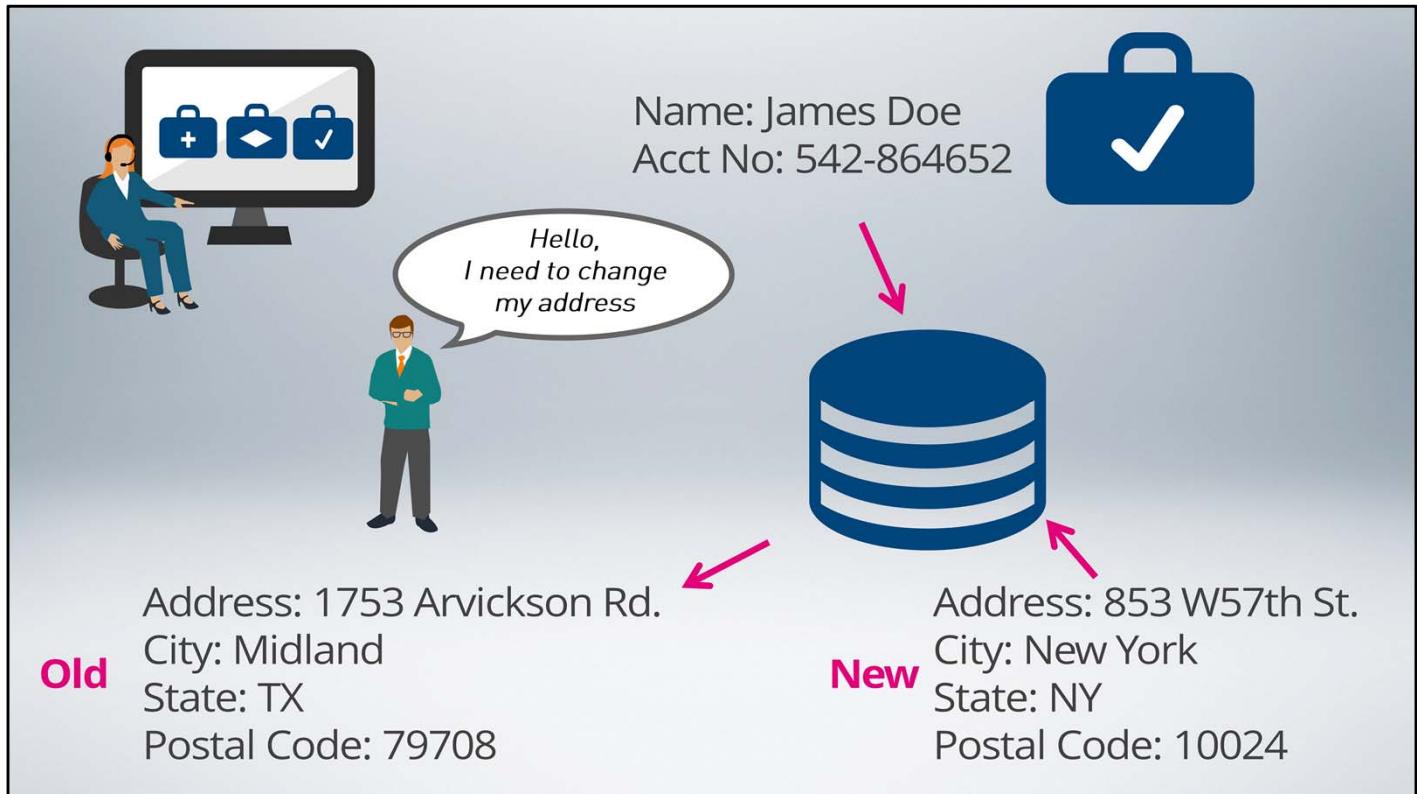
Normally, as part of a Pega solution implementation, we would progress from our process to creating user interface drafts, or mock-ups. This allows the implementation team to design the appearance and flow of a solution without the need to create lots of other rules – namely, those that implement the data elements and business policies that we just mentioned.

However, to more effectively teach the basics of Pega solutions, we will switch that order slightly. In this course, we discuss data elements and data modeling *first*, before moving on to the user interfaces, and ultimately business policies.

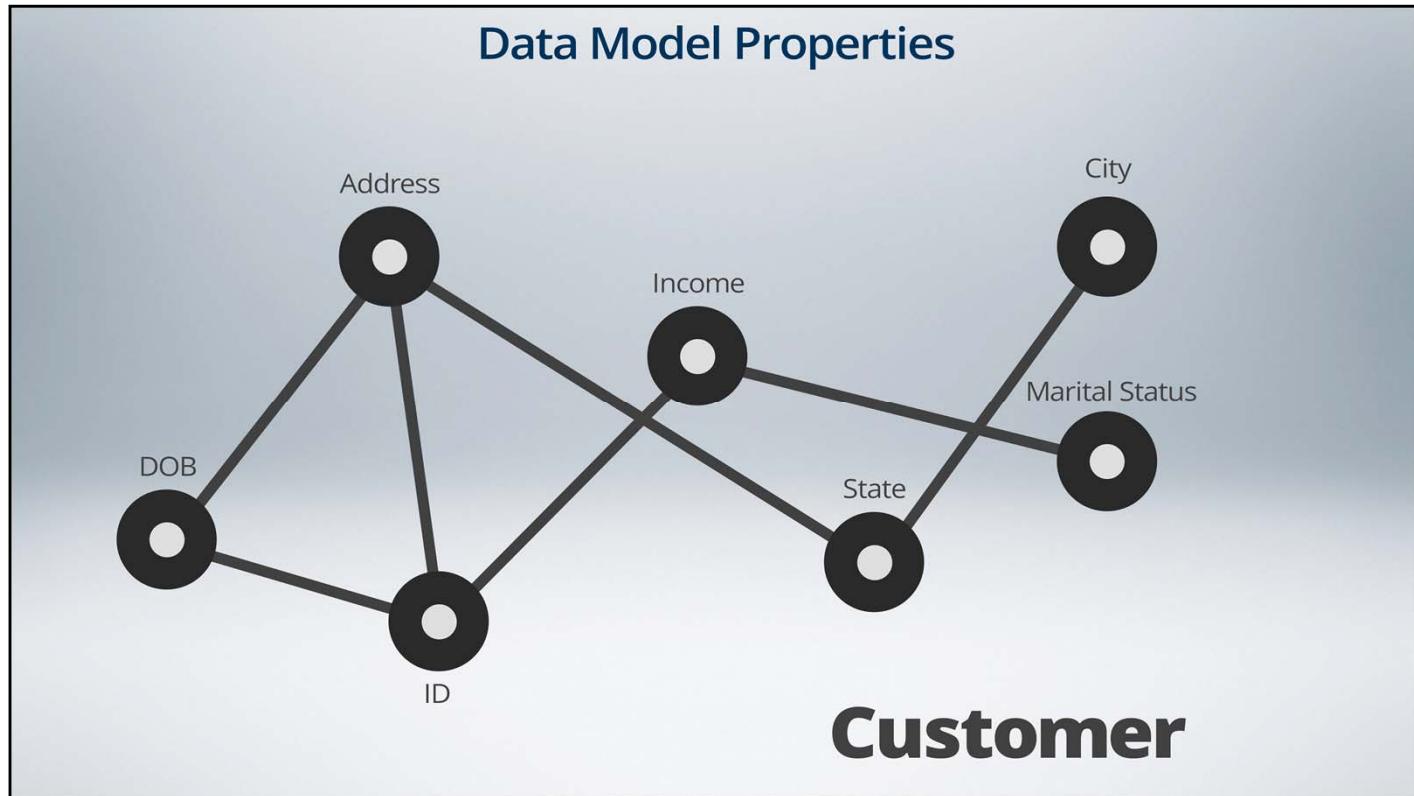
To understand how Pega solutions process cases, we need to understand how PRPC handles data. To do that, we start with the fundamental unit of data: properties.



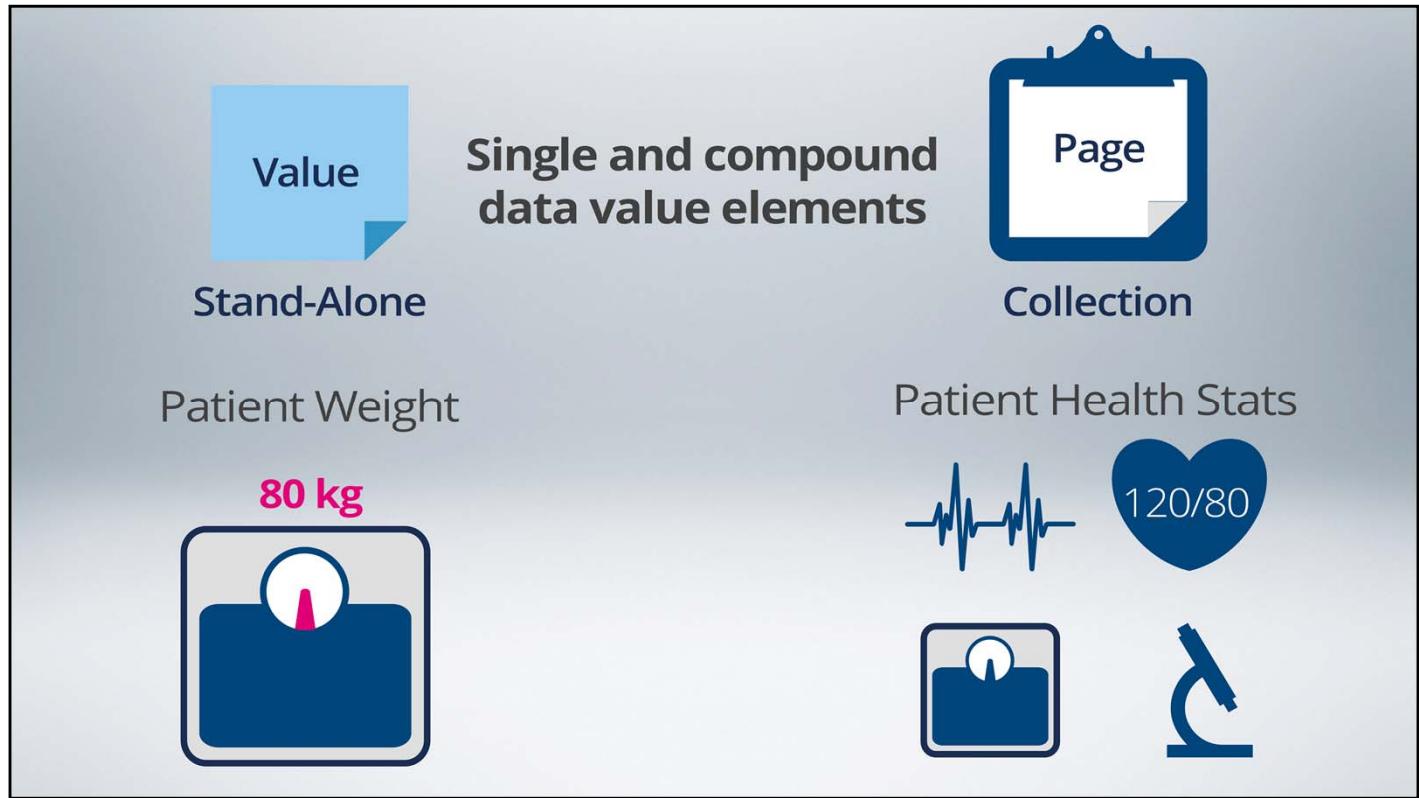
In a Pega solution, we want to allow end users to create, process, and resolve cases. This requires information – we collect information important to the case, which we then use to make decisions to best process and resolve the case. Without data, we don't have a case to process.



For example, if we want to create a case to process a change of address for a customer – say for a checking account or an insurance policy – we need data: the identity of the customer, the current address, and the new address.



As part of our implementation of a case, we create a data model to represent and store information. The fundamental unit of the data model – the entity that stores the data we collect from and present to an end user – is called a property. In the next lesson, we will learn how a property relates to the over-arching data model. For now, however, let's just focus on what a property is, and how PRPC uses it to store data.



When we look at data, we generally look at it in one of two ways: as a series of independent, unaffiliated elements, or as a grouping or collection of related elements.

If a patient asks their doctor for their weight, the doctor puts them on a scale and answers their question with a number.

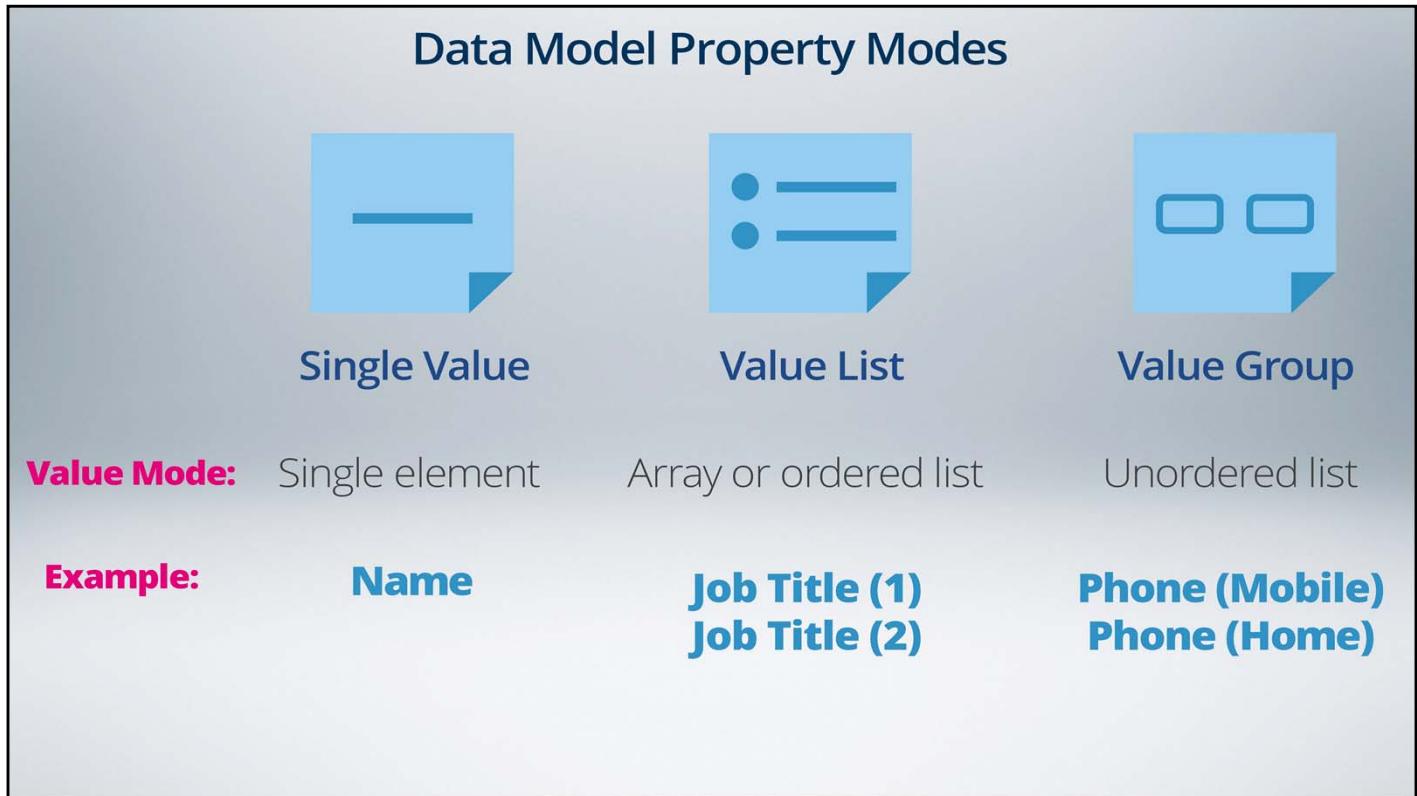
Now, if instead they ask their doctor about their *health*, they get *much* more information – not just their weight, but other information, such as: how their weight compares to the population in general, their blood pressure, their pulse, and the results of a blood test or two. By reviewing all of this information, the doctor can answer the question “How am I?”

So, we have two different types of information, depending on the question we ask: a single value – our weight – or a grouping of information.

In a Pega solution, data is treated the same way; as either a discreet, single value with no intended correlation with any other value, or as a collection of related values. This distinction is explained by the mode of a property.

As a general rule, system architects worry about two types of modes for a property: value modes and page modes. Value modes describe a single piece of information, such as weight, while page modes describe an aggregation of information, such as a patient.

For the rest of this lesson, we will focus on the value modes – specifically, single-value properties. In the next lesson, we discuss the page modes, and how they relate to value-mode properties. PRPC also supports two other types of property modes: Java objects and Java properties, which are more advanced topics and beyond the scope of this course.



Within our solution, we can use one of three value modes: Single Value, Value List, or Value Group. That sounds complex, but really isn't. Let's look at some examples to better understand what each mode represents.

A property of mode Single Value – also known as a single-value property – represents a single piece of information, with no implied relation to any other information. Name is an example of a single-value property.

A Value List is an ordered grouping of related information. If you wanted to discuss someone's employment history – where the order is important – you could look at the job title as an ordered list. The first item in the list might be "Associate Editor", the second might be "Editor", and the third "Senior Editor". Each is still a single piece of information, but there's a clear relationship between all three, and we need a way to identify each one.

A Value Group is an unordered grouping of information. If we ask someone for their telephone number or email address, they might respond with the question, "which one?" So, it's important to qualify which telephone number we want: the number for their telephone at home, or their telephone at work, or the number for their mobile telephone. We don't usually need to know all of their numbers – we only dial one number at a time – but it's important to have access to all of them, so we know *which* one we need to dial. That means we need to be able to identify the contextual meaning of each number: home, work, or mobile.

So, if we were to refer to each of these data elements, we might do so thusly:

- Name,
- Job Title (1), Job Title (2), or Job Title (3),
- And Phone (Home), Phone (Work), or Phone (Mobile).

Property Types

Age

Name

Date of Birth

Text

Number

Date

10 Standard Property Types...



As a system architect, the property mode we use most often is single-value, so we'll concentrate on that particular property mode for the rest of the lesson. However, remember that no matter which mode we use, these concepts still apply.

When we want to store information, we first need to understand what type of information we have. Is it a number? Text? A date or time?

To differentiate between these differing types of information, we can assign a type to a property. This allows us to identify the type of information we want the property to store. By assigning a type to a property, we can also ensure that users provide valid information. For example, if we ask someone to provide their name, we want text, and not a number. If we ask their age, we want a number, not a description such as "young" or "old". And if we ask their date of birth, we want a specific type of number – a date.

When we create a property, we can assign to it one of ten different types. Let's discuss them briefly.

Single Value	Text	Identifier	Password	Encrypted Text
	A string of characters The defendant said "No".	Text strings that disallow certain characters Street Address	Encrypted displayable characters *****	Encrypted displayable characters *****
Example	A string of characters The defendant said "No".	Text strings that disallow certain characters Street Address	Encrypted displayable characters *****	Encrypted displayable characters *****

First, we already discussed one type: text. Text properties store a series of characters – sometimes referred to as a “string” of characters – such as “Steve”, “duck, dodge & weave”, or an entire chapter of a book. Text properties can store any allowable Unicode character – both graphical characters and non-graphical characters such as tabs, line breaks, and carriage returns.

We also have several variations on textual data:

- Identifier, which can only store text strings that do not contain double quotation marks (""), tabs, carriage returns, or line breaks.
- Password, which can only store encrypted graphical characters.
- Encrypted Text, which can store encrypted graphical characters – similar to the password type – but can be decrypted for display.

Single Value	Date	Time of Day	DateTime
Example	Calendar date in the format YYYYMMDD	Local time in the format HHMMSS	Combination of date and time of day
	20131202	152709	December 2, 2013, 7:27:09AM

Then we have time-keeping types:

- Date, which stores a calendar date in the format YYYYMMDD – four digits for the year (YYYY), two digits for the month (MM), and two digits for the day (DD).
- TimeOfDay, which stores a local time in the format HHMMSS – two digits for hours (HH), two digits for minutes (MM), and two digits for seconds (SS). Local times are not dependent upon a time zone.
- DateTime, which stores a UTC (Coordinated Universal Time) value. DateTime data is stored as a combination of Date and Time Of Day formats, and all values are normalized to Greenwich Mean Time (GMT).

With these property types, the value presented to the end user may differ from the internal representation. An internal date of 20131202T052709 may appear to the user in a variety of formats.

Single Value	Integer	Decimal	TrueFalse
Example	Positive and negative whole numbers and Zero 453556345- 95304	Non-whole numbers 1.5798€52.58	Boolean values (Binary outcomes) True/FalseYes/No

Next, we have number types:

- Integer, which can only store positive and negative whole numbers, and zero.
- Decimal, which can store non-whole numbers. Decimals are well-suited for recording numbers with a few decimal digits, such as currency amounts.

And finally, we have a Boolean property type, named TrueFalse. TrueFalse properties are used to store data that *must* be one of two values – typically true/false, yes/no, or on/off.

Demo



Create single-value properties to store data

Now that we know the types of data that we can store, let's discuss how to create a property.

In Pega 7, we can create properties from a flow action. When we do, we enter the name of the property, select its mode and type, and provide some basic information about how we plan to use the property. We can also provide some basic configuration information about the property. We can define a property as a calculated value, a required value, or a read-only value, dependent upon how we intend to use the property in the user interface.

A calculated property is computed automatically by PRPC. We'll learn how to compute a value with a particular type of rule called a Declare Expressions rule in an upcoming lesson.

A required property must have its value provided by the end user when completing the form. If the user neglects to enter a value for the property and submits the form, the application returns an error message and prevents the user from advancing until they provide a value.

A read-only property is uneditable on the form. Read-only property references allow end users to review data, prevent them from editing the data.

When we finish, PRPC automatically creates the properties we specified, as well as the UI rule – called a section – that it needs to display them to the user. This allows us to test the properties by running our process and entering values into the provided fields.

But what if we need to restrict user input to a set of specific values? For example, we want users to enter an urgency for their purchase request – but we need them to enter that urgency consistently, from one purchase request to another.

With a property, we can define allowable values, from which the end user can choose the most-appropriate value. This list is part of the property definition, so whenever we use this property in the UI, the user will always have our allowed choices. Since we want to display a user-friendly label for each value, we can use a specific type of list called a prompt list.

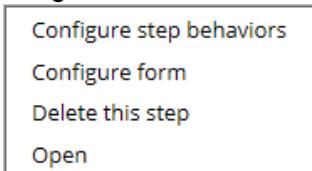
And the next time we view the property on the UI, we see our list of allowed values.

Pega 7.1.6 Update Notes

Edit Fields form has been replaced by Form Configuration

Form Configuration can be accessed in one of three ways.

- Configure form has been added to the step options menu



- Configure form has been added to the step options menu
- Configure form has replaced Add fields on the process outline.

A screenshot of a process outline for 'Assignment: Personal Info'. It shows a step labeled 'Actions' with the sub-step 'Collect Personal Info'. Below it, there is a button labeled 'Configure form' which is highlighted with a red box. Other options include 'View action specification' and '+ Add action'.

- You can configure a form while testing the case type.

A screenshot of a dialog box titled 'No form has been added to Enter Request Details'. It asks 'Would you like to work on this now?' with two buttons: 'No thanks, maybe later' (blue) and 'Yes, I'll add a form now' (orange).

The Edit Fields form has been replaced by Form Configuration

A screenshot of the 'Form Configuration' dialog. It shows a field configuration section with a placeholder 'Enter field here ...' and dropdown menus for 'Text' and 'Optional'. There is also a '+ Add field' button and 'Save' and 'Cancel' buttons at the bottom.

The List of available field types while configuring a form has been updated

Single value
Text
Text (multiple lines)
Search
Date
Date and time
Time
Number (integer)
Number (decimal)
Currency
Checkbox
Dropdown
Complex types
Object
List of objects

Exercise: Create Single-Value Properties

Exercise: Add Allowed Values to a Property Definition



Introduction to Data Modeling

In addition to properties, we also need to create data objects as part of our data model – to establish a correlation between properties in a set, and differentiate one unique set from another

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

- Understand the concept of a data model
- Explain the difference between a page, a page list, and a page group
- Use a page, page list, or page group to store data

In the previous **lesson**, we discussed...

- Numbers**
 - ▼ A person's age
- Text Strings**
 - ▼ Name
 - ▼ Occupation
 - ▼ Home Address
 - ▼ Birthdate
 - ▼ Martial Status



In the previous lesson, we conducted a basic discussion of data elements in PRPC, and focused primarily on single-value properties – the most-basic elements of data storage for our solutions.

As part of that discussion, we looked at simple types of data: numbers, such as a person's age, and text strings, such as their name or home address.

Data Modeling

Establishing Relationships Between Data

Frank James

1750 East 93rd Street
17 April 1976
Married
Mechanical Engineer





Page



Page List

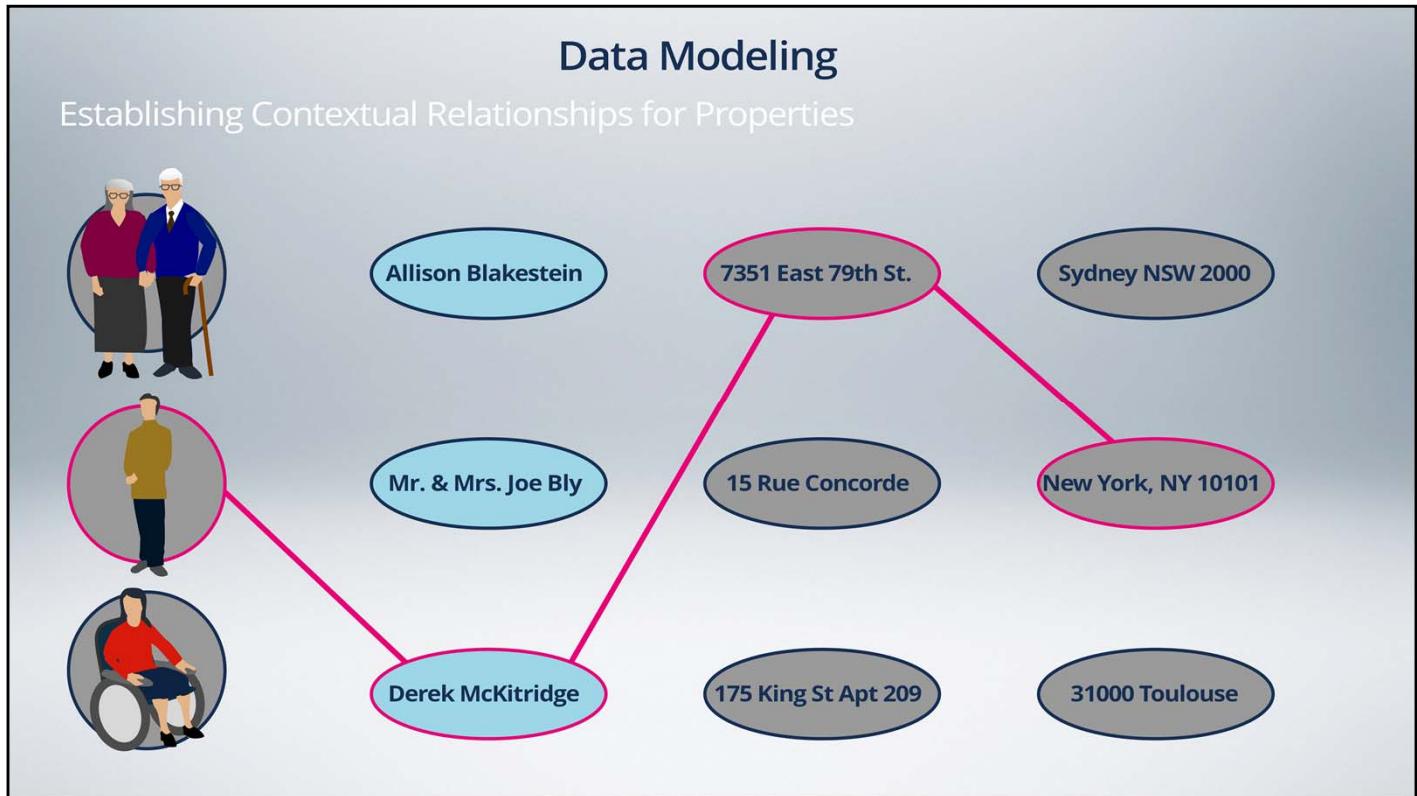


Page Group

We also discussed how single-value properties are all distinct from each other.

By itself, a name has no contextual relationship with any other information, such as an address – the two are unique bits of information. But what if a relationship between name and address *is* important? Perhaps we want to correlate a specific person's name and their home address. If we want to mail our roommate a postcard from our ski vacation, that relationship between name and address becomes *very* important, and we need a way to enforce it.

If we need to establish a contextual relationship between two single-value properties, we can use one of the three page-mode properties defined in PRPC: pages, page lists, and page groups. Let's see how we can use these property modes to establish a contextual relationship.



To understand the importance of contextual relationships, we need to talk about data modeling.

Let's say we want to send a postcard to someone while we're on vacation. The name of the person is important, as is the address. More important, though, is that the name and address correlate. We want a specific person, at a specific address. How can we ensure that we use the right combination?

A data model is a representation of a data structure. It explains to us what our data objects are, and tells us what distinctive elements define them. In our example, a person – such as the recipient of our postcard – is represented by a data object. Let's call this object "person".

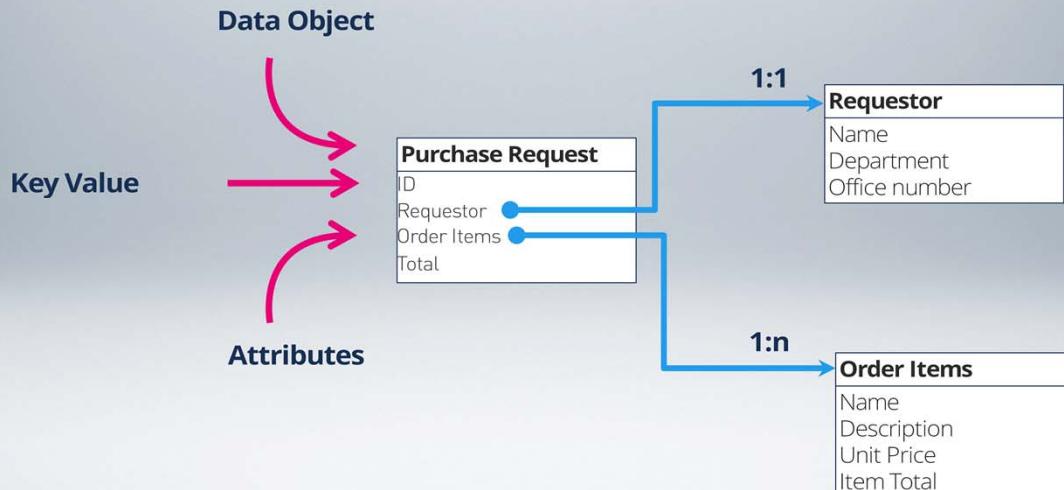
Each person has a set of distinct attributes. In this case, we're not concerned with physical attributes – tall, blue eyes, or brown hair – but rather other attributes, such as street address, unit number, city state/province, country, and postal code.

So, if we look at our problem this way, we need a unique way to identify the data object that corresponds to our friend – a type of "key value" – and then obtain from that data object the attributes that define his or her address. In essence, this is the basic building block of a data model: a data object that consists of a set of attributes, identified uniquely by its key value.

The remainder of our data model describes the relationship between data objects: some objects have a one-to-one relationship, such as a person and their home address. Other objects may have a one-to-any relationship, such as parents to children.

Creating a Data Model

Establishing a Relationship between Data Objects



So, for a Pega solution, we might have a data model that looks fairly complicated. But, with what we've learned so far, it's not that tough to figure out. Let's think through what a data model for a purchase request might look like.

First, we have a data object that represents our purchase request.

This purchase request has several attributes: a unique identifier, the person placing the request, the items they're requesting, and a total price.

Two of these attributes are themselves data objects: the Requestor, with the attributes name, department, and office number; and the Order Items, with the attributes name, description, unit price and item total.

And we can describe the relationship between these objects. Each purchase request can have one requestor, and any number of order items.

Creating a Data Model

Establishing Page-Mode Properties

	Value-mode	Page-mode
Single element	Single Value Name	Page Position Details
Array or Ordered List	Value List Job Title (1)	Page List Order Item (1)
Group (Semantic List)	Value Group Phone (Mobile)	Page Group Address (Home)

Now that we understand how a data model works, we can discuss how one is implemented in PRPC.

First, the attributes for our data object are represented by value mode properties – most frequently single-value properties, the kind we discussed in the previous lesson.

The data object itself is represented by a page-mode property. In a Pega solution, we have three different modes; a Page, a Page List, and a Page group.

If we remember the previous lesson, this sounds somewhat familiar. When we discussed value-mode properties, we learned that there were three types: Single Value, Value List, and Value Group.

Page-mode properties are organized similarly. A Page is a single entity, such as an open position, or an item for sale.

A Page List is a numerically ordered list, such as the specific items that make up a purchase request.

And a Page Group is a semantically ordered list, such as a set of addresses – home and work.

One thing we need to remember: the choice between a page, page list, or page group depends upon the relationship of the object we're describing to its parent. If we want to capture a single address for a person, we would use a page. If we want to capture information about their work history – employers, job titles, and dates of employment – we would use a page list, since the sequence is important, and the number of jobs may be extensive. If we want to capture information about their undergraduate and graduate degrees, we could use two page groups – one for an undergraduate degree, and the other for a graduate degree.

Demo



Use a Page to Collect and Store Data

So, now that we understand the use of pages and page lists, we can begin to use them in our solution.

In our Purchase Request case, we need to create properties to store the shipping and billing address. Let's look at how we can create a page of properties to store the requestor's shipping address. Eventually, we might want to populate this information automatically, but for now, we just need to define the properties and the page that contains them.

As we saw in the last lesson, we can create a property from a flow action. Since we want to create a page that contains properties, the process is similar. The one difference is that we need to select that the property is a page. If we wanted to create a page list or page group, we would select the corresponding option.

When we select a page, page list, or page group, the dialog adds a new section to capture the properties that belong on the page. At the bottom of the dialog, we can still create a property not on the page.

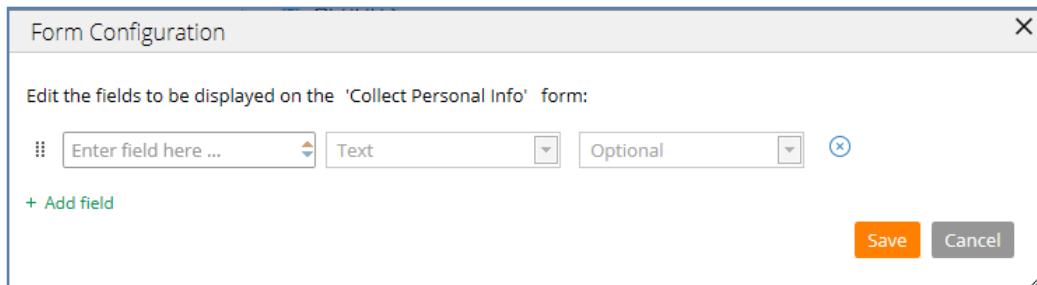
So, all of the properties we create for the address go on the page.

And when we finish, PRPC automatically creates all of the needed rules – UI, properties, and everything else – needed to use the property in the UI.

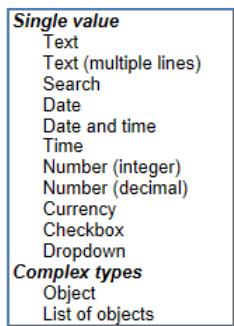
Pega 7.1.6 Update Notes

Edit Fields form has been replaced by Form Configuration

In this lesson, we will again see the use of form configuration.



... and a list of types available for new property creation.



Exercise: Create and Use Property Pages



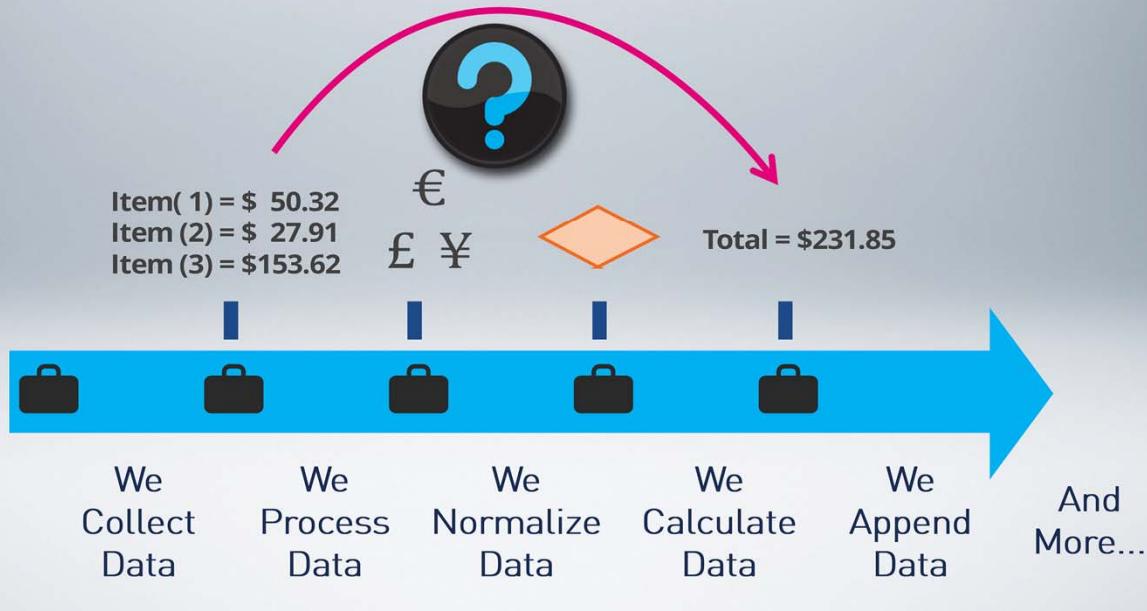
Setting Values of Data Elements

To process a case, we need to collect, act upon, and display data. To do this, we must understand how to reference data, and how to manipulate data with a data transform.

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

- Reference a property in PRPC
- Use a data transform to set a property value

Manipulating Data: Collect and Process Data



When we create and process a case, we need data: we collect it, we process it, we act upon it, and we present it back to the end user.

Some of the time, our data isn't in a form that we require, so we need to find a way to manipulate it into an acceptable form. For example, we collect an individual's first name and last name, but want to combine them into a full name. Or perhaps we need to sum all of the elements of a list.

To do this, we need to understand how to refer to our data elements. These data references allow us to link a field on a form, such as First Name, to the property that contains the value. Or add two numbers together. Or evaluate properties as part of an automated decision that reflects a business policy.

Referencing Data: Value-Mode & Page-Mode Properties

- ▼ Value-Mode = single strings of data: text; numbers; dates
- ▼ Page-Mode = containers for value-mode properties



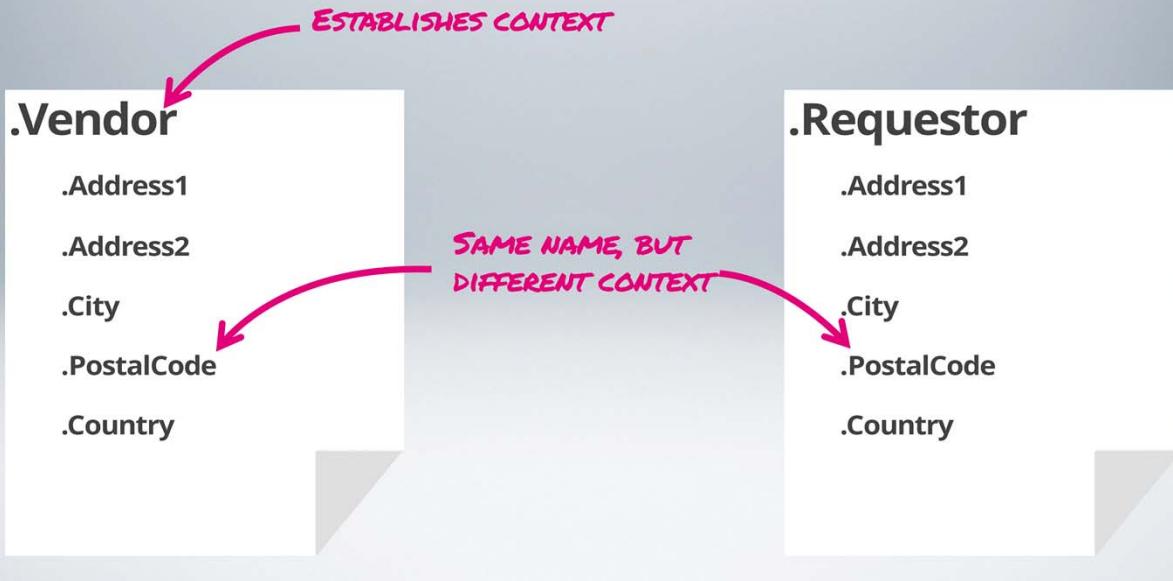
So far, we've looked at two property modes: value, and page. Value-mode properties store single strings of data – text, numbers, or dates – while page-mode properties act as a container for value-mode properties.

When we refer to a property in PRPC, we prefix the property name with a period “.”. So, to refer to a single-value property named “City”, we type **.City**. For a value-group property, such as a mobile telephone number, we type **.Phone(Mobile)**.

Page-mode properties are no different. To refer to a page that contains contact information for a vendor of equipment or services, we might type **.Vendor**. And to refer to the third page of a page list that contains information about our financial assets, we might type **.Asset(3)**.

And if we want to refer to a specific property on a page, we use the name of the page as a prefix for the property name. So, “city on the vendor address page” becomes **.Vendor.City**. And “type of the third asset on our list” becomes **.Asset(3).Type**.

Referencing Data: Establishing Context



By using the name of the page as a prefix for a property, we establish an important piece of information about that property – its *context*.

As we learned in previous lessons, each property in our solution is unique. By establishing the context for a property, a page – by itself or as part of a page list or page group – acts as a container for the properties it contains.

Since each page is a distinct container, each property defined on that page only needs to be unique to the page. So, to ensure the correct context for our property references, we need to include the page – or pages, since pages are themselves properties that can be defined for a specific page – that contains the property we want.

Property Context: 2 Important Rules

1

We always define properties on a **specific** page

2

Context matters only in **ambiguous** situations



Before we continue, there are two very important rules regarding property context that we need to understand.

1. Every property we use is defined for a specific page.
2. Once we establish our context, we don't need to prefix any property within that context, as long as we stay within that context.

In PRPC, a page is basically a portion of system memory set aside for us to define properties and read their values. Every property we use is defined on a page *somewhere* in system memory. That includes value-mode properties such as the ones we discussed at the beginning of the lesson – .City and .Phone(Mobile). We didn't need to provide the context for these properties because they are defined on a special page in memory called *pyWorkPage*.

When we create a case, we store all of its data on this special page, *pyWorkPage*. When we process our case, we create, read, update, and delete properties on this particular page. So, as long as we confine our operations to this page – by not accessing any other pages to define or read data – we don't need to provide a page reference, because the system does that for us as we process our case.

So, since our case processing occurs within the context of *pyWorkPage*, we don't need to use a prefix for any property defined on that page. Though, since we have defined pages within *pyWorkPage* – which we refer to as *embedded* pages – we need to remember the prefix for properties defined on *those* pages. There are a few situations in which we must *clearly* establish the context of our data elements. One is when we use a type of rule called a data transform to

read and set property values.

Manipulating Data: Data Transforms



In the Candidate process, end users provide a substantial amount of information about themselves when they apply for a position. As we process their case, we may need to copy data from one property to another. Or, we may need to manipulate it – to transform it somehow. And maybe we need to do both at the same time.

One of our options for copying and manipulating data in our cases is the data transform. The purpose of a data transform is summed up in its name: it transforms data in our application. That can be as simple as copying a value from one property to another, but can also include more complex manipulations, such as adding numbers, concatenating strings, and even copying entire pages.

Data Transforms: Copy and Manipulate Data



Let's look at a fairly simple example. Suppose we're purchasing items online: we fill our cart with the items we want, and begin the checkout process. We provide a shipping address, and our credit card information, and are prompted to provide a billing address.

If we plan to have our purchase delivered to our home, then our billing address and shipping address are likely the same. Rather than re-entering our address, wouldn't it be helpful if we could just *reuse* it when we're prompted for our billing address?

Enter the data transform. We can create a data transform that reads property values from one page, such as the shipping address, and copies them to properties on another page – in this case, the billing address.

We can use a data transform several ways: we can call it from a flow action rule, we can call it from a connector, and we can even use a special data transform rule – pyDefault, which the New Application wizard creates for us automatically – to initialize property values when we create a case.

Demo



Use a Data Transform to Set a Property Value

Let's investigate how we can use a data transform this way.

We start by looking at a very simple example. Once our purchase request has been submitted, it needs to be approved. To facilitate the approvals process, we need to copy the manager from the Cost Center page to pyWorkPage.

In the Get Manager Approvals process, we can call a data transform immediately upon beginning the process.

With this data transform, we want to read a value from the Cost Center Manager property on the Cost Center page, and write that value to the Cost Center Manager property on pyWorkPage. We don't need to add a page reference to this property because it's already within context – the property is defined on pyWorkPage, along with all other case data.

Data transforms can be used to iterate over page lists or page groups, and copy entire pages at a time, but these actions are – from a conceptual standpoint – similar to our example here. The most important thing to remember when using a data transform is to establish our context correctly when reading and writing property values.

Exercise: Set a Property Value with a Data Transform



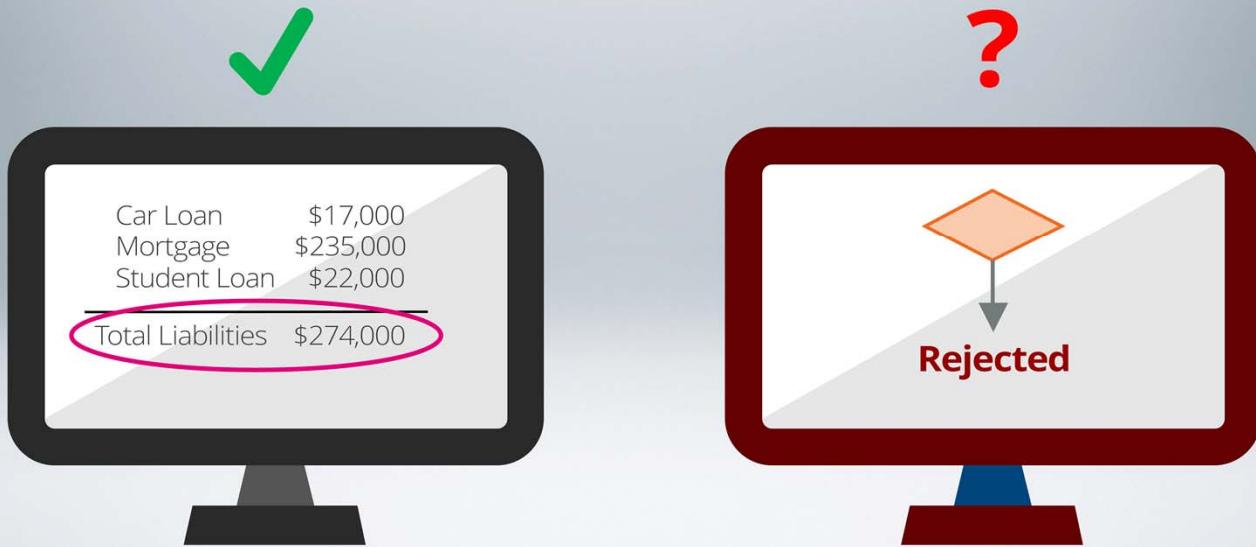
Inspecting Data Elements

The Clipboard provides a way to view and interact with in-memory data, which helps us understand how an application manages that data and allows us to debug issues with our data.

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

- Navigate the clipboard
- Use the Clipboard tool to monitor property values

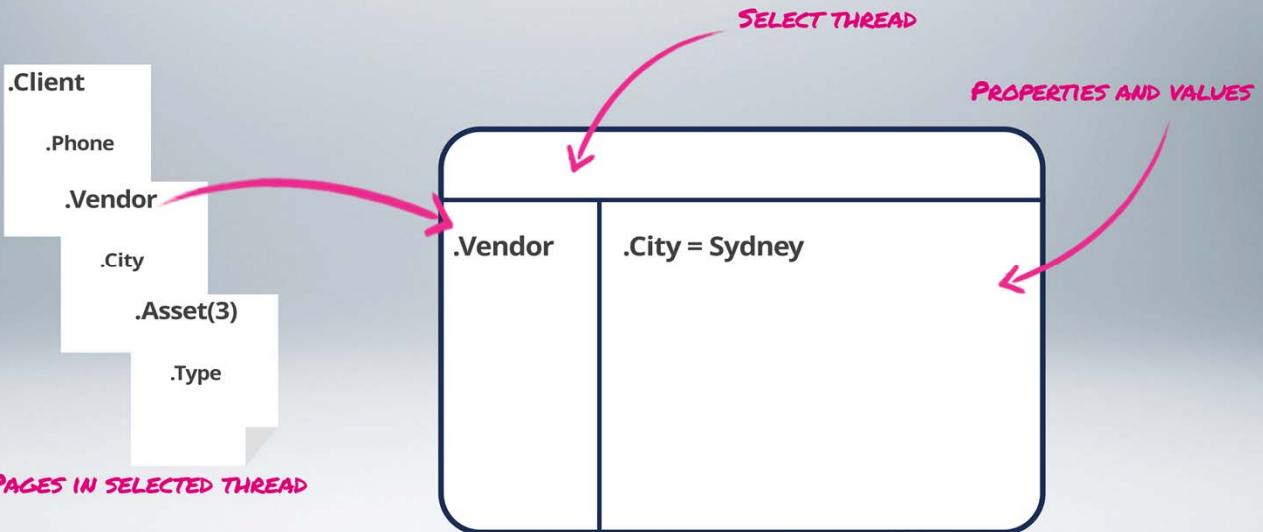
Inspecting Data: On-Screen vs. Off-Screen



As we learned previously, PRPC uses pages to store data in memory as we process a case. If that data appears on-screen while we work on a case, we can easily ensure that our data is accurate.

But what if the data *doesn't* appear on-screen? How can we ensure that the data we *don't* see is accurate?

Navigating the Clipboard: Understanding the Layout



To help us verify all of the data in our solutions – including the data we can't see, but that our case relies upon, and which *must* be correct – we can use a tool called the Clipboard.

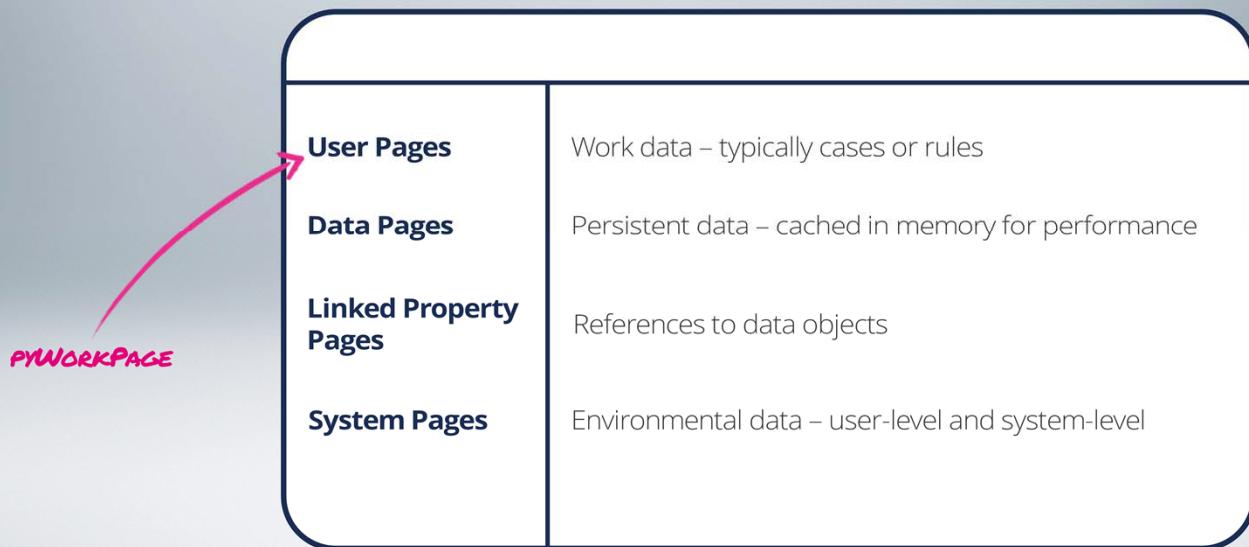
The purpose of the Clipboard tool is simple: for any page in memory, it displays the properties defined on that page, and their values. The tool is divided into three areas.

The top area allows us to select the thread to view. Each thread corresponds to a unique action currently managed by PRPC. One thread is dedicated to the Designer Studio environment in general. Other threads are dedicated to open rule forms. And, if we're currently processing a case, PRPC assigns the case to its own, unique thread. A full discussion of threads is outside the scope of this course, but essentially a thread is a container in memory – similar to a page, but bigger.

When we open the Clipboard tool, it identifies the active thread – generally the active tab in Designer Studio – and displays all of the pages defined for the thread.

To view the contents of a page, we simply identify it in this left pane, and click on it. The tool displays all of the properties on the page – and their values – in the right pane.

Navigating the Clipboard: Understanding Page Categories



The left pane lists all of the pages defined within the selected thread, grouped into four categories:

User pages, which contain data about the work being performed – such as updating an open case (for end users) or configuring a rule (for application designers).

Data pages, which contain any active data pages. Data pages are persistent, in-memory data sources that we can use to cache data and boost performance in our applications. We'll learn about data pages in an upcoming lesson.

Linked property pages, 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), and outside the scope of this course.

And **system pages**, which contain information about the operating environment, such as the active user and the active application.

For the remainder of this lesson, we focus on user pages – specifically, pyWorkPage – though the concepts we discuss apply to any clipboard page, in any category

Demo



Use the Clipboard to Monitor Property Values

Once we create or open a case, we can use the Clipboard tool to locate pyWorkPage within the User Pages category, and view all of the data defined for our case. This data includes not only the information we provide by completing and submitting forms, but also information from external sources, along with information automatically defined by PRPC.

If we want to review data on an embedded page, we can expand each page – and its children – until we drill down to the page we want.

Some of these pages are created automatically by PRPC to properly handle cases. For example, pxFlow contains information about the current process, such as the current step in the process – which in our case is the flow PurchaseRequestEntry.

At this point, pyWorkPage contains some basic data about our purchase request. We can also add, modify, or delete properties. For example, we can change the cost center. Adding and modifying property values can help us to debug rules, by allowing us to provide values for data elements that may not yet be incorporated into our data model.

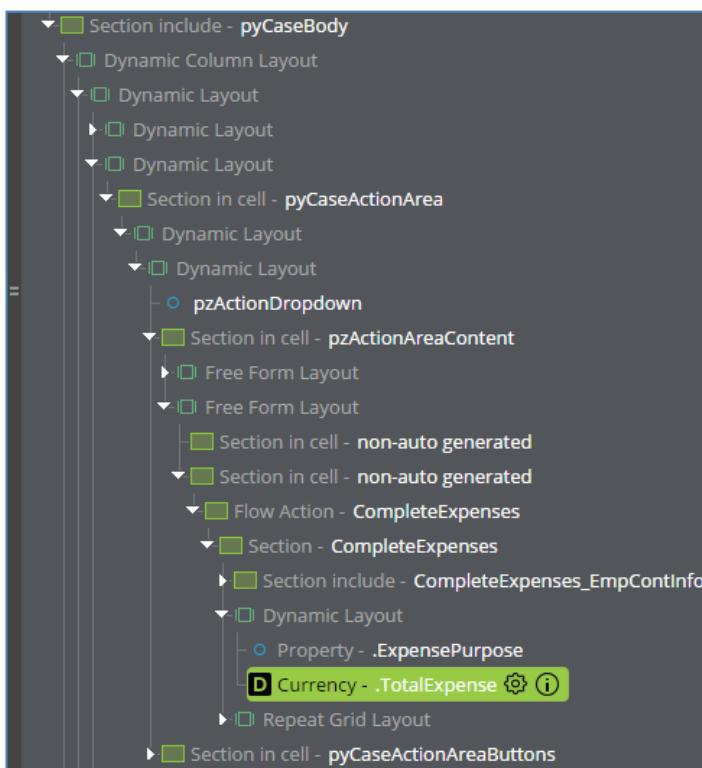
pyWorkPage also contains a page called LineItems, which contains information about the items we request. As we continue to process our case by adding line items, we can refresh the contents of the clipboard to reflect this additional data – or any other data we collect, either from user input, integrations, data transforms, or declarative expressions. And when we close the case, its information disappears from pyWorkPage.

Pega 7.1.6 Update Notes

New UI Tree

The UI Tree tool helps you to understand the structure of the currently displayed page at runtime and in Designer Studio.

- Once activated, a sidebar column is displayed listing a tree of all artifacts on the current page.
 - Clicking any of the items collapses the list to show only the hierarchy specific to the selected element



- You can navigate through Pega artifacts on the page either:
 - Using the tree representation
 - Directly hovering over the artifact on the page itself.

Exercise: Review Case Data with the Clipboard

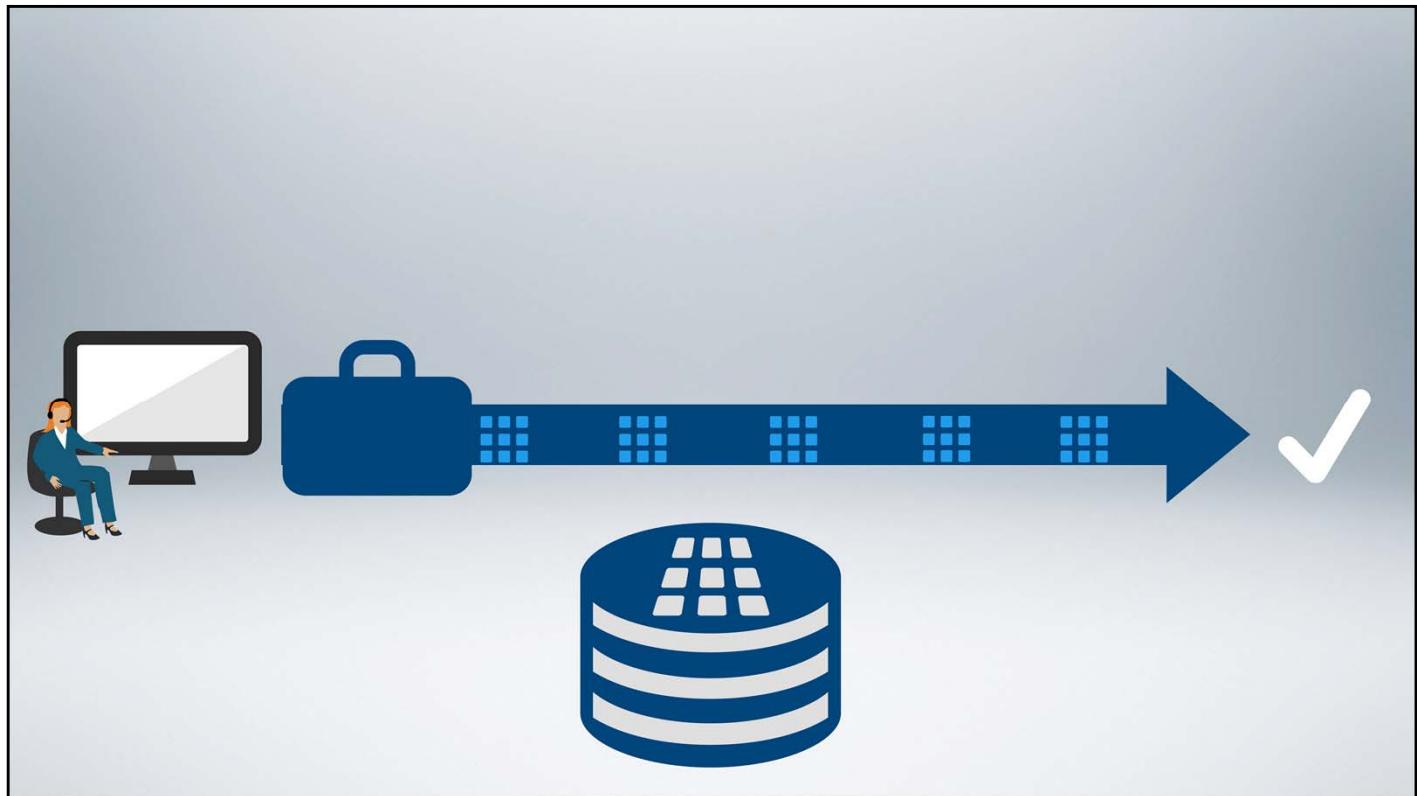


Review of Defining the Data Elements

Proper construction, use, and maintenance of a data model requires that we understand how to properly reference properties to read and write data.

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

- Understand the difference between value-mode and page-mode properties
- Reference value-mode and page-mode properties
- Set property values with a data transform
- Review case data with the Clipboard



In the previous four lessons, we discussed the concept of data modeling in a solution.

We learned that a data model consists of properties, which we use to store data generated while processing a case. As end users process each case, its data is written to a database, allowing other users to reopen the case at a later date and continue processing it, and to report on the data across multiple cases.

Value	Page
Stores an actual value (data)	Contains single-value properties
Supports one of 10 data types, to store text, number, date, time, or Boolean data	Represents a data object, defined by its characteristics (single-value properties), and identified by a key value
Supported usage :Single element, ordered list, or unordered list	Single page, ordered list of pages, unordered list of pages

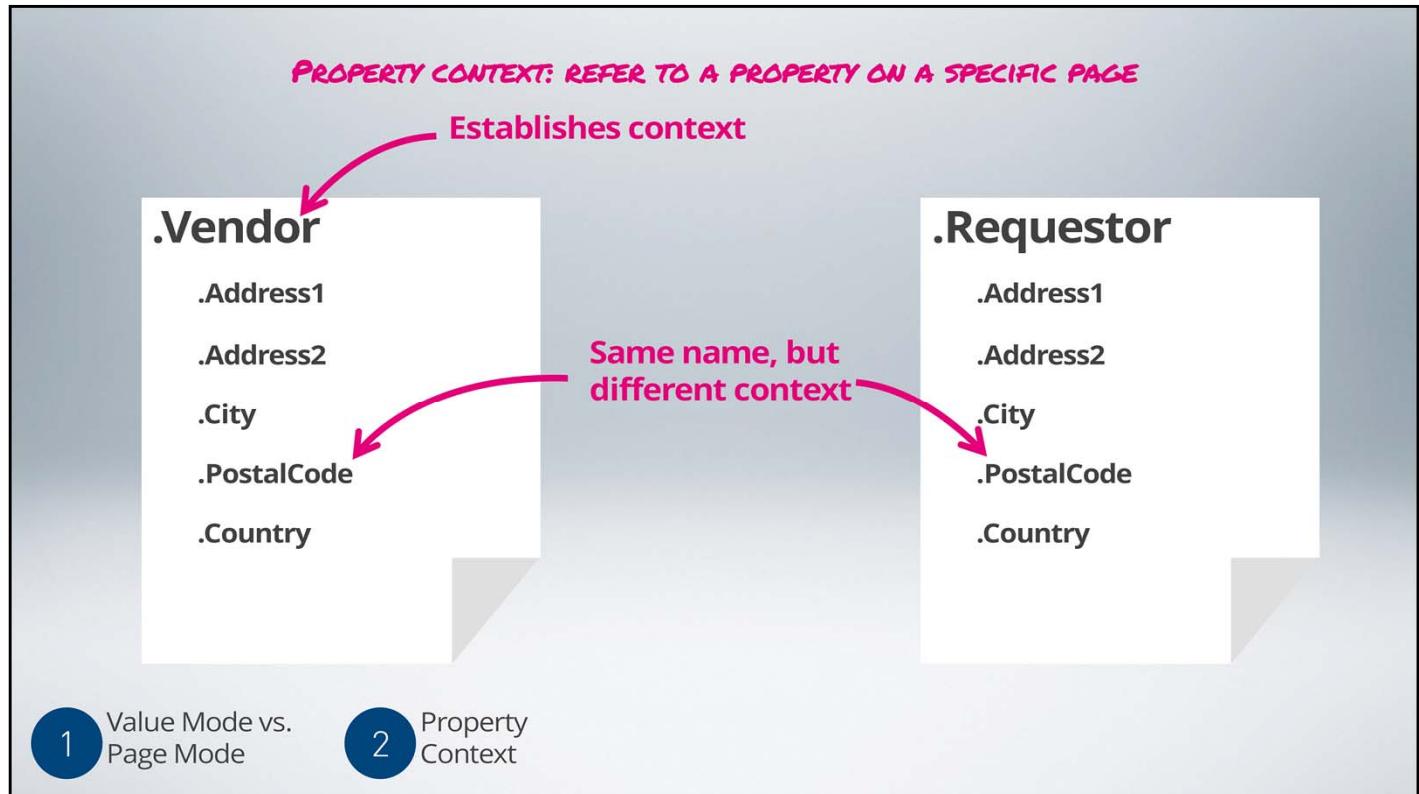
1 Value Mode vs. Page Mode

We also learned the two property modes used most often in Pega solutions: value and page.

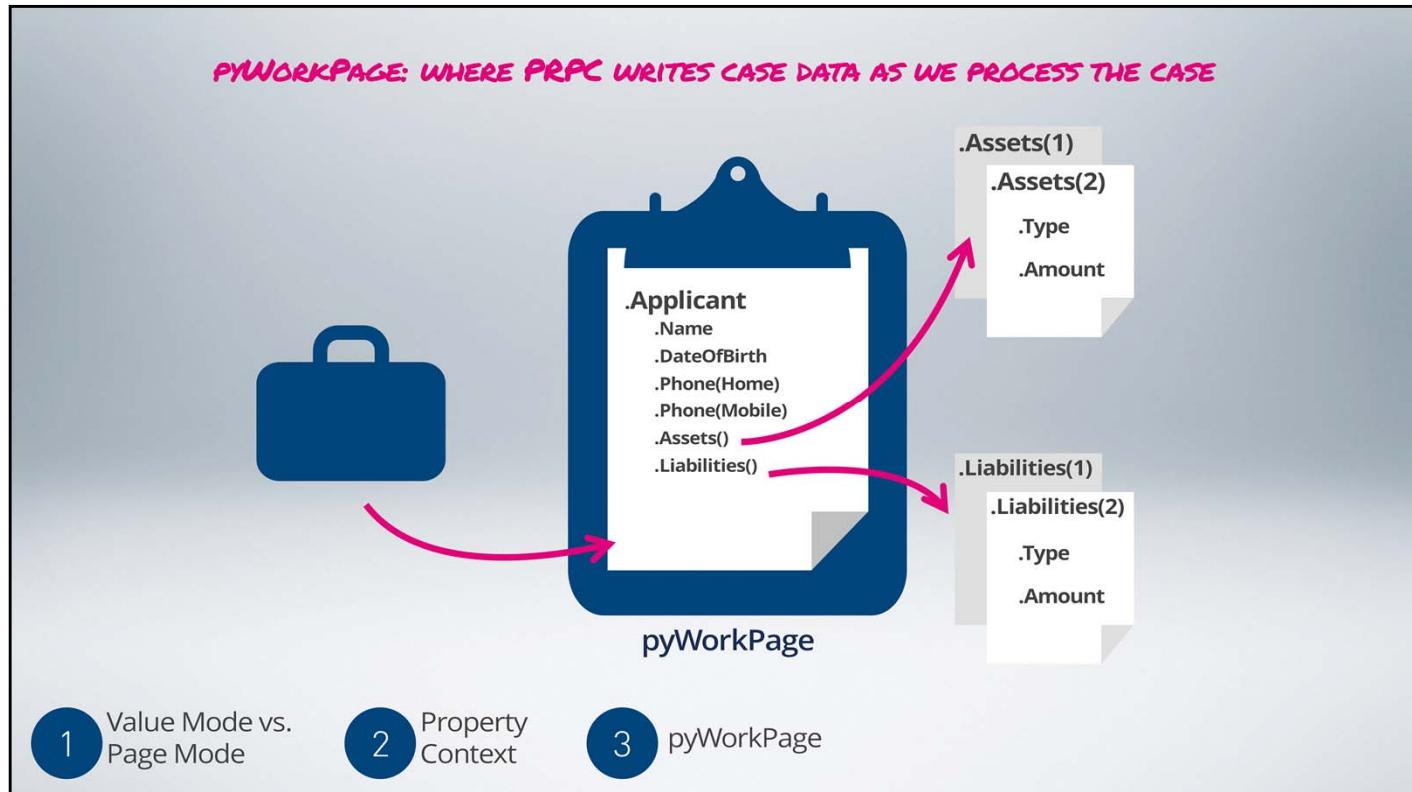
Value-mode properties contain a single value, such as a date, or a name, or a telephone number. We can define a value-mode property to store one of 10 different types of data to represent text, number, date, time, or Boolean data.

Page mode properties represent a data object – a representation of an entity such as a person, an order, or a vehicle. Each data object is defined by its constituent properties, and identified by its key value.

We also learned that each mode can represent a single element, an array or ordered list of elements, or a semantic grouping of elements.



But, since page-mode properties contain value-mode properties – the data elements to which users write and from which they read – we need to remember the context of each data element. When our property reference is not “in context”, we need to include in our reference any pages that contain the property itself. So .City refers to a property named City that has been defined on the current page, while .Vendor.City refers to the same property defined on the Vendor page.

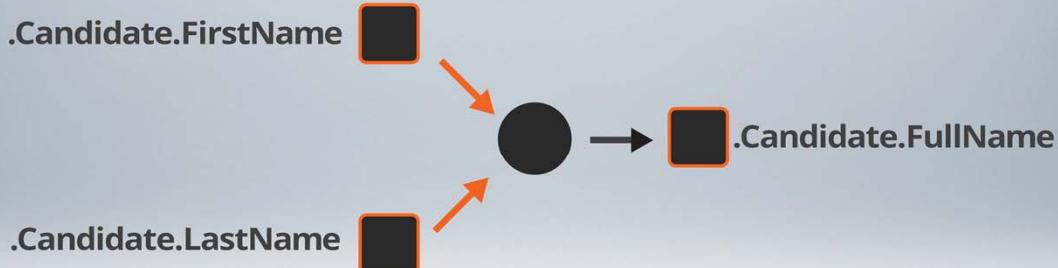


Since most of our case data – the data we use in this course – is defined on `pyWorkPage`, this is our usual context, and most of our property references are relative to this page. So, a data model might lead to a data structure that looks something like the following example for a bank loan.

On `pyWorkPage`, we define a page named `Applicant`, to capture information about the applicant. On this page, we might define data such as:

- Single-value properties such as `Name` and `Date of Birth`.
- A value group consisting of `Phone/Home` and `Phone (Mobile)` to collect the telephone numbers at which we can contact the applicant.
- And two page lists – one for assets and one for liabilities – each with single-value properties to capture the type and current value for each item on these lists.

DATA TRANSFORMS: COPY AND MANIPULATE DATA IN MEMORY



1 Value Mode vs.
Page Mode

2 Property
Context

3 pyWorkPage

4 Data
Transforms

Once we understand how to create a valid property reference, we can read and set property values – not just in our UI, but elsewhere. We can reference properties in decision rules, declarative expressions, and – as we learned two lessons ago – data transforms.

A data transform manipulates data, transforming it in our application. Data transforms can be simple – copying a value from one property to another. They can also be complex, creating a data page populated with data from existing cases, and copying that data into a list of options in our UI.

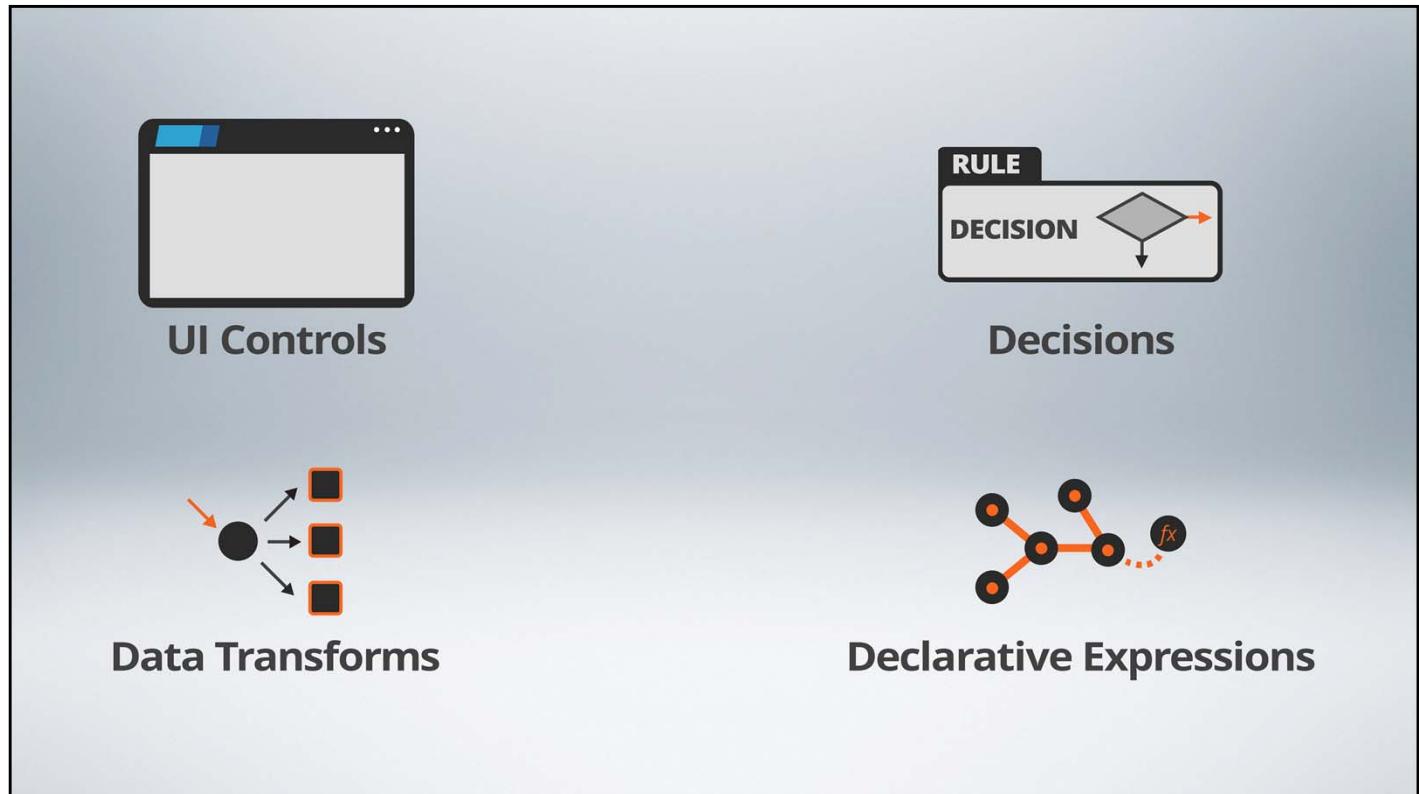
We built a simple data transform to combine the first name and last name of a person into their full name. When we did, we remembered to establish the context for each of our properties, to ensure that we read from and write to the correct data elements.

THE CLIPBOARD: REVIEW DATA IN MEMORY TO DEBUG APPLICATIONS

A blue briefcase icon is on the left, with a pink curved arrow pointing from it to a blue clipboard icon. The clipboard icon has a white document on it with the text '.CostCenter=1005'. Below the clipboard is the text 'pyWorkPage'.

1 Value Mode vs. Page Mode 2 Property Context 3 pyWorkPage 4 Data Transforms 5 The Clipboard

Finally, we learned how to use the Clipboard to review – and even update – data as we develop and debug our solution.



Now that we understand how to create data elements and reference them within our solution, we can use UI controls to collect and display property values. We can also create decisions and declarative expressions that enforce business policies based upon actual case data.

Module 06: Enhancing the User Interface

This lesson group includes the following lessons:

- Introduction to UI
- Designing Layouts
- Creating Repeating Layouts
- Formatting Data Elements
- Validating Data Elements
- Review of Enhancing the User Interface

Introduction to User Interface

Designing the presentation layer is a key component in projects as it provides the mechanism for users to interact with systems.

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

- Understand how user interfaces are used in applications
- Identify the three main types of User Interfaces
- Understand the relationships and how they can be used
- Describe the key characteristics of User Interfaces

User Interface is a core component in presenting the content to users



The User Interface or UI is a key component in case management along with Logic, Data and Process.

The screenshot shows the Pega 7 Designer Studio interface. On the left is a sidebar with a 'Recent' section containing items like 'SAE1 Administrators', 'Program Fund', 'Purchase Application', 'CaseType for this work...', 'Application - Overview', 'SAE1 Architect', 'pyDefault', 'CourseDev_Branch_Ca...', 'Purchase Request', 'Purchase Application', and 'Application - Structure'. Below this is a 'Records' section with 'CourseDev_Branch_CamtasiaRules' and 'PurchaseFW 01.01.01'. At the bottom of the sidebar is a 'Favorite' section with 'Purchase Request'. The main content area has tabs for 'Home', 'Guardrail Warnings' (last 7 days), 'Pega Task', and 'PDN News'. The 'Guardrail Warnings' tab shows two charts: 'Introduced by you' and 'Introduced by team', both with 0 Severe, 0 Moderate, and 0 Caution cases. The 'Pega Task' tab shows 'No items'. The 'PDN News' tab shows 'No items for the filters applied' and a link to 'PDN Forums'. The bottom navigation bar includes icons for Tracer, Clipboard, UI Inspector, Performance, Alerts, Inspection Prefs, and Pega 7.1.0.

Our user interface ranges from the entire user portal- Designer Studio or the Case Manager Portal, for example down to individual content items such as text, buttons, and other display controls.

The screenshot shows the Pega CaseManager interface. At the top, there's a navigation bar with 'Pega® 7' logo, 'CaseManager', 'Create', 'Next Assignment', a search bar, and a user profile for 'SAE1 Architect'. A dropdown menu shows 'Default WorkGroup'. On the left, a sidebar has links for 'Dashboard', 'My WorkList', 'My Cases', 'Calendar', 'Reports', 'RECENT' (with items like 'Program Fund (PF-1)', 'Purchase Request (PR-6)', etc.), and 'FOLLOWING' (with a note 'No items for the filters applied'). The main content area is titled 'Summary for Purchase Application'. It shows a 'Case stages for Program Fund' with four stages: 'Program Fund Entry (0)', 'Approval (0)', 'Disbursement (1)', and 'Closure (0)'. Below this are buttons for 'Enter Program Details', 'Get Manager Approvals', 'Wait for Program End', and 'Close Program Funding', along with a 'Send Email' button. A section titled 'URGENT WORK FOR DEFAULT WORKGROUP' lists five tasks: 'PR-7 Add Line Items', 'VM-2 Add/Edit Vend...', 'PR-6 Verify Addresses', 'PR-2 Enter Purchas...', and 'PR-1 Enter Purchas...'. Each task has columns for Urgency (10), Due date, ID, Description, Category, and Owner (all listed as 'SAE1 Architect'). To the right, there's a 'PEGA PULSE' box and a 'TEAM MEMBERS' section with a table:

MEMBER	ROLE	ACTIVITY COUNT
SAE1 Architect	Curriculum Developer	5
Integration Sample U...		0
StockTrader		0
SAE2 Architect	Curriculum Developer	0
Bob Ryder	Account Manager	0
PRPC_SOAPOper		0
Steve	System Architect	0
Administrator	Please do not change the access groups	0
Batch Processing	Batch Processor	0
External User	External User Access	0

User Interfaces in case management are mainly used to provide screens for users to create, update or resolve their cases

Case Processing

- Forms used to collect information
- Pages used to display information
- Other presentation layers

PLACE ORDER

Review the Purchase order and then place it to Amazon, check the Information tab for address and other contact information for the vendor.

ITEM ID	ITEM CATEGORY	ITEM DESCRIPTION	QUANTITY	UNIT OF MEASURE	UNIT PRICE	SUBTOTAL
1	Hardware	Laptop	2	Each	1200	\$2,400.00
2	Hardware	Mice	9	Each	30	\$270.00
3	Hardware	Keyboard	10	Each	60	\$600.00

Shipping Method:

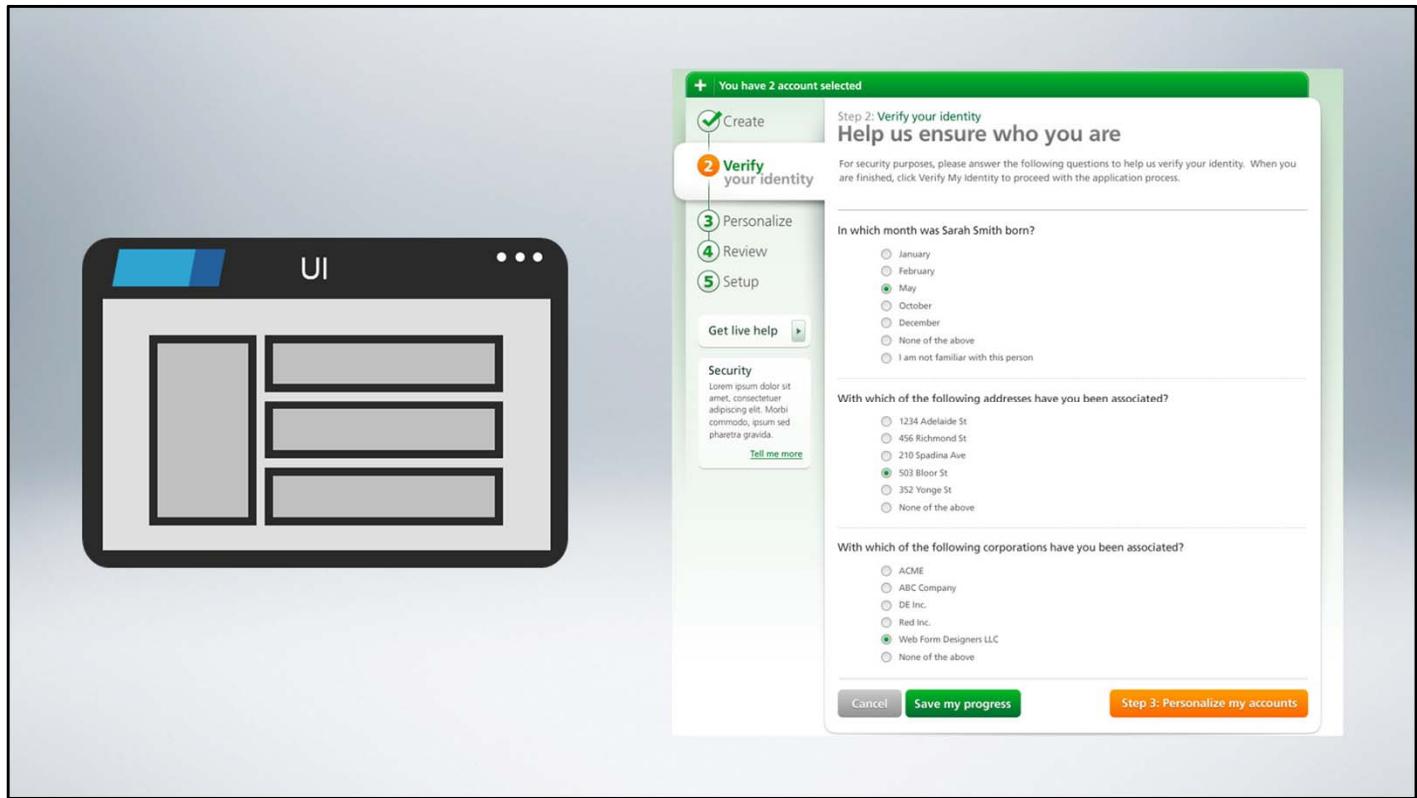
Tax:

Order Total: USD3,270.00

In our application, these screens can either collect data from an applicant, or present the collected data to a loan officer. They can also be used to display the account details of the customer after querying from an external system or present a static content.

The screenshot displays the Pega CaseManager interface within a 'User Workspace'. The main title 'User Workspace' is at the top. On the left, a sidebar provides navigation links for Dashboard, My Worklist, My Cases, Calendar, Reports, and RECENT activity logs. The central area is titled 'Purchase Request (PR-2)' and contains a 'COMPLETE QUOTE' table with data for two items from Amazon. Below the table are tabs for Overview, Information, and Audit, with 'Information' selected. The 'CASE CONTENTS' section lists a single item named 'Complete Quote(Quote)'. The 'CASE DETAILS' section shows case-specific information like Case ID (PR-2), Urgency (10), and Status (Pending-Quote). A sidebar on the left also lists recent Purchase Requests.

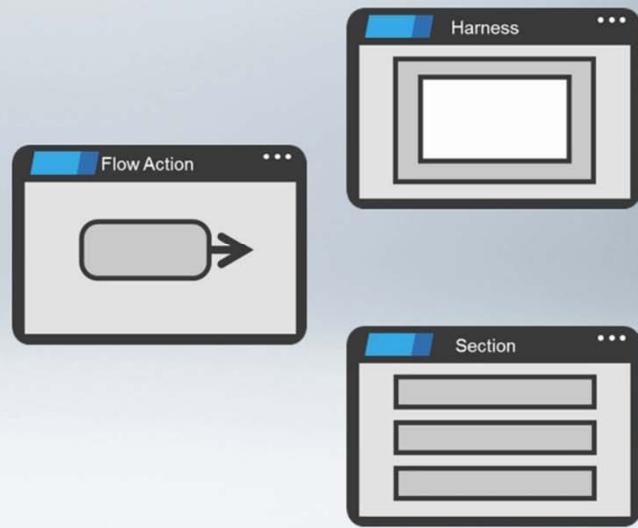
Though it's important to understand how the entire UI fits together, for this lesson we will talk about this working area, and how we create and display content within it.



In this lesson we'll also take a look at how these User Interfaces are created and the terminology used. Now that we have seen the different purposes, let's see which rules are used to build the User Interfaces.

Three UI Interface rules

- Harness
- Flow Actions
- Sections



There are three main rule types of user interfaces that are used extensively while building applications. These are Harnesses, Sections and Flow Actions.

Three UI Interface Rules

- Harness
- Sections
- Flow Actions



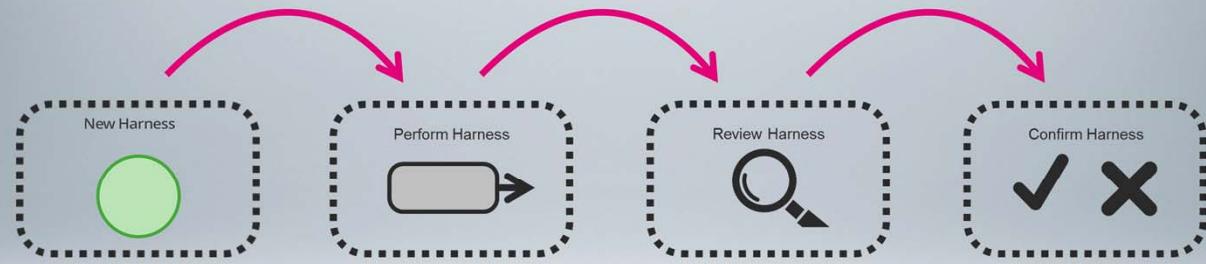
Let's start with harness. Harnesses can be thought of more as a web page that the user views while creating, updating or resolving cases.

Standard Harnesses



There are four harness rules that are shipped in the product and are tightly integrated with case management and are used during Case Processing. Let's look at each of these rules.

Standard Harnesses



Lastly after the case is resolved, it shows a confirmation screen for the users to view the details. The **Confirm Harness** is referenced by the process for this functionality.

These harnesses are tightly integrated with the business process. Let's open up a flow and see how these harnesses are referenced.

Demo



Demo: How Harnesses Are Referenced in the Business Process

The New harness is only referenced on the starter flows and you can change the name to something other than New if you wish.

This harness is optional and we could elect to skip the rendering of this harness. There is a flag to select to skip the creation of New harness.

The Confirm harness is also referenced in this flow, and again you can change the name. The Confirm harness is also referenced on flow actions. We will learn about flow actions in a little bit.

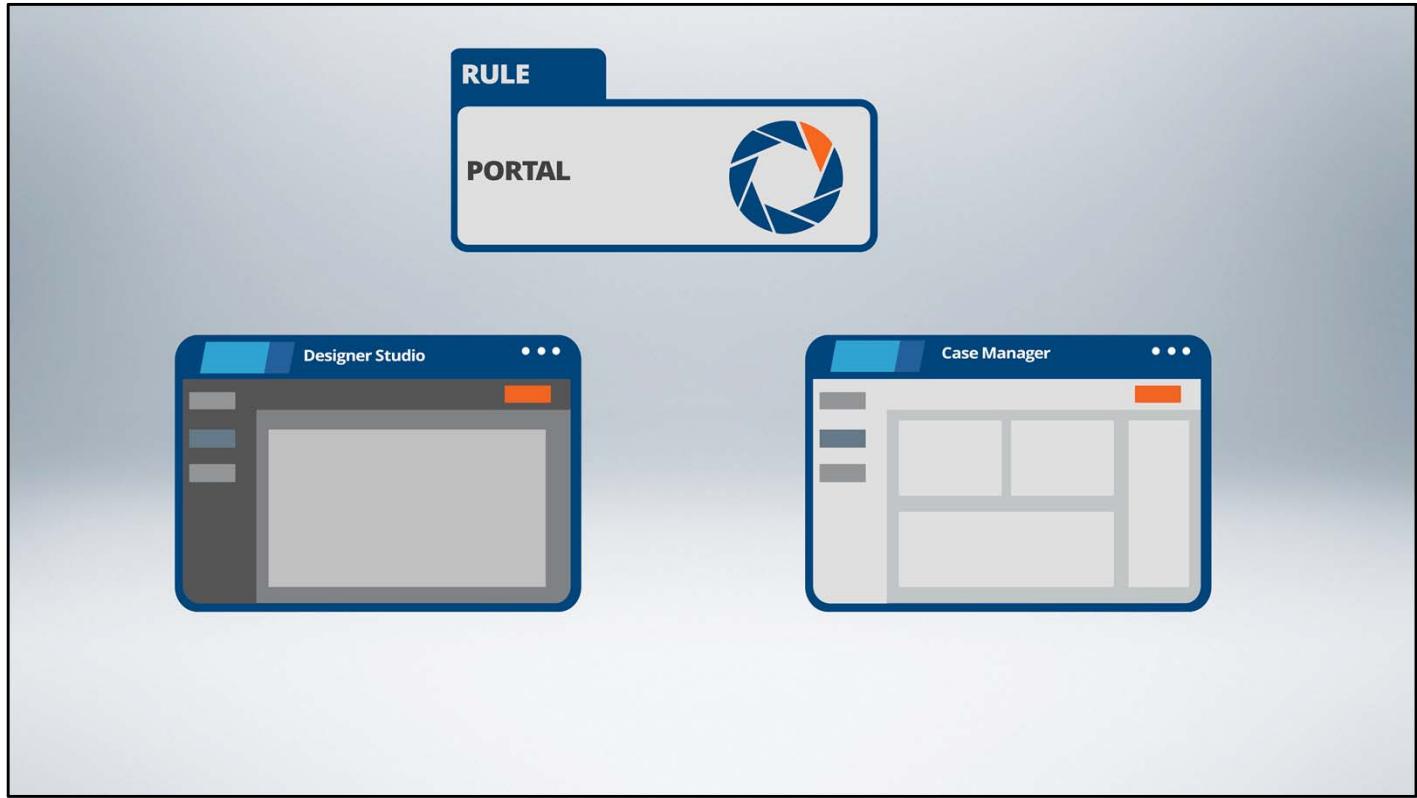
Let's go to the case designer and open up a flow.

The user actions are represented by an assignment shape. The assignment shape indicates that the case is assigned to a specific user or a group of users and has to wait for them to act on it. The Perform harness is referenced in this shape. Again we could opt to use the same name or use a customized harness on any step.

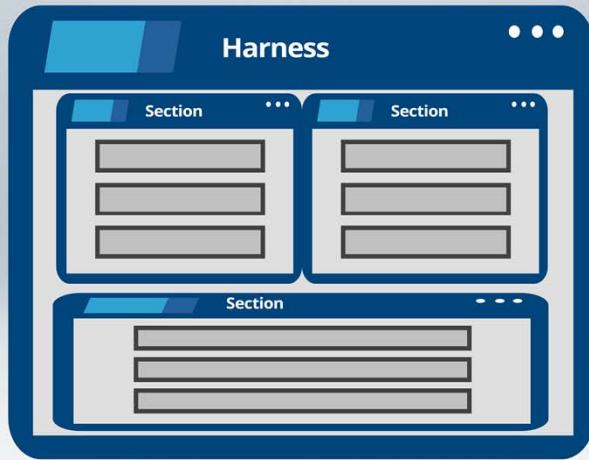
Standard Harnesses



The Review harness is being referenced internally in the process engine, so the name cannot be changed, however the content can be changed making it specific to a case or at the application level. All other harnesses can also be customized while still using the same name. We recommend that if you are going to customize the functionality of a harness that you modify the name as well. Harnesses used in business processes impact the case creation, update and resolution. Therefore, these modifications are usually done by senior system architects.



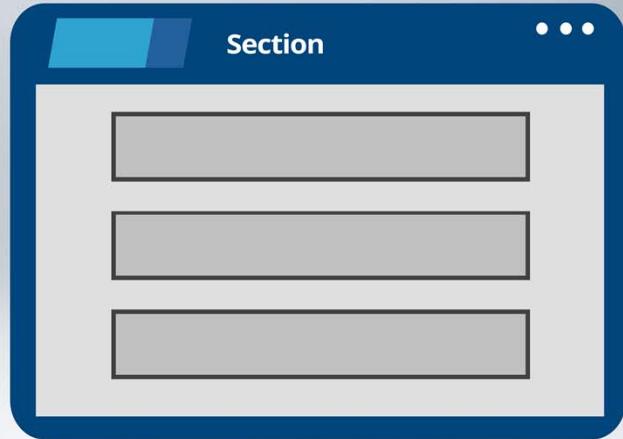
Harness rules are also referenced in a portal record, so Designer Studio, Case Manager and other end user portals use a custom harness. We will learn about this in the advanced courses.



A harness is nothing but a top level frame that provides the framework of a User Interface screen. Harnesses do not provide the capability to add content in it directly. Harnesses use sections, which is another type of User Interface rule.

Three UI Interface rules

- Harness
- Sections
- Flow Actions



Sections are the building blocks of an application UI. There are several sections shipped in the product, however your project team will primarily work in creating new sections since that is what determines what users see.

Demo



Demo: Sections and Flow Actions

Let's look at an example. This is the perform harness used in the Purchase request, sections are added in a construct named container.

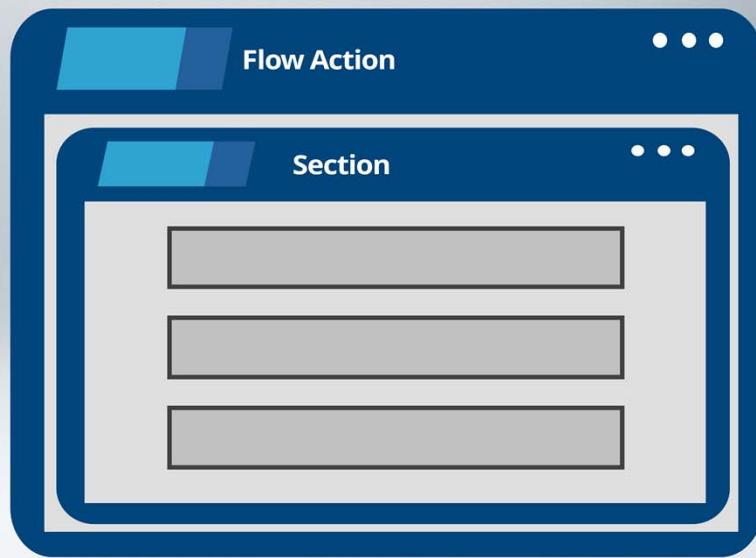
Sections use a construct named layout which helps in aligning the presentation of the data elements in rows, columns and images. We will learn about the layouts and data element formatting in the next few lessons.

The Perform harness uses standard sections and application developers can modify the content to suit their application needs.

The Other Actions menu in the case header section lists the different actions that users can perform in that step, the default action always displays when the harness is rendered. The default action and other actions use a third UI rule named Flow Action.

Three UI Interface rules

- Harness
- Sections
- Flow Actions



Flow Actions are nothing but a place holder for a specific action that users can take while processing a case.

Demo



Demo: Sections and Flow Actions (continued)

Flow Actions must be used on all assignment shapes because it determines which path or action users can take.

When a manager is reviewing the purchase request, he or she can either approve or reject the purchase request. The Other Actions menu offers the capability for the manager to select either of them.

When the managers select Approve, they are actually accessing the corresponding flow action which renders this screen.

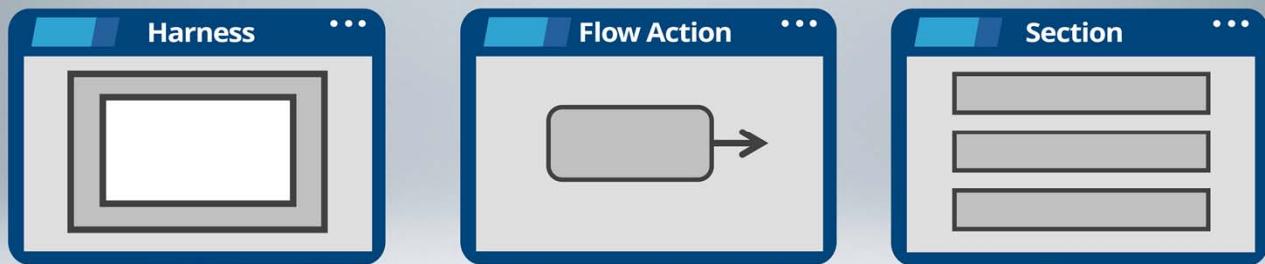
If they want to switch to Reject, then they can use the Other Actions menu again, notice Approve is not a choice here since its already opened.

Each assignment shape must have at least one flow action connected to it. There is no restriction on the number of flow actions that can be added to an assignment, however anything more than a few means that the process should be properly designed so users are not presented with too many choices.

Submitting the flow action progresses the case to the next step in the process.

Let's open the actual flow action, so we can see that this references a section. Sections in general are the only rule that is used to save the content that displays to users.

Three UI Interface rules



To sum it up, we have harnesses which are the top level page, flow actions which are the specific paths or actions a user can perform and sections which are used in creating the content that the user views.

Demo



Demo: Understanding the Relationships of the Three UI Rule Types

Let's view the relationships of these User Interface records at runtime.

When creating a new purchase request, it shows the Enter purchase request screen for user to enter their inputs. To see which rules are used here, let's use the UI Inspector tool.

When hovering over the area outside the border it shows the container element type which is part of a harness, this does not contain the content that the user sees.

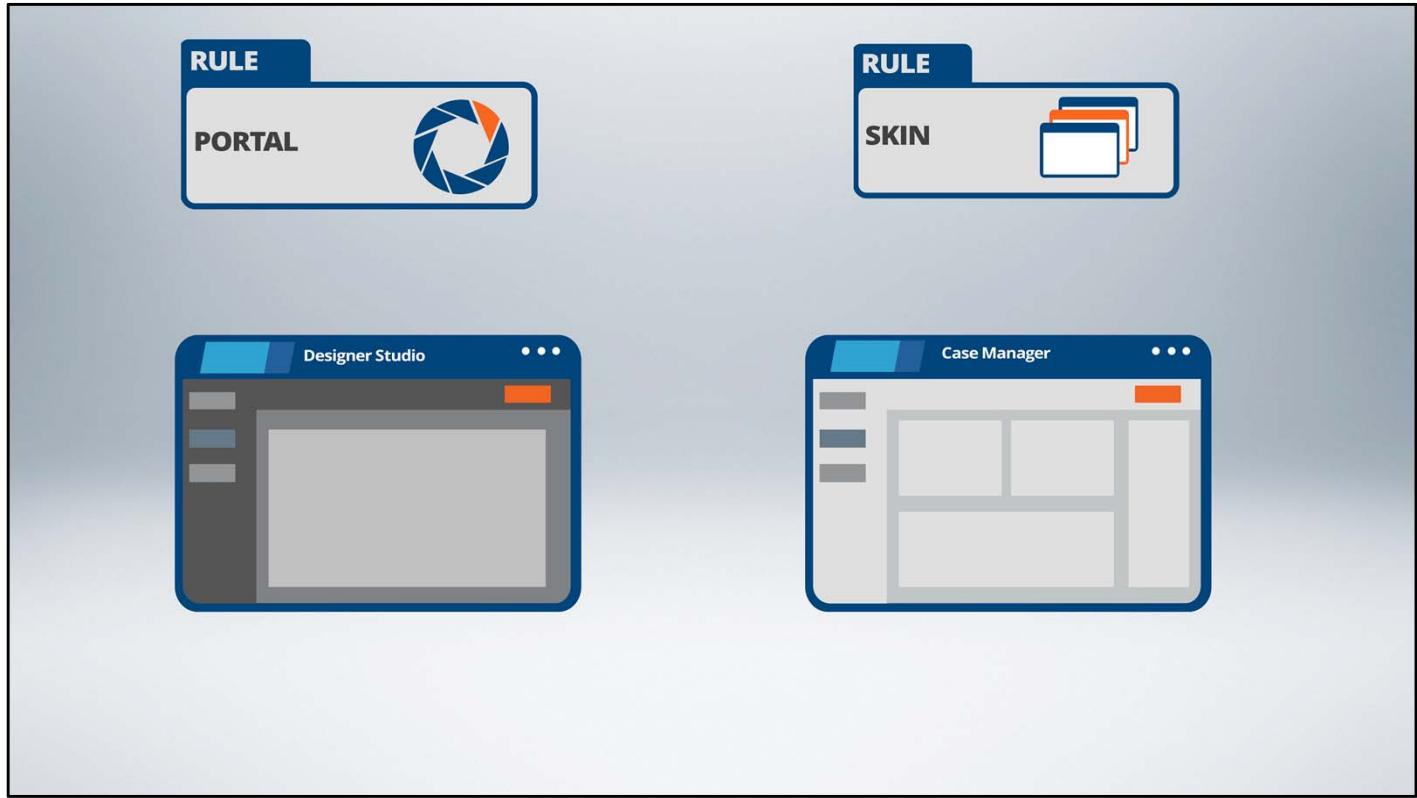
Narrowing it further we can see the entire UI Structure — the EnterPurchaseRequest section which is referenced in the EnterPurchaseRequest flow action which is used in the harness.

If we hover over a cell it shows the property name that is referenced in the section.

Now let's now open the end user portal built for this application.

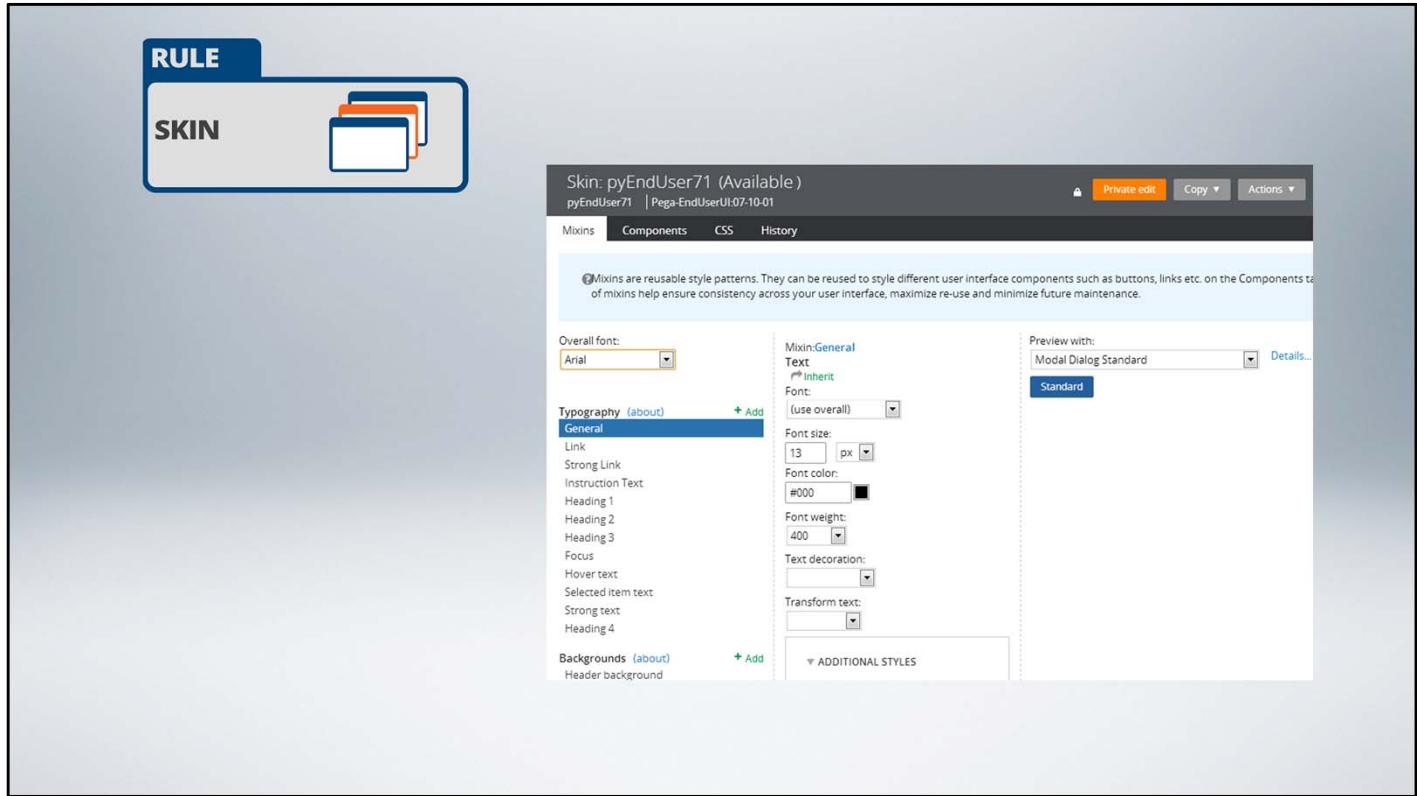
The UI inspector is available on the portal for developers to verify the rules.

When we hover over the section it displays the UI structure. When used as end user portals, the harness directly references sections and there are no flow actions. Portals use harnesses that are configured a little differently from how the harnesses are configured in case management. The only thing that they have in common is fact that these harnesses also use sections for their content.

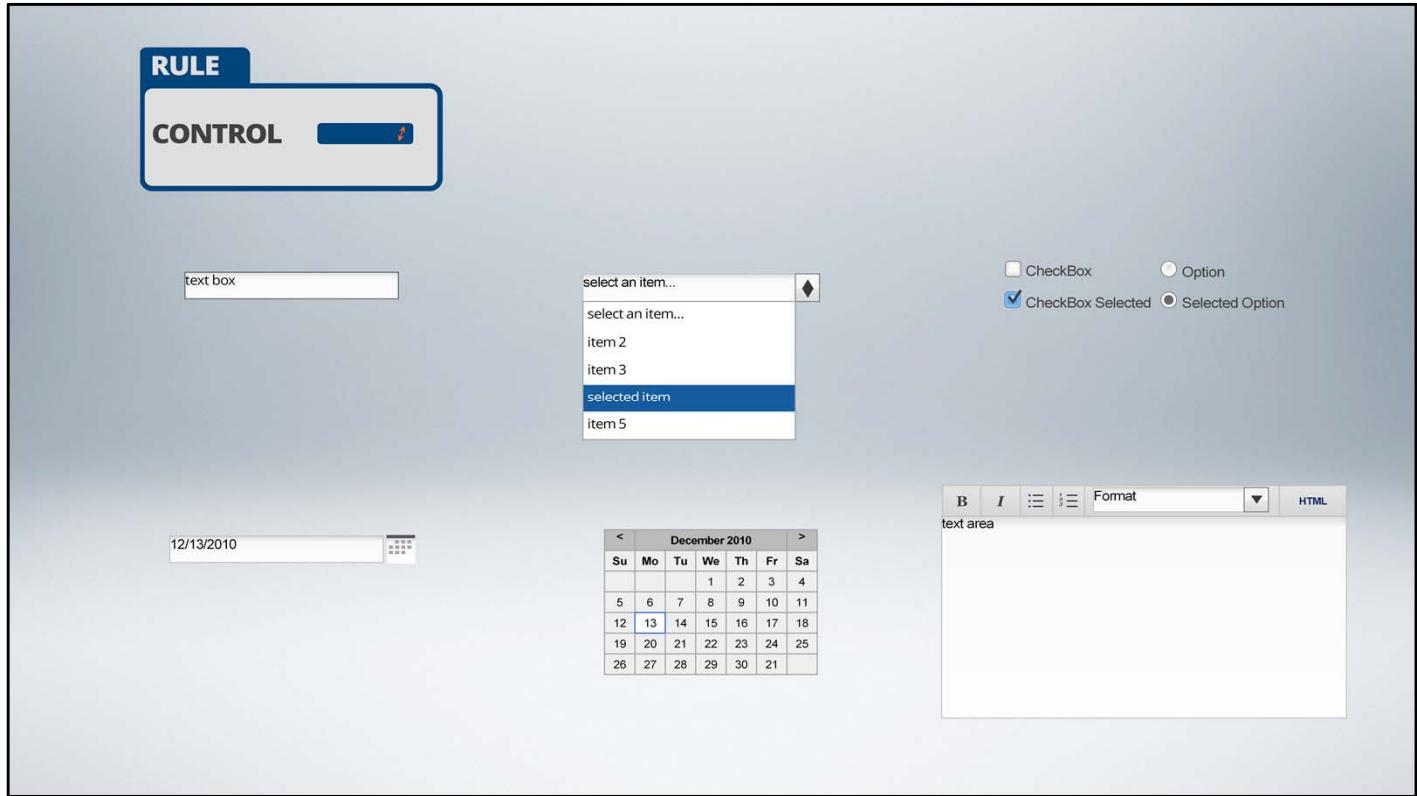


There are other types of User Interface rules other than these three rules. Let's quickly see what each of them can be used for.

Portals are user interface rules and are primarily used to display the workspaces for users. Portals use harnesses for content. They also reference another rule type named skins which controls the styling.



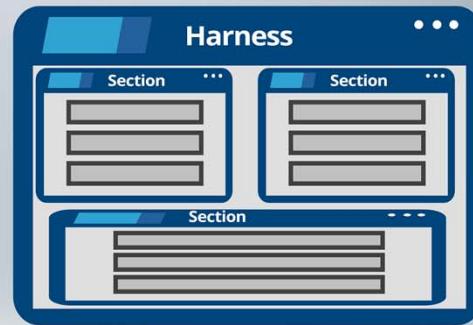
And that leads us to Skins, which are used for defining the styling elements such as color schemes, font schemes, images, etc. that are used in the application. Pega provides a wizard to generate skins and it does not require end users to know CSS and other styling elements to create them.



Controls are used in configuring the presentation of data elements, for example, whether it should be a text box, dropdown, radio button or checkbox and so on. Pega 7 comes with a list of standard controls for all the most commonly used presentation choices and can be used as such. There are other UI components that are used in the application, however these are the most prominent ones that we will be using most often.

UI Characteristics

- Model Driven
- Auto-Generated
 - Drag and Drop interface
 - Quicker development cycle, easier maintenance and easy to upgrade



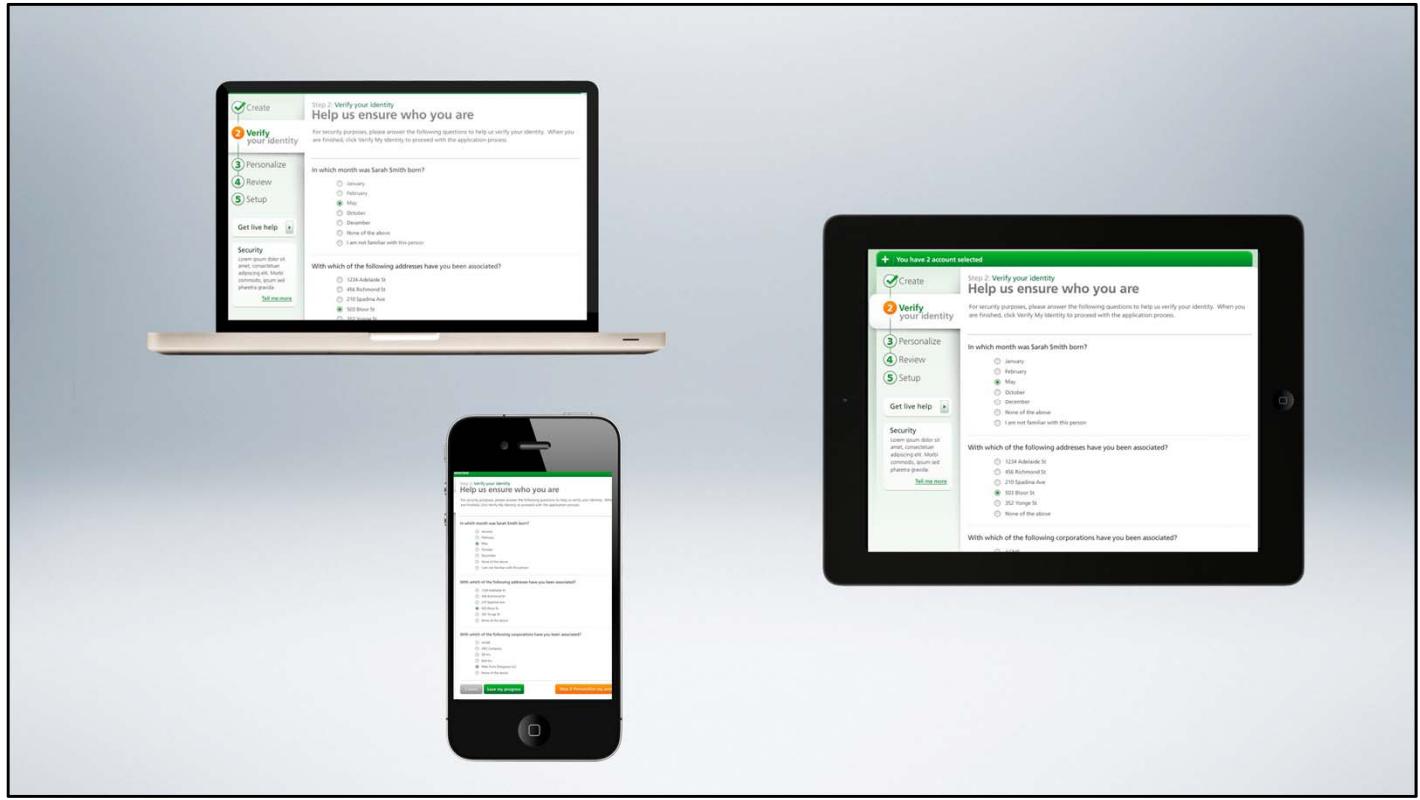
Now that we know the different UI rules and the structure, let's look at some of the key characteristics.

The UI records are model driven. What does this mean? The User Interface rules that we see here are tightly coupled with the process. They are accessed from the process and not outside the process. In short the process determines which User Interface is rendered and this helps in UI.

Pega7 UI's are completely auto generated, the sections and harnesses present a visual component that allows us to easily drag and drop fields or controls. The html code is generated by the system so users do not need to be proficient in scripting, styling or HTML tags for creating User Interfaces.

Auto generation is a very powerful concept, and it lets us quickly create user Interfaces without coding. Users are presented with a very user friendly interface that allows us to drag and drop fields or controls to present a UI.

The Auto generated UI offers multiple benefits for the development team in terms of quicker development cycles, lesser training, lesser maintenance of resources, and an easier upgrade to higher versions and so on.



The other key benefit in the auto generation of UI is that it allows us to build the user Interface once and deploy it in various presentation options such as desktop, tablet or in smart phone.



Auto generated harnesses and sections are supported in various browsers such as chrome, safari, Firefox and IE.



UI rules generate HTML5 and CSS3 and hence despite being auto generated they are very capable of generating good looking user Interfaces.

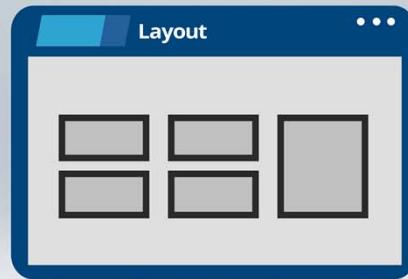
Designing Layouts

When creating sections, we need to design layouts that can be used to group properties. The properties that are added into sections are placed inside a layout. Layout plays a huge role in how the properties are aligned in the user interface.

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

- Understand what a dynamic layout is
- Use the different standard formats
- Add layouts into sections

What is a Layout?

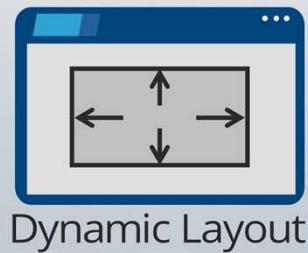
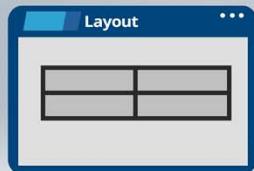


A Construct used to align properties in sections

Welcome to the lesson on Designing Layouts. What is a “Layout”? A layout is used in a section to align data elements for display.

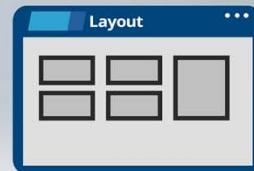
Layouts Supported in Pega 7

Smart Layout



Dynamic Layout

Free-Form Layout

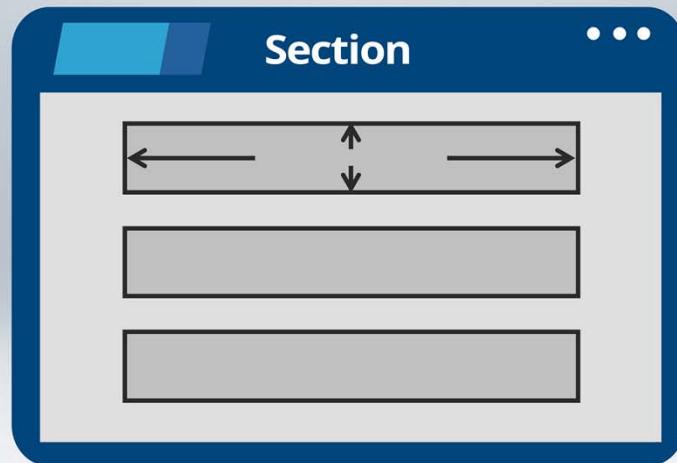


Column Layout



Repeating Layout

There are several layout types supported in Pega 7. In this lesson we'll focus on dynamic layouts, which is one of the new layout types supported in Pega 7 and it will be one of the more commonly used layouts in applications that are built in Pega 7. Smart and Free form layouts are supported in the product for backward compatibility but have been superseded by Dynamic layouts. Repeating layouts will be covered in detail in the next lesson and Column layouts is another new layout type which is covered in detail in the advanced courses.



Layouts can be included only in sections. Sections must include at least one layout to contain the data elements we want to display.

The figure consists of three separate wireframe diagrams, each representing a different layout for address data elements. Each diagram contains the following fields:

- Attention
- Address Line 1
- Address Line 2
- Address Line 3
- Address Line 4
- City
- ZIP / Postal Code
- Phone
- Fax
- State/Province
- Country

In the first layout, all fields are stacked vertically in a single column.

In the second layout, the address lines (Address Line 1 through Address Line 4) are grouped together in one column, while the city, ZIP/postal code, phone, fax, state/province, and country are grouped together in another column.

In the third layout, the address lines (Address Line 1 through Address Line 4) are distributed across three columns, with the city, ZIP/postal code, phone, fax, state/province, and country grouped together in the third column.

Layouts help in displaying the data elements in single, double or triple columns; depending on the screen width we could go up to six columns.

The image shows two versions of a form side-by-side, demonstrating different ways to label fields.

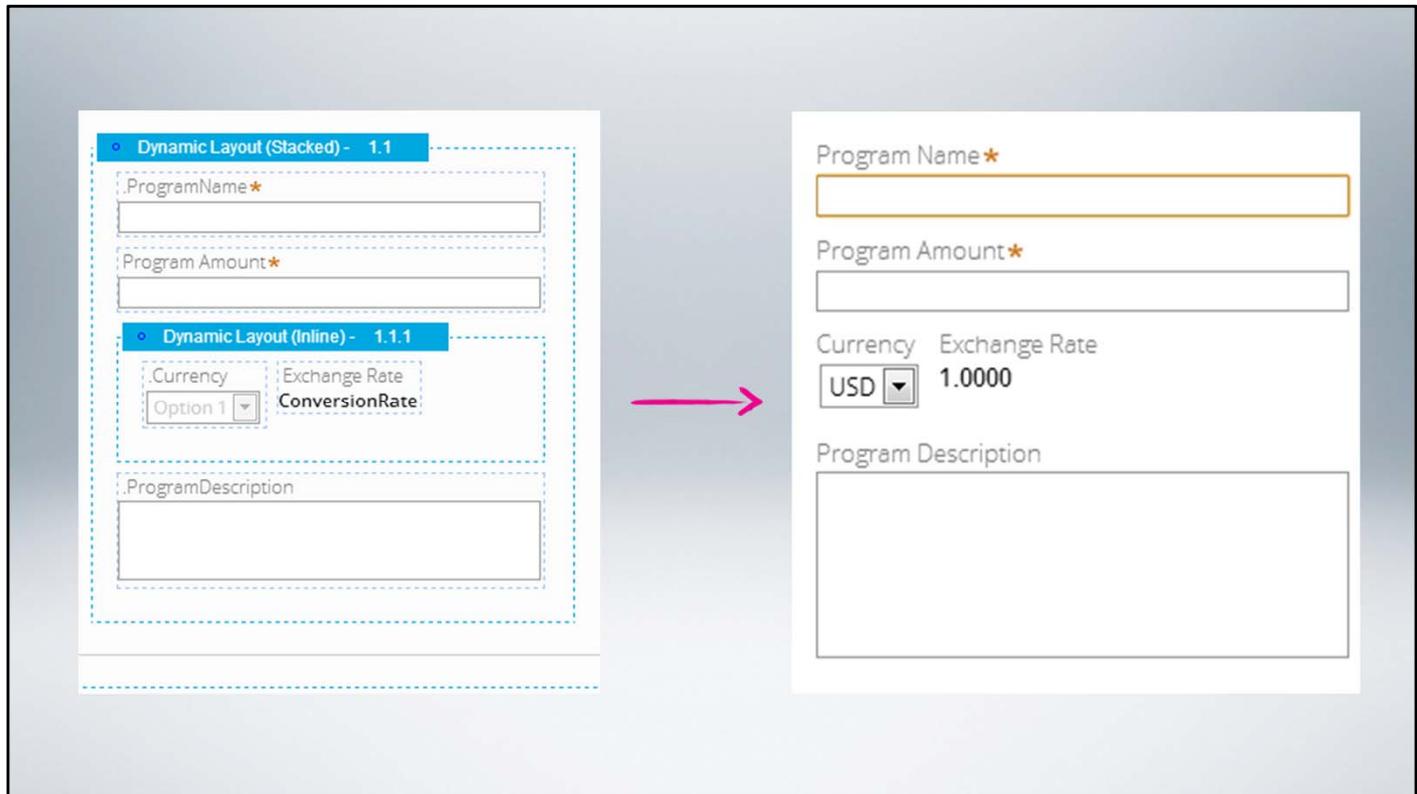
Left Form (Label Inside Field):

- Attention:
- Address Line 1:
- Address Line 2:
- Address Line 3:
- Address Line 4:
- City:
- State/Province:
- ZIP / Postal Code:
- Country: United States ▾
- Phone:
- Fax:

Right Form (Label Outside Field):

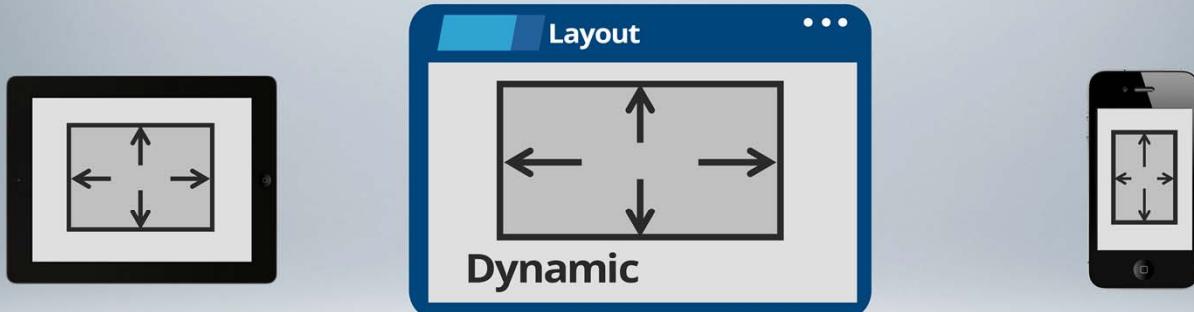
- Attention:
- Address Line 1:
- Address Line 2:
- Address Line 3:
- Address Line 4:
- City:
- State/Province:
- ZIP / Postal Code:
- Country: United States ▾
- Phone:
- Fax:

They can help in positioning the labels either at the top or at the left of the field they identify.



Dynamic layouts help us to create the UI screen quickly and also support complex configurations. They support nesting, which allows us to embed a dynamic layout inside another, like in this example we need to display both the currency and exchange rate in one row while the other rows display only one column

Dynamic Layouts



Fit multiple screen layouts Responsiveness

Dynamic layouts are responsive – when the display area changes, the look and feel of the application adapts to the smaller or larger display area. This attribute - known as “Responsiveness” - lets us create a UI that responds to changes in the display area in a predictable manner. This feature is significant in making the application render correctly for different forms such as in tablets and smart phones.

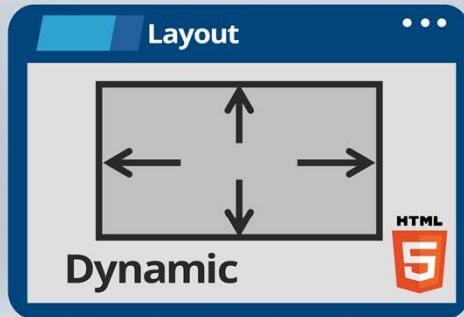
The image displays three versions of a form layout for address input, illustrating how the layout adapts to different display areas:

- Top Version (Large Display):** This layout is designed for a wide display area. It features three columns of input fields. The first column contains "Attention" (with a dropdown menu), "Address Line 3", "State/Province", and "Phone". The second column contains "Address Line 1", "Address Line 4", "ZIP / Postal Code", and "Fax". The third column contains "Address Line 2", "City", "Country" (with a dropdown menu set to "United States"), and "Fax".
- Middle Version (Medium Display):** This layout is designed for a medium display area. It shows the same four groups of fields as the top version, but each group is split into two rows. For example, "Address Line 1" and "Address Line 2" are in the first row of the first group, while "Address Line 3" and "City" are in the second row.
- Bottom Version (Small Display):** This layout is designed for a small display area, such as a mobile device. It shows the same four groups of fields as the top version, but each group is further subdivided into two rows. For example, "Attention" and "Address Line 1" are in the first row of the first group, while "Address Line 3" and "City" are in the second row.

For example, a triple layout can become either a double layout or a single layout depending on the display area.



Dynamic Layouts



Fit multiple screen layouts Responsiveness



Supported in all browsers



Dynamic layouts use HTML5 tags which are supported in all browsers. When a dynamic layout is added in a section, the system auto generated code uses the DIV tag that makes it more flexible in terms of aligning fields. We'll look at adding dynamic layouts in this lesson and we'll see how easy it is to work with them.

- **Various Formats of Dynamic Layouts**
- **How to Add Dynamic Layouts into Sections**

Demo



Pega 7 provides us with a set of dynamic layouts that we can use. Here we are seeing the default skin rule that is shipped in the product because it's easier to see the entire list as part of the skin rule. The layout can be previewed on the right side without saving the rule. The application team can also create additional formats to support functionality that is specific to their application.

It is more common to build additional formats specifically around customizing the width or the height or label position, etc. In general, the data elements are arranged in one of two ways – the first one is using the Inline grid which positions the data elements in a grid format using columns and rows. The Inline grid formats require developers to specify the number of columns during definition. Let's see some examples of formats that are shipped as part of the product.

The Stacked layout uses one column per row, whereas the Inline grid double uses two columns per row and the Inline grid triple uses three columns per row. The second way to arrange fields is Inline, let's check a standard format that is shipped in the product. In this arrangement, the fields are placed one after another similar to the way we write, from left to right. The labels can be placed either above the field, or to the left of it. The previous examples we saw had the labels on top. There are other standard formats shipped in the product which have the labels aligned to the left. There is also a layout without any labels like the Simple list which displays the fields without labels.

The dynamic layout supports both vertical and horizontal alignment. For example, the Inline middle format vertically aligns the fields to middle. There is also the default format, unlike other standard formats the product allows us to make changes to this one. The responsiveness can be configured by setting the responsive breakpoint where we specify which layout to use when the layout size is different.

Let's open the UI gallery to see some specific examples using dynamic layouts. Note most of the layouts here

use dynamic layouts, which we can see in the product. Let's look at one specific example. In this example, we see that this layout is used to collect information about a person. It uses the information such as name, address and other details. It displays the generic information about dynamic layouts and various standard layouts as well. To see the underlying rule that is developed to render this screen, let's click the link to see the design time configuration.

It uses m

Now that we have seen different dynamic layouts, let's see how we can add them into sections.

To create sections using stage designer, we need to enter a flow action name, it can be an existing or a new flow action. In this case let's add a new flow action and then add a few fields that will display on the screen. Again we can use existing properties or enter new ones here. When we enter a new name here, the system automatically creates new properties. Let's add another new field and also change the data type to Date. Once we save the changes, let's open the section.

The system creates a new section with a dynamic layout using the default format and places the fields in it. We can also use the preview option in the actions menu to see how this screen will be rendered at runtime. We could change it to a different format by opening the properties panel for the layout. This panel allows us to choose any of the available formats. When we select a format it is immediately applied and we could use the preview option to see how it displays.

To include a new layout, we use the layout palette and select layout from it. After selecting the layout we should drop it in the section where we want to include the layout. It shows an orange bar to indicate where it can be dropped. Once dropped, it shows the dialog for us to select the layout type, let's click OK so it creates a dynamic layout. Again it creates a dynamic layout with the default format and then we have the choice to switch to a different format. If we want to nest or embed one dynamic layout into another, we can drag the layout from the layout palette and drop it inside the layout at the place where want to use it. This provides us with a lot of flexibility in terms of how to present a User Interface.

Some layouts have the potential to be reused in multiple screens, for example displaying the list of selections. In such cases the layouts can be saved into sections for reusability. The Layout palette also allows us to add another section so multiple sections can be embedded inside each other.

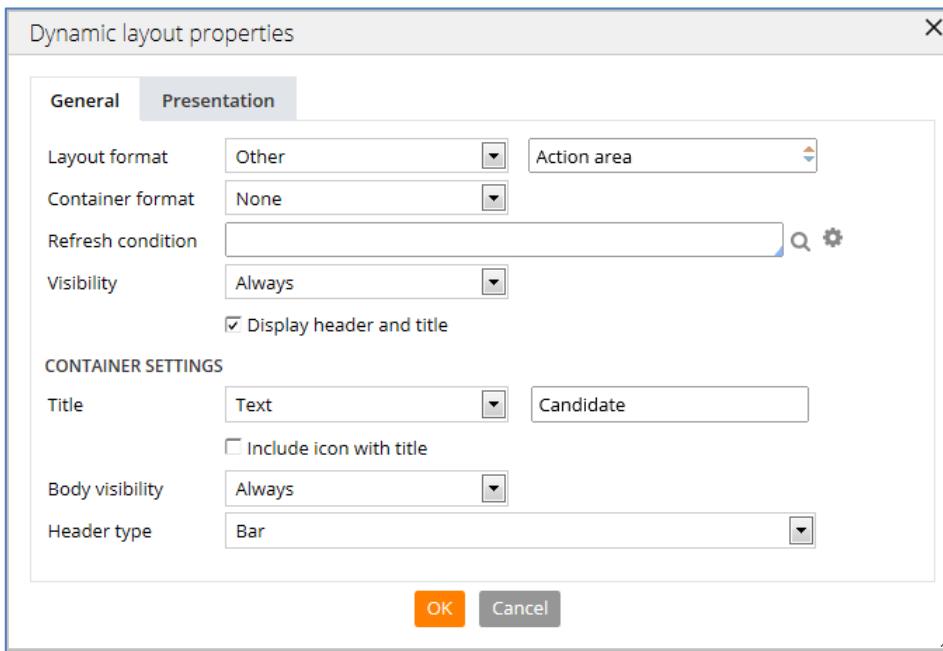
That brings us to the conclusion of this lesson on layouts. While building applications, it's always easier to use the stage designer and create sections and fields inside it and then customize the section after getting to know more. In the lesson on controls, we will see how to add the data elements in the sections.

Multiple dynamic layouts – an outer layout to display the fields in one column and then multiple layouts are embedded inside that to render the screen that we saw earlier.

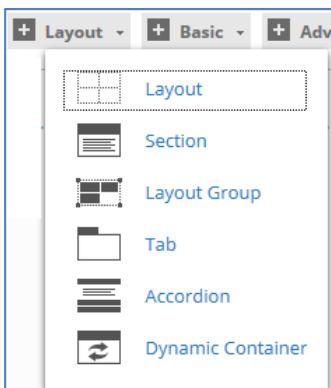
Pega 7.1.6 Update Notes

Updates for Pega 7.1.6 include:

- Dynamic Layout form improvements:
 - Refresh conditions can be specified on a dynamic layout. You no longer need to create sections to create refresh boundaries.
 - Layout format list has been simplified. Second dropdown for "other" formats



- Process diagram layout pallet has been simplified



- Flow actions are given default names (based on step name)
- Create replaces New in the Application Explorer Menu
- Property Record rule form has been updated
 - Label replaces Short Description
 - Application layer context can be specified

The following updates have been visited before:

- Process outline has been updated
- Edit Fields Form Has Been Replaced by Form Configuration
- List of Field Types Has Been Updated

Exercise: Adding Layouts

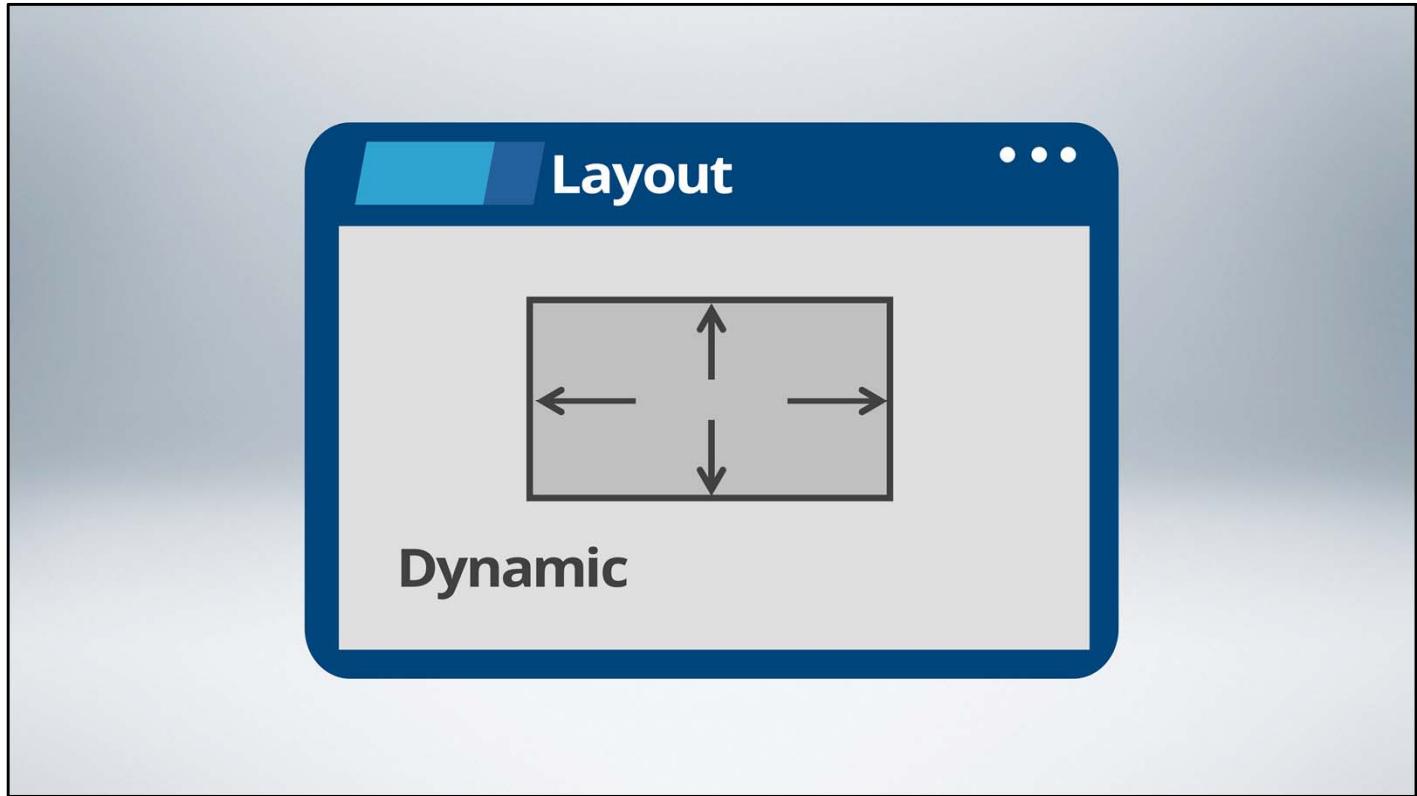


Creating Repeating Layouts

When displaying a list, we require the layout to have the ability to display any number of rows without requiring any additional effort.

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

- Understand which data elements are used in the configuration of repeating layouts
- Add Repeating layouts in sections



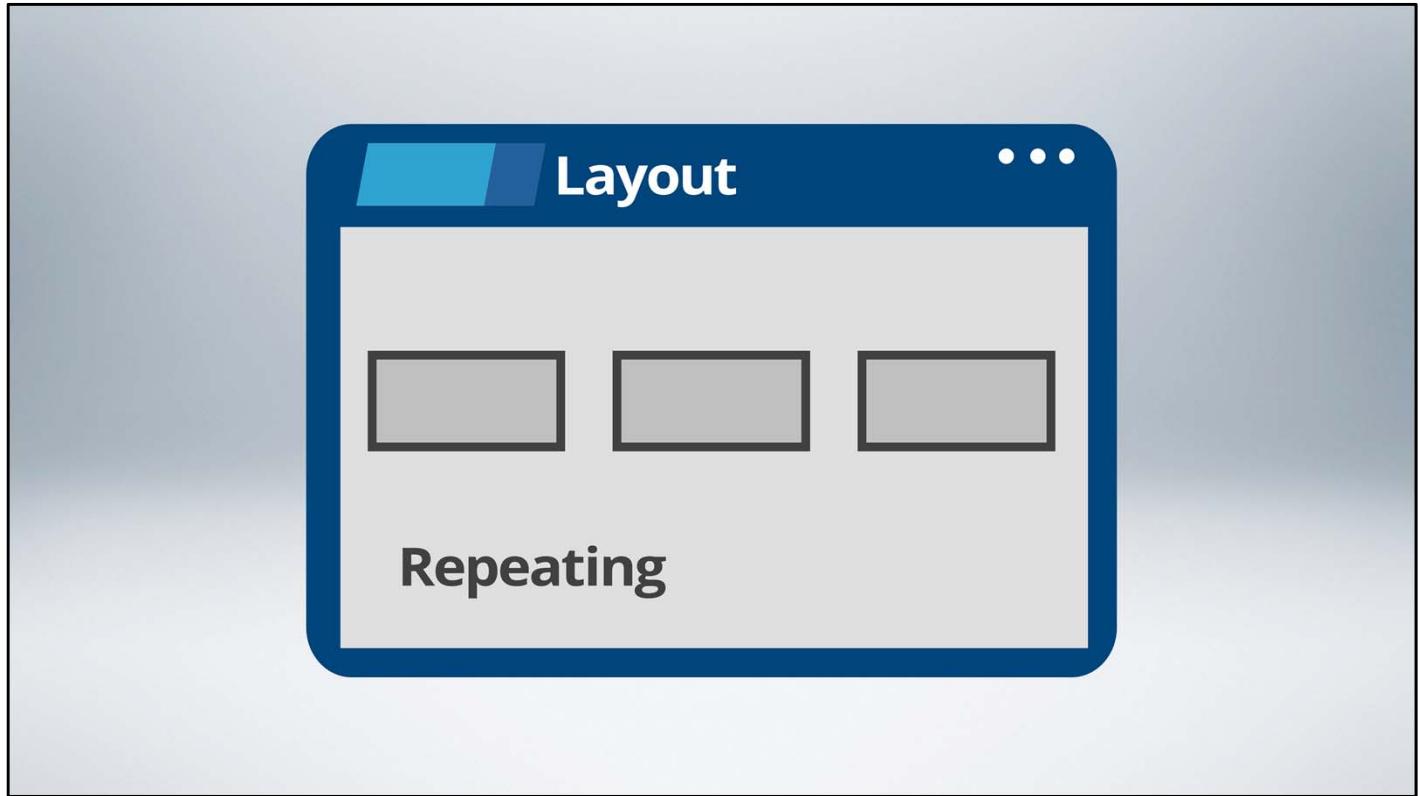
In the previous lesson we learned about layouts and specifically about dynamic layouts. Dynamic layout formats are very useful in displaying single value properties and embedded page properties.

The screenshot shows a software interface titled "Products". A message at the top says "This is a list of products." Below the header, there are several buttons: "Quick Find", "New Products", "Actions", "Report", and "View: Products". The main area displays a table with the following data:

Product Name	Category Name	Quantity Per Unit	Unit Price	Units In Stock	Units On Order	Reorder Level	Discontinued
Chef Anton's Cajun Seasoning	Condiments	48 - 6 oz jars	\$22.00	53	0	0	No
Chef Anton's Gumbo Mix	Condiments	36 boxes	\$21.35	0	0	0	Yes
Louisiana Fiery Hot Pepper Sauce	Condiments	32 - 8 oz bottles	\$21.05	76	0	0	No
Louisiana Hot Spiced Okra	Condiments	24 - 8 oz jars	\$17.00	4	100	20	No

At the bottom right of the table, it says "Showing 1-4 of 4 items" with a refresh icon.

However to display a list of data elements that belongs to a page list or a page group, we need to use repeating layouts.



However to display a list of data elements that belongs to a page list or a page group, we need to use repeating layouts.

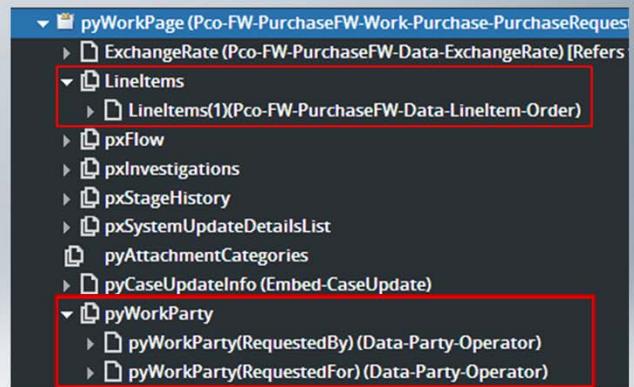
▼ Page list : Item(1), Item(2), Item(3),....

Ordered list



▼ Page group: Address(<Home>),
Address(<Work>),...

Unordered list



We learned about Page lists and page groups in the Data modeling lesson. Both are a list of pages that contain data elements – one is an ordered list while the other is unordered. Let's talk specifically about page lists since they are used more frequently.

Products								
	ProductName	Supplier	Category	UnitPrice	UnitsInStock	UnitsOnOrder	Discontinued	
Edit	Delete	Côte de Blaye	Aux joyeux ecclésiastiques	Beverages	263.5000	17	0	<input type="checkbox"/>
Edit	Delete	Chartreuse verte	Aux joyeux ecclésiastiques	Beverages	18.0000	69	0	<input type="checkbox"/>
Edit	Delete	Sasquatch Ale	Bigfoot Breweries	Beverages	15.0000	111	0	<input type="checkbox"/>
Edit	Delete	Steeleye Stout	Bigfoot Breweries	Beverages	13.0000	20	0	<input type="checkbox"/>
Edit	Delete	Laughing Lumberjack Lager	Bigfoot Breweries	Beverages	14.0000	52	0	<input type="checkbox"/>
Edit	Delete	Queso Cabrales	'Cooperativa de Quesos 'Las Cabras'	Dairy Products	22.0000	22	30	<input type="checkbox"/>
Edit	Delete	Queso Manchego La Pastora	'Cooperativa de Quesos 'Las Cabras'	Dairy Products	38.0000	86	0	<input type="checkbox"/>
Edit	Delete	Escargots de Bourgogne	Escargots Nouveaux	Seafood	13.2500	62	0	<input type="checkbox"/>
Edit	Delete	Aniseed Syrup	Exotic Liquids	Condiments	10.0000	13	70	<input type="checkbox"/>
Edit	Delete	Chang	Exotic Liquids	Beverages	22.0000	16	41	<input type="checkbox"/>

1 2 3 4 5 6 7 8

We'll start with an example - this list displays all the products that the company is selling. If we look at the data structure of this object, we see a product page which has product name, supplier, category name, unit price, units in stock, etc. The bigger requirement that is satisfied by repeating layouts is that the user interface does not require modifications when data that is associated with the products change.

There are various types of repeating layouts supported in the product, let's see them one by one.

Demo

- **Repeating Layouts**
- **Repeating Grids in Applications**
- **Adding a grid into a section**

1. Let's see some examples in the UI gallery by selecting the Designer Studio > User Interface > UI gallery.

The repeating layouts are grouped under Tables & Grids.

Let's open the Grids example first.

This example displays a repeating grid which is one of the more commonly used repeating layouts. The grid looks like a spreadsheet with rows and columns.

The header row identifies the columns, while each row is actually a page that is part of the page list.

This example displays a read-only list of companies. If we click on any column header then the results are sorted on that column, and if we click again the rows are sorted in reverse order.

In addition to sorting, Grids support a wider variety of features and these can be configured after adding grids in sections. This is another example for using a repeating grid – here the rows are grouped based on a category.

Grids support pagination which is useful in displaying large lists.

There are various ways to style the paging results.

It can display based on a set of rows, or the first 10 results, or progressive where the list loads only the set of rows that are requested in memory, and then when we scroll down it loads the additional rows. This is very useful in cases where all the results are not required to be loaded in memory unless required.

Filtering is another important feature that can be configured in grids.

Filtering provides different choices depending on the data type - for date fields we have the option to include a date range for numeric fields, we have the option to include a range, and for other text fields, it allows multi-selection as well as the ability to search using the text string entered in the field.

Grids also allow editing, in this example, we have to double click the row for editing, and there is an add a

row icon to add rows, it can add rows before or after the selected row.

Besides inline editing there are other ways to edit rows.

Let's look at one more example which uses expand pane.

Again there are various configuration options available in the expanded pane.

When we click the icon, it shows the details row which is collapsed otherwise.

The repeat grid is the most commonly used repeating layout and the UI gallery has various examples of the grids.

There are use cases where we may need to display hierarchical relationships, in which case we use trees, like in this case we are displaying the organization structure- Abigail Lippman has two direct reports and if we expand Jeremy we see that he has additional people reporting to him. Trees are better suited for this requirement.

The details of each node can be viewed by clicking on the node.

If we need to display details in the same fashion as the grids, then we have tree grids which is actually a combination of tree and grids. They can display hierarchical relationships and display as a spreadsheet.

2. Now that we have seen various examples in the UI gallery, let's open a purchase order case to see the use of grids.

The place order screen shows the list of all items that are ordered.

When we open the underlying section rule, we see that it uses a repeating grid layout.

To look at the configuration of the grid we need to click on the grid layout, and then click the icon to open the properties panel.

It references the page list named Lineitems for getting the list of items.

Pagination if required must be enabled and these are all the supported formats that we saw in the previous example.

When enabled, it displays an additional field for us to select the number of items we want to display in one page.

In the Operations tab we can see that this grid is configured to allow editing, there are also the flags to enable sorting or filtering, and selecting these flags automatically enables those features, it's that simple!

Grids can also be styled differently in the skin rule – there are three standard formats shipped in the product as part of the skin rule and if required we could also create any number of custom formats.

This grid displays the line items that are selected for ordering.

Let's create a new purchase request and go to the screen where we can add the line items.

In the Add line items screen, we see a section that looks different than a normal grid.

How do we know it's a grid? If we click the append icon, it creates a new row, and it repeats the set of fields that shows up in the repeating area.

We can add any number of rows by clicking the append icon.

We can also delete an item using the delete icon.

Let's look at the section rule to see the configuration.

We can see that it uses a grid layout, and in the properties panel we see that it uses .Lineitems which is a page list property that stores the entered values.

When we click the cell inside the repeating grid , and open the properties panel, we can see it includes a section in a cell directly which makes the entire section repeating when a new row is added.

When a section is added in the grid, it does not allow sorting or filtering because the one column inside the repeating grid does not reference any field but references the whole section.

3. In the stage designer let's add a new step to demonstrate how we can add a grid layout in the section.

After adding the step, let's use the outline view to configure this step.

In the Outline view, click the newly added step name, and to create a screen in the process we need to create a flow action rule.

After entering the name, we tab out and then click the Add fields link to enter the properties that we want to display in the screen.

The grid uses a page list as its source, so let's select a page list.

If the page list does not exist, the system creates it.

We also need to enter the fields that we want to be part of the grid.

We can add as many fields as we want, and these properties will also be created if they do not exist.

Once we are done adding, we click Ok to save the entries.

The system creates a flow action rule, section, and the pagelist and properties. Let's click the open icon to see the flow action, which references a section, and open it.

We see that it created a grid repeat layout.

The source is .Lineltems, and the system created columns for all the fields that we entered.

It also created icons for adding and deleting rows automatically.

The stage designer is one way to add the grid, it is usually used in POC implementations where we need to create all those rules and associated properties with one click.

In most cases, we need to add a grid layout on an existing section. To do it we click the layout palette, and then select layout, and drop it below the existing layout.

This pops up the set layout type dialog, which has the dynamic layout selected. Let's select repeating layout, and we see Grid in the select box.

Clicking OK adds the grid.

Let's see what other choices we have before clicking OK.

We see trees and tree grids.

Dynamic, Column and Tabbed are other choices which we will learn about in the advanced courses

Now let's click OK.

It adds a new repeating grid below the existing grid.

Note there is no source specified yet, we can enter the source for the grid by opening the view properties panel icon and then selecting the page list name here.

We should add properties inside the layout and we will learn more about that in the next lesson.

That sums up this lesson, we learned about various repeating layouts, then we learned more about grids and the different features supported in grids. We concluded by seeing how we can build repeating grids from stage designer, which we will use in the exercise.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6, seen in this lesson:

- Flow actions are given default names based on step name
- Process outline has been reformatted for easier use
- Configure form replaces Edit Fields
- List of field types available during form configuration has been updated

Exercise: Adding a Repeating Layout



Formatting Data Elements

The data elements must be formatted to display the values or accept values in the format we want the users to enter. We will learn about the rich set of controls shipped in the product that we can use.

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

- Understand what a control rule is
- Apply controls on the properties used in sections
- Use from a list of shipped controls
- Configure the control for additional formatting



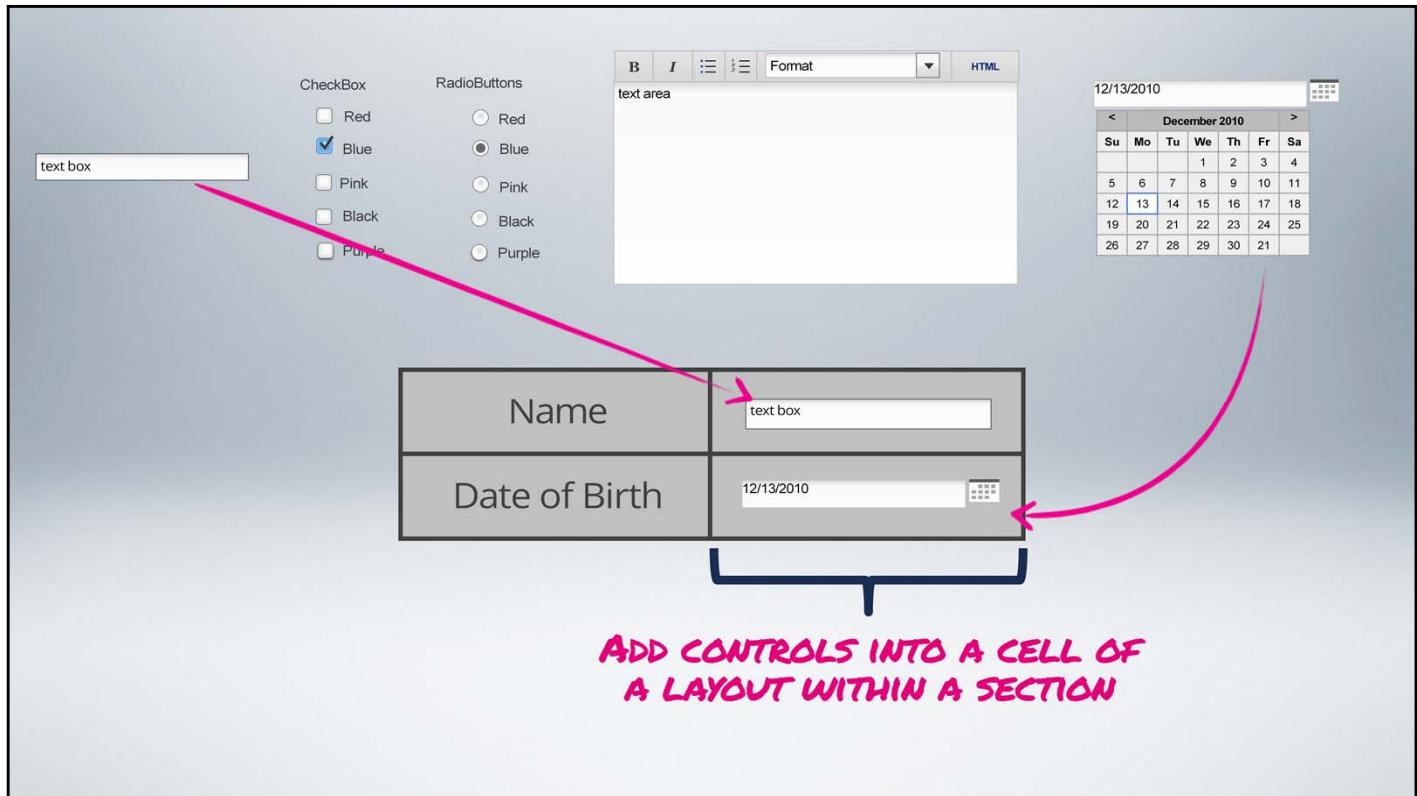
Controls

- Control the display and presentation of data elements
- Formatting the values
- Lot of commonly used controls shipped in the product

A Control rule is a user interface (UI) rule in PRPC that is primarily used to control the display and presentation of properties for end users who are working on the case.

Control rules play a key role in formatting the data elements that appear in the user interface.

There is a wide variety of standard controls that come with the product that we can use to format the data elements.



Some controls that come with PRPC are the Text Box, checkbox, Radio button, text area, and calendar. We can use these to display the data elements in those formats.

All of these are standard auto-generated controls which means, we can simply add them to our application, specifically to the sections, and they will automatically produce the necessary rendering code. We usually add a control in the cell of a layout which then requires some configuration. We will discuss the configuration required in a little bit.



Using auto-generated controls help our application to work in various browsers and various devices.



Properties and Controls are always connected to each other. When we add a property into the section the control associated with the property is automatically used although it can be changed. Let's see this in action to better understand what is happening.

Demo



Adding Controls to Sections

When a user is working on the case, they see a screen with all the fields that they need to enter. To see which rules are used, let's use the UI inspector. We need to hover over a specific field to see the rule names that are used to render this field. The field references a property named ProgramName, the control used is pxTextInput which is one of the standard controls that comes with the product and the control displays a text box for the field.

The field programname is placed in a section named EnterProgramFund. To summarize, the inputs a user sees at runtime are properties that have been added to sections using a specific control to optimize the data entry into those fields based on the type of data they expect to hold.

The EnterprogramFund section has all of these properties added inside the layout. If we open the properties panel on the programname field,

we can see that it uses the Text Input control and the property that is referenced is ProgramName. When we open the property rule we see that it also references the control here. The control is referenced in both places – in the layout cell as well as in the property. Now that we have learned about the relationships, how can we associate controls to fields? The first option is to use the stage designer approach. In the assignment shape, when we define a flow action we can also define all the properties that we want to display on the screen.

When we enter the fields here the system creates these properties if they do not exist and also puts them in the section. By default, it creates a dynamic layout which uses the Default Format, and then adds all the properties directly. As we see here, we can either enter a new or an existing property. We can also define the property type to change the fields.

When we open the section it displays the three properties we added. Let's open the properties panel for the

first. We can see it that has an entry which the control inherited from the property.

To better understand this let's open the property rule. We can see that there is a field and it has the control name entered here. When a property is defined, the system automatically assigns the control based on the property type. It assigns pxTextInput since this is a text field. When we go back to the section and look at the properties panel for the next field we see it uses the same setting as in the previous field.

When we open the property it uses a date control named pxDateTime since the property type is defined as date. At runtime we see that these fields use the controls that are defined in their respective property. However in some cases we would like the property to use different controls rather than the default ones that are assigned. This is when we select a control using the properties panel of the field in the section.

In the section we just created, we have a notes field, usually the notes field may need more than just a text box so users can enter additional comments. To change it to a different control we click the change link and then select one of the controls here – let's add the text area now. Clicking OK replaces the control with Text Area. After saving the changes, we can go back to the case that we were working on and we can see the changes. We can also add controls directly in the cell by using one of these two palettes – basic and advanced.

The Basic palette lists all the basic controls such as text box, text area, radio buttons, check box, select box, buttons and so on.

The Advanced palette has controls such as paragraph, autocomplete, smart label, and so on.

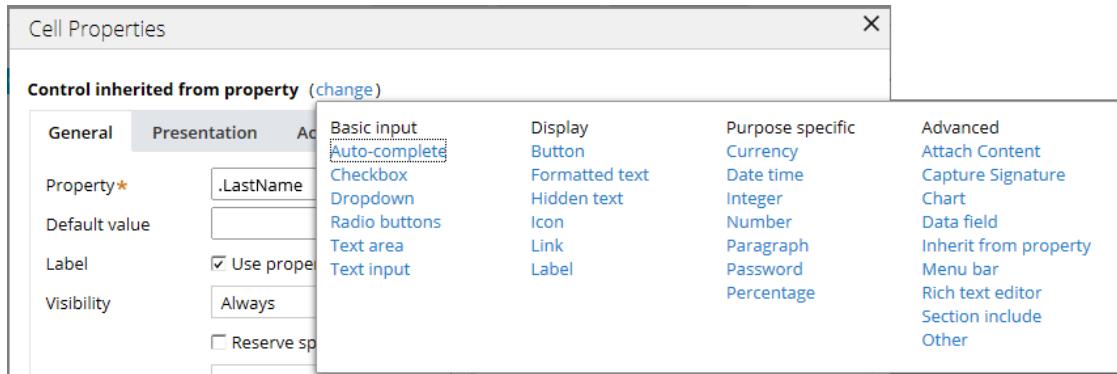
Let's add a control – to do this we click on the control name and then drag and drop it into the layout either above or below an existing cell. After adding the control, we open the properties panel to configure the control. The first step when we add a control directly from the palette is to associate the control to a property. Controls support granular configurations such as setting the width, height and style. Most of these are found in the Presentation tab. We can also set the field to be read only or editable. When setting the width and height we use a pixel width or a percentage. The last tab Action is covered in detail in a separate lesson in this course. The available configuration options of controls also vary based on the type of control. For example, let's look at the rich text editor control. This particular control lists additional options not available on other controls such as the toolbar for the Rich Text Editor – enabling some selections here displays those icons at runtime. If we open the second property which uses the date time control, we see these additional configurable parameters for the date field. The date can either use a calendar or use dropdown lists. We will look at this in a little bit, for now let's leave it as-is. Lastly, if we look at the text property we can see additional parameters such as the min/max chars which allows us to specify the minimum and maximum characters allowed in that field. Now, let's save the rule since we made some changes in the rich text area control and see how those toolbars appear at runtime.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6, seen in this lesson:

Cell Control Properties list has been updated:

- Attach content to attach and upload media files
 - In a desktop browser, launches file browser to select file
 - In a mobile web browser, allows users to attach image files
- Capture Signature
 - Allows you to sign your name on a mobile device form



Demo



Learn About Most Common Controls

The UI gallery is a great place to see how the controls are configured and we can use it as a learning tool. Let's open up the UI gallery to see some of these controls that we saw in the previous slide. Controls are listed under the available components area. Let's open the text input control that is all the way down at the bottom. Text Input displays a text box at runtime; this page has several examples showing how text input can be configured. The required field shows a backdrop text which goes away once we click on the field, not adding anything in the required field throws an error. This example restricts us – we can only enter a maximum of 10 characters. The next example highlights how to use a tooltip – this is useful when we want to provide additional instructions for users which will appear when we hover over the field. Most of these examples use a width setting of 100% so they display in the width of the entire section except for the one that is configured to display in only a portion of the entire width. And in the last example we show how the color of the field can change when the user focuses on the field.

There are more examples here. Please take a look at them and if we want to see how they are configured use the view design time configuration link. Let's use the navigate link to switch to a different control that displays as a sample in the UI gallery. Again the components area is where all the controls that are listed in the UI gallery exist, let's select Date Time.

Date Time can be presented in one of three ways – in the first option there is a calendar control which users can click on to see the calendar and they can also enter the date in the text box.

The No Typing style prevents users from typing in the text box forcing them to always select from the calendar. The last style dropdown displays the calendar icon

in addition to the dropdown for month day and year. The three dropdowns that are in the next row are for selecting the time – hour, minute and second.

Going back to the home page of the UI gallery, we see that there are samples of the other controls that we learned about in this lesson such as the Text Area and the Rich Text Editor.

In addition, there are more examples that we will learn about in the next video such as the dropdown, radio button, link, icon and button control. But first, let's apply what we've learned by working on a brief exercise.



- ▼ Text Input
- ▼ Date / Time
- ▼ Text Area
- ▼ Rich Text Editor

In the previous video we learned about control rules, how they can be added and where they are referenced.

We also learned about a few of the controls – Text Input, Date Time, Text Area and the Rich Text Editor.



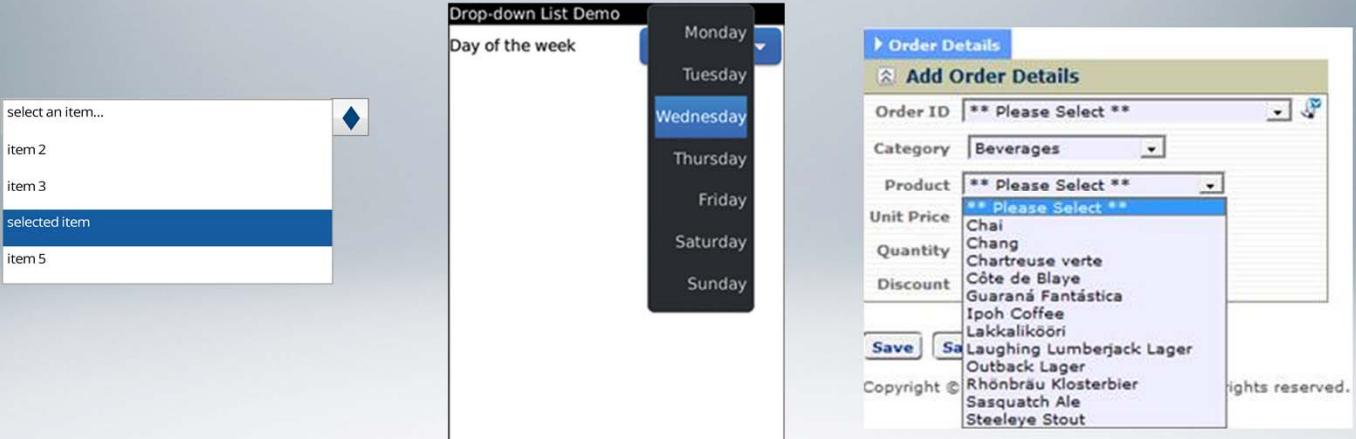
- ▼ Referenced in the property **CONTROL:'XTTEXTINPUT'**
- ▼ Can be changed to a different control in the cell
- ▼ Can be configured in the properties panel of the cell

And, we learned that the controls are referenced directly in the property rule form and they can also be changed in the section using the properties panel of the field. We can also configure the control using the properties panel.

Exercise: Adding a Control on an Existing Field



List based controls: Dropdown



In this lesson we will learn about a few controls that are commonly used in applications.

The first one is the dropdown- dropdown which is useful when we want to select from a list of choices.

These choices can be a static list such as day of the week,
or a dynamic list such as product names that the company is supplying or ordering.

List based controls: Radio buttons

- Option
- Selected Option
- Option

Horizontal

Do you drink coffee every day?

Yes No

Vertical

Do you drink coffee every day?

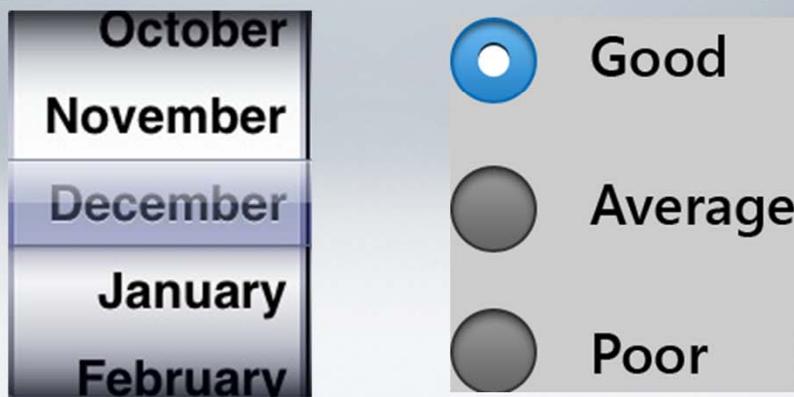
Yes
 No

In some scenarios, we want the list of choices to appear as radio buttons.

We will learn about radio buttons and also how we can configure the presentation of the lists to appear in different orientations.

Static List of Choices

Where to define these choices that appear in radio buttons and dropdowns



Remember we learned earlier that the control rule is always associated with a property and now we will see how we can leverage the property rule form to present the static list of choices.

Exercise: Adding List Based Controls



Action Controls



Icon



Link

Cancel

Button

A control is not always meant to display the value or to allow users to enter a value on a field. There are few controls that are useful when performing an action.

We will learn about these specific controls that are available to us in the product and let us perform actions.

Demo



pxDropDown

To add a dropdown let's open the basic palette, Click basic palette, click the dropdown control, drag the control and add it in the layout below an existing cell.

Now we need to associate a property with a control. Open the properties panel. We can use an existing property named Gender. Use down key and select Gender. This control has the option to select the listing source which provides the values that appear on the dropdown. Let's leave it as-is for now. Callout and then show the values. After adding the control, let's go back to the property rule form. Click Gender tab.

The list of values can be added in the Display and the validation section of the rule form. Expand the Display and validation. The Table type has various choices available to configure the values. Show the choices for table type. Let's select local list. It opens up another field and we can add the values in this field. As the name suggests Local list indicates the list we are adding here is only applicable in the context of wherever the property is being used. Users can only select from one of these choices, they cannot enter any other values.

After entering the values, let's open up the runtime view for this section. Click the case tab. We see the newly added field and the options that we added in the property. Click refresh.

Let's go back to the section and add another dropdown in the layout.

Click the section tab in the properties panel. Open the properties panel for the new dropdown let's reference another existing property. Use the down arrow and select Urgency.

If we look at the property definition. Click the open icon we see that it is an integer property Callout on property type integer and the table uses a different choice – prompt list.

Expand the display and validation. Prompt list is similar to local list in that it applies only in the context of the property and it also has two values - standard value, which the system expects (in this case an integer) and Prompt value, which the user sees and which can be a text value.

Callout on prompt list, standard and prompt value as the presenter speak

This is useful when we want to show the label to the user instead of displaying IDs. At runtime the dropdown appears in the same way as the local list, it does not look like an integer field.

Demo



Radio Buttons

Local and prompt lists can also be a source for radio buttons.

Let's change the control we have for Gender to use a radio button instead of the dropdown. Now the choices appear in radio buttons, even though we see it in the section. Let's look at how it appears at runtime. The choices appear one below the other. We can also change them to appear beside each other.

The list source is the same as what we had for dropdown. The presentation tab controls how the data elements appear in the screen. The orientation field is selected as vertical by default, so let's switch it to horizontal.

We see that the radio button choices now display next to each other.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6, seen in this lesson:

Cell Action Properties list has been updated:

- Get Directions action added
 - Launches mapping tool

All actions (Common actions)			
Display	Process work	Launch	List
Apply conditions	Add new work	Flow in modal dialog	Add child
Close	Cancel	Get Directions	Add item
Expand/Collapse	Contents	Harness	Delete item
Menu	Enable action section	Landing page	Edit item
Post value	Explore	List view	Open local action
Print	Finish assignment	Local action	Open selected item
Refresh	Perform action	Open URL in window	Refresh current row
Refresh row	Review	Report definition	Refresh list
Set focus	Save	Wizard	Set focus
Set style	Show flow location	Get work	Other
Set value	Show reopen screen	Create work	Invoke action
Show smart info	Update	Get next work	Log off
Show smart tip	View Attachments	Open assignment	Open rule
Spell check	View history	Open work by handle	Open rule by keys
		Open work item	Open rule by name
		Re-open work item	Run activity
			Run data transform
			Run script
			Toggle Mobile/Full-Site

Demo



Action Controls

Controls are not just meant to display a user input field for users to enter or select values. There are controls that can be used for making some actions. In the basic palette we see buttons, icons and links. These are controls that we use to perform some action. Let's see an example in the gallery. There are some sample buttons available in the gallery.

The buttons can be customized to show a different background color, font and other style elements. When we add a button control we can configure the label, add an image and select a style that is defined in the skin. The action buttons perform a specific action – the launch button opens a screen in the modal dialog and the menu button displays a menu. We could also do the same for links.

The UI gallery presents the same example we saw for actions. The last action control we have is the icon. Let's quickly look at the design time configuration for these action buttons.

The section has all these buttons. Let's view the properties panel for the launch button. When the button is selected the control is not associated with any property since it does not store or display a value.

Let's go the Presentation tab. The format field lists all the different styling options that are saved as part of the skin rule. Optionally, the image source can be used to display an image on the button. The actions menu lists the configuration that is required to perform the action. The Actions menu uses an action set. There are two separate parts in the action set.

The first is to select an event – so when using a button, link or icon, the event has to be clicked. This means that the action happens when the user clicks on the button, link or icon.

After selecting the event, we can configure the action we want taken.. Here it launches a local action which is a flow action rule.

There are various other events available for selection in the event field. Not all events are applicable on all controls. We can perform many actions on the same event.

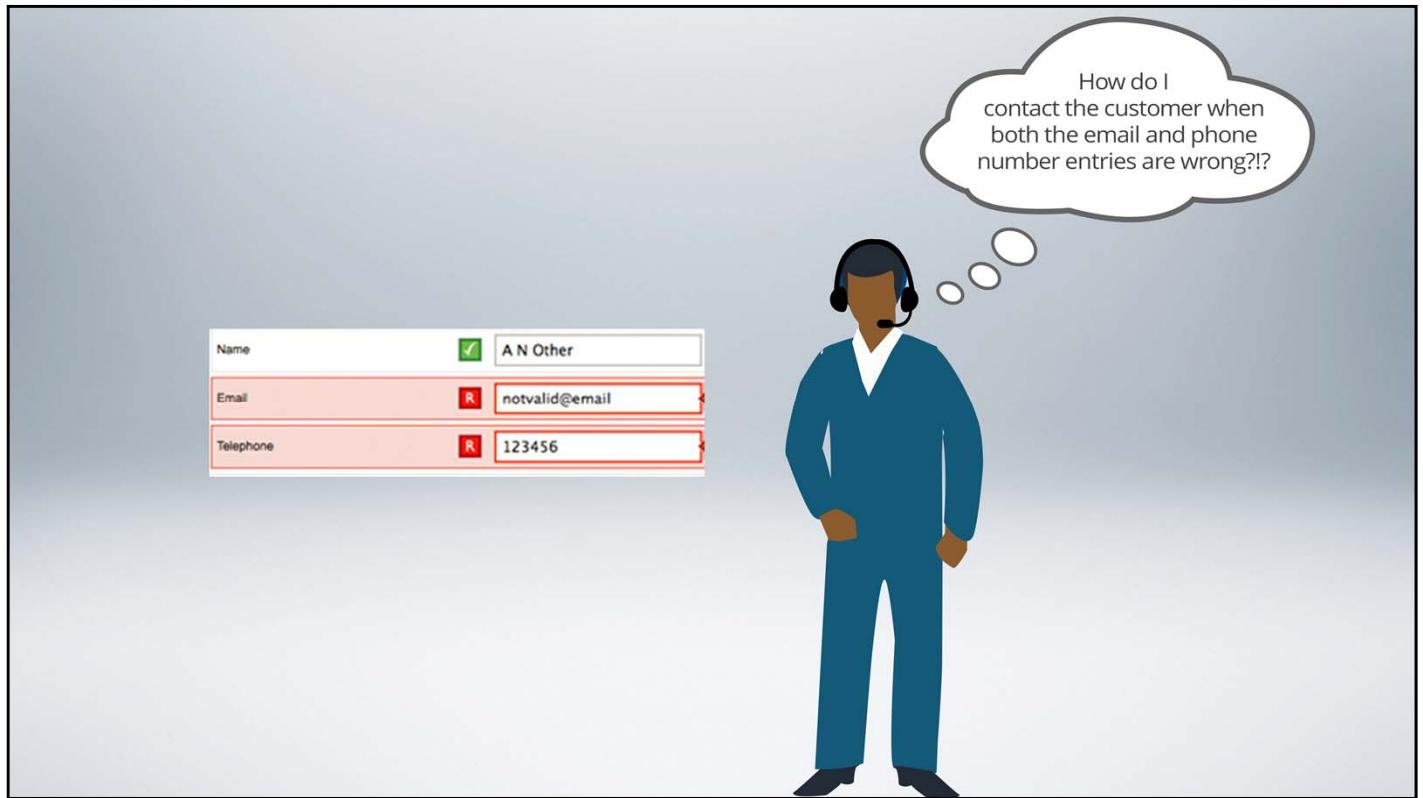
The system displays the most commonly used actions when we add an action. To see more we can click the All action link. The system supports a wider variety of actions that can be performed. These actions are grouped in categories, and we will learn about adding a few events in the dynamic user interface lesson. Now let's try another exercise using some of these controls we just learned about.

Validating Data Elements

While working on UI screens its highly important to ensure that the UI inputs enforce users to submit valid values.

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

- Understand various forms of validation
- How the validation rules are referenced
- Perform mandatory field checks



When users are working on a case it's very important that the data the user is entering is the data that the system expects.

The data entered in these fields may be used in the subsequent stages of the case. So we want to make sure the case does not contain any invalid data.

Phone Number:	() -	(###) ####-####
Phone Number with optional Extension:	() - Ext.	(###) ####-#### Ext\#, 99999
Social Security Number:	- -	###-##-####
Zip Code:		#####
Alphanumeric text - all characters required:		AAAAAAA
Alphanumeric text - all characters optional:		

In some cases users may not be aware of the format in which the system needs the data unless informed. To provide a good user experience it is important that we guide users to enter valid data.

The screenshot displays a user interface with four distinct sections:

- Select Field:** A dropdown menu labeled "- Select One -".
- Combo Box:** An input field with a dropdown arrow.
- Multi-select:** A two-panel interface for selecting items from a list. The left panel, titled "Full List", contains four items: Item 1, Item 2, Item 3, and Item 4. The right panel, titled "My Items", contains the text "none". Between the panels are four buttons: up and down arrows for navigating between lists, and a double-right arrow and a double-left arrow for moving multiple items at once.
- Radio Button Array:** A group of five radio buttons. The first button, "Option 1", is selected (indicated by a filled circle). The other four options ("Option 2", "Option 3", "Option 4", "Option 5") are empty circles.

One way to handle this is to make sure the user interface forces users to enter the proper values thereby eliminating the need to validate the data they enter. This is a win-win scenario — it's simple for users because they cannot enter anything wrong and it's good for the system since it always gets valid data.

Phone Number:	() - (###) ####-####
Phone Number with optional Extension:	() - Ext. (###) ####-#### Ext\#, 99999
Social Security Number:	- - ###-##-##-##
Zip Code:	#####
Alphanumeric text - all characters required:	AAAAAAA
Alphanumeric text - all characters optional:	

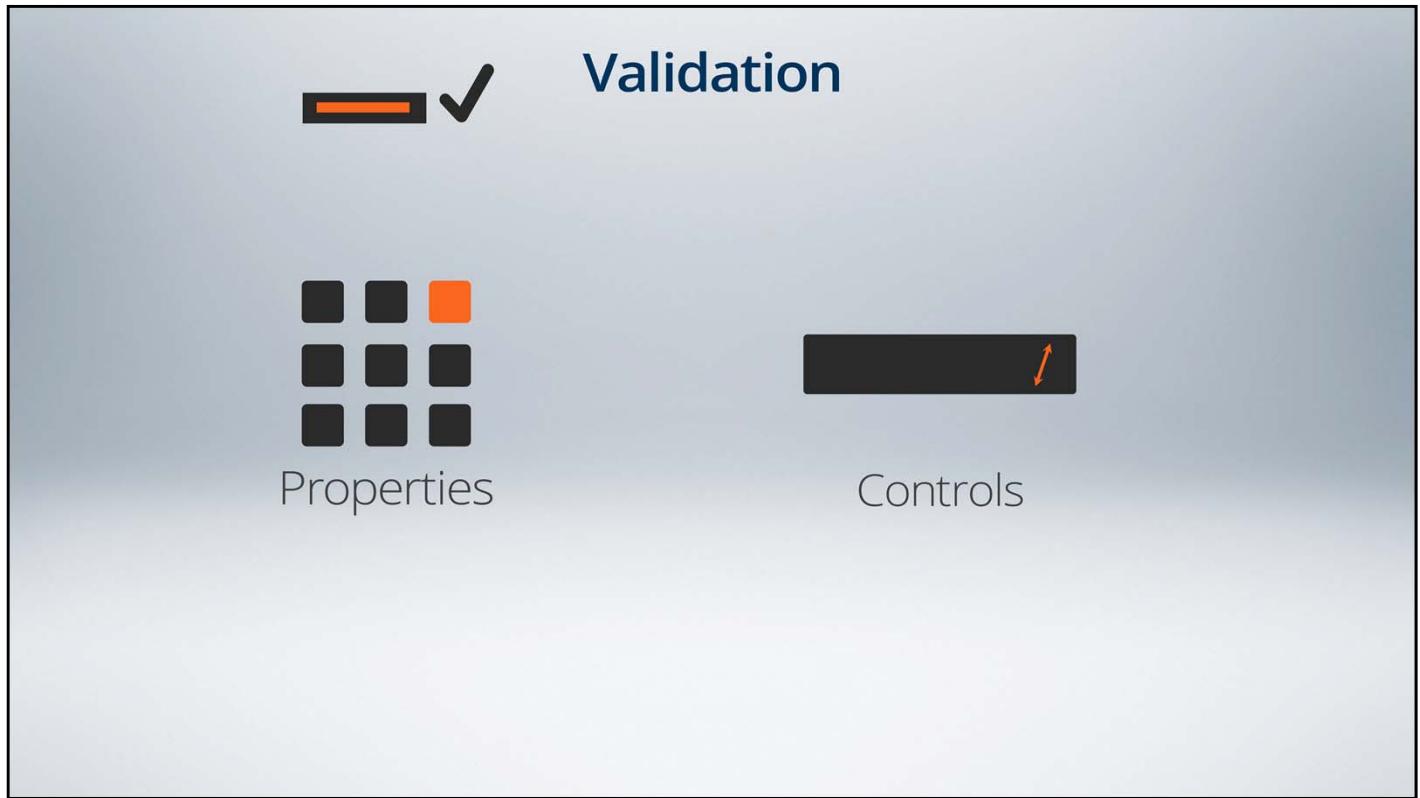
However, it is not always practical to follow this, which means we need to create validation rules. In such cases, it is important for us to provide the appropriate messages so users enter valid values. These messages can either appear on screen along with the field or

Register

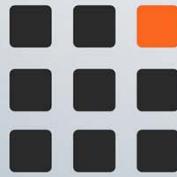
Username:	PeterS	
Password:		Password is required!
Confirm password:		Password is required!
Real name:	Peter Stoev	
Birth date:	11/05/2006	11
E-mail:		E-mail is required!
SSN:	—-—	Invalid SSN!
Phone:	(_)—-	Invalid phone number!
Zip code:	—-—	Invalid zip code!

I accept terms You have to accept the terms

display as the error message when the inputs are validated.



PRPC offers a couple of options that we can leverage to enforce users to enter valid data. These are not thought of as validation rules; however they play a key role in enforcing that users enter valid data. They are – Properties and Controls



Property

Identify the property type

- Date
- DateTime
- Decimal
- Identifier
- Integer
- Password
- Text
- TextEncrypted
- TimeOfDay
- TrueFalse

Properties- A property rule not only provides the premise to define a data element but is also useful in making sure the valid data is collected.

The property rule form offers various ways to enforce validation - when we define a property, we must specify the data type.

When selected as single value, it can be one of the following data types – Date, DateTime, Decimal, Integer, Text, etc.

Property

Select appropriate data type

Unit Price must be defined as Decimal

UnitPrice
\$14.00
\$9.80
\$34.80
\$18.60
\$42.40
\$7.70
\$42.40
\$16.80
\$16.80
\$15.60
\$16.80
\$64.80
\$2.00

We always need to make sure that we define the appropriate data type.

For example, when we define a property for unit price, we define it as decimal instead of text so users can enter only numeric values.

Control

Enforces the user selections at runtime



Control – Controls which we learned about in the previous lesson are not only useful in formatting values but are also useful in enforcing what users can enter while processing a case.

For example, the calendar control forces users to select a date, month and year using the calendar so they cannot enter an invalid value.

Recommended Practices

Define properties with appropriate data type

Use controls that guide the user to choose valid choices

Defining the properties with the appropriate data type and using a control that restricts users from entering invalid data goes a long way in making sure users enter valid data.

Property: Local list

Finite list of choices

- Local list
- Data page

▼ DISPLAY AND VALIDATION

UI Control
RadioButtons 

Table Type
Local List 

TABLE VALUES

1	Male
2	Female

Search and Select: Select a Manager  

Search 

Results

Select Ename	Actions
FORD	 
SCOTT	 
JAMES	 
TURNER	 
MARTIN	 
WARD	 
ALLEN	 
MILLER	 
ADAMS	 
CLARK	 

 Previous 1-10 of 13  Next 3 

The property rule can also be used for another type of validation. When we need users to select from a finite list, we could use the property rule form to list all the choices by using a local list or we can use other rules like data pages to control the list of values.

Validation Requirements

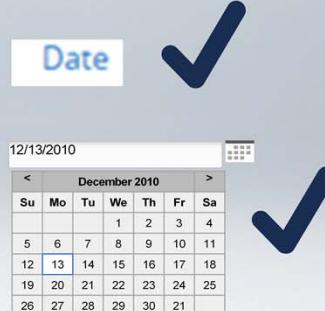
Date of Birth

- **Property Type:** Date
- **Control:** Date-Calendar

Date of Birth

11/18/2041

Enter a valid past date



Despite using proper data types and controls we will still end up with cases where these two controls are not enough.

For example, when we want users to enter their date of birth,

we want them to only enter dates, so we define the property to be of date type

and use the calendar control so then at runtime users can only select from a calendar.

Now we want to restrict users so that they cannot enter a date in the future for the date of birth field since it is not valid. Defining date of birth as a date property and using the calendar control users cannot choose anything other than a date value, however users can select a future date.

The image displays two side-by-side screenshots of a web application's user interface, illustrating validation errors.

Left Screenshot:

- Header:** "1 error prohibited this user from being saved"
- Message:** "There were problems with the following fields:"
- Error List:** "• Password doesn't match confirmation"
- Fields:** "Password" (containing "*****") and "Confirm password" (containing ".")
- Submit Button:** A standard "Submit" button.

Right Screenshot:

- Header:** "2 Errors"
- Errors:**
 - Username:** Must be 10 characters or less.
 - Age:** Must be in the 5 to 120 range.
- Fields:** "Username" (containing "MostAwesomestUserNameEver") and "Age" (containing "-42")
- Submit Button:** A standard "Submit" button.

Similar to the date of birth example, we will have validation requirements where we need to make sure the value entered in a field must be compared against another field, the value has to match a specific format, the value entered should not exceed a specific size or the value is in a specific range and so on.

Validation Rules

Validate

- Rule useful in building conditions

Edit Validate

- Rule useful in making sure the pattern is matched

There are multiple ways in which this can be accomplished. Let's take a look at two of the most commonly used validation rules in the system. They are – Validate and Edit validate.

Validate Rules

Validate

Rule useful in building conditions

The screenshot shows a web form with two input fields: 'Username' and 'Age'. The 'Username' field contains 'MostAwesomestUserNameEver' and the 'Age' field contains '-42'. A red error summary box at the top indicates '2 Errors'. Below it, two specific validation messages are listed: 'Username Must be 10 characters or less.' and 'Age Must be in the 5 to 120 range.' Each message has a small info icon to its right. At the bottom of the form is a 'Submit' button.

The Validate rule is useful in establishing conditions for a data element – such as comparing a property with a value

Validate Rules

Validate

Rule useful in building conditions

The screenshot shows a user interface for a password confirmation form. At the top, a red error box displays the message: "1 error prohibited this user from being saved" and "There were problems with the following fields:". Below this, a bulleted list indicates: "• Password doesn't match confirmation". The form contains two input fields: "Password" (containing "*****") and "Confirm password" (containing "*"). A "Submit" button is located below the input fields.

or comparing two properties (password and confirm password)

Edit Validate Rules

Edit Validate

Rule useful in making sure the pattern is matched

The image shows a portion of a web form. At the top, there is a dropdown menu labeled "*Sex:" with "Male" selected. Below it is an input field labeled "*Email:" containing the value "abc@abc". A red box highlights this input field. To the right of the input field, a red callout box contains the error message "Please enter an email address." A red arrow points from the bottom of the input field towards the error message.

Edit validate rules are used in cases where we want the values to match a specific pattern such as a phone number, SSN or email address.

Demo



Validation Rules

Now, let's look at some of these rules in our application to see them in action.

This is a validate rule, here we see that it can be configured to validate more than one property in a single validate rule. Each property has a validate condition which we can see by clicking the Expand all button.

Now the conditions appear next to each field, the program name has a condition where it displays the error message when the value is null. We can see the actual condition by clicking the Edit link.

The validation condition is configured in this dialog. In this rule it checks to see if the value stored in the program amount is less than or equal to 0.

If the condition is satisfied, the error displays as per the message rule here. The flag ContinueValidation is very important. If it is enabled it continues validating the rest of properties that are defined in the validation rule, and if it is not enabled it stops validation and displays the error message.

Just as we saw in the examples, validate rules can also be used to compare two properties. This is an example where the total cost in a purchase request is verified if it exceeds the total program amount. Notice in this dialog, the actual message is entered here instead of a message rule.

Let's take a look at another example.

Here the line items are validated. There are two conditions on the same property - the first condition is checking to see if the length is at least 1 and if the second condition references another validate rule.

Since it's a page list this validate rule is applied for each line item entered by the user. The validate rule can

also call additional validation rules in the additional validation field and this is the same rule we saw a little bit earlier. Ok, so we defined the validate rules, how do we access them? In the Actions menu, we click View References which displays all rules in which this rule is referenced. This displays one entry - one validate rule. We can open the validate rule from here by clicking the icon next to the rule name. The validate rule opens and we need to dismiss this window. This is the same rule we saw a little while earlier.

Let's see where the validateLineItem rule is referenced. This rule is validating the list of line items that the user enters. It is referenced in the Add Line items flow action, which is where line items are added.

Let's open the flow action. We need to close the window again to see the flow action. The validate rule reference is configured in the validation tab. When users submit the action all the user fields in that screen are validated. We need to make sure that the validate rule includes only the fields that exist on this screen, otherwise it throws exceptions and does not proceed further. We will learn more about validate rules in System Architect Essentials II course.

Now let's look at edit validate rule. There are various edit validate rules that are shipped as part of the product. To see them, let's use the Records Explorer.

Edit validate rules are part of Data model category. Like we learned in the designer studio lesson, the Records Explorer displays all rules that are in the system and not just part of the application. Let's open one of them – IsNonNegative. This rule checks the value entered in the field to make sure it is positive. Edit validate uses Java Code and so it is very important that you have Java expertise to create them. There is a rich set of standard Edit validate rules that we can leverage. Typically a Java developer creates these rules, we just need to know how to use them. We can search by entering a string to see what rules are available. There is a rule for checking if the phone number is valid, and there is also the rule that validates the email format.

Let's open the rule to see how this rule can be used. We will do it by checking the references to this rule. A property rule named pyEmailAddress references this rule. Let's open the property rule to see where the reference is made. This is a standard property shipped in the product for work party information. The reference is made in the Advanced tab, in the Use Validate field. We will not be asked to write these rules, however we should be able to use the shipped ones or the ones created by a senior architect.

Demo



Identify Mandatory Fields

One of the most common requirements is to make sure users enter values for all the required fields.

Let's see how we can address this requirement. We can do it in couple of ways. One way is to use a validate rule to handle required fields. But, let's look at the other option.

Let's say we need to identify the required fields in the Enter request details step. We see the flow action that is being used and the fields that display in this flow action are defined using the stage designer.

It has two fields – requested for and needed by. Let's say we want to make the requested for field required, since we do not want users to create a purchase request without entering requestor. We can do this just by enabling the required field. Let's click OK and then open the section rule to see what changes the system made after this selection.

In the section, we see an asterisk next to Requested For label to identify that it is a required field. When we open the properties panel for the field, we can see that there is a flag to mark the field as required. Let's say the customer looked at it and decided to have the Neededby field be required as well. We can make the change directly here by opening the properties panel and enabling the field. After clicking OK, we see that there is now an asterisk symbol next to the neededby field as well.

At runtime, when we reach the screen, we see that all required fields have this flag next to them indicating users must enter some value. If the user continues without entering valid values, the system throws this error which is automatically configured. Actually, the required flag is validated even before clicking Submit. For example, if we are in requested for field and move to the needed by field we get the error.

That sums up the lesson. We saw the need for validation, how properties and controls can restrict users to enter valid data, and then we also learned about validate and edit validate rules. We saw some specific examples of each of them and how they are referenced. Lastly we learned how to enable required field validations in a PRPC application.

Exercise 1: Identifying Mandatory Fields
Exercise 2: Validating User Entries

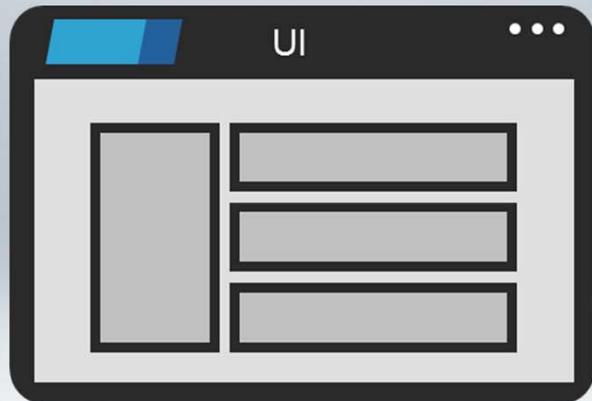


Review of Enhancing the User Interface

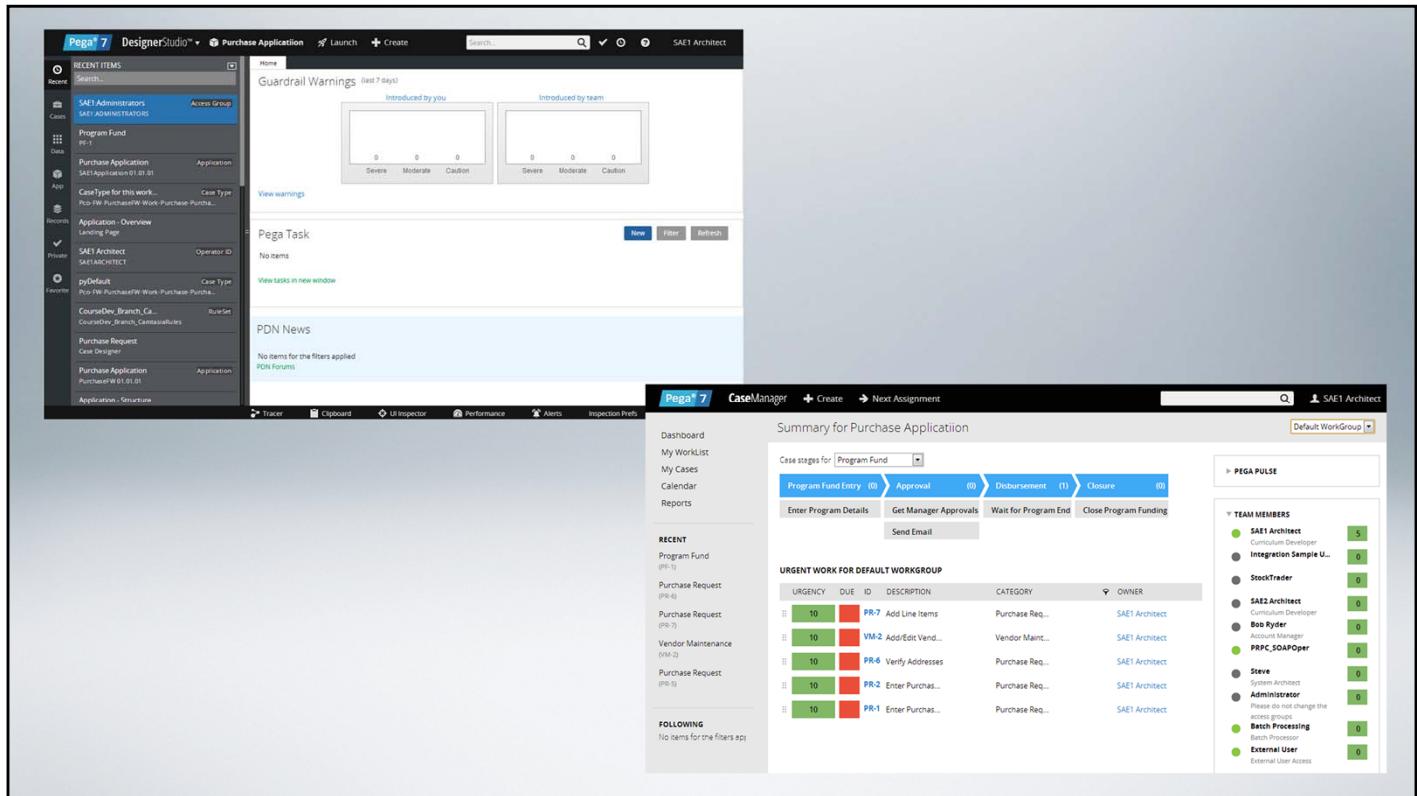
Building UI screens is a critical component in the application development. Sections, controls, flow actions and harnesses are some of the rules we will use to build user interfaces that are used in case processing.

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

- List down the list of UI rules that are available to use
- Format the data elements that appear on user screens
- Validate the values that are submitted



We started this lesson group by identifying where UI rules are referenced in a Pega application.



The user work space or the portal that the user sees after they login is created by using UI rules.

You have 2 account selected

Step 2: Verify your identity
Help us ensure who you are

For security purposes, please answer the following questions to help us verify your identity. When you are finished, click Verify My Identity to proceed with the application process.

In which month was Sarah Smith born?

- January
- February
- May
- October
- December
- None of the above
- I am not familiar with this person

With which of the following addresses have you been associated?

- 1234 Adelaide St
- 456 Richmond St
- 210 Spadina Ave
- 503 Bloor St
- 352 Yonge St
- None of the above

With which of the following corporations have you been associated?

- ACME
- ABC Company
- DE Inc.
- Red Inc.
- Web Form Designers LLC
- None of the above

PLACE ORDER

Review the Purchase order and then place it to Amazon, check the information tab for address and other contact information for the vendor.

ITEM ID	ITEM CATEGORY	ITEM DESCRIPTION	QUANTITY	UNIT OF MEASURE	UNIT PRICE	SUBTOTAL
1	Hardware	Laptop	2	Each	1200	\$2,400.00
2	Hardware	Mice	9	Each	30	\$270.00
3	Hardware	Keyboard	10	Each	60	\$600.00

Shipping Method:

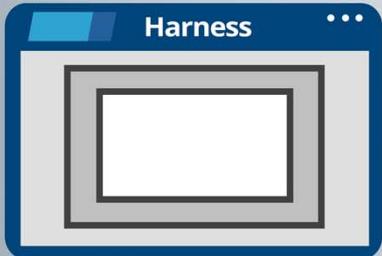
Tax:

Order Total: USD3,270.00

Step 3: Personalize my accounts

When a case gets created, it triggers its starting process and all assignments that require a specific user action is rendered by a set of UI rules.

UI Rules



What are the different sets of UI rules? We have:

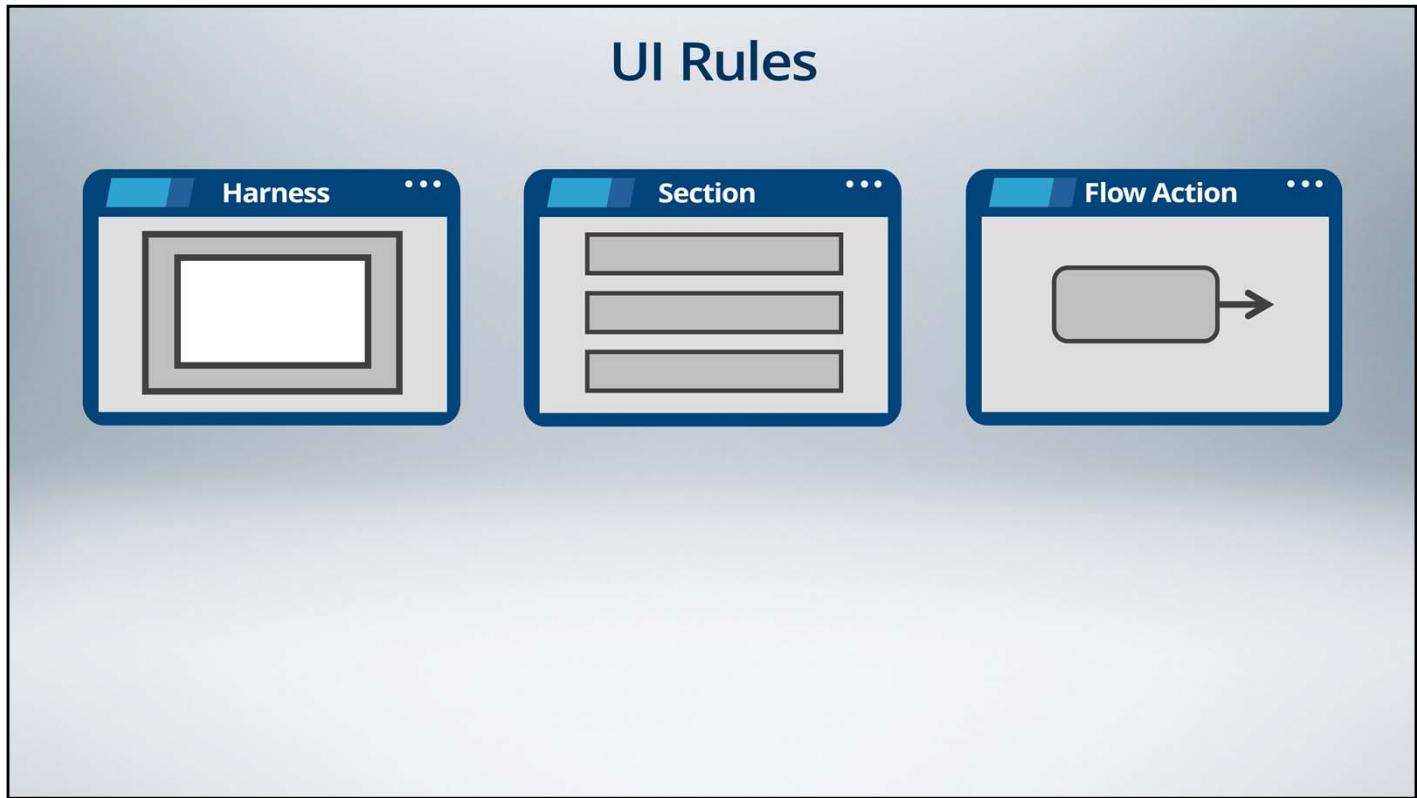
Harness rules –

Standard Harnesses



We use four standard harnesses which are used when a case is created, processed and resolved.

The New harness is used before a case is created, the perform harness is used while performing an action on the case, the review harness is used to enable a user to see the case when they do not have permission to perform an action on the case and the confirm harness displays after the case is resolved.

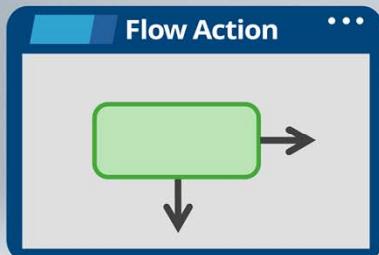


Section rules –

These are the building blocks of the UI, it's one of the primary rules that controls the content that end users see.

Flow Action rules –

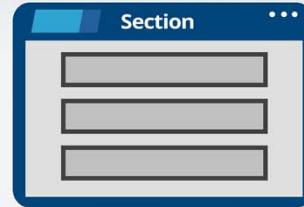
Flow Actions



Referenced in the connectors coming out of the assignment shape

An assignment shape can have more than one connector

Flow actions reference sections

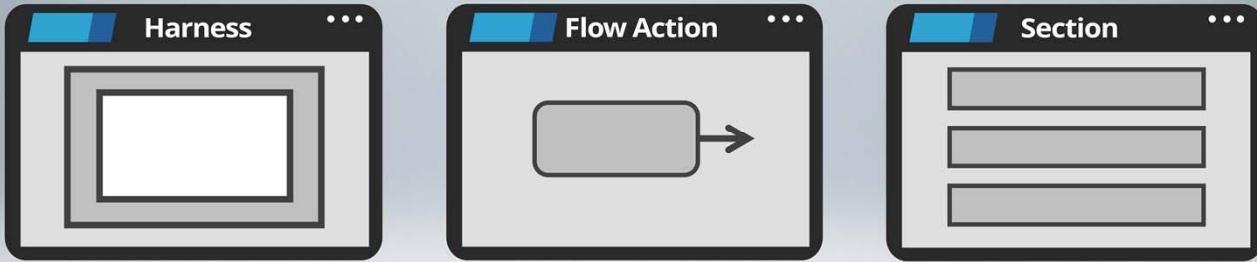


These must be used on all assignment shapes because they determine which path or action end users can take.

Some assignment shapes have more than one connector coming out of it – For example, when a manager is reviewing the purchase request, they can either approve or reject it. So in this scenario there will be two connectors coming out of an assignment shape. Each connector shape references a flow action rule

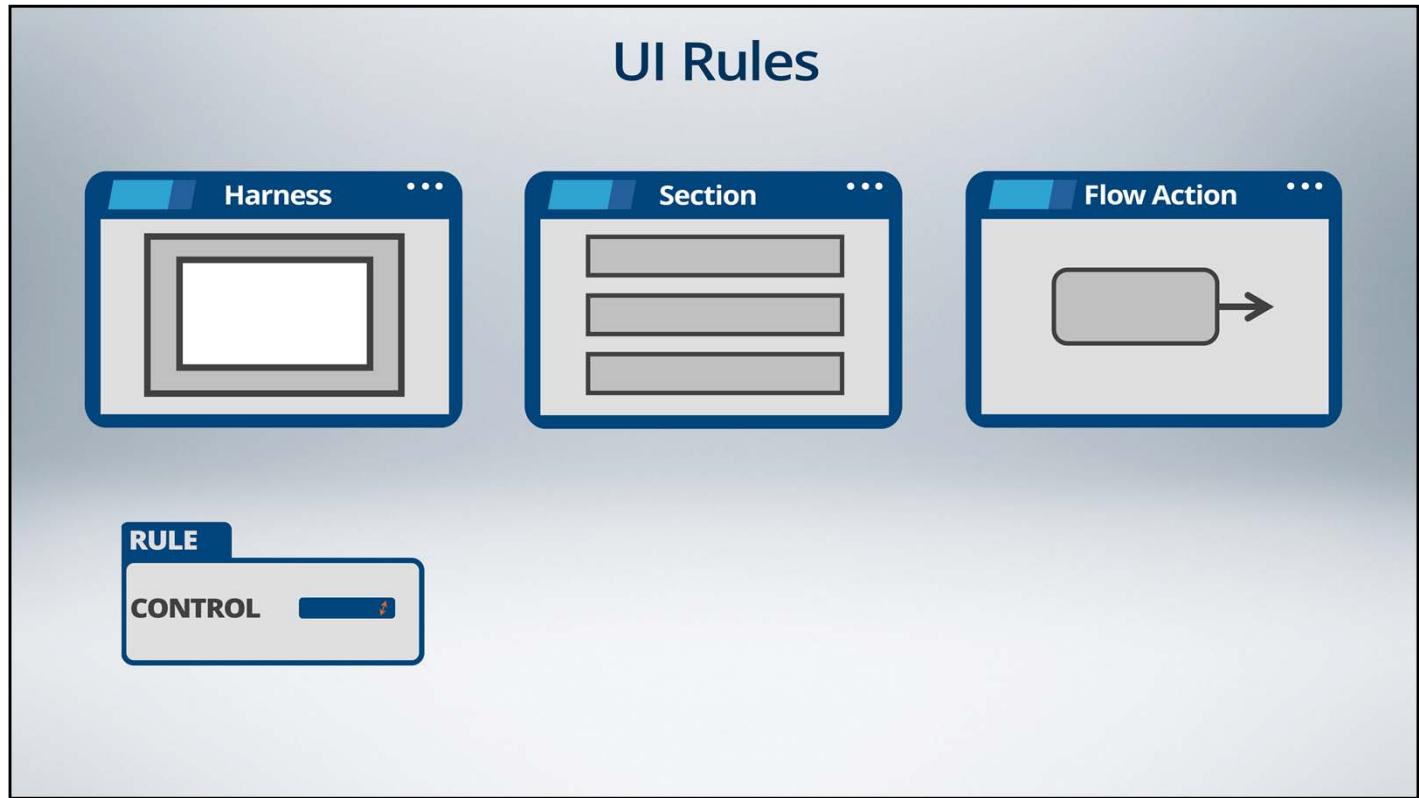
and each Flow action in turn references a section to render screens for both Approve and Reject.

Three UI Rules Impacting Case Processing

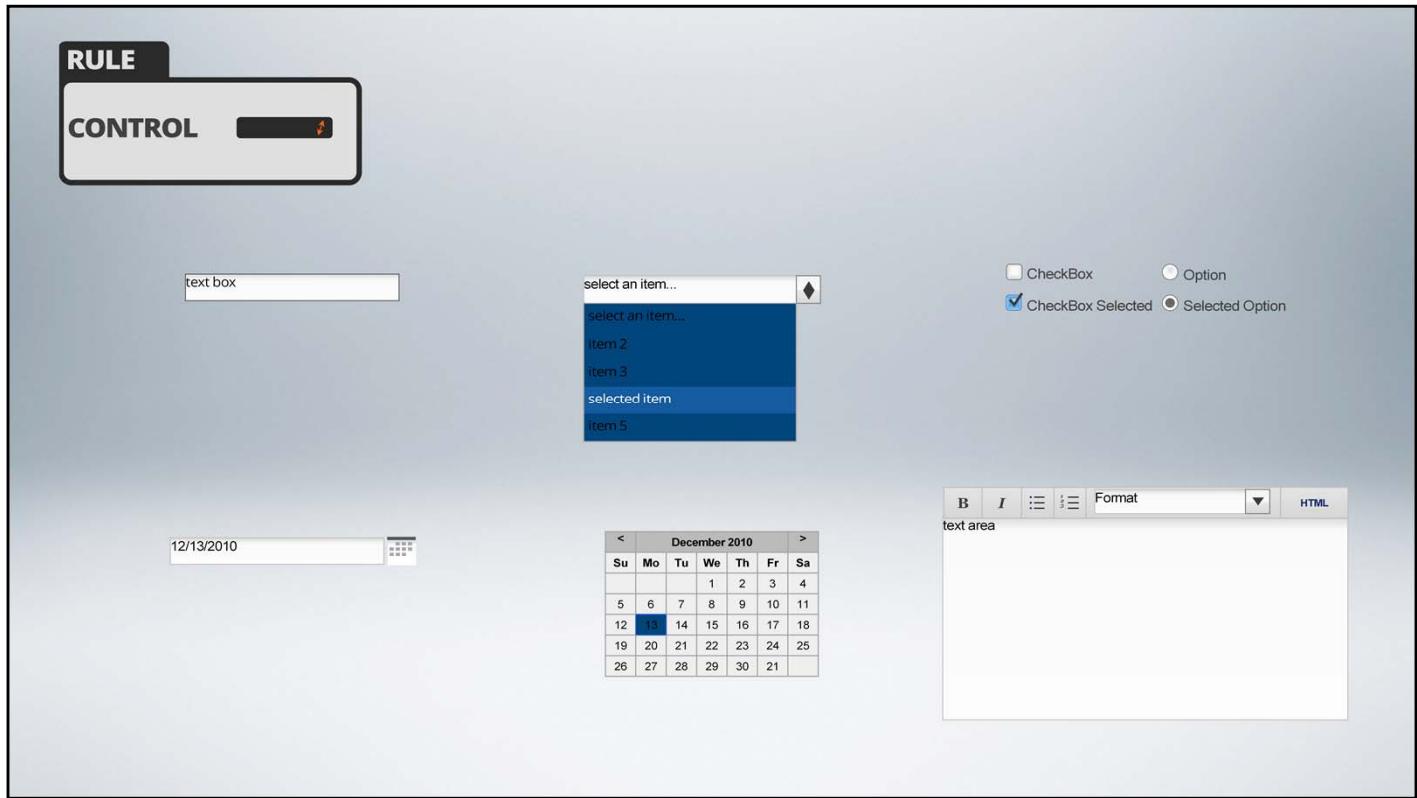


These three rules play a key role in how the user interfaces are rendered during case processing.

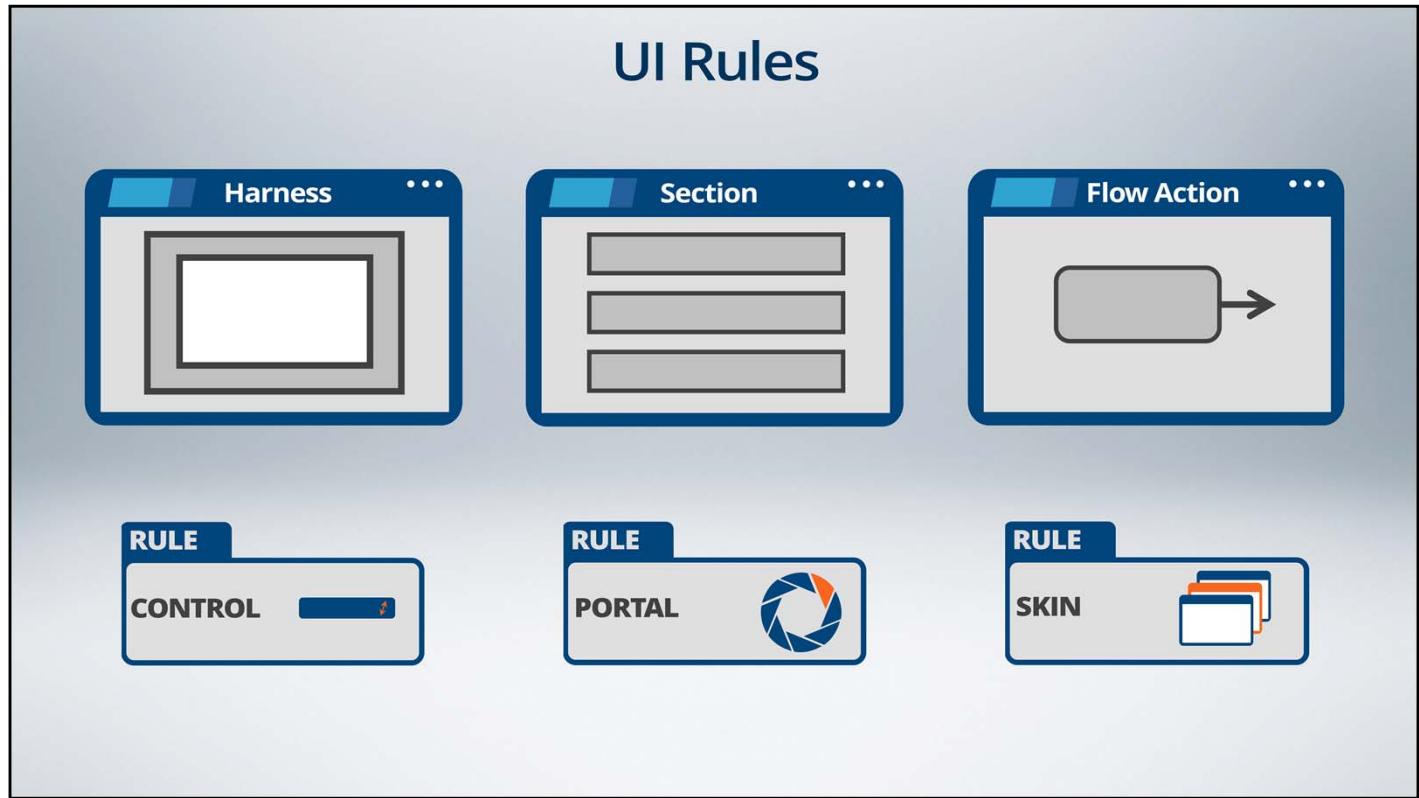
We have harnesses which are the top level page, flow actions which are the specific paths or actions a user can perform and sections which are used in creating the content that the user views.



Controls –



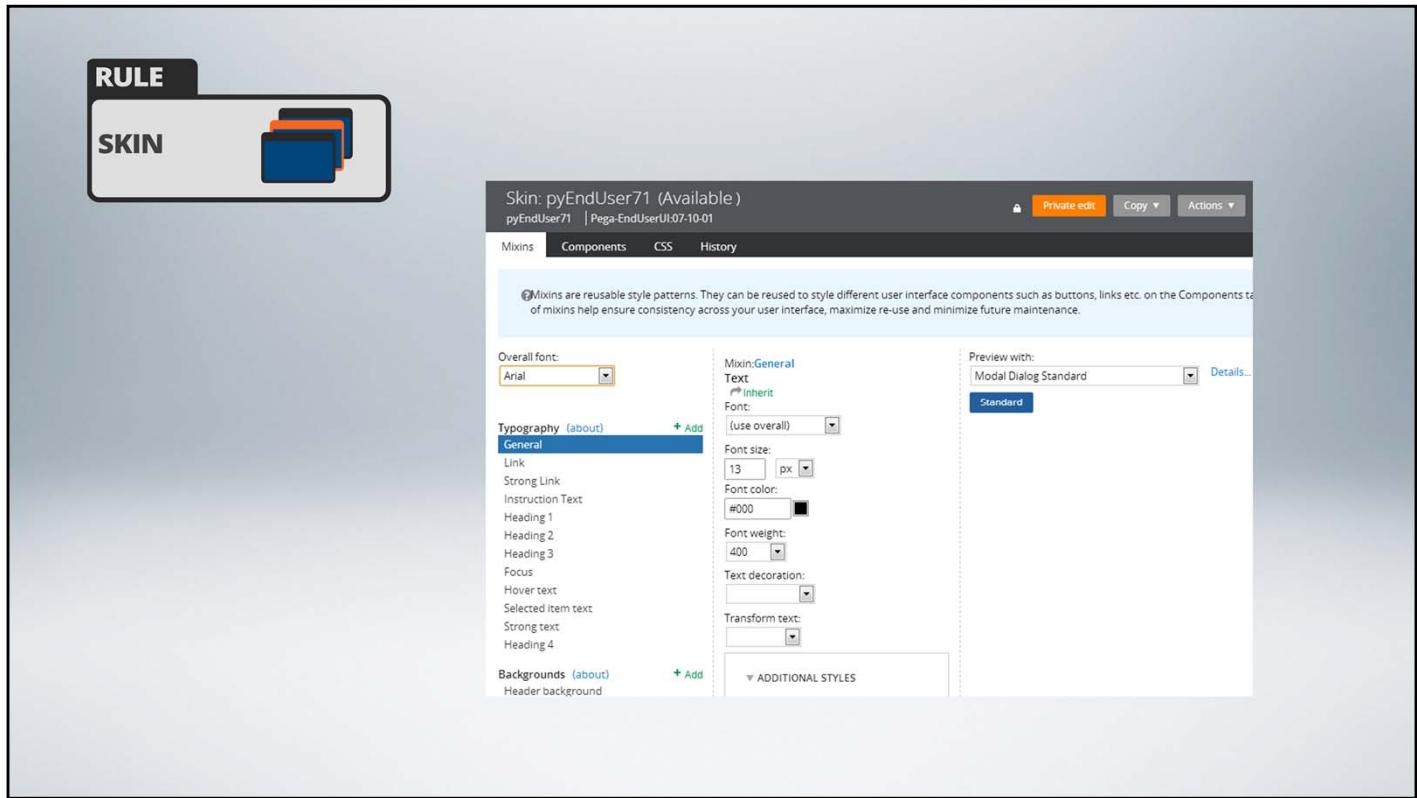
These are used in configuring the presentation of properties in sections. For example, a property could use a text box, dropdown, radio button, checkbox, and so on. Pega 7 comes with a list of standard controls for all the most commonly used presentation choices which can be used without additional coding.



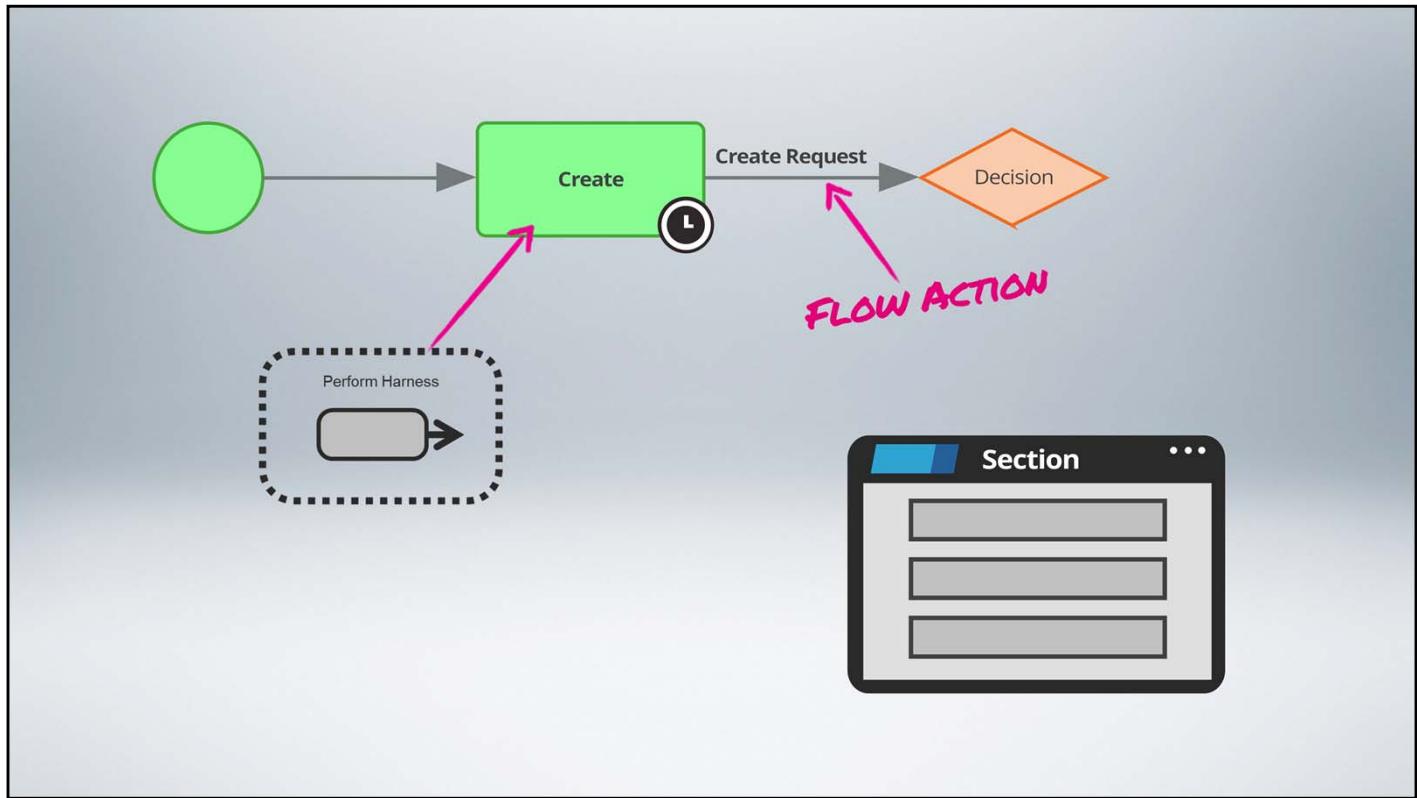
Portals –

These are user interface rules and are primarily used to display the workspaces for users. Portals use harnesses which contain information on what the user sees.

Skins –



The skins are used for defining the styling elements such as color schemes, font schemes, images, that are used in the application. Pega provides a wizard to generate skins and it does not require end users to know CSS and other styling elements to create them.

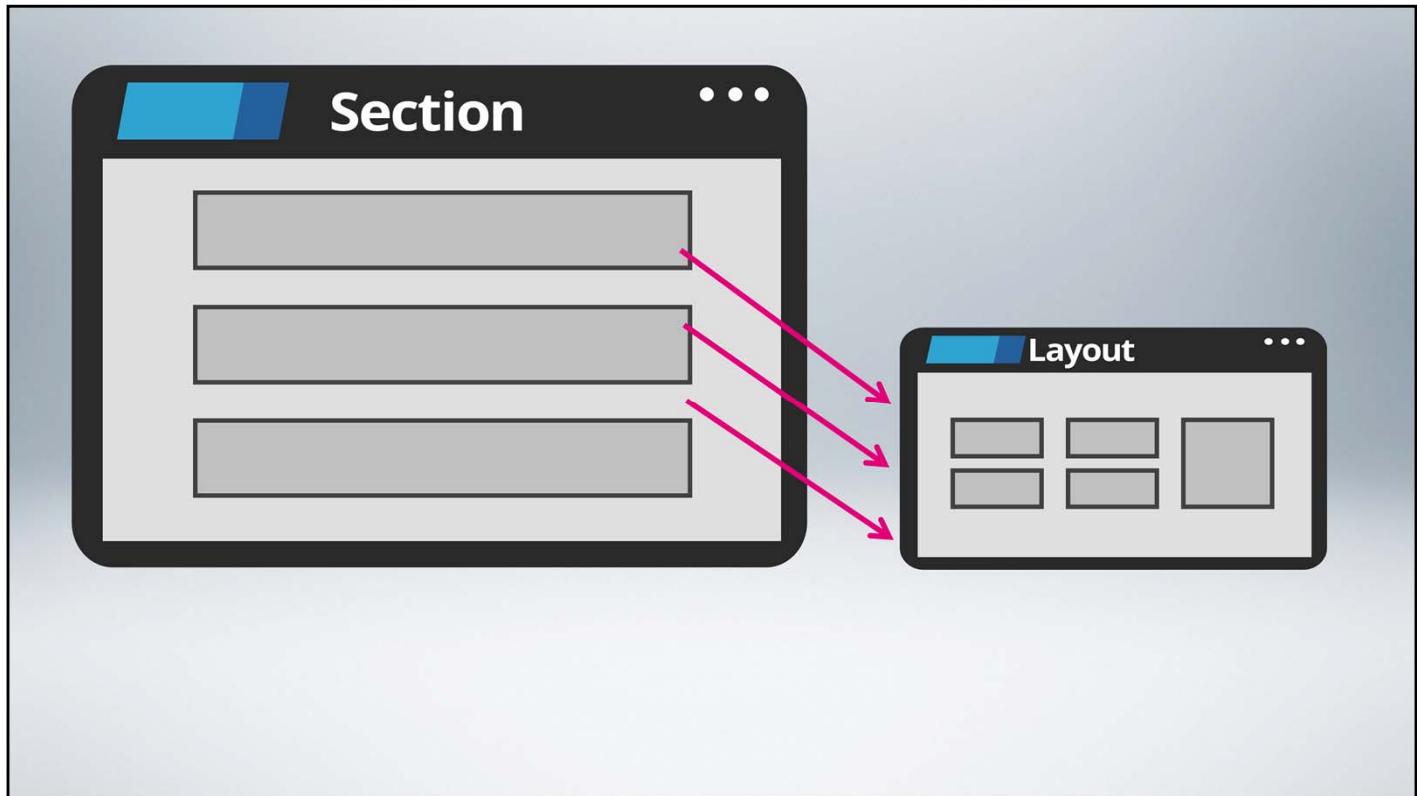


When the flow reaches an assignment shape it waits for the user to perform an action.

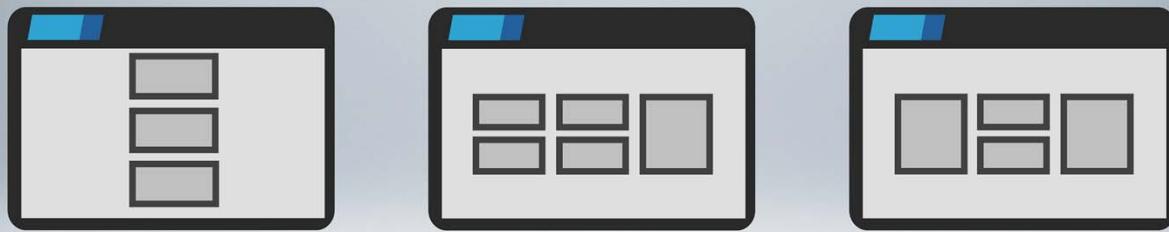
This shape references a standard harness named perform

and the connector coming out of the assignment shape references a flow action.

Both the perform harness and the flow action reference sections that render the content that we see in that screen.



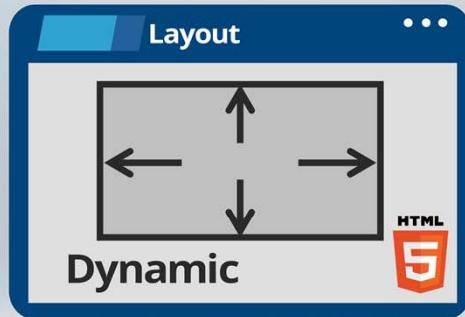
A Section is where we add the properties or the data elements that we want to display at runtime. Sections use a construct named layouts to group all these properties.



Layouts play a key role in aligning the properties, such as displaying them in a single column, two columns or three columns.



Dynamic Layouts



Fit multiple screen layouts Responsiveness



Supported in all browsers



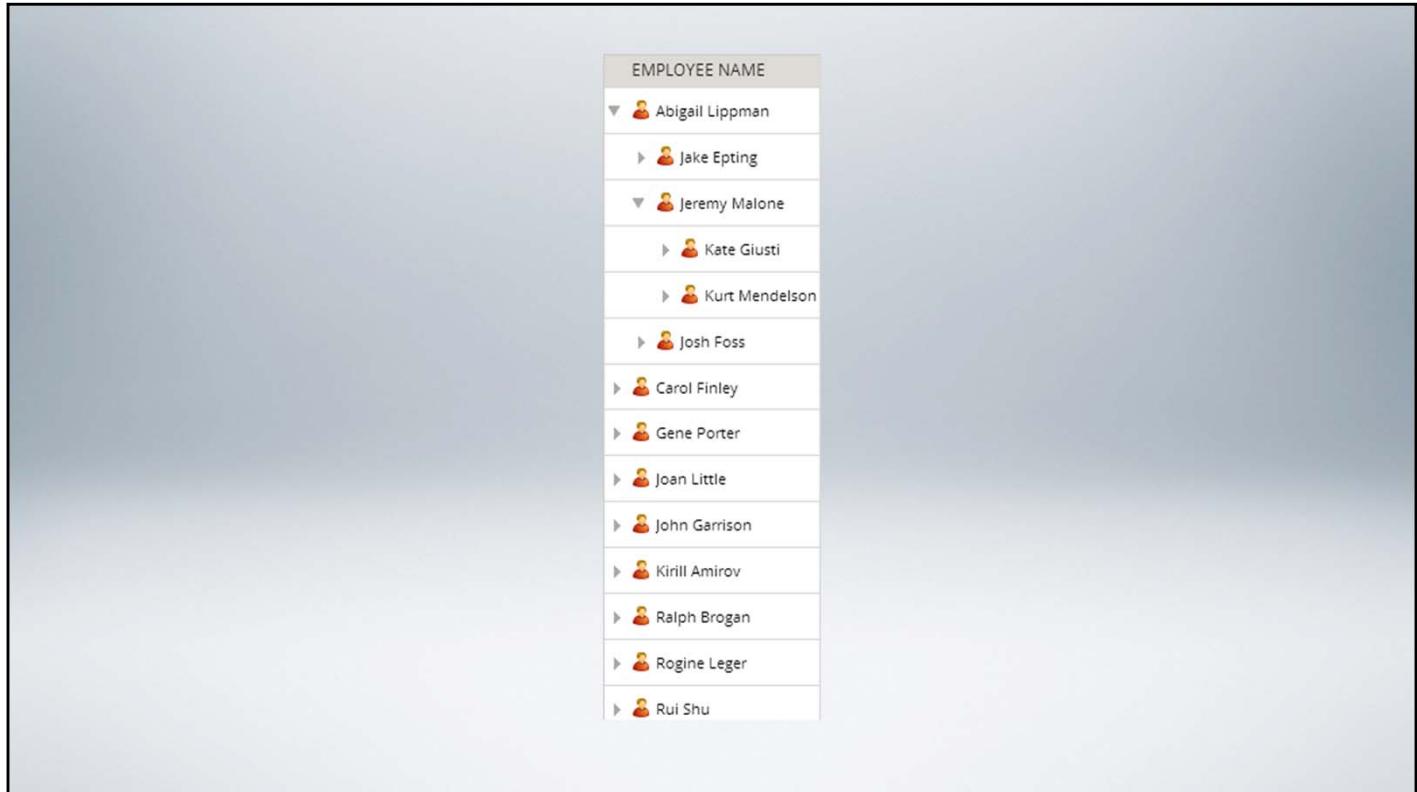
Pega 7 uses dynamic layouts that are very useful in making sure the application can be rendered in multiple browsers and can be used to automatically adjust to different screen resolutions.

Products								
	ProductName	Supplier	Category	UnitPrice	UnitsInStock	UnitsOnOrder	Discontinued	
Edit	Côte de Blaye	Aux joyeux ecclésiastiques	Beverages	263.5000	17	0	<input type="checkbox"/>	
Edit	Chartreuse verte	Aux joyeux ecclésiastiques	Beverages	18.0000	69	0	<input type="checkbox"/>	
Edit	Sasquatch Ale	Bigfoot Breweries	Beverages	15.0000	111	0	<input type="checkbox"/>	
Edit	Steeleye Stout	Bigfoot Breweries	Beverages	13.0000	20	0	<input type="checkbox"/>	
Edit	Laughing Lumberjack Lager	Bigfoot Breweries	Beverages	14.0000	52	0	<input type="checkbox"/>	
Edit	Queso Cabrales	Cooperativa de Quesos 'Las Cabras'	Dairy Products	22.0000	22	30	<input type="checkbox"/>	
Edit	Queso Manchego La Pastora	Cooperativa de Quesos 'Las Cabras'	Dairy Products	38.0000	86	0	<input type="checkbox"/>	
Edit	Escargots de Bourgogne	Escargots Nouveaux	Seafood	13.2500	62	0	<input type="checkbox"/>	
Edit	Aniseed Syrup	Exotic Liquids	Condiments	10.0000	13	70	<input type="checkbox"/>	
Edit	Chang	Exotic Liquids	Beverages	22.0000	16	41	<input type="checkbox"/>	

1 2 3 4 5 6 7 8

When we need to display a list that can expand depending on the associated data, then we use grids.

We saw three different flavors of grids – a Repeating grid that displays the list in rows and columns.

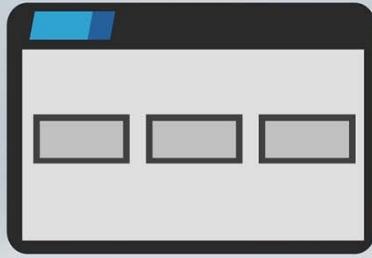
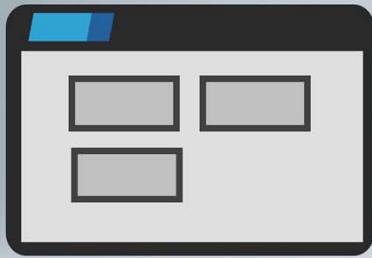


A Tree that is useful in displaying a hierarchical list

EMPLOYEE NAME	ROLE	EMPLOYEE TYPE	GENDER
▼  Abigail Lippman	Site Services	Full Time	Female
►  Jake Epting	Technical	Full Time	Male
▼  Jeremy Malone	Technical	Part Time	Male
►  Kate Giusti	Systems	Full Time	Female
►  Kurt Mendelson	Tech Services	Contract	Male
►  Josh Foss	Marketing	Full Time	Female
►  Carol Finley	Software	Full Time	Female
►  Gene Porter	Site Services	Part Time	Male
►  Joan Little	Site Services	Part Time	Female
►  John Garrison	Tech Services	Part Time	Male
►  Kirill Amirov	Finance	Full Time	Female
►  Ralph Brogan	Software	Full Time	Male
►  Rogine Leger	Technical	Full Time	Male
►  Rui Shu	Tech Services	Full Time	Female

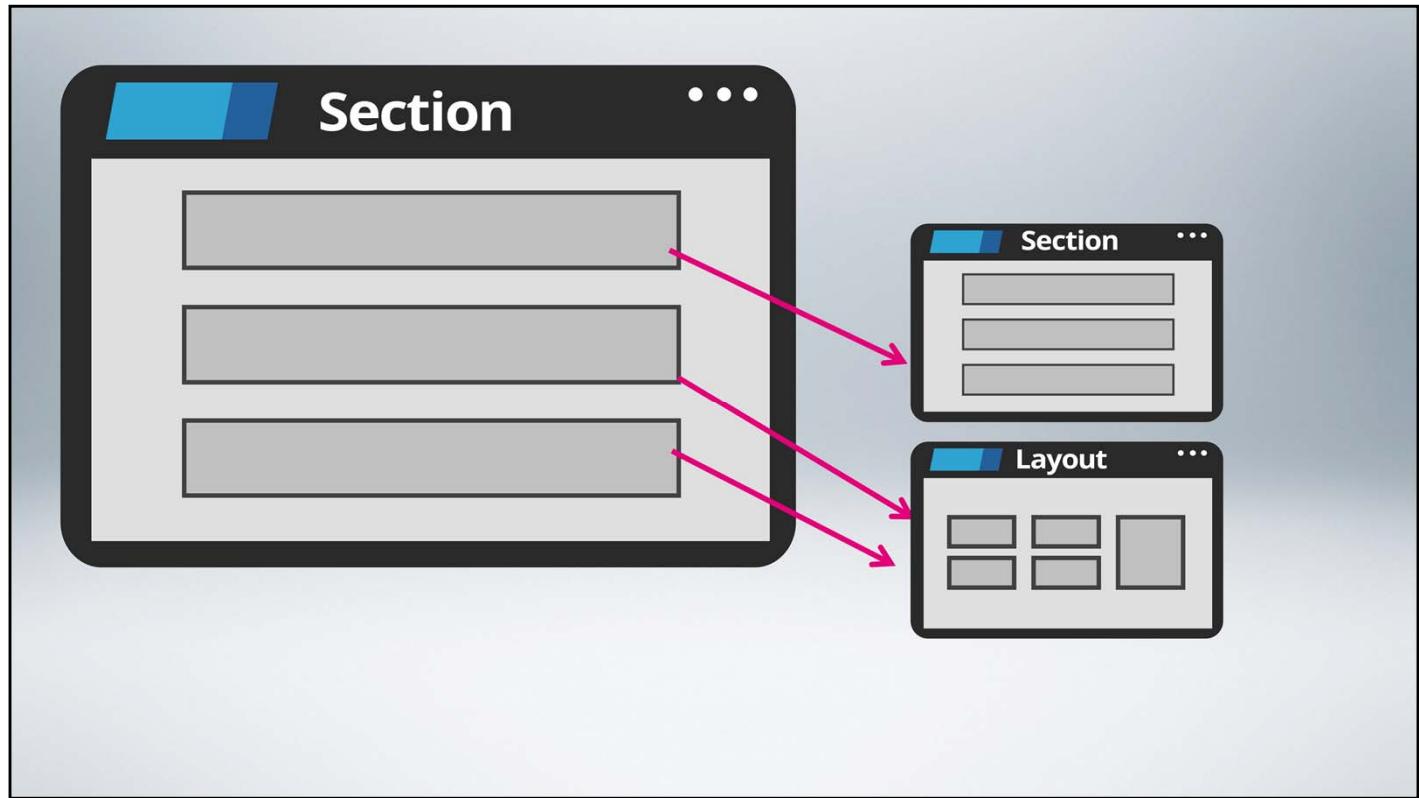
And Tree grids that can display the hierarchical list in a tabular format.

Repeating Layouts



Pagination
Filtering
Sorting

Repeating grids make configuration easier in terms of adding functionality like pagination, sorting, filtering without requiring us to write a lot of code to implement each of the functionality.



We can also reference another section inside a section. This allows us to save layouts as sections for reusability.



Controls

Control the display and presentation of data elements

Assigned based on property type

Formatting can be changed in the UI using properties panel

Support additional configuration

Lot of commonly used controls shipped in the product

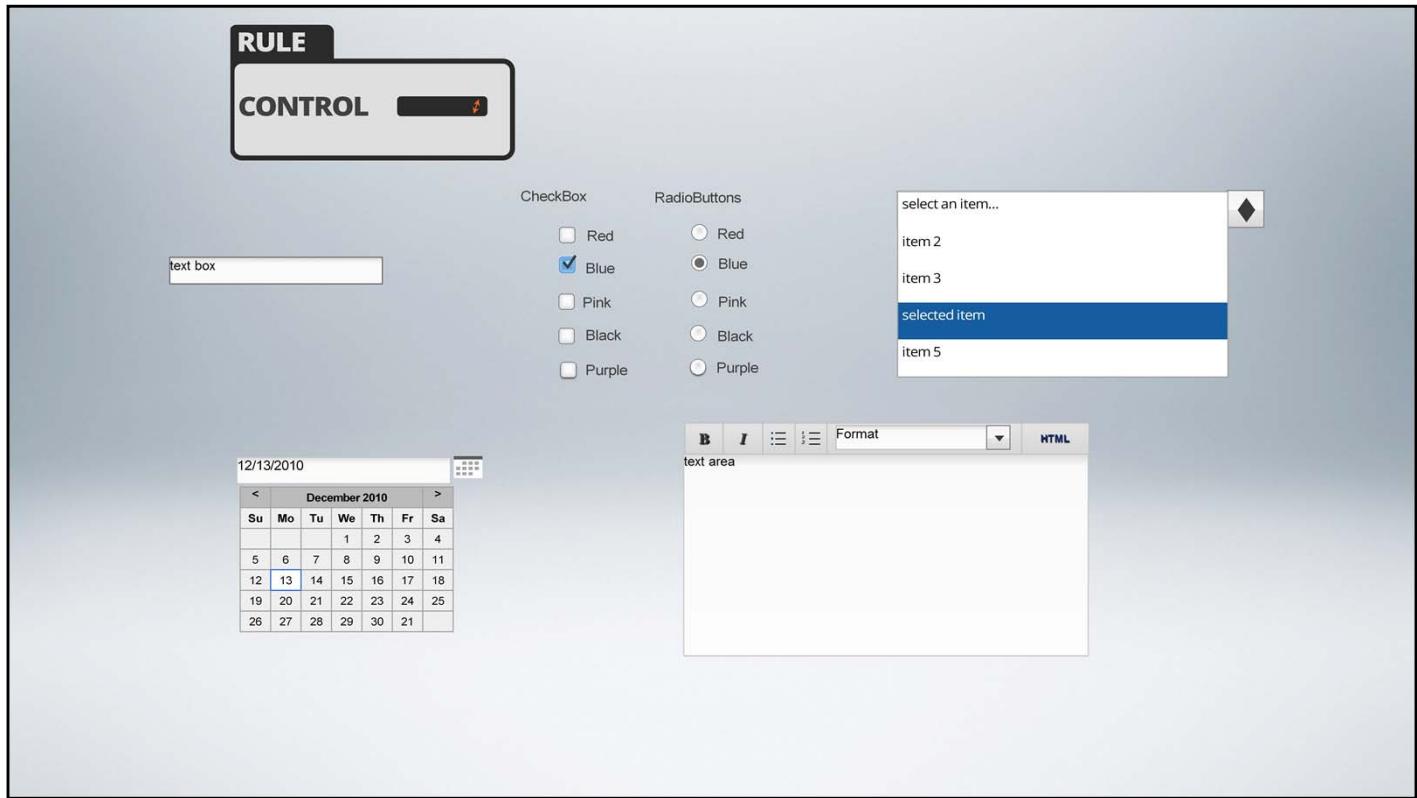
When properties are added in a layout, they are presented in a specific format- this is done by using control rules.

Controls are assigned to all properties by the system using its property type.

However we can override by default controls by specifying a control on the properties panel of the cell where the property is added.

When overriding a control on a cell within the section, we can apply additional configuration options. For example, we can configure the width and height of the cell, min and max characters allowed and so on.

There is a wider variety of standard controls that can be used to format data elements. Let's take a look at few of them.



We have text input to render a text box,

radio buttons,

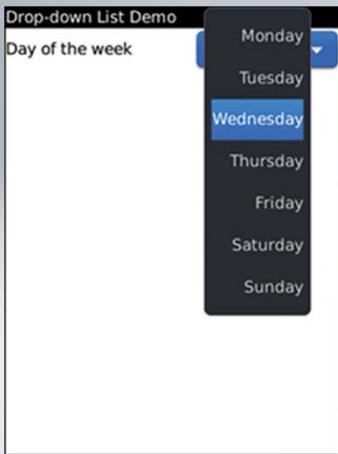
dropdown,

date control,

And the new Rich Text Editor which is useful in cases when we want to enter formatted text that enables the end users to enter a longer text.

List Based Controls

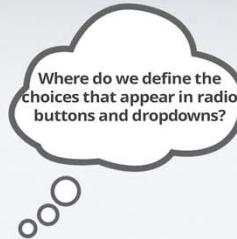
Dropdown



Radio buttons

Horizontal

Do you drink coffee every day?
 Yes No



Property form
(Local and prompt list)

Some controls are list based – meaning they display a list of choices for the users to select.

This list can be stored as part of the property using a local or a prompt list.

Action Controls



Icon

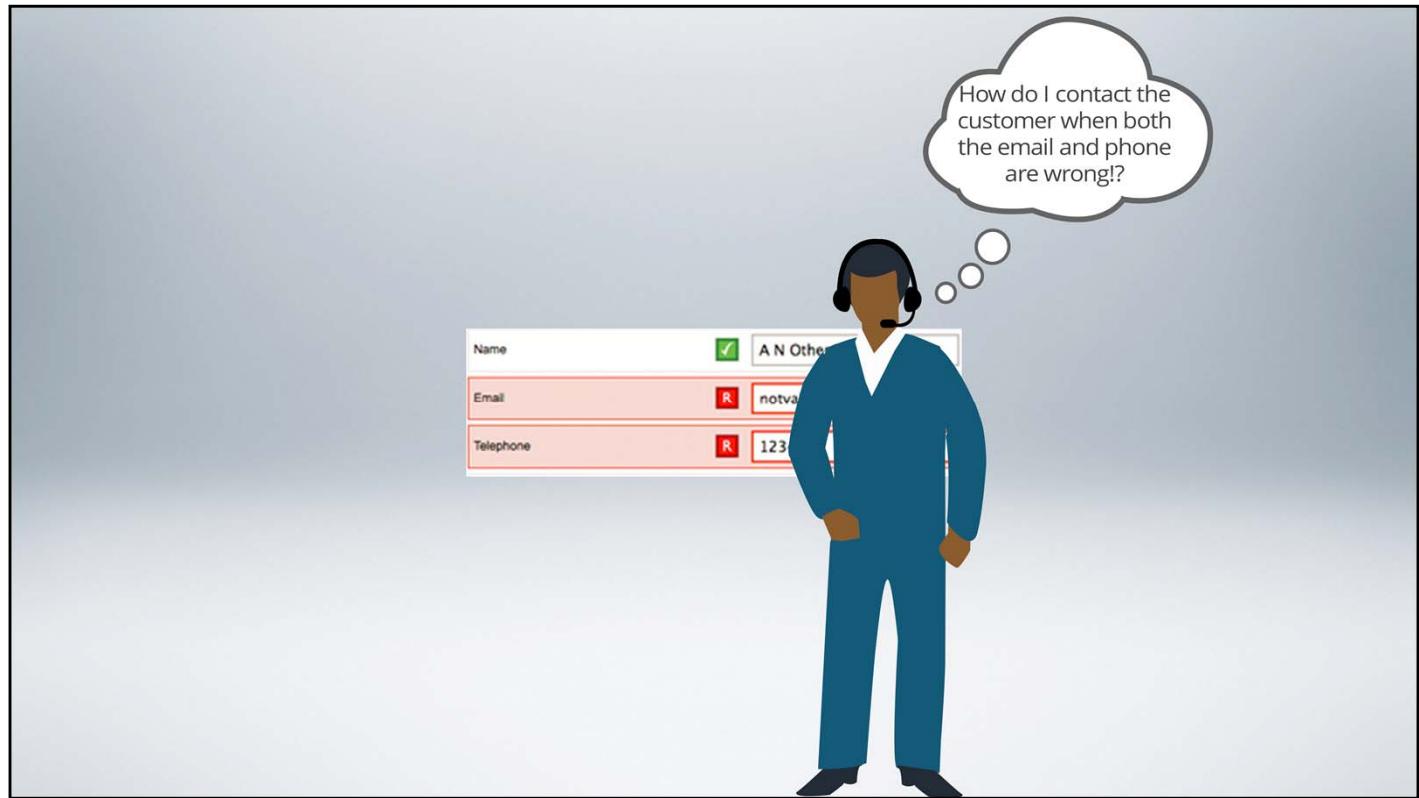


Link



Button

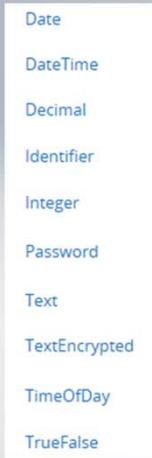
There are action based controls such as icons,
Links,
and buttons
which can be configured to perform a specific action like cancelling the changes.



Data elements play a critical role in case processing hence it is mandatory for us to validate the inputs entered by a user.

Properties and Controls

Define properties with appropriate data type



Use controls that guide the user to choose valid choices



One way to do this is by simply defining the properties with the appropriate data type and using a control that restrict users from entering invalid data. This goes a long way in ensuring the validity of captured data.

Validation Rules

Validate

Rule useful in building conditions

Edit Validate

Rule useful in making sure the pattern is matched

Pega provides two rule types – validate and edit validate for implementing the validation.

The Validate rule is useful in establishing conditions for a data element such as comparing against another property or a value.

Edit validate rules are used in cases where we want the values to match a specific pattern such as an email address.

We covered a lot of information in this lesson group. If you would like to review any of the concepts we learned, please return to the applicable lesson and watch it again.

Module 07: Accessing Data in the Application

This lesson group includes the following lessons:

- Introduction to Data Tables
- Accessing Data in the User Interface
- Creating a Dynamic User Interface
- Accessing External Systems
- Review of Accessing Data in the Application

Introduction to Data Tables

Data used in applications are stored in a dedicated database table outside the application. When Pega is used as a system of Record for such data we need to use data tables to store them.

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

- Describe data tables
- Know how a data table is created
- Populate the data tables in the application

Data Tables



A structure used to store reference data outside the application in a dedicated database table



CAN BE UPDATED WITHOUT IMPACTING CASES

Now that we have learned about using the User Interfaces, let's see how we can populate data for these data elements. A data table is one way we can store data that we use in our user interfaces.

What is a data table? A data table is a structure that is useful in storing reference data outside the application using a dedicated database table.

The table can be updated periodically, without impacting the business process.

Product Catalog

ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder
Chai	10 boxes x 20 bags	18,0000	39	0
Chang	24 - 12 oz bottles	19,0000	17	40
Aniseed Syrup	12 - 550 ml bottles	10,0000	13	70
Chef Anton's Cajun	48 - 6 oz jars	22,0000	53	0
Chef Anton's Gumbo	36 boxes	21,3500	0	0
Grandma's Boysenberry	12 - 8 oz jars	25,0000	120	0
Uncle Bob's Organic	12 - 1 lb pkgs.	30,0000	15	0
Northwoods Cranberry	12 - 12 oz jars	40,0000	6	0
Mishi Kobe Niku	18 - 500 g pkgs.	97,0000	29	0
Ikura	12 - 200 ml jars	31,0000	31	0
Queso Cabrales	1 kg pkg.	21,0000	22	30
Queso Manchego La	10 - 500 g pkgs.	38,0000	86	0
Konbu	2 kg box	6,0000	24	0
Tofu	40 - 100 g pkgs.	23,2500	35	0
Genen Shouyu	24 - 250 ml bottles	15,5000	39	0

Let's look at a couple of examples of data that an application can use for reference. A list of all products sold by the company which has the product name, quantity per unit, unit price, units in stock, etc.

Supplier List

Company Name	Contact Name	Contact Title	Address	City	Region	Postal Code	Country	Phone	Fax
Exotic Liquids	Charlotte Cooper	Purchasing Manager	49 Gilbert St.	London	n/a	EC1 4SD	UK	(171) 555-2222	n/a
New Orleans Cajun Delights	Shelley Burke	Order Administrator	P.O. Box 78934	New Orleans	LA	70117	USA	(100) 555-4822	n/a
Grandma Kelly's Homestead	Regina Murphy	Sales Representative	707 Oxford Rd.	Ann Arbor	MI	48104	USA	(313) 555-5735	(313) 3349
Tokyo Traders	Yoshi Nagase	Marketing Manager	9-8 Sekimai Musashino-shi	Tokyo	n/a	100	Japan	(03) 3555-5011	n/a
Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra	Export Administrator	Calle del Rosal 4	Oviedo	Asturias	33007	Spain	(98) 598 76 54	n/a
Mayumi's	Mayumi Ohno	Marketing Representative	92 Setsuko Chuo-ku	Osaka	n/a	545	Japan	(06) 431-7877	n/a
Pavlova, Ltd.	Ian Devling	Marketing Manager	74 Rose St. Moonie Ponds	Melbourne	Victoria	3058	Australia	(03) 444-2343	(03) 444-6588
Specialty Biscuits, Ltd.	Peter Wilson	Sales Representative	29 King's Way	Manchester	n/a	M14 9SD	UK	(161) 555-4448	n/a
PB Knäckebrot AB	Lars Peterson	Sales Agent	Kaladagatan 13	Göteborg	n/a	S-345 67	Sweden	031-987 65 43	031-987 65 91
Refrescos Americanas LTDA	Carlos Diaz	Marketing Manager	Av. das Americanas 12.890	Sao Paulo	n/a	5442	Brazil	(11) 555 4640	n/a

A list of all suppliers from whom the purchase can be made, which has company name, contact information, address, etc.

Product Catalog					Supplier List									
ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	Company Name	Contact Name	Contact Title	Address	City	Region	Postal Code	Country	Phone	Fax
Chai	10 boxes x 20 bags	18,0000	39	0	Exotic Liquids	Charlotte Cooper	Purchasing Manager	49 Gilbert St.	Boston	n/a	EC1 4SD	UK	(171) 555-2222	n/a
Chang	24 - 12 oz bottles	25,000	17	40	New Orleans Cajun Delights	Shelley Burke	Order Administrator	P.O. Box 78934	New Orleans	LA	70117	USA	(100) 555-4822	n/a
Aniseed Syrup	12 - 550 ml bottles	10,0000	13	70	Grandma Kelly's Homestead	Regina Murphy	Sales Representative	707 Oxford Rd.	Ann Arbor	MI	48104	USA	(313) 555-5735	(313) 555-3349
Chef Anton's Cajun Gumbo	48 - 6 oz jars	22,0000	53	0	Tokyo Traders	Yoshi Nagase	Marketing Manager	9-8 Sekimai Musashino-shi	Tokyo	n/a	100	Japan	(03) 3555-5011	n/a
Chef Anton's Gumbo	36 boxes	21,3500	0	0	Cooperativa de Quesos 'Las Cabras'	Antonio del Valle Saavedra	Export Administrator	Calle del Rosal 4	Oviedo	Asturias	33007	Spain	(98) 598 0254	n/a
Grandma's Boysenberry Spread	12 - 8 oz jars	25,0000	120	0	Mayumi's	Mayumi Ohno	Marketing Representative	92 Setsuko Chuo-ku	Osaka	n/a	545	Japan	(06) 431-7877	n/a
Uncle Bob's Organic Salsa	12 - 1 lb pkgs.	30,0000	15	0	Pavlova, Ltd.	Ian Devling	Marketing Manager	74 Rose St.	Melbourne	Victoria	3058	Australia	(03) 444-2343	(03) 444-6588
Northwoods Cranberry Sauce	12 - 12 oz jars	40,0000	6	0	Specialty Biscuits, Ltd.	Peter Wilson	Sales Representative	Moonee Ponds	Manchester	n/a	M14 6SD	UK	(161) 555-4448	n/a
Mishi Kobe Niku	18 - 500 g pkgs.	97,0000	29	0	PD Knäckebrot AB	Lars Peterson	Sales Agent	Kaloadagatan 13	Göteborg	n/a	S-345 67	Sweden	031-987 65 43	031-987 65 91
Ikura	12 - 200 ml jars	31,0000	31	0	Refrescos Americanas LTDA	Carlos Diaz	Marketing Manager	Av. das Americas 12.890	Sao Paulo	n/a	5442	Brazil	(11) 555 4640	n/a
Queso Cabrales	1 kg pkg.	21,0000	22	30										
Queso Manchego La Mancha	10 - 500 g pkgs.	38,0000	86	0										
Konbu	2 kg box	6,0000	24	0										
Tofu	40 - 100 g pkgs.	23,2500	35	0										
Genen Shouyu	24 - 250 ml bottles	15,5000	39	0										

What do these examples have in common? They are a list of items, the item has attributes, their values change periodically and they are used as a reference data for the application

Data Table Structure

The screenshot shows a database interface for the 'PR_OTHER' table. The table has 13 columns with the following details:

COLUMN_NAME	DATA_TYPE	TYPE_NAME	COLUMN_SIZE	BUFFER_LENGTH	DECIMAL_DIGITS
PXCREATEDATETIME	93 DATE		7	0	(null)
PXCREATEOPNAME	12 VARCHAR2		128	0	(null)
PXCREATEOPERAT...	12 VARCHAR2		128	0	(null)
PXCREATESYSTEMID	12 VARCHAR2		32	0	(null)
PXINSNAME	12 VARCHAR2		128	0	(null)
PXOBJCLASS	12 VARCHAR2		96	0	(null)
PXUPDATEDATETIME	93 DATE		7	0	(null)
PXUPDATEOPNAME	12 VARCHAR2		128	0	(null)
PXUPDATEOPERAT...	12 VARCHAR2		128	0	(null)
PXUPDATESYSTEMID	12 VARCHAR2		32	0	(null)
PYLABEL	12 VARCHAR2		64	0	(null)
PZINSKEY	12 VARCHAR2		255	0	(null)
PZPVSTREAM	2004 BLOB		4000	0	(null)

To save such a list of items we use data tables. Data tables use a proprietary database table structure in terms of columns. These tables are stored in the same database where the Pega 7 application is installed.

Product Catalog

Products		
Column Name	Data Type	Allow Nulls
ProductID	int	<input type="checkbox"/>
ProductName	nvarchar(40)	<input type="checkbox"/>
SupplierID	int	<input checked="" type="checkbox"/>
CategoryID	int	<input checked="" type="checkbox"/>
QuantityPerUnit	nvarchar(20)	<input checked="" type="checkbox"/>
UnitPrice	money	<input checked="" type="checkbox"/>
UnitsInStock	smallint	<input checked="" type="checkbox"/>
UnitsOnOrder	smallint	<input checked="" type="checkbox"/>
ReorderLevel	smallint	<input checked="" type="checkbox"/>
Discontinued	bit	<input type="checkbox"/>

Depending on the list, these tables will have additional columns so in the previous example we saw, in the case of products - product name, quantity on hand, unit price, etc. will be saved as separate columns. Even though data tables are saved as database tables, they can only be edited inside PRPC. We will see how to add, modify or delete content in each table in a little while.

Demo



Know How a Data Table is Created

Now let's see how we can create the data tables. The data table can be quickly created using the data table wizard. The wizard automatically creates all the rules that are required for data tables. The landing page displays all the existing data tables. To add a new one lets click the button that starts the wizard. The Class name field is required to group the rules created for data tables. The data table must be created in the data layer of the class structure. In this example, we are actually creating it as part of the classname that ends with Int- instead of Data-. We will learn more about class structures in the System Architect Essentials II course. We are creating a data table for shipping rates, so let's just enter the class name here.

We can add a description for the data table so it's easier to know what it is used for. The Derives from field indicates the parent for the class that we need to create, identifying the parent helps in inheritance in terms of using base rules shipped in product. The Create Data Pages flag controls the creation of data pages which are useful in accessing the results of the data table. We will learn more about data pages and how they can be accessed in the application in the next lesson. Notice it creates two data pages. The lookup data page populates the page with values specific to a particular line item or the product key. In the example, we saw before the lookup data page loads a specific product page when the corresponding product ID is entered.

The list data page is used in rendering the entire list. This is used to display all products sold by the company in a repeating grid or to select a specific product in a dropdown. As the name suggests, Create Dedicated Database table creates a new database table for the data table in the same database where Pega 7 is installed. It is always important to do this so the system can keep the data table records separate from each other. At times, developers may not have access to create tables directly. In such cases this field cannot be enabled and it is up to the Lead System Architects (LSAs) to hand over the script to the DBAs for creating them. If this flag is disabled, PRPC uses a table named PR_Other to save the data instances.

Continuing along, let's add the columns that are needed for this specific data table. We see that it already uses the label field and also opened up an editable field for product key. In this case let's enter shippingrateid which is the key for this field. Let's add one more column— rate and define it as a decimal. Once defined, let's accept the default choices in the RuleSet field and click Generate.

The wizard generates all the rules that are required to render the data table in both design and runtime. These are the list of rules created by the wizard. It creates properties for each column we created. It also creates other rules such as the data transform to set defaults, a validate rule to validate the data associated with the properties, creates list rules to display the data table in read only and edit mode and lastly it creates the two data pages that we defined earlier.

In addition to data pages, it also creates a report definition to display the results. If we have a case when data pages are not created this report definition can be used to display the results.

Demo



Populate the Data Table

After the wizard is run, we can see the newly added table in the list, it displays 0 in the row count.

Let's create some data in the table. Click the icon to open the table.

To add a row we click the Add a row icon, and we see the text boxes. After entering the values, click the Save icon to save the instances. If we enter a text value in the rate field PRPC throws an error. Once we make entries to the table, we can continue adding additional rows. We could use this interface to populate rows and PRPC also allows us to support writing procedural rules to populate this table.

In this lesson, we learned about data tables, why they are needed in applications, how to create them and how to populate them. We will learn about accessing data using data pages in the next lesson.

Exercise: Create a new Data Table

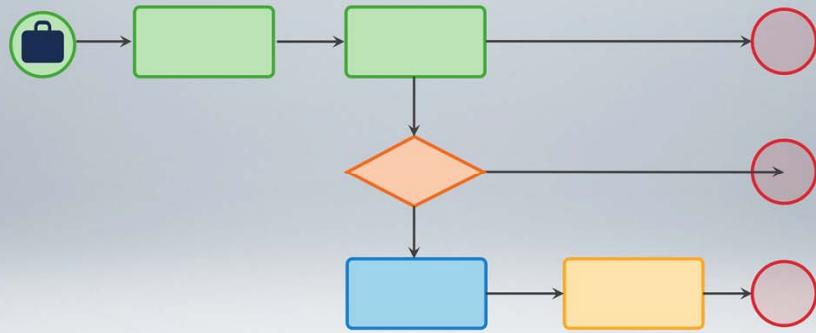


Accessing Data in the User Interface

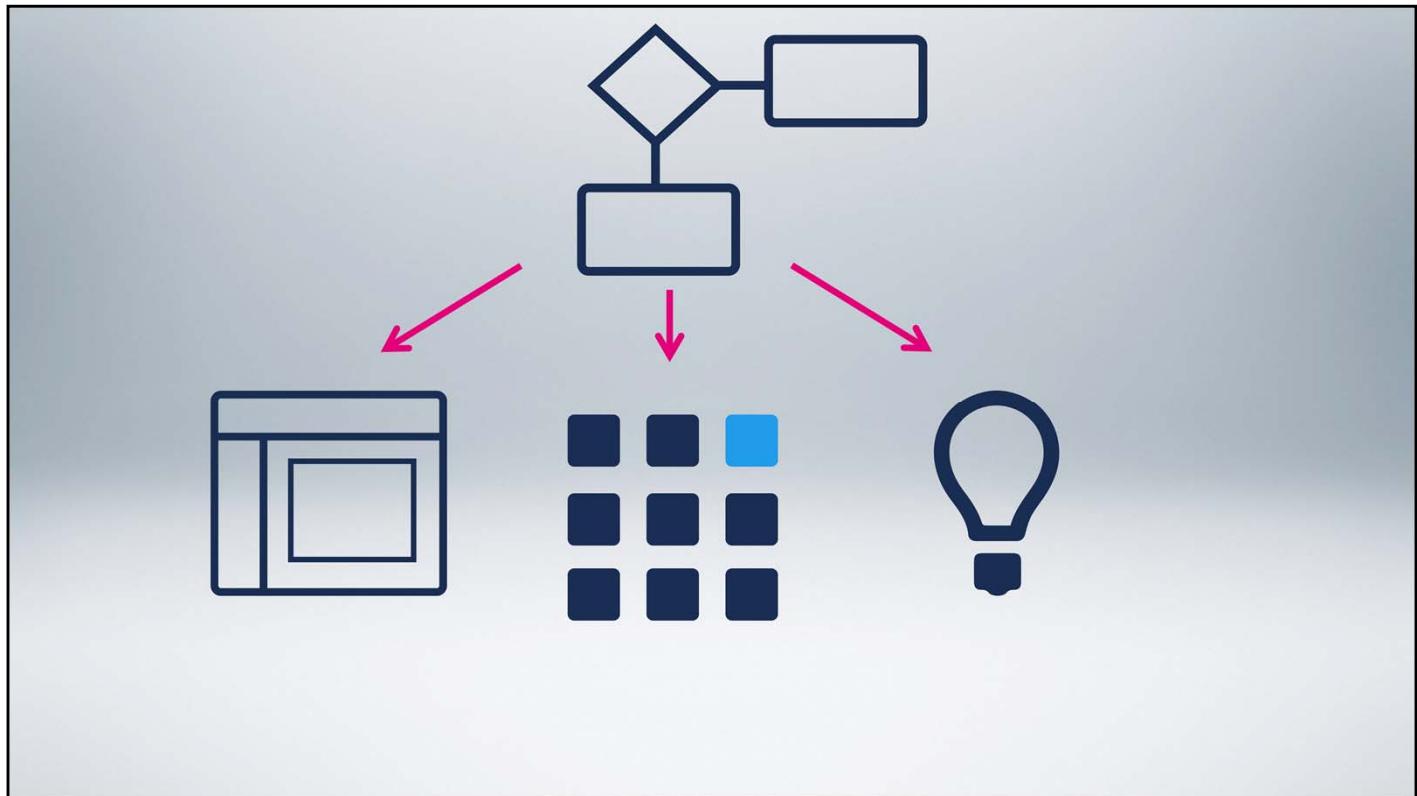
Applications require access to reference data for displaying them in UI screens. Data Pages play an important role in application development to define data as a separate layer from the case processing logic.

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

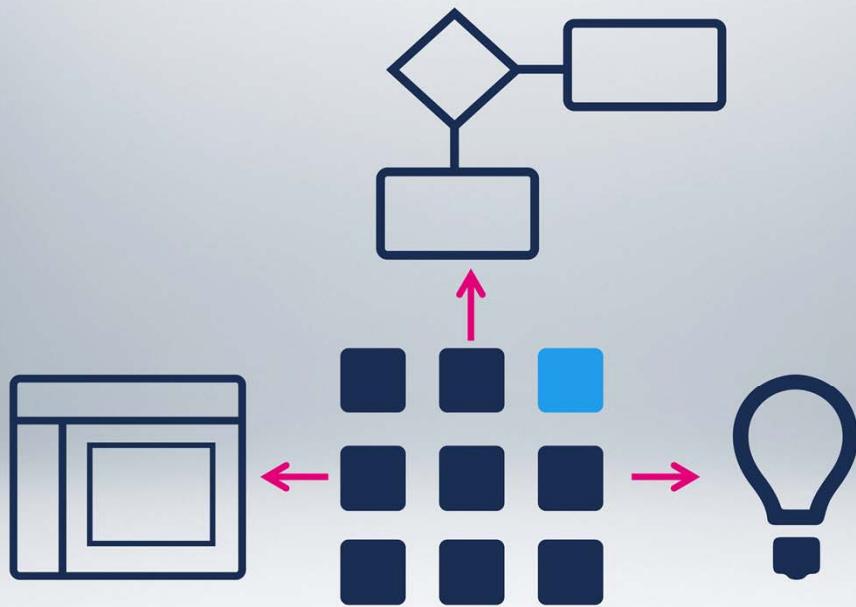
- Learn how data can be stored for access
- Access data from a data page
- Source the controls using a data page



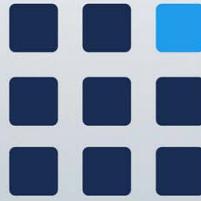
When a case is created, it starts a flow (process) and the flow decides how the case will be processed .



Processes require UI, Logic and Data- all these are built using rules in a Pega application.



Data is a key component and is used by the Process, UI and Logic. In this lesson we will focus on how the data is accessed by properties that display in the UI screens.



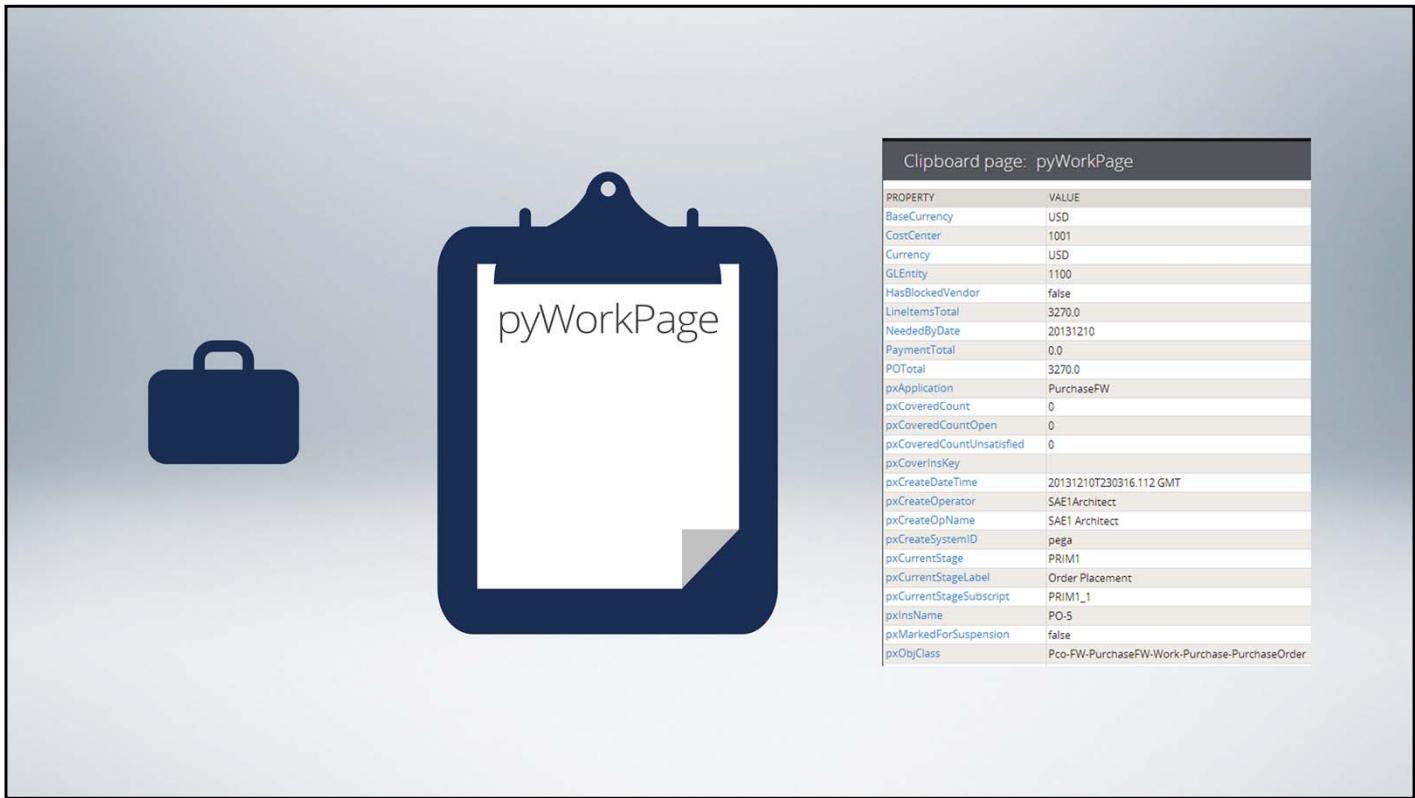
Applications request data before displaying the data elements

Data is stored in clipboard



When we present a case to the user, we want data to populate in the properties used in UI screens to help the user perform an action. For example, when a customer calls, the customer support representative handling the case should be able to see the account details for that customer.

Data needed by applications is stored in the clipboard.



We learned in one of the earlier lesson, that when a case is created, it creates a named page called `pyWorkPage` in memory, which can be seen in the clipboard.

When users work on the case, the values entered on data elements are saved as part of this page.



When a case is being processed, often we require additional data other than the ones supplied by the user or the ones that are saved in a case.



This is a list of products.

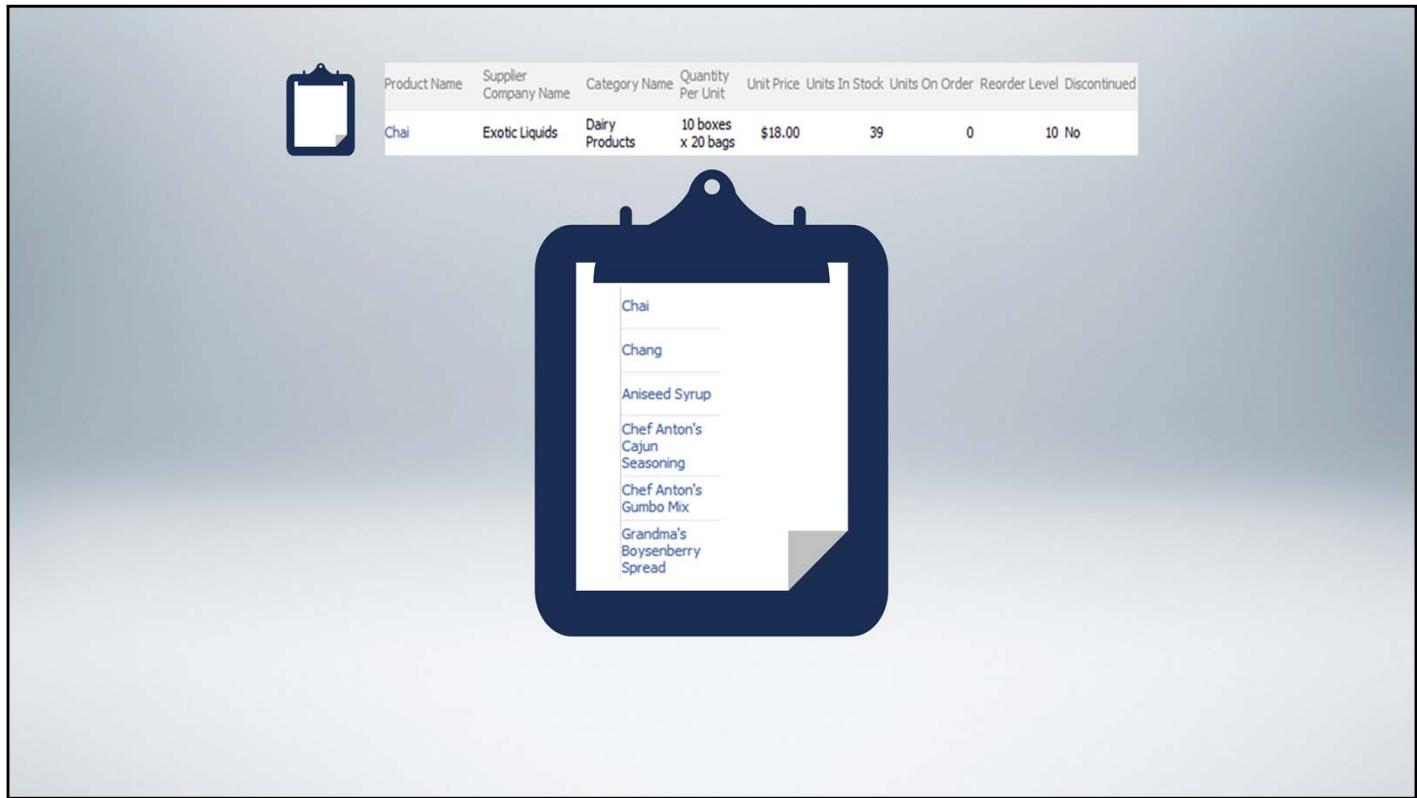
Product Name	Supplier Company Name	Category Name	Quantity Per Unit	Unit Price	Units In Stock	Units On Order	Reorder Level	Discontinued
Chai	Exotic Liquids	Dairy Products	10 boxes x 20 bags	\$18.00	39	0	10	No
Chang	Exotic Liquids	Beverages	24 - 12 oz bottles	\$19.00	17	40	25	No
Aniseed Syrup	Exotic Liquids	Condiments	12 - 550 ml bottles	\$10.00	13	70	25	No
Chef Anton's Cajun Seasoning	New Orleans Cajun Delights	Condiments	48 - 6 oz jars	\$22.00	53	0	0	No
Chef Anton's Gumbo Mix	New Orleans Cajun Delights	Condiments	36 boxes	\$21.35	0	0	0	Yes
Grandma's Boysenberry Spread	Grandma Kelly's Homestead	Condiments	12 - 8 oz jars	\$25.00	120	0	25	No
Uncle Bob's Organic Dried Pears	Grandma Kelly's Homestead	Produce	12 - 1 lb pkgs.	\$30.00	15	0	10	No
Northwoods Cranberry Sauce	Grandma Kelly's Homestead	Condiments	12 - 12 oz jars	\$40.00	6	0	0	No
Mishi Kobe Niku	Tokyo Traders	Meat/Poultry	18 - 500 g pkgs.	\$97.00	29	0	0	Yes
Ikura	Tokyo Traders	Seafood	12 - 200 ml jars	\$31.00	31	0	0	No

< Previous | Page: 1 2 3 4 5 6 7 8 | Next >

Items per page: 10, 15, 20, 25 | Showing 1-10 of 77 items | 

For example- when a user is working on a purchase request, the item selection screen should list all the items that a user can order. The selected item is saved as part of the case and the list of all items comes from the data component.

Also when an item is selected the system should display all the details corresponding to that item.

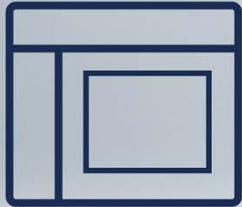


To display both the list of items and the details for the selected item we need to have the product list and product details in the clipboard.

Pega uses Data Pages to store the data that is required by the application. Data Pages are saved in memory and can be seen in the clipboard.



Data Pages



Separate Data from Process, UI and Logic

Data pages play a key role in data management and are extremely powerful in terms of providing access to the data that is required by users while working on a case.

What does a data page offer? For starters it separates the definition of the data from the Process, UI and logic so any changes in the definition do not impact the process, UI or logic. Let's see what gets saved as part of the data page definition.



Data Pages

Source from where data is read

Data table, external system

Uses lookup, data transform or report definition

It contains the details on the source of the data and the means to collect data-whether it's accessed from a data table or coming from an external source and also whether it uses a lookup or uses a rule like report definition or an activity or data transform.



**Data
Pages**

Data Structure



page list



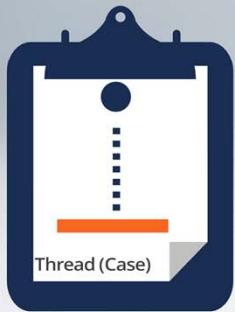
page

It contains the data structure of the data page – whether it's a page list or a single page.



Data Pages

Scope



Thread (Case)

Requestor
(user session)

Node (across all sessions)

It defines the scope where the results are available – whether the data page is accessible only on a case level, or on the entire user session or across all user sessions



Refresh Strategy



and lastly when the data gets refreshed. We will learn about refresh strategy in the advanced courses.



Data Pages

Separate Data from Process, UI and Logic

- Source from where data is read
 - Data table, external system
 - Uses lookup, , data transform or report definition
- Data structure
 - Pagelist or page
- Scope
 - Thread(Case), requestor(user session) and node (across all sessions)
- Refresh strategy

Defining all these in a data page offers so much flexibility in terms of designing the application especially UI screens. The UI screen need not be designed to suit where the data is coming from or check whether the data is stale or if the page exists. The UI is not impacted by the changes that a data page has to go through.



Data Pages



- Data pages are invoked declaratively
 - System looks for the page in memory
 - If not in memory, system must load the page
 - Page, once loaded, exists in memory until the scope expires
 - Data is refreshed depending on the refresh strategy

How does a data page get accessed? Data pages are invoked declaratively, when a rule accesses a data page, the system looks for that data page in memory, if the page does not exist in memory, the system looks for the data page definition and then loads the data page in memory.

The data page exists in memory for the next access, so when we access it for the second time the system picks the data page that is already loaded in memory. The data page remains in memory as long as the scope is applicable, so if the scope is REQUESTOR, the page exists until the user session expires.

Depending on the refresh strategy, the data gets updated on the data page automatically.

Demo



Data Pages

Let's look at a data page that is used in our base application.

The data pages belonging to the application can be seen in the data explorer. We learned in the previous lesson that a data table wizard creates data pages. Let's take a look at the Shipping Rates data table now. here

These are created by data table wizard, let's open the first one.

This data page is a pagelist. The data page cannot be modified in the user interface since it comes from a data table.

The scope for this data page is defined as thread which means it can be accessed only on a specific case running on that thread – since shipping rates are typically used only in the Purchase Order case.

Look at the Data Sources area – it uses a report definition rule to get the list of shipping rates defined as part of the data table.

Now, let's look at the other data page. The structure is page, so the results of a data page is always one page. The source field uses a different value. The scope for this field is also Thread, let's open up the countrylist data page. The country list data page is a node level data page which means the data page is loaded in the node and is shared with all users logging on to that node. This is because the country list does not differ by the operator. Similarly, if we look up the operator data page we see that this page is set to requestor, because each operator has different data.

The refresh strategy is set in the load management tab, the operator data is set as 1 day. Some data like Program Fund is set to refresh a little quicker than a day since program fund can be created by operators.

The program fund is set to 1 hour since all program funds require an approval and 1 hour is more realistic to get all the changes made in the interim. However, there are other pages which require more current data.

And we need the refresh to occur much faster.

In the advanced courses, we will learn more about data pages refresh strategies, data load strategies and other information to help you.



Data Pages in the UI



Repeating Grids

If the structure is a list

CheckBox

- | | |
|--|---------------------------------------|
| <input type="checkbox"/> Red | <input checked="" type="radio"/> Red |
| <input checked="" type="checkbox"/> Blue | <input checked="" type="radio"/> Blue |
| <input type="checkbox"/> Pink | <input type="radio"/> Pink |
| <input type="checkbox"/> Black | <input type="radio"/> Black |
| <input type="checkbox"/> Purple | <input type="radio"/> Purple |

RadioButtons

select an item...

- item 2
- item 3
- selected item**
- item 5

List based controls such as dropdown or radiobutton

Now that we have seen how a data page is defined, let's take a quick look at how can we use the data page in our UI screens.

If the data page is a list, and it displays a list of results,

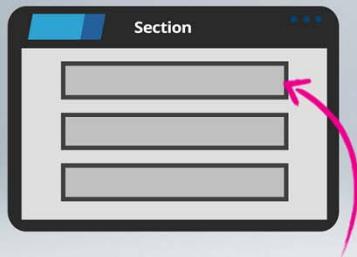
we can use the data page in a repeating grid

and also in a list based control such as dropdowns or radio buttons.



Data Pages in the UI

If the structure is a page



An embedded section using the Data page



A field cell using the reference
D_Page.propname

If the data page is a page structure,

we can display the data page in a section using a page property as a reference either including the page as a section or referencing a single property in the data page.

Demo



Controls

Let's look at this by using the vendor maintenance case,

in here we are editing an existing vendor named CDW and the section below that displays the information about the vendor. Let's look at the section rule that is written to render this screen. The section include EditVendorDetails displays the details of the selected vendor. The section include uses a Data page as the page context and is pointing to D_Vendor. The Data page uses a parameter field VendorID which is referencing a property name.

This property VendorSearchValue is selected by the user at runtime in this field. Once the user selects CDW in this field, this value is passed to the data page and it loads the details on this section.

The list based controls like dropdown or radio buttons can get data from the data page. The country field has a drop down and is configured to read from a data page. The List source field uses a data page and the dropdown allows referencing different properties for the value and what the user sees at runtime. When the screen is loaded, it looks for the data page and if it doesn't exist in memory it loads the data page in memory.

Data pages are very useful and are very essential to know how to access them in user interfaces. Let's do a couple of exercises to learn about accessing data from the data pages.

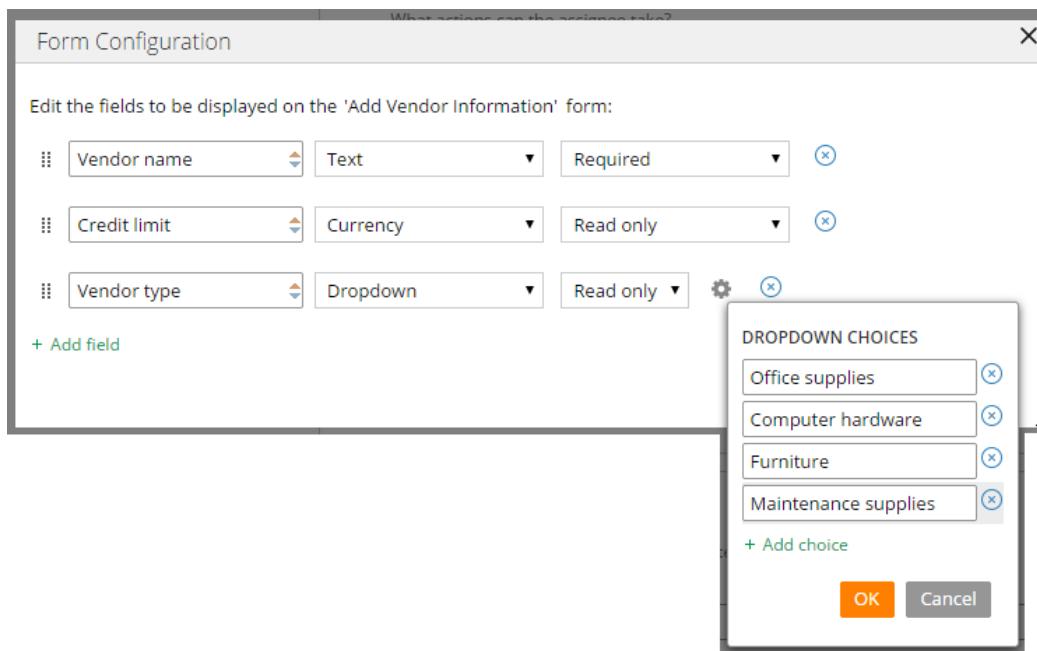
Exercise 1: Populate Dropdown from a Data Page
Exercise 2: Populate a Data Page in an Embedded Section
Exercise 3: Auto Populate a Page Property



Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- Update to form configuration dialog dropdown field
 - Can specify list choices directly in a pop-up dialog



Creating a Dynamic User Interface

UI needs to be refreshed based on user selections or actions. Dynamic UI helps in rendering changed portions of screen at a quicker load without refreshing the whole screen.

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

- Create user interfaces that changes part of a screen with user selections
- Apply client side refresh when, visible when on an event
- Apply refresh when condition on a section using the expression builder

Dynamic UI



Impacts part or the whole screen

In this lesson we will learn about building highly dynamic user interfaces that help in delivering a good user experience.

A dynamic user interface may change portions or even the entire screen based on a specific event

- such as when the user clicks a button or

Dynamic UI



Values changed on a specific field

when the user provide inputs to a specific field. Let's look at some examples where a dynamic UI is helpful.

The screenshot shows a user interface for managing line items in a purchase request. At the top, there's a header with a downward arrow labeled "LINEITEM", a "Add Item" button with a plus sign, and a "Delete" button with a minus sign. Below this is a table with four columns: PRODUCT, QUANTITY, PRICE, and SUBTOTAL. There are two rows of data:

	PRODUCT	QUANTITY	PRICE	SUBTOTAL
1	Laptop	3	200	600
2	Printer	1	80	80

Three specific cells are highlighted with pink circles: the "3" in the Quantity column for the Laptop row, the "200" in the Price column for the Laptop row, and the "600" in the Subtotal column for the Laptop row.

When line items are added or modified in a purchase request, we need to recalculate and display the new total on the screen before the user submits the request.

- Customer is single = collect personal information
- Customer is married = collect personal and spouse information

Name _____	
Address _____ City _____ Zip _____	
Birthdate _____ Age _____	Home phone _____ <input type="checkbox"/> *
Occupation_____	Business phone_____ <input type="checkbox"/> *
Marital Status _____	* check boxes if NOT okay to call or leave message
If married, spouse's name _____	Birthdate _____ Age _____
Spouse's occupation _____	Year married _____
Emergency Contact: Name _____	Relationship _____
Home phone _____ <input type="checkbox"/> *	Work phone _____ <input type="checkbox"/> *

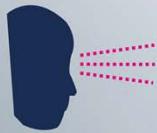
HIDE THESE FIELDS IF THE
CUSTOMER IS NOT MARRIED

When a customer applies for a loan and indicates they are married, they should be presented with additional fields to capture the spouse details.

Users want to see the changes while at the screen
before submitting for changes to take effect

LINEITEM					
		PRODUCT	QUANTITY	PRICE	SUBTOTAL
1	Laptop		2	400	800
2	Printer		1	100	100

A Dynamic UI is interactive immediately updating or modifying the screen based on a user action creating a positive, easy to use experience.



Visible when – hide/display fields



Refresh when – displays the changed values



Read only when

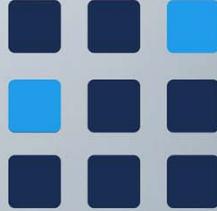


Disable when

Some examples of a Pega dynamic UI includes changing the screen to display or hide fields, displaying modified values of properties on the screen, changing the edit option of the fields, or disabling the properties based on a condition.

NEXT


esc



Click of a control Link, Button	Keyboard actions Enter, Esc, Key press, Key down	Property based Property value changes
---	--	---

This condition is triggered based on an event such as a click of a button or a link, keyboard actions like enter, esc, key press, key down, or a change in the value of a property.

Key Parts of Dynamic UI

Source (Controlling field)

NEXT



There are two key parts in creating a dynamic UI.

The Source or the controlling field – this is where the event is tied to.

This can be a button a user clicks

or a dropdown where a user makes a selection.

Controlling Field

Identify the field or action control

Choose an event

Click, Change, Hover, Focus

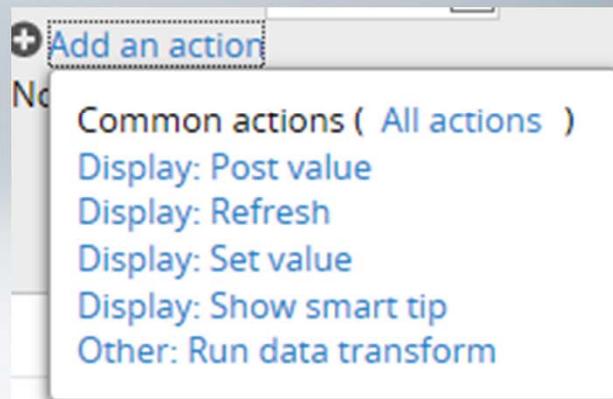
Keyboard actions

Mouse events	Keyboard events	Other events
Click	Enter key	Change
Double-click	Up key	Focus
Hover	Down key	
Right-click	Left key	
	Right key	
	Esc key	
	Tab key	
	Any key	

When defining a dynamic UI, the first step is to identify the source and select the event associated with it.

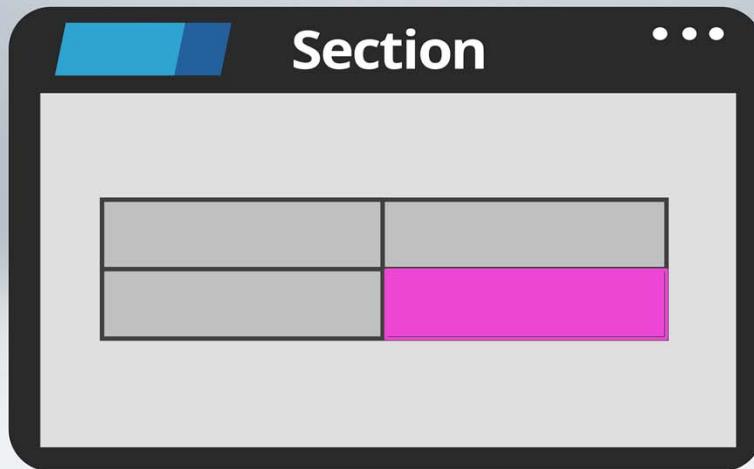
Controlling Field

Associate one or more actions



We also need to associate an action that can occur when the event happens – submit the value selected,
or refresh the section or part of a section.

Controlled Field



The second part is identifying the target fields or the fields that we want to be controlled by the source.

When the event defined on the source field is triggered, it performs the action associated with it and additionally it impacts the target identified which could be a cell, layout or a section embedded in another section.

Identify the controlled fields

Marital status – event is triggered

Name _____	_____
Address _____	City _____ Zip _____
Birthdate _____ Age _____	Home phone _____ <input type="checkbox"/> *
Occupation _____	Business phone _____ <input type="checkbox"/> *
Marital Status _____	* check boxes if NOT okay to call or leave message
If married, spouse's name, _____	Birthdate _____ Age _____
Spouse's occupation _____	Year married _____
Emergency Contact: Name _____	Relationship _____
Home phone _____ <input type="checkbox"/> *	Work phone _____ <input type="checkbox"/> *

THE MARKED LAYOUT MUST
BE IDENTIFIED AS THE TARGET
BASED ON MARITAL STATUS

When the event is triggered, the changes affect only the identified target, so we need to identify them correctly. For example, if the event is triggered on marital status, we want the target to be the layout containing spouse information so that it shows or hides based on the selection.

Benefits of Dynamic UI



Decrease the number of full page refreshes



Increase the performance



Compress more functionality into single page



Create a compelling modern user experience

Before we see how we can create dynamic user interface let's take a brief look at some benefits we can get

A dynamic user interface decreases the need to refresh a whole page to see changes.

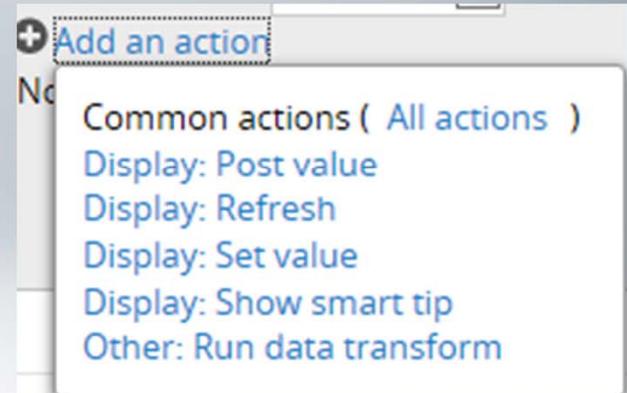
This in turn, increases the performance of the application in terms of quicker load of UI screens.

We could show more data elements in the UI screens since not all of them are displayed at the same time.

And more importantly, we can create a modern engaging user experience.

Refresh a section

Mouse events	Keyboard events	Other events
Click	Enter key	Change
Double-click	Up key	Focus
Hover	Down key	
Right-click	Left key	
	Right key	
	Esc key	
	Tab key	
	Any key	



In this lesson, we will learn various ways to refresh a part of a section based on an event. We will also learn about how we can implement more functionality in the same screen by displaying only the applicable part.

Demo



Dynamic User Interface

Let's see a short demonstration before looking at how the rules are implemented. When the user is working on the vendor maintenance case they see this screen to add a new or edit an existing vendor. By default it shows the screen for us to add a new vendor. When we select edit vendor, the new field shows up here for users to select an existing vendor and also hides the other section we had earlier.

Let's continue along, when we select an existing vendor in the field it displays the section containing the details for the selected vendor.

Let's look at the section rule that is used to display this screen- the section name is AddEditVendor, let's look at the fields that are configured to render the dynamic UI. The source or the controlling field we have is the checkbox field. Let's look at the configuration in that field. The property name used is VendorType – this property stores true or false values to indicate whether it's an existing or a new vendor. The actual configuration for capturing the details on the event is done in the Actions tab. The Actions tab can include one or more action sets. The action set includes two things: Event, here it uses click, so when the user clicks on the checkbox field it triggers the action, that leads us to the second thing we configure in the action set- the action. Here we have two actions.

The first action Post Value submits the value based on the users selection, in this case true if it's checked and false if it's unchecked. The second action Refresh Section refreshes the entire section. Let's continue along.

Now, let's now open the second field – this field acts both as a target and a source. First let's look at how the target part is configured – the cell is visible only when VendorType is true. VendorType is the property associated with the checkbox. So this field appears only when the checkbox is checked. This field also acts

as a source – when a value was entered, we saw that the section appeared below with vendor details. The action set uses the Change event for this field, since the event needs to be triggered when the value changes. The actions look similar to what we saw earlier if we need to add an action we can click the add action link. It displays the most common actions applicable and we can see both PostValue and Refresh exist there.

Let's look at the sections that are visible or hidden according to these values. The Add Edit vendor section includes a section named EditVendorDetails and if we open the properties panel, we see that it uses a condition .VendorType =false, so this appears when the checkbox is not checked. Look at the page context field here – the section include is using a page property named .NewVendor to display the new vendor screen when the condition for .VendorType is false. The Add Edit Vendor section also includes the section that displays the existing vendor details. This is the section that displayed the vendor details screen here.

Let's open up the properties panel. This uses the same embedded section named EditVendorDetails that was used to display the vendor screen, however the page context is defined to use the D_Vendor data page.

The data page uses a page structure and the VendorSearchvalue is the property where we selected the vendor name at runtime. The section is visible only when the checkbox is selected and the vendor is selected in the control. When we embed a section inside another it also provides another option to do a refresh. So in this scenario we could have also configured the refresh when condition in the refresh when field. While either approach works and accomplishes the same end user experience, configuring the refresh at the included section may be easier to maintain and implement. Consider a scenario where three different conditions may all require a refresh. Rather than configuring three source controls to trigger the section refresh, you could configure it all here in one central place.

To configure the condition, first we need to add a property and then open the condition builder to add the condition to refresh. The Condition uses the property that we select earlier and uses the keyword Changes

and can be changed to use other conditions like < , >, =, and so on. The Expression builder also supports adding other conditions by using one of the keywords AND, OR. We can use the expression builder on scenarios when we want to use the target to refresh based on more than one of the controlling fields.

That sums up this lesson on Dynamic User interfaces.

This is a very important concept to learn, we will learn more about it in the System Architect Essentials II course and other advanced courses using complex scenarios.

Exercise: Creating a Dynamic User Interface

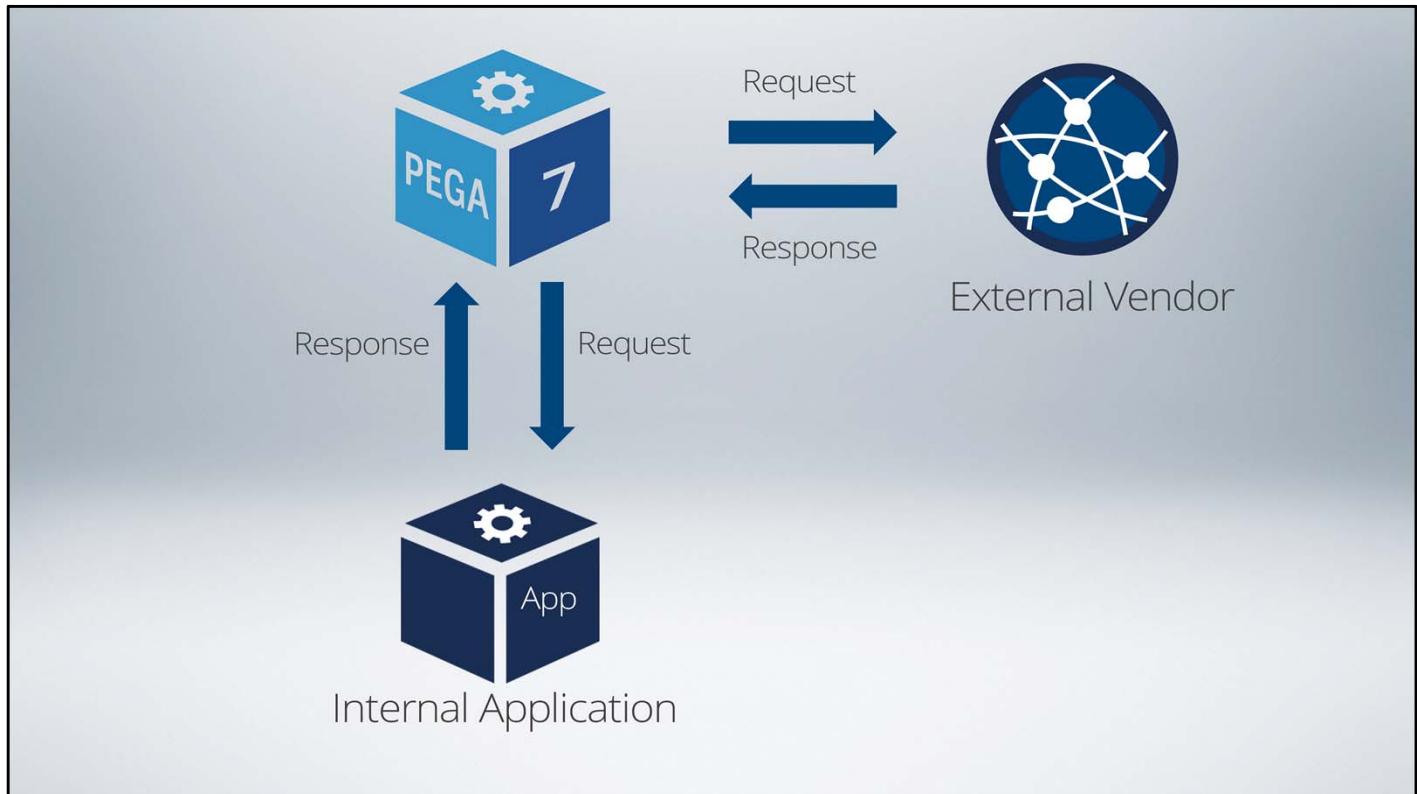


Accessing External Systems

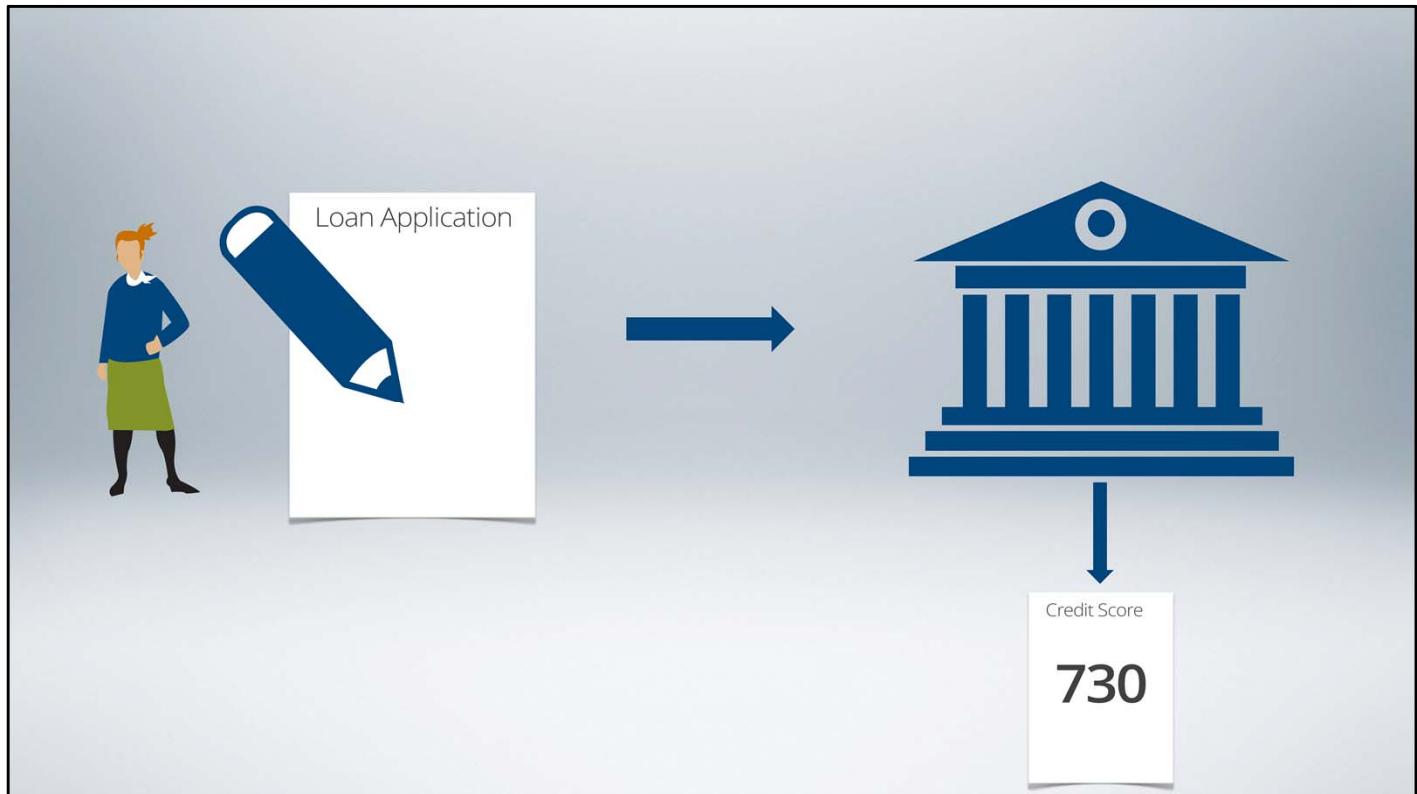
Enterprises use a wider variety of systems to handle various processes which requires systems to have the ability to communicate with each other.

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

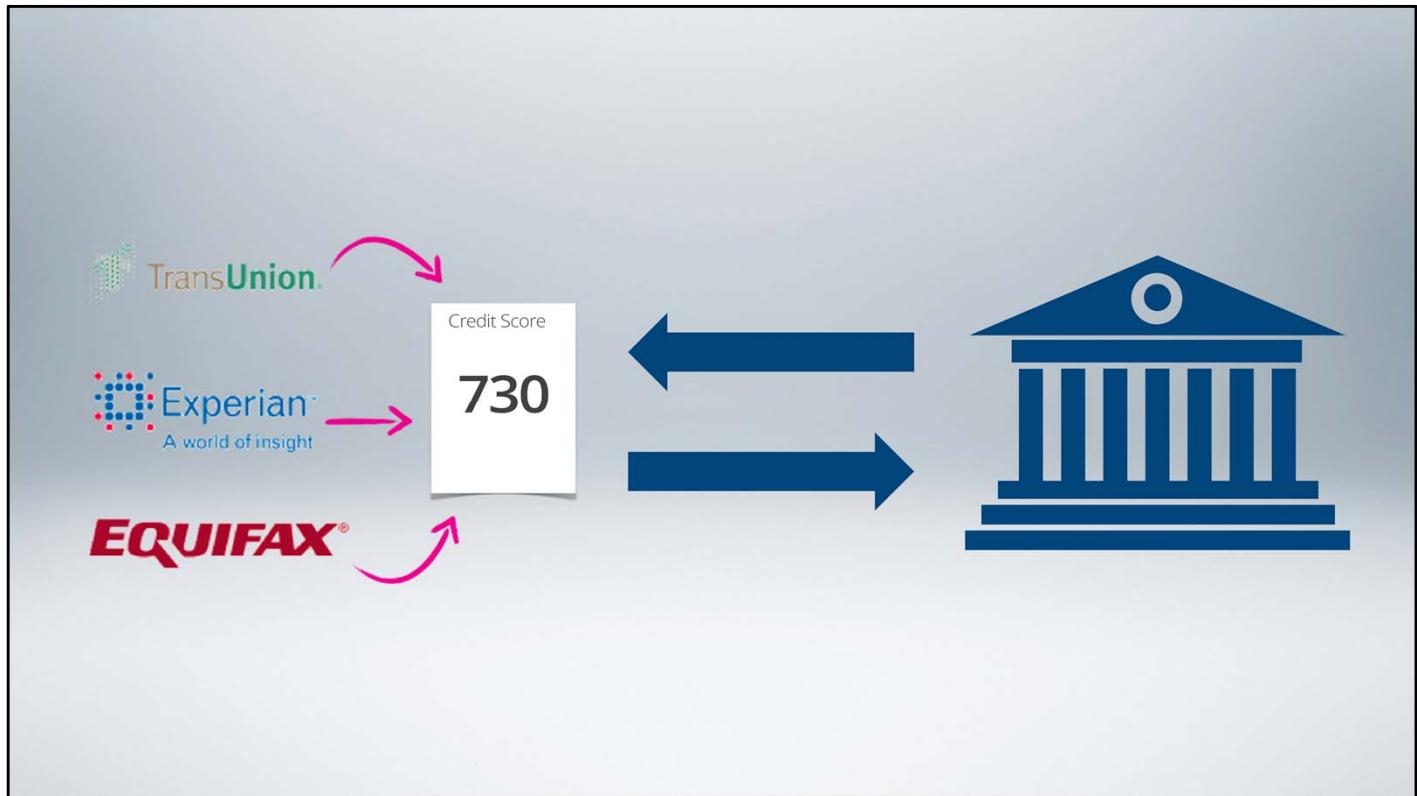
- Define Connectors and Services
- Use the Connector Wizard to create connectors and associated rules
- Understand how connectors and services are referenced in business processes



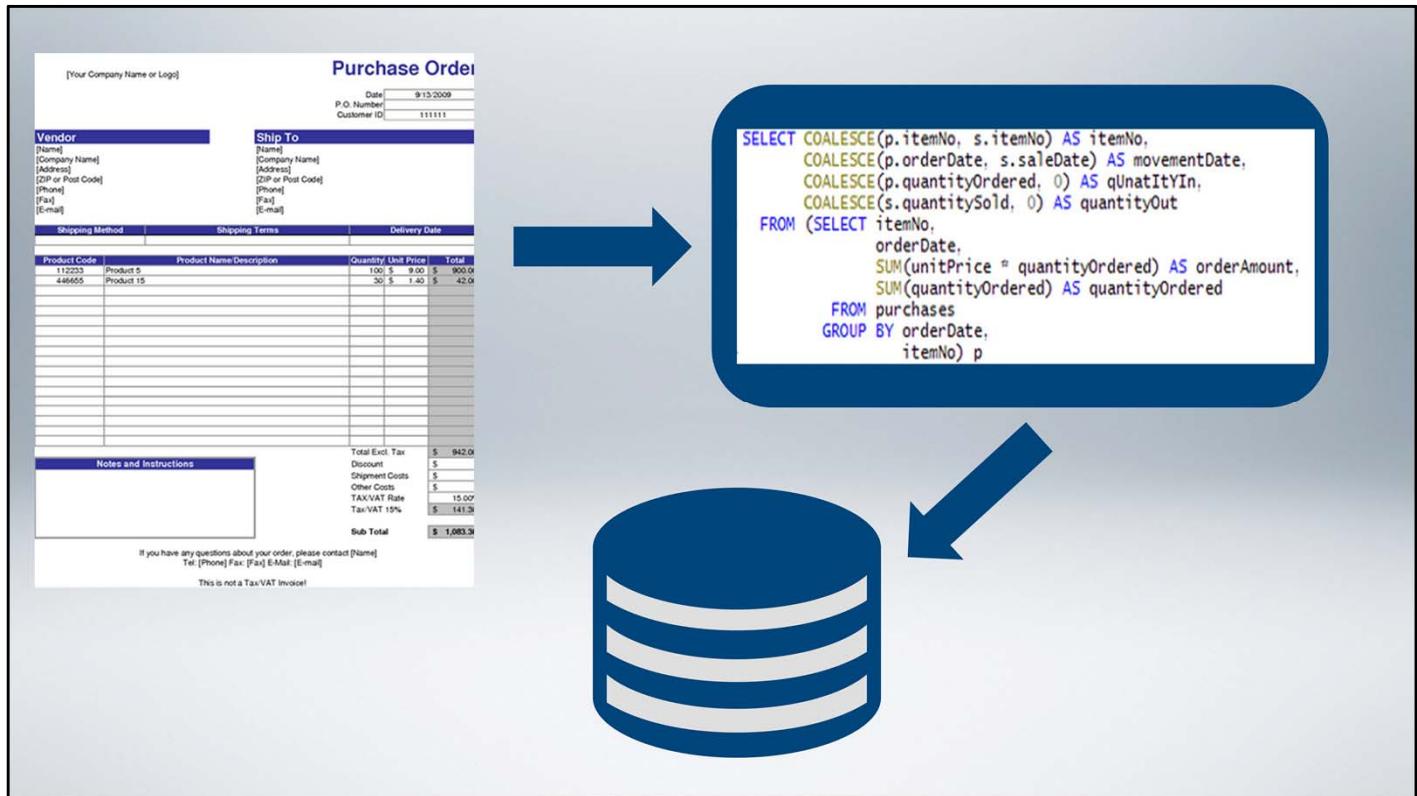
In all our enterprise projects we have requirements to interact with external systems that are running outside PRPC during case processing. These systems may run in the internal network or may be hosted on the internet by an external vendor. PRPC can integrate with all systems by passing a request and receiving the response.



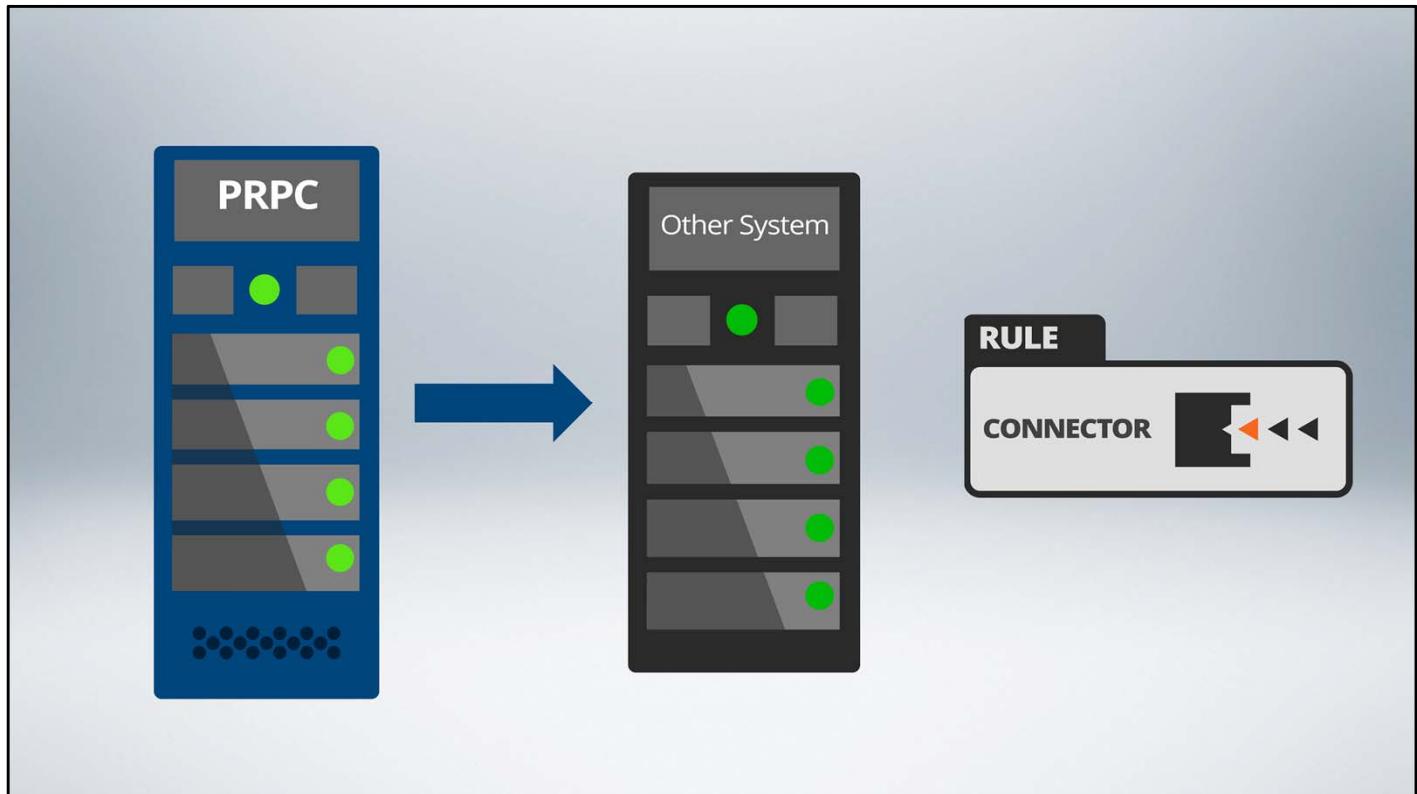
Let's look at an example, when a customer is entering a loan application and submits it to the bank, the bank will collect the customer's information and will also check or verify their credit score.



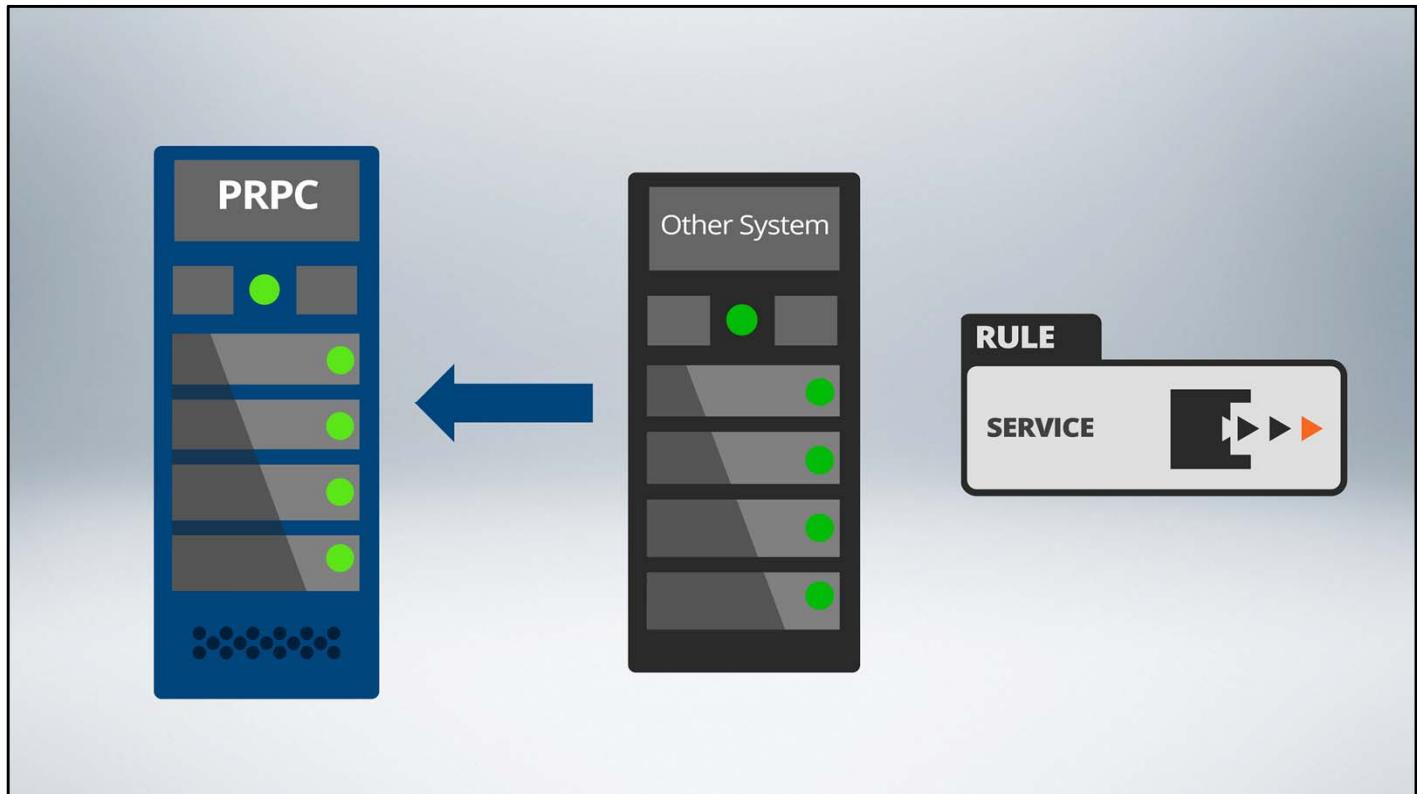
The credit score for a US based customer is supplied by three credit rating agencies - Transunion, Experian or Equifax. The application must integrate to the systems hosted by those agencies to get the credit report to verify the score.



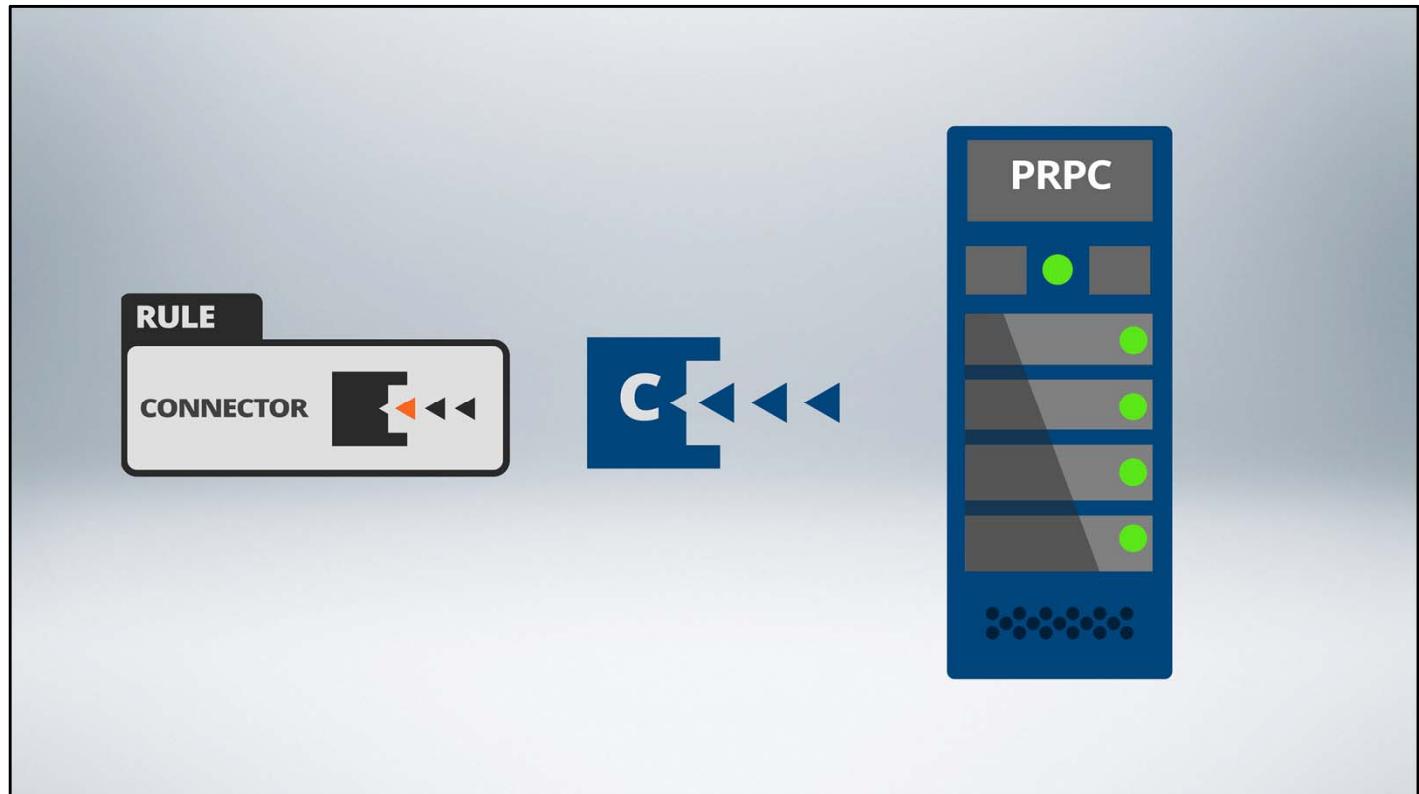
Similarly, when the order fulfillment agent is processing a purchase order submitted by a user, the application must query the internal database to check the inventory of the selected product.



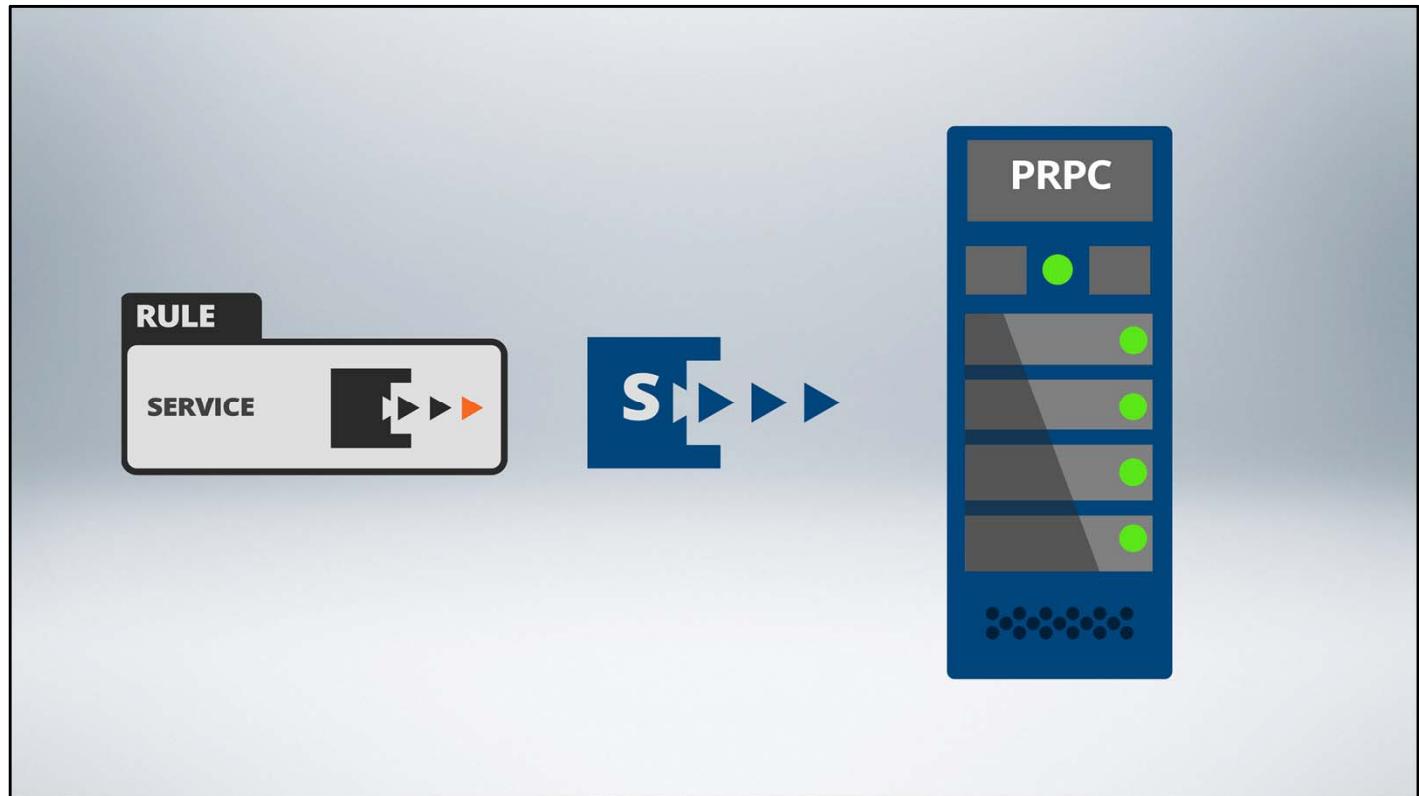
In both of these examples PRPC is requesting data from another system. PRPC uses “**Connectors**” to facilitate this type of integration.



In some cases, these external systems request data from a PRPC system. PRPC uses “**Services**” to facilitate this type of integration.



When acting as a connector, PRPC is a client requesting information from an external system.



And when acting as a service, PRPC provides resources to any client requesting it.

1. PRPC assembles the request

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Published by JBoss at https://www.jboss.com -->
<?wsdl?>
<definitions xmlns:ns1="soapwebservice.jdevelop.eu" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://jboss.com/webservices/jdevelop" name="SOAPWebServices" targetNamespace="soapwebservice.jdevelop.eu"
    >
    <xsd:schema>
        <xsd:import namespace="ns1" schemaLocation="http://localhost:8084/soapwebservice?xsd=1" />
    </xsd:schema>
    <types>
        <message name="calculateValues">
            <part name="calculateValues" element="ns1:calculateValues" />
        </message>
        <message name="calculateValuesResponse">
            <part name="calculateValuesResponse" element="ns1:calculateValuesResponse" />
        </message>
        <message name="SOAPWebService">
            <operation name="getCalculateValues">
                <input message="ns1:calculateValues" />
                <output message="ns1:calculateValuesResponse" />
            </operation>
        </message>
        <portType name="SOAPWebServicesPortBinding" type="ns1:SOAPWebServices">
            <binding style="document" transport="http://schemas.xmlsoap.org/soap/http">
                <operation name="getCalculateValues">
                    <soap:operation soapAction="urn:https://blog.jdevelop.eu/services/getCalculateValues" />
                    <input>
                        <soap:body use="literal" />
                    </input>
                    <output>
                        <soap:body use="literal" />
                    </output>
                </operation>
            </binding>
        </portType>
        <service name="SOAPService">
            <port name="WebServices" binding="ns1:SOAPWebServicesPortBinding">
                <soap:address location="http://localhost:8084/soapwebservice" />
            </port>
        </service>
    </definitions>
```

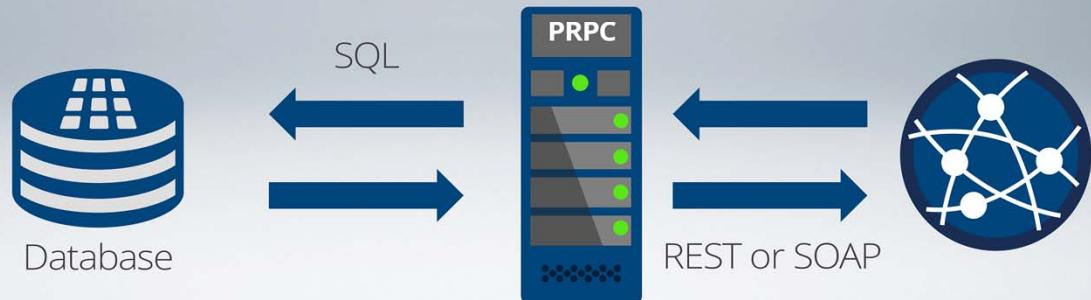
```
SELECT COALESCE(p.itemNo, s.itemNo) AS itemNo,
       COALESCE(p.orderDate, s.saleDate) AS movementDate,
       COALESCE(p.quantityOrdered, 0) AS quantityIn,
       COALESCE(s.quantitySold, 0) AS quantityOut
  FROM (SELECT p.orderDate,
               SUM(unitPrice * quantityOrdered) AS orderAmount,
               SUM(quantityOrdered) AS quantityOrdered
         FROM purchases
        GROUP BY orderDate,
                 itemNo) p
```



When acting as a connector, the system does the following:

1. Assembles the request – if we are connecting to a web service, we need to use the Web Service Definition Language(WSDL) supplied and enter values for all the parameters used in the request XML, and in the example of database, we need to create the SQL that is used to get the data from the database.

1. PRPC assembles the request
2. Communicate with the external server using the correct protocol



2. Communicates using the right protocol – We need to make sure the request is routed to the appropriate service and also uses the correct protocol – SQL to query from the database, REST or SOAP in case of web services.

1. PRPC sends request
2. Communicate with the external server using the correct protocol
3. PRPC maps the response to properties in the case

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
    <soapenv:Header>
        <t:payloadManifest xmlns:t="http://www.standards.org/webservices/transport"/>
    </soapenv:Header>
    <soapenv:Body>
        <t:trans:ProcessMessageResponse xmlns:t="http://www.standards.org/webservices/transport">
            <content xmlns="http://www.groovyinsapui.com/getContent">
                <Response value="http://www.groovyinsapui.com/response">
                    <DeliverySource>ip://www.groovyinsapui.com/response</DeliverySource>
                    <RefId>123</RefId>
                    <InquiryDate>2008-11-27</InquiryDate>
                    <Status>OPERATION SUCCESSFUL</Status>
                </Response>
            </content>
        </t:trans:ProcessMessageResponse>
    </soapenv:Body>
</soapenv:Envelope>

```



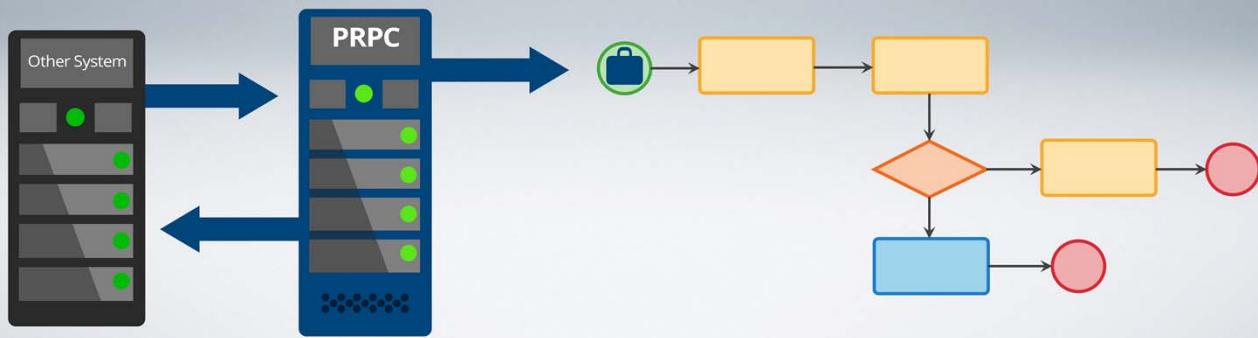
3. Receives the response from an external system – The external system provides the response for our request and we need to map the data back to the case.



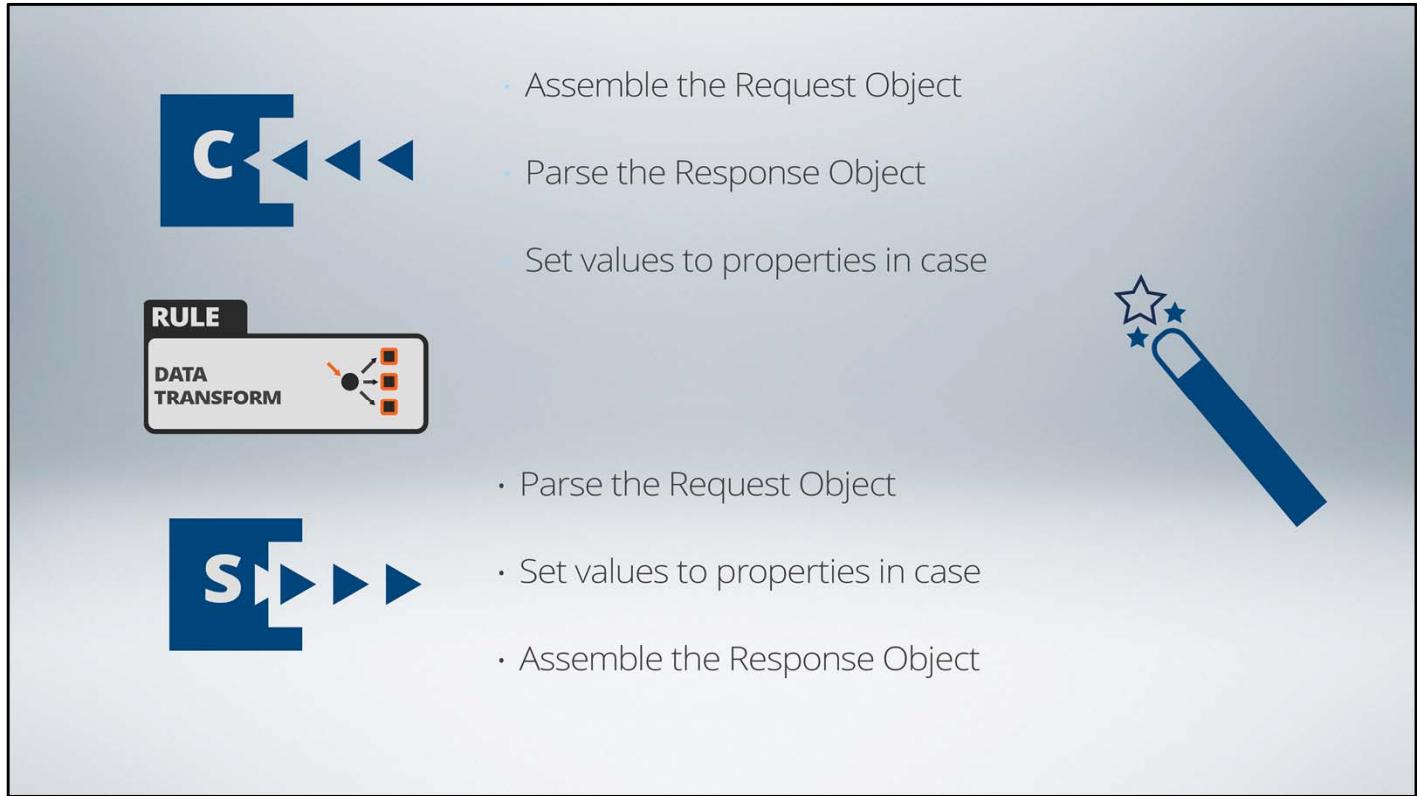
1. PRPC sends request
2. Communicate with the external server using the correct protocol
3. PRPC maps the response to properties in the case

The good news is that all of these can be created quickly using the connector wizard, which we will learn about in this lesson.

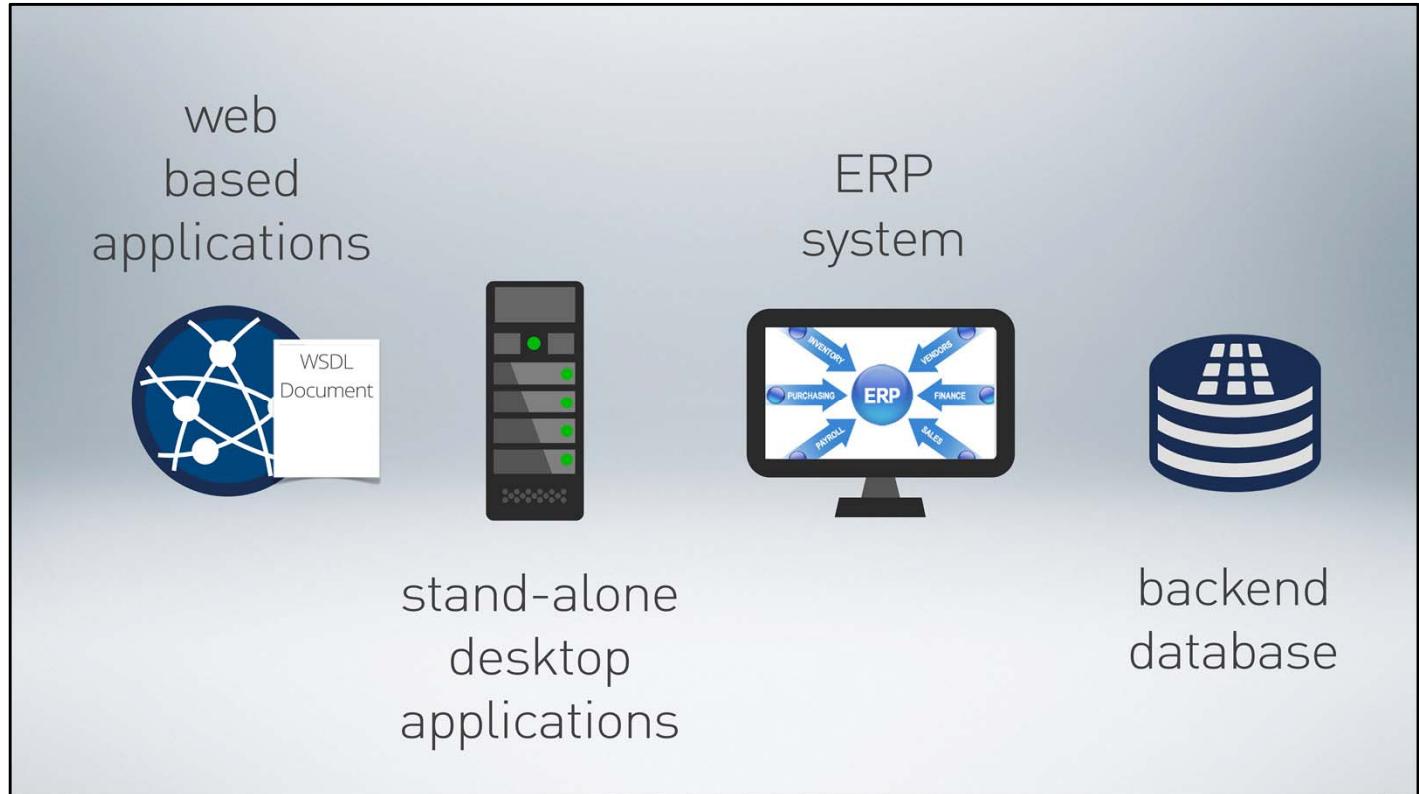
1. External system sends request
2. Communicate with the PRPC using the correct protocol
3. PRPC processes request and sends back response
 - a. Usually creates a case



In the case of Services, PRPC acts as a server or resource supplying information. So an external system assembles the request, and sends it to the PRPC instance using the expected protocol. PRPC processes the request, performs some tasks and then sends the response back to the external system.



Both services and connectors involve the exchange of data which involves parsing the input or response, creating a response object and mapping rules to set values on properties. To facilitate development, Pega 7 comes up with wizards to build connectors and services.



Enterprises use a wide range of software vendors ranging from web based applications, stand-alone applications running on a desktop, an ERP system or a backend database.



These systems are built using a wide variety of technologies such as java, .NET or EJB, SAP or they use a variety of databases such as Oracle, DB2, and so on.

Targets	   
Technologies	   
Protocols	  

This requires communicating over a wide variety of protocols.

Connectors

- File
- JAVA
- JMS
- HTTP
- EJB
- MQ
- SOAP
- JCA
- CMIS
- SOAP (.NET)
- SQL
- SAP
- REST
- BPEL
- SAPJCo

PRPC connectors can connect to a file system, HTTP, SOAP or REST to access web services, java services, EJB or JCA, SQL and BPEL.

PRPC also has direct connectors to specific ERP systems like SAP, and content management solutions like CMIS.

Services

- File
- Email
- HTTP
- SOAP
- SOAP (.NET)
- JAVA
- EJB
- JMS
- MQ
- JCA
- COM
- BPEL
- CMIS
- Portlet
- JSR94
- SAP
- SAPJCo

Similarly when PRPC is acting as a service it is available to be contacted through Email, File, HTTP, SOAP, JAVA, EJB, JMS and others. The list is little different from the connectors.

The wizards support most of the connector and service protocols and hence they are generated automatically. This list shows that PRPC can connect to any system that exists in an enterprise.

Planning: Connector

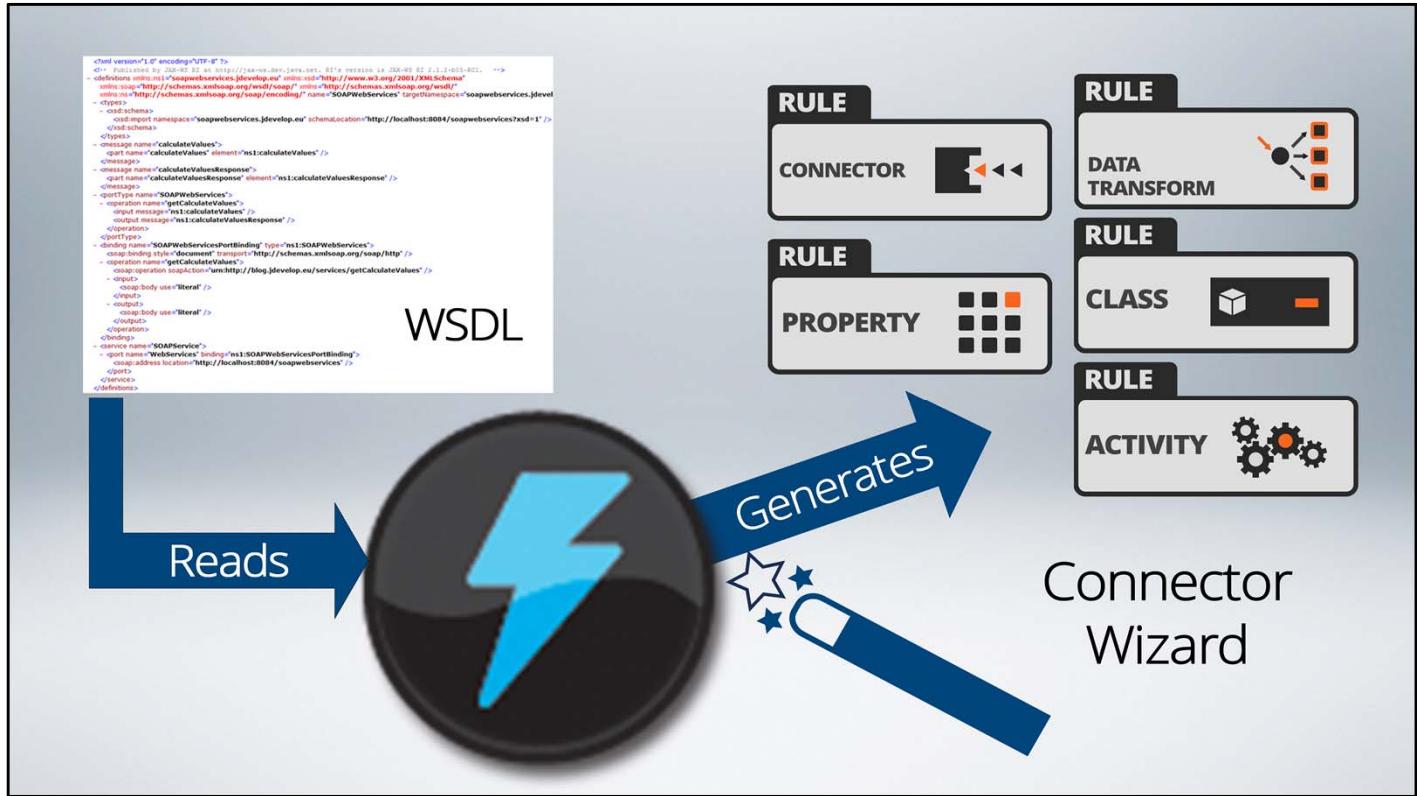
- Access the external service
- Protocol to use
- Assemble the request
- Mapping from response

When we connect to an external system, we need to plan or decide on the following points:

1. How we are going to access the external service?
2. What protocol are we going to need to use?
3. How are we going to assemble the request?
4. What type of response do we want to get?



In this lesson, we will see an example of connecting to a web service using the SOAP Protocol.



The external system publishes a WSDL document which is the binding contract published by web services. The good news is that we do not need to be experts in interpreting a WSDL, the connector wizard can use the WSDL as an input to generate the rules used for assembling the request, accessing the external service and interpreting the response.

Demo



Running the Connector Wizard

Let's see this in action-

Let's select the Designer Studio > Integration and then select Create Soap Integration since we are creating a soap connector.

The wizard provides a guided step by step interaction gadget and we will have the opportunity to review our changes before the final submission.

There is also the inline contextual help available for each step.

Now let's continue along. In the first step we need to provide the WSDL document – the system can read from a URL or we can import it. Let's select the second option – Upload WSDL from a file and then locate the file in the local system. After choosing the file, click next to go to the next step.

The system quickly parsed the information in the WSDL and it provides the information such as the endpoint URL where the service is hosted. There is authentication and other details that can be configured here, let's leave them as-is for now.

The wizard lists all the methods supported in the WSDL in the operation area. We can select whatever method we want to use to connect to the other system.

This service offers the GetExchange rates method, so let's go ahead and select it. In the last screen, we can review the choices and also make some changes. We can rename the service to make it more meaningful.

And for the other fields, we would normally change these values to make sure the connector is created in the required place. However, for our learning purpose, let's accept the default choices and click Create. The wizard created 15 records for the connector to function.

In rare cases we could undo the generation by clicking the undo generation button. We can look at the connector rule by clicking the data source.

In the advanced courses, we will learn more about the different types of connectors and view all the rules created by the wizard.

Demo



How Connectors are Referenced in the Business Process

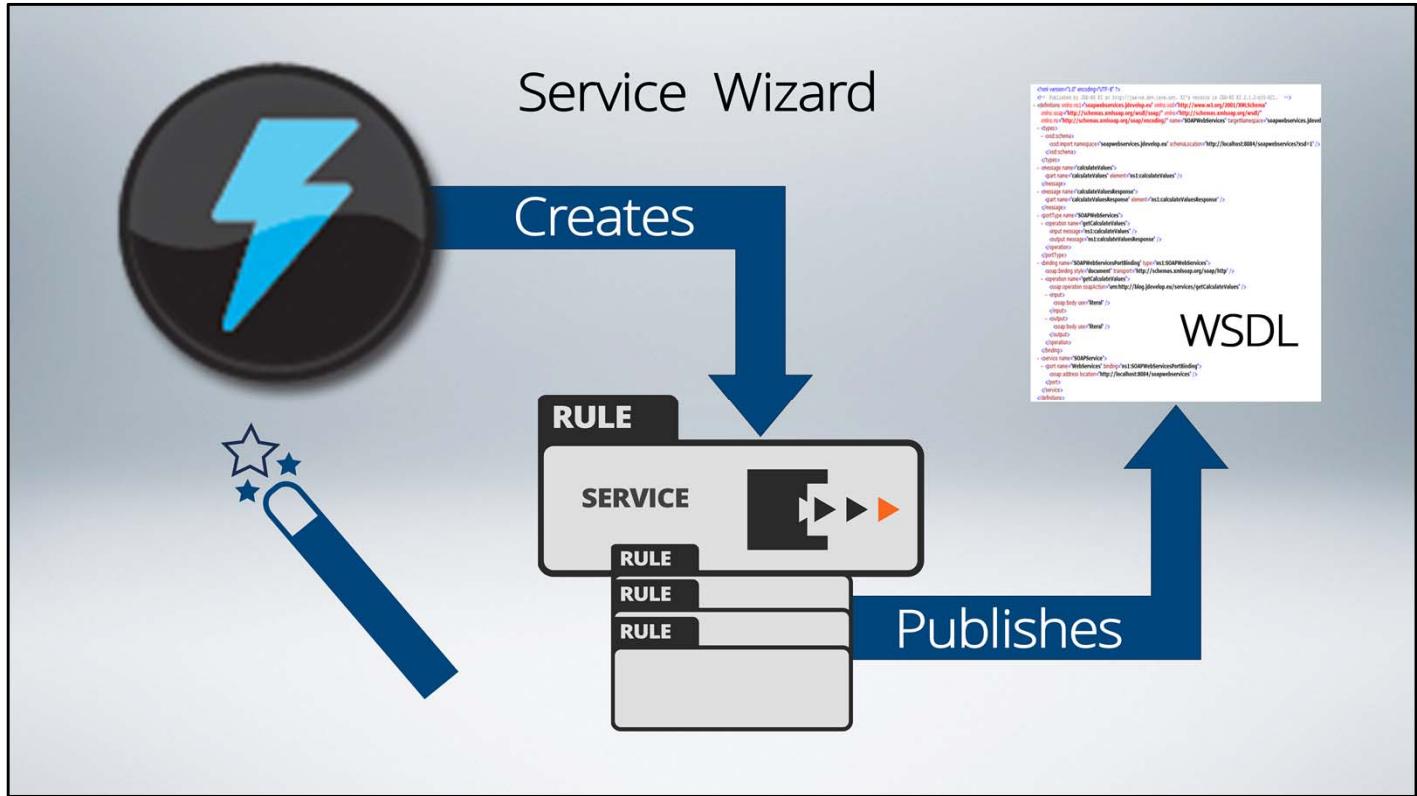
We have seen the connector rule that is generated by the wizard. How does this rule get invoked in the business process?

The recommended option is to create a data page and then define a data source for the page.

In the Source field, select Connector. In the Type select the type of protocol used and enter the name. Also use the data transforms and the list where the response is captured if available. Alternately, the connector can also be invoked from the process using a special shape. In the process modeler, click the add shape icon

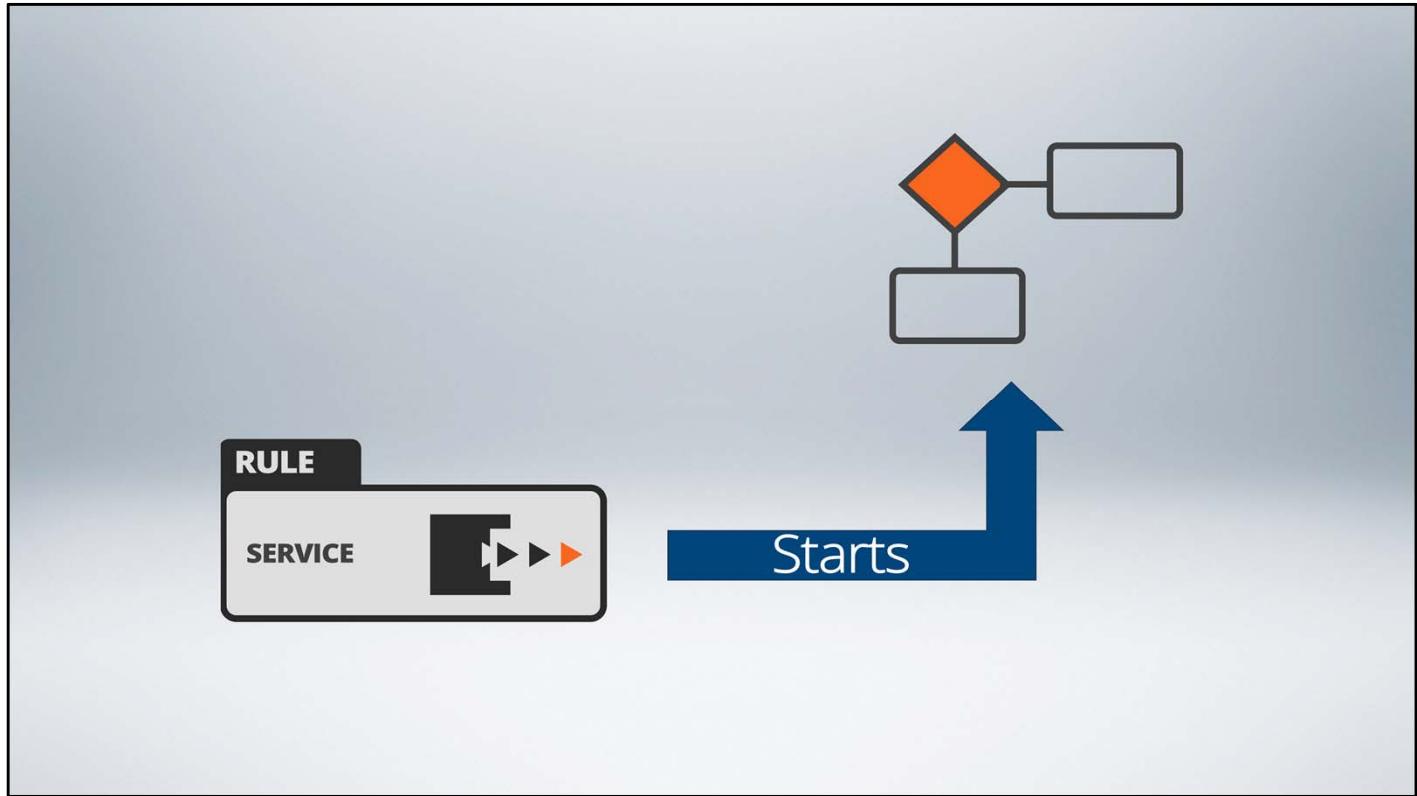
and then navigate to advanced shapes and select Integrator.

We can configure the properties of the shape to access the connector.



PRPC supports wizards to generate a service, when running the wizard it creates a service rule and other related rules that are required for the service to operate.

In a few cases, the service rule supports publishing the WSDL or the binding document that other connecting systems require for communicating with PRPC services.



When PRPC is implemented as a service, the PRPC service receives the request from the external system and then typically starts a flow.

That concludes this lesson, this is a basic introduction to the integration capabilities in PRPC , most of the integration tasks are performed by senior architects and this topic is covered extensively in the advanced courses.

Optional Exercise*
Reusing Data Transform for Background Check
**This lab can take some time*

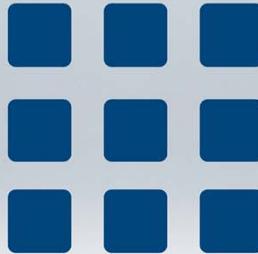


Review of Accessing Data in Application

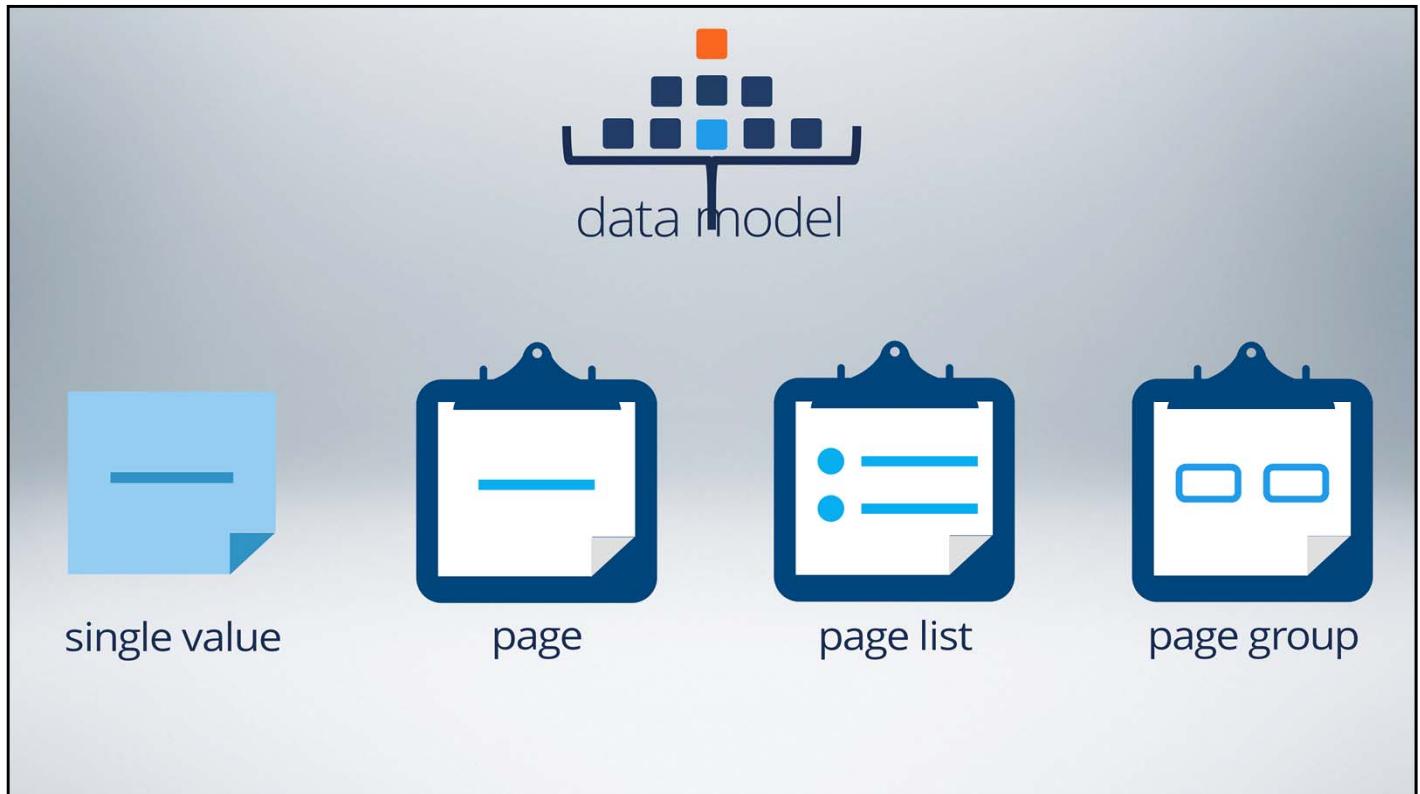
Applications require data for processing cases, we learn how to use data pages to create data in a separate layer and how we use them in application.

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

- Review how data is saved as a separate entity from the case
- Leverage data pages to keep the data definition outside case processing
- Understand how to integrate with external systems to collect data
- Understand how to create data tables to store data inside Pega
- Learn how to build a dynamic user interface



Almost all applications work with data in one way or another. As a result, it is critical to understand how PRPC can access both internal and external data and the options available for persisting data. This lesson group covered lessons teaching us how to define internal data, how to connect to access external data and how the data remains available for subsequent access.



Before PRPC can work with data in any way, we need to define a data model which we learned earlier - the properties we define can be a single value, page, page list and page group.



Some of the data elements are defined only to store reference data that is required in processing a case.



Data Pages



pyWorkPage

Clipboard page: pyWorkPage	
PROPERTY	VALUE
BaseCurrency	USD
CostCenter	1001
Currency	USD
GLEntity	1100
HasBlockedVendor	false
LineItemsTotal	3270.0
NeededByDate	20131210
PaymentTotal	0.0
POTotal	3270.0
pxApplication	PurchaseFW
pxCoveredCount	0
pxCoveredCountOpen	0
pxCoveredCountUnsatisfied	0
pxCoverInSKey	
pxCreateDateTime	20131210T230316.112 GMT
pxCreateOperator	SAE1Architect
pxCreateOpName	SAE1 Architect
pxCreateSystemID	pega
pxCurrentStage	PRIM1
pxCurrentStageLabel	Order Placement
pxCurrentStageScript	PRIM1_1
pxInName	PO-5
pxMarkedForSuspension	false
pxObjClass	Pco-FW-PurchaseFW-Work-Purchase-PurchaseOrder

We learned about the importance of data pages and how they can source any type of data. Remember that a data page is created in memory outside pyWorkPage and so the case can access this data page whenever it is required.



Data Pages

Separate Data from Process, UI and Logic

Source from where data is read

Data table, external system

Uses lookup, , data transform or report definition

Data structure

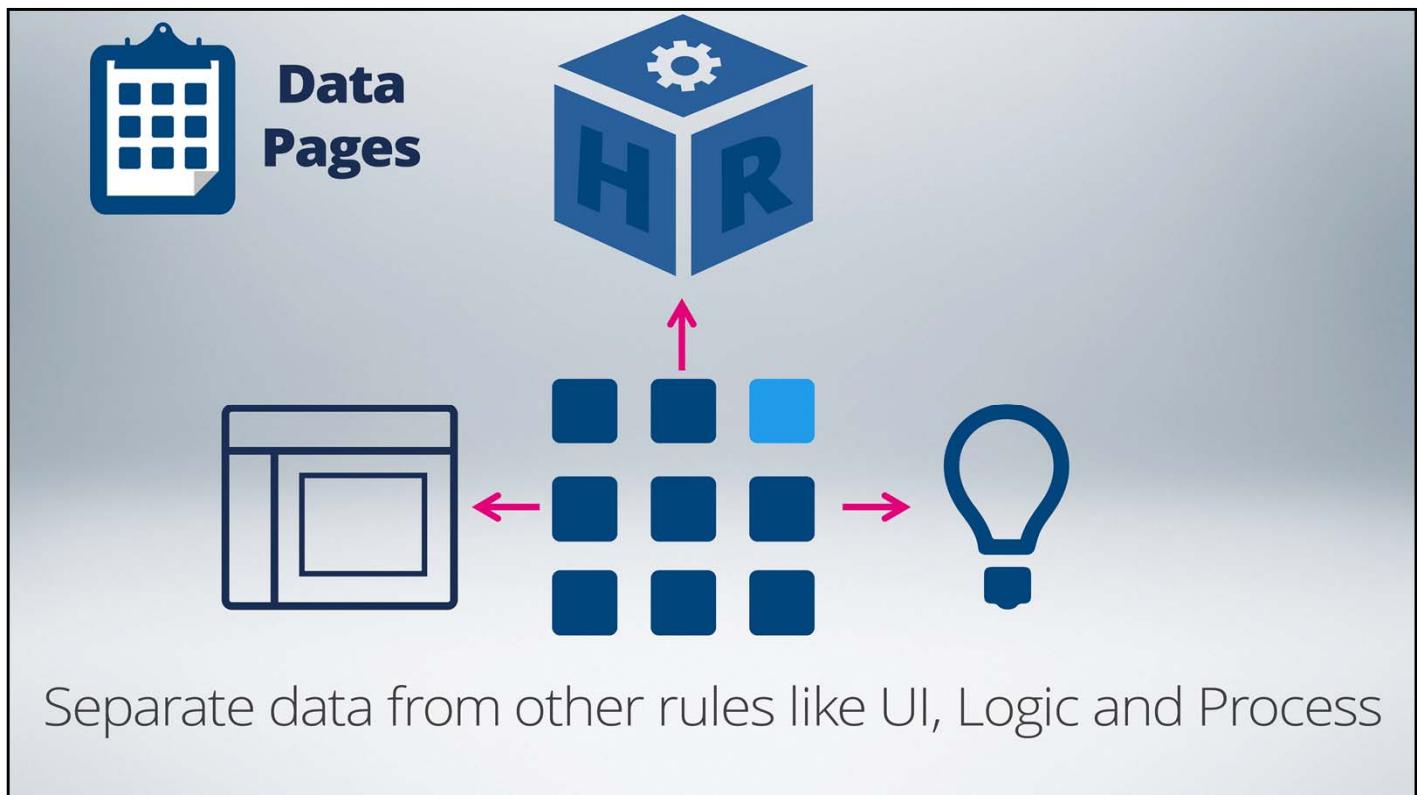
Pagelist or page

Scope

Thread(Case), requestor(user session) and node (across all sessions)

Refresh strategy

Data page definitions control how the data is sourced, their page structure, where it is getting created or the scope of the page, and when it gets refreshed as well as if it is editable or not.



Data pages help in designing the UI and other rules that require access to data by abstracting the definition of the data page. In other words, a UI developer for example can simply reference a data page without having to worry about the complexities of where and how the data is loaded.



Data Pages in the UI



Repeating Grids

If the structure is a list

CheckBox

RadioButtons

- | | |
|--|---------------------------------------|
| <input type="checkbox"/> Red | <input checked="" type="radio"/> Red |
| <input checked="" type="checkbox"/> Blue | <input checked="" type="radio"/> Blue |
| <input type="checkbox"/> Pink | <input type="radio"/> Pink |
| <input type="checkbox"/> Black | <input type="radio"/> Black |
| <input type="checkbox"/> Purple | <input type="radio"/> Purple |

select an item...

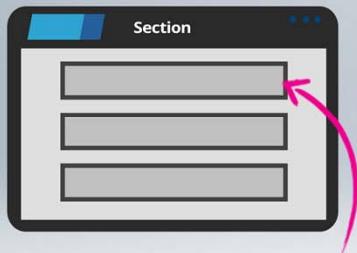
- item 2
- item 3
- selected item
- item 5

List based controls such as dropdown or radiobutton

When the data page is a list structure, we can use the results to render a repeating grid or populate a list based control such as a dropdown or radio buttons.



Data Pages in the UI



An embedded section using the Data page

If the structure is a page



A field cell using the reference
D_Page.propname

When the data page is a page structure we can use it to populate all the fields on an embedded section. In this case the data page would likely be configured to accept a parameter value so that the fields display the appropriate content for a particular case.



Data Pages in the UI

If the structure is a list

#	Description	Category	Frequency	Amount	Balance
1	Paycheck	Income	Twice a month	\$1500	\$1730.52
2	Cel Phone	Bills	Monthly	-\$96.58	\$1649.94
3	Rent	House	Monthly	-\$900.58	\$749.94
4	Car Payment	Auto	Monthly	-\$185.58	\$564.36
5	Website Contract Work	Income	Once	\$465	\$1029.36

If the structure is a page

Transaction Detail close or Esc Key

Transaction Information	
TransactionID: 3828965819	TransactionDT: 12/02/2010
Amount: \$173.92	SettlementDT:
Status: APPROVED	
Response:	
Donor Information	
Name: RG Sign Print Center	Username: rgsignprintcenter38
Account Type: Visa	
Allocations	
Default Designation \$173.92 	

Let's see them with an example – if we need to look at a list of all transactions made by the account holder, we would require a data page with the list structure.

If the user wants to select a specific transaction to look at the transaction details then we require a data page that is a page structure to display the selected transaction.



Data Pages



Data pages are invoked declaratively

System looks for the page in memory

If not in memory, system must load the page

Page, once loaded, exists in memory until the scope expires

Data is refreshed depending on the refresh strategy

Data pages are invoked declaratively

– when a rule accesses a data page, the system looks for that data page in memory, if the page does not exist in memory, the system looks for the data page definition and then loads the data page in memory.

The data page exists in memory for the next access, so when we access it for the second time the system uses the data page that is already loaded in memory. The data page remains in memory as long as the scope is applicable, so if the scope is REQUESTOR, the page exists until the user session expires.

Depending on the refresh strategy, the data on the data page may be reloaded upon subsequent access.

Data Tables



A structure used to store reference data outside the application in a dedicated database table



Some applications require data that is part of what is saved in PRPC. We use data pages if the data has to be maintained in PRPC.

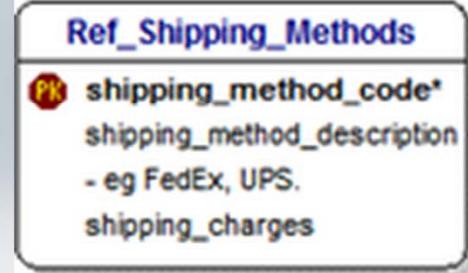
A data table is a structure that is useful in storing reference data in a dedicated database table which can be updated periodically.

Table Structure

Table: PR_OTHER

Lab27/PEGA/TABLE/PR_OTHER

COLUMN_NAME	DATA_TYPE	TYPE_NAME	COLUMN_SIZE
PXCREATEDATETIME	93 DATE		7
PXCREATEOPNAME	12 VARCHAR2		128
PXCREATEOPERAT...	12 VARCHAR2		128
PXCREATESYSTEMID	12 VARCHAR2		32
PXINSNAME	12 VARCHAR2		128
PXOBJCLASS	12 VARCHAR2		96
PXUPDATEDATETIME	93 DATE		7
PXUPDATEOPNAME	12 VARCHAR2		128
PXUPDATEOPERAT...	12 VARCHAR2		128
PXUPDATESYSTEMID	12 VARCHAR2		32
PYLABEL	12 VARCHAR2		64
PZINSKEY	12 VARCHAR2		255
PZPVSTREAM	2004 BLOB		4000



Similar to cases, data tables use a proprietary database table structure to persist their data. These tables are stored in the same database where the Pega 7 application is installed.

Even though data tables are saved in database tables, they are usually edited through PRPC. A data table created for a specific use case includes additional columns specific to the table. Data tables can store information such as a list of shipping options available for a purchase order case.

Data Tables

Data Table wizard

Provide ability to define columns, data type and so on

Create all rules and database tables and columns



Access Data Table

Two data pages – to access the list of all rows, the specific row based on the id

Landing Page

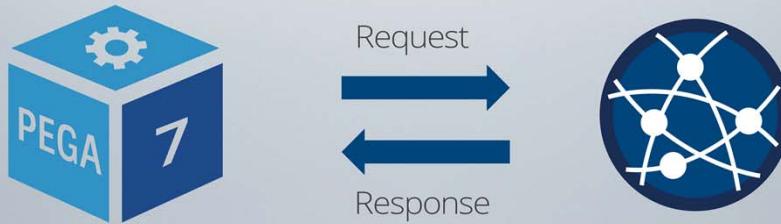
Add rows to table

A data table can be quickly created using the Data Table wizard. This wizard automatically creates all the rules necessary for a data table.

The wizard also creates two data pages – one for displaying the list of all products and the other one to display a single product based on a parameter search, in this case it would be ProductID.

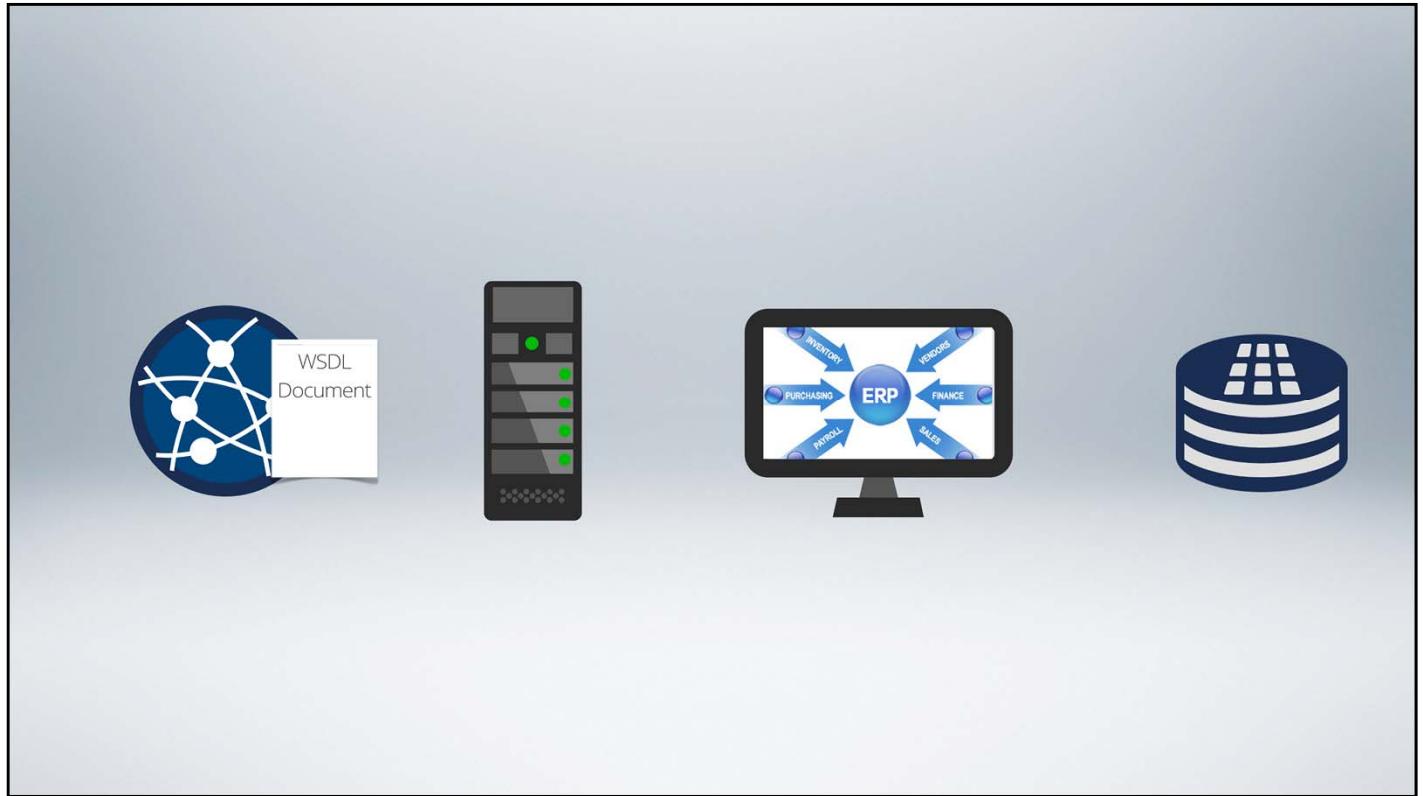
Once the table is created we can go to the landing page and use the edit option to add rows in the data table.

Accessing External Systems



Date	Description	Category	Frequency	Amount	Balance
AUG 01	Paycheck	Income	Twice a month	\$1500	\$1736.52
AUG 02	Cell Phone	Bills	Monthly	-\$86.58	\$1649.94
AUG 02	Rent	House	Monthly	-\$900.58	\$749.94
AUG 02	Car Payment	Auto	Monthly	-\$185.58	\$564.36
AUG 02	Website Contract Work	Income	Once	\$465	\$1029.36

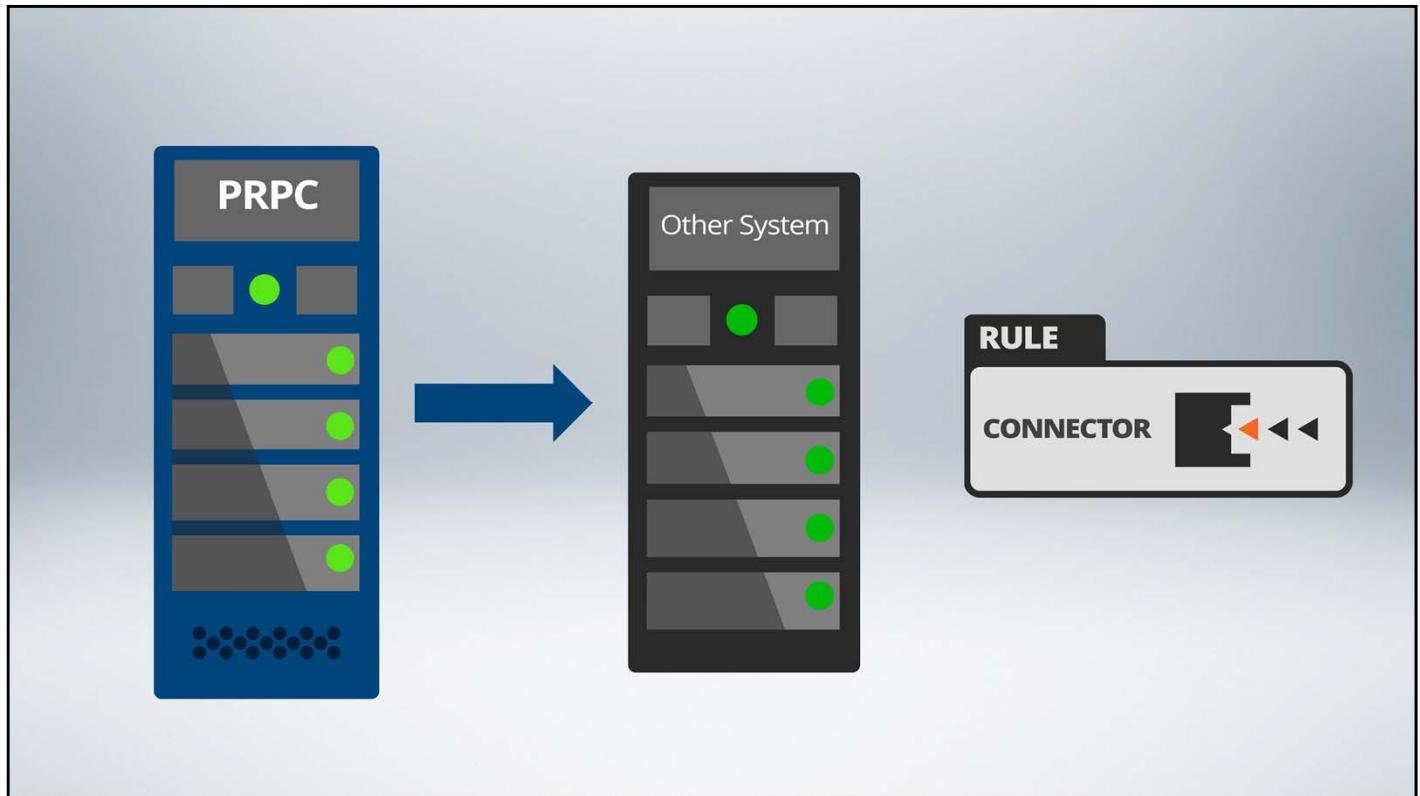
PRPC is not always used as the system of record (SOR) for storing all reference data. In most enterprise projects we access an external service to get the reference data. For example, the list of transactions that an account holder has made over the past one month is usually stored in an external system and we normally access this external service to get this data.



Enterprises have a wide variety of systems that act as a source for providing data to PRPC. These can be external to the enterprise like a public hosted web service for getting the credit score of the applicant or the backend SAP system that holds the inventory data for the company.



These systems may be written to using different technologies and therefore often require accessing their data using a different set of protocols.



PRPC can integrate with these systems using CONNECTORS.

Connectors act as a client to these systems.

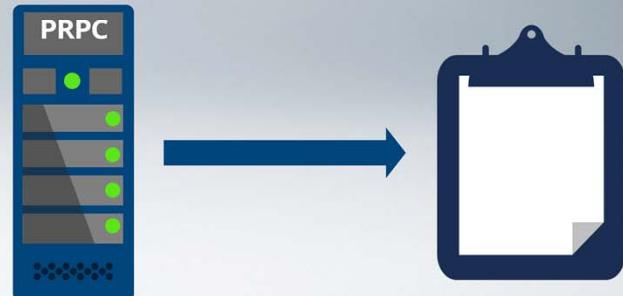
1. PRPC sends request
2. Communicate with the external server using the correct protocol
3. PRPC maps the response to properties in the case



```

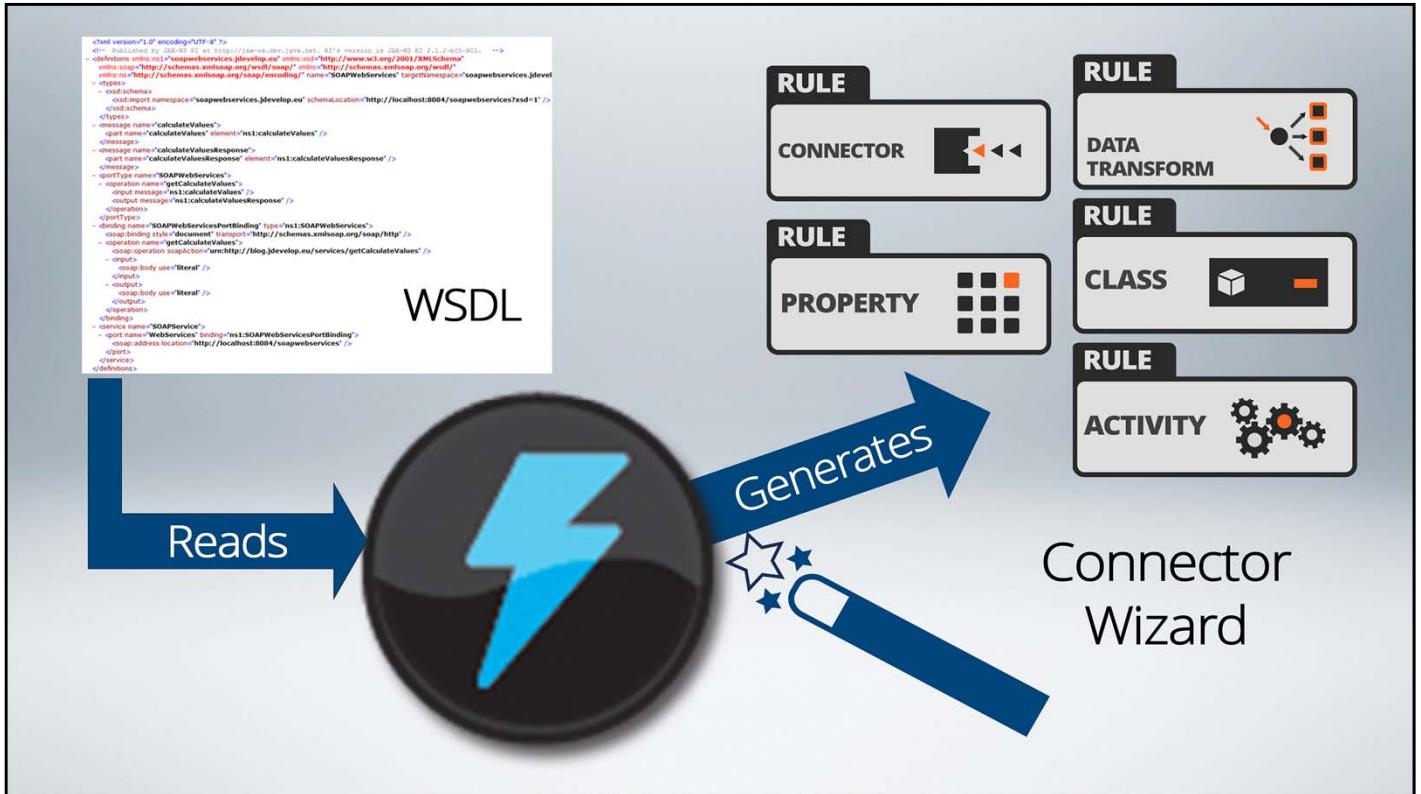
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header>
    <t:payloadManifest xmlns:t="http://www.standards.org/webservices/transport"/>
  </soapenv:Header>
  <soapenv:Body>
    <t:trans:ProcessMessageResponse xmlns:t="http://www.standards.org/webservices/transport">
      <content xmlns="http://www.groovyinsapui.com/getContent">
        <Response value="http://www.groovyinsapui.com/response">
          <DeliverySource>ip://www.groovyinsapui.com/response</DeliverySource>
          <RefId>123</RefId>
          <InquiryDate>2008-11-27</InquiryDate>
          <Status>OPERATION SUCCESSFUL</Status>
        </Response>
      </content>
    </t:trans:ProcessMessageResponse>
  </soapenv:Body>
</soapenv:Envelope>

```

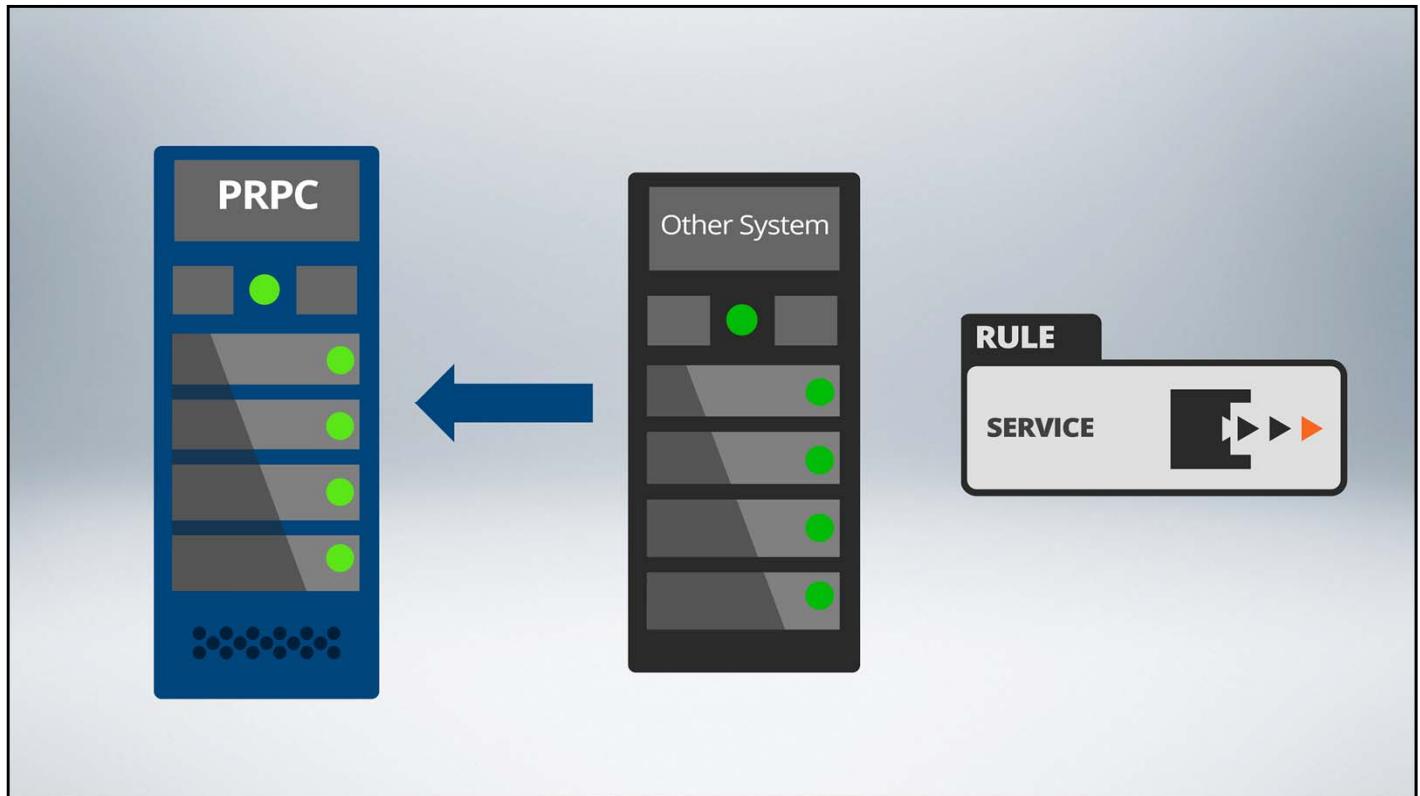


Connectors assemble a request, use the appropriate protocol to access the external system and then receive the response sent by an external system.

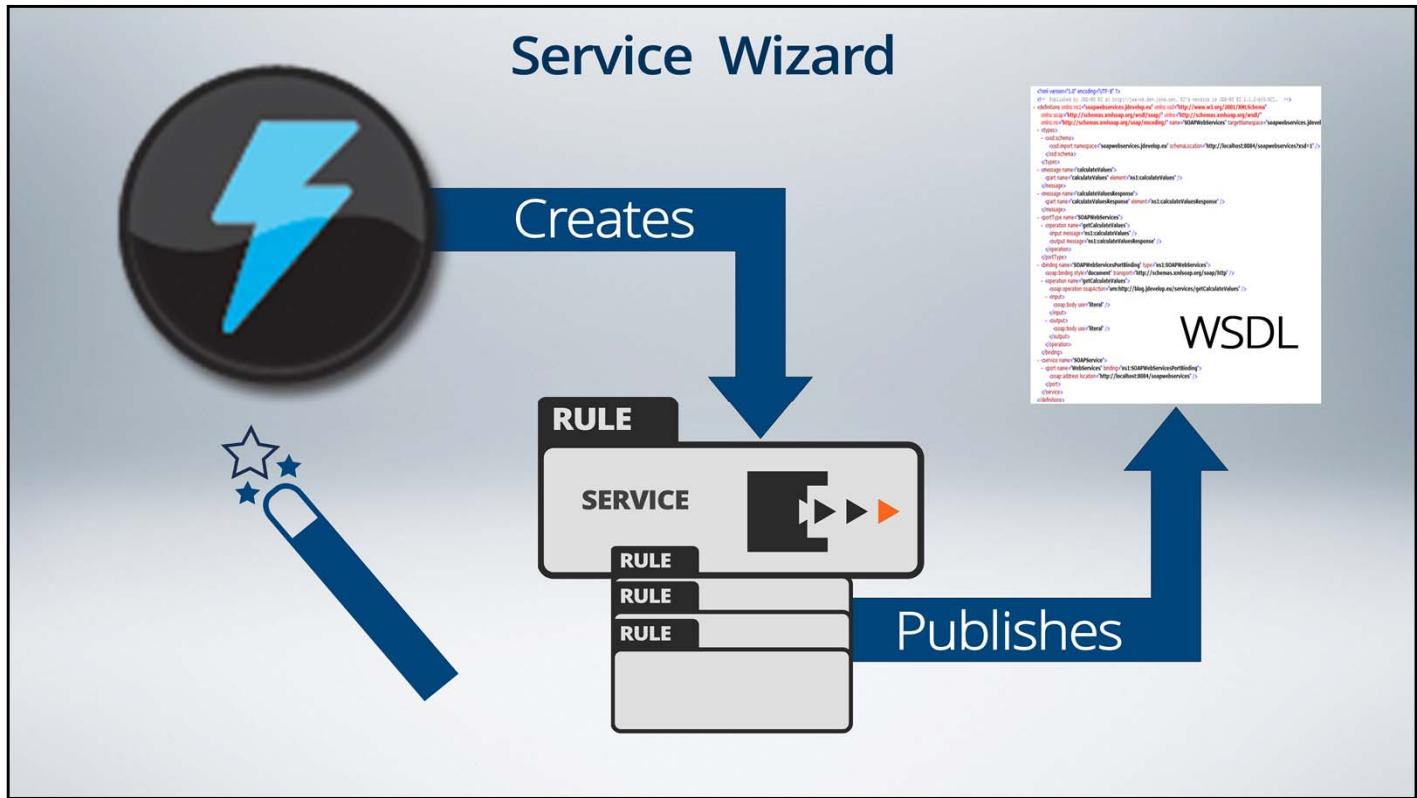
PRPC has the connector wizards for different protocols to automate these three steps for most of the supported connectors.



The Connector wizard generates the connector and other associated rules that are required to perform all these operations. The results can be stored in a data page so the user interface can use the data page to display at runtime.



We also learned about PRPC Services where PRPC acts as a service to external clients, who send a request to PRPC. PRPC then creates a new case or access an existing case, performs some processes and sends back the response.



PRPC also comes up with a Service wizard that can create all the rules associated with a PRPC service rule.

Connectors

- File
- HTTP
- SOAP
- SOAP (.NET)
- REST
- JAVA
- EJB
- JCA
- SQL
- BPEL
- JMS
- MQ
- CMIS
- SAP
- SAPJCo

We also looked at the list of all connectors

Services

- File
- JAVA
- JCA
- Portlet
- Email
- EJB
- COM
- JSR94
- HTTP
- JMS
- BPEL
- SAP
- SOAP
- MQ
- CMIS
- SAPJCo
- SOAP (.NET)

and services that are supported in PRPC.

Dynamic UI

Impacts part or the whole screen
Values changed on a specific field

Apart from learning how to access data and where the reference data can be saved in a Pega application we also learned about dynamic user interfaces.

Dynamic user interfaces change portions or even the entire screen based on a specific event such as a change in the value of a specific property, say if we choose a specific color.

Based on the value, it may conditionally display a new screen

or display the new calculated subtotal based on the changes in quantity or unit price for the line item.

Key Parts of Dynamic UI

Source (Controlling field)

NEXT



Click of a control

Link, Button

Keyboard actions

Enter, Esc,
Key press,
Key down

Property based

Property value changes

Dynamic behavior requires configuring two fields – a source which can be based on an event such as a click of a button or a link,

keyboard actions such as enter, esc, key press, key down, or
a change in the value of a property.

Controlled Fields



Visible when – hide/display fields



Refresh when – displays the changed values



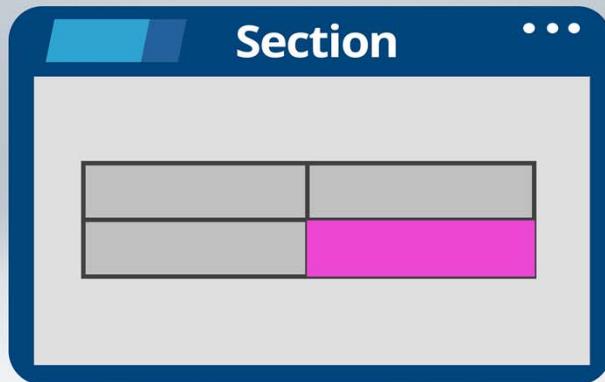
Read only when



Disable when

And a target which will be one or more dependent properties which might require us to perform one of these things – refresh the screen to conditionally display or hide portions of the screen, display the modified values of the properties on the screen, change the edit option of the fields, or disable the properties based on a condition.

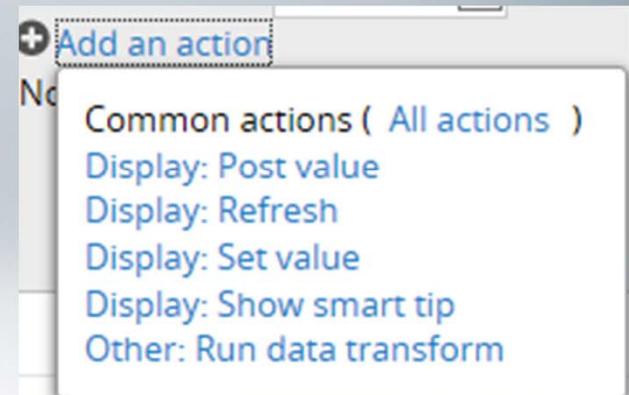
Controlled Field



When the event defined on the source field is triggered, it performs the action associated with it and additionally it impacts the target identified which could be a cell, layout or a section embedded in another section.

Refresh a Section

Mouse events	Keyboard events	Other events
Click	Enter key	Change
Double-click	Up key	Focus
Hover	Down key	
Right-click	Left key	
	Right key	
	Esc key	
	Tab key	
	Any key	



We concluded the lesson by looking at how we can refresh a section based on an event – first we select the event

and then select the action by defining them together in the action set.

Module 08: Automating Business Policies

This lesson group includes the following lessons:

- Introduction to Declarative Processing
- Conditional Processing
- Automating Business Processes Using Decision Rules
- Review of Automating Business Policies

Introduction to Declarative Processing

Declarative processing is processing of specified actions or determination of specified target property values automatically, when certain input properties are set or change. This lesson provides an introduction to declarative processing and different types of declarative rules. It also covers the creation and usage of Declare Expression rules and usage of Declarative Rules Inspector.

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

- Understand Declarative Processing
- Understand different types of Declarative Rule Types
- Use and create Declare Expression Rules and use Declarative Rules Inspector

Declarative Processing

Computational relationship among property values

Line Item Cost = **Quantity** x **Unit Price**

When a **Line Item** is added to a Purchase Request that is part of the Program Fund, check if **funds are available** in that Program.

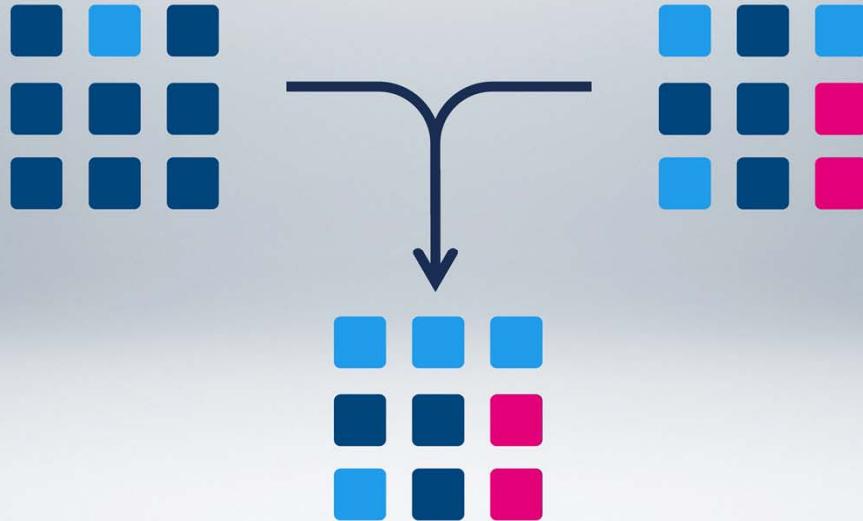
Do not allow **loans longer than three years** for **vehicles more than five years old**.

Declarative processing helps to simplify our application by describing a computational relationship among property values, such as:

- Determining a property value as a mathematical operation such as sum or the multiplication of other property values
- Performing an action when a property value changes
- Restricting the allowed value of a property based on the value of another property

In each case, we have one or more input values. These values are used to determine the value of another property.

Declarative Processing



Update values automatically as inputs change

The primary benefit of declarative processing is that PRPC processes the relevant actions and sets of properties automatically. They run automatically when a value is set or changes.

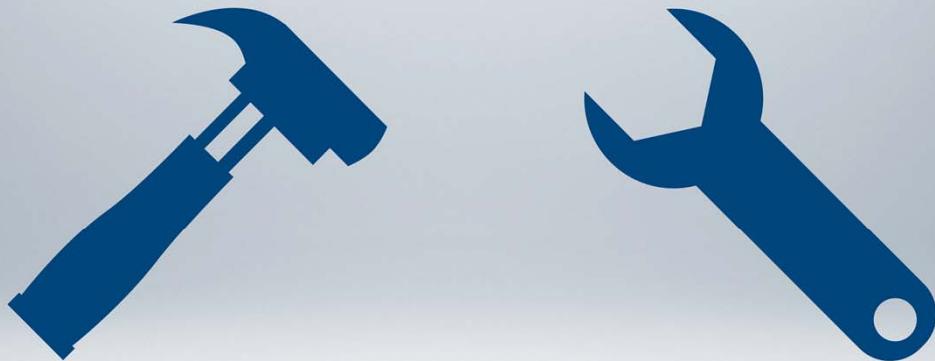
Declarative Processing



No need of explicit programming
configuration

The declarative rules do not need to be called explicitly or run in a sequential fashion. This allows us to express the relationships between property values, without the need to create other rules – such as data transforms or activities – to perform related actions

Declarative Processing



Faster Development
Easier Maintenance

Since the declarative rules provide a way to determine property values automatically, the system generates the programming behind the scenes and the declarative rules model relationships between properties. Using a declarative rule speeds development and simplifies maintenance.

<u>Assets:</u>		<u>Liabilities:</u>	
Checking account	1259.63	Credit card	2257.49
Savings account	4729.11	Credit card	6123.41
CD	11865.21	Charge card	653.21
Investment account	37963.78	Unsecured loan	9758.23
Vehicle	6429.00	Vehicle loan	2921.73
Total assets	62066.73	Total liabilities	21714.07
Net worth	40352.66		

Let's say we're totaling an individual's assets and liabilities to determine their net worth. Each time we add – or remove – an asset or a liability, or update its value, we want the respective total value and the overall net worth to adjust accordingly. If we determine these values declaratively, PRPC automatically computes these totals every time we add, remove, or update one of the input values.

Declarative Rule Types

Category	Purpose	
Constraints	Decision	Enforce relationship between properties
Declare Expressions	Decision	Set a property value with an expression that references other rules.
Declare OnChange	Decision	Run an activity at activity step boundaries when a property satisfies a specific condition
Declare Trigger	Decision	Run an activity when a class instance is created, updated, or deleted
Declare Index	SysAdmin	Expose (in the database) properties on aggregate pages for faster access
Data Pages	Data Model	Create clipboard pages containing relevant data

© 2011 Pegasystems Inc.

PRPC provides six declarative rule types:

- **Constraints** — define and enforce comparison relationships among property values. Constraints can provide a declarative form of property validation every time the property's value is "touched", rather than the explicit validation provided by the property definition or validation rules. The system evaluates constraints automatically each time a property identified in a constraint is changed.
- **Declare Expressions** — define automatic computations of property values based on expressions. This expression may consist of property references, decision rule results, or other declare expressions. We'll discuss declare expressions in more detail later in this lesson.
- **Declare OnChange** — performs an action, specifically an activity, automatically when the value of a property that we specify to be monitored, changes. For example, we can use a Declare OnChange rule to increment a counter each time a work item is updated. Declare OnChange can also force all processing on a case to be suspended. This may be needed when a case suddenly violates some business policy or federal guidelines.
- **Declare Trigger** — causes an action, or specifically, an activity to run when instances of a specific class are created, updated, or deleted in the database. For example, we can use a declare trigger to send an email whenever a customer case is updated, perhaps with a changed address or preference.

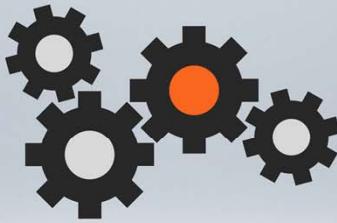
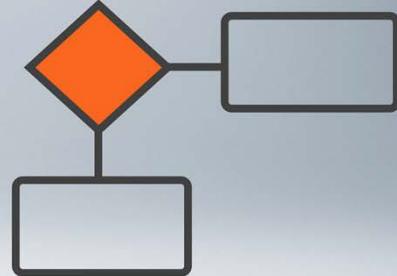
□ **Declare Index** — defines the criteria that PRPC uses to automatically maintain the index instances for faster access. An index can improve search and reporting access for properties that cannot be exposed as database columns. This occurs when properties are embedded within an aggregate property, in other words, when the properties are contained within a page list or page group property. Indexes are created automatically by the Property Optimization tool. PRPC saves indexes as instances of concrete classes derived from the Index- base class.

□ **Data Page** — used to cache data on demand to a clipboard page that can be used by one or more applications. A data page defines its source data, scope, refresh strategy, editability, and structure. This declarative rule should be the primary means of sourcing data and will be discussed in more detail in the “Accessing Data in User Interface” lesson.

Evaluation of Declarative Rules



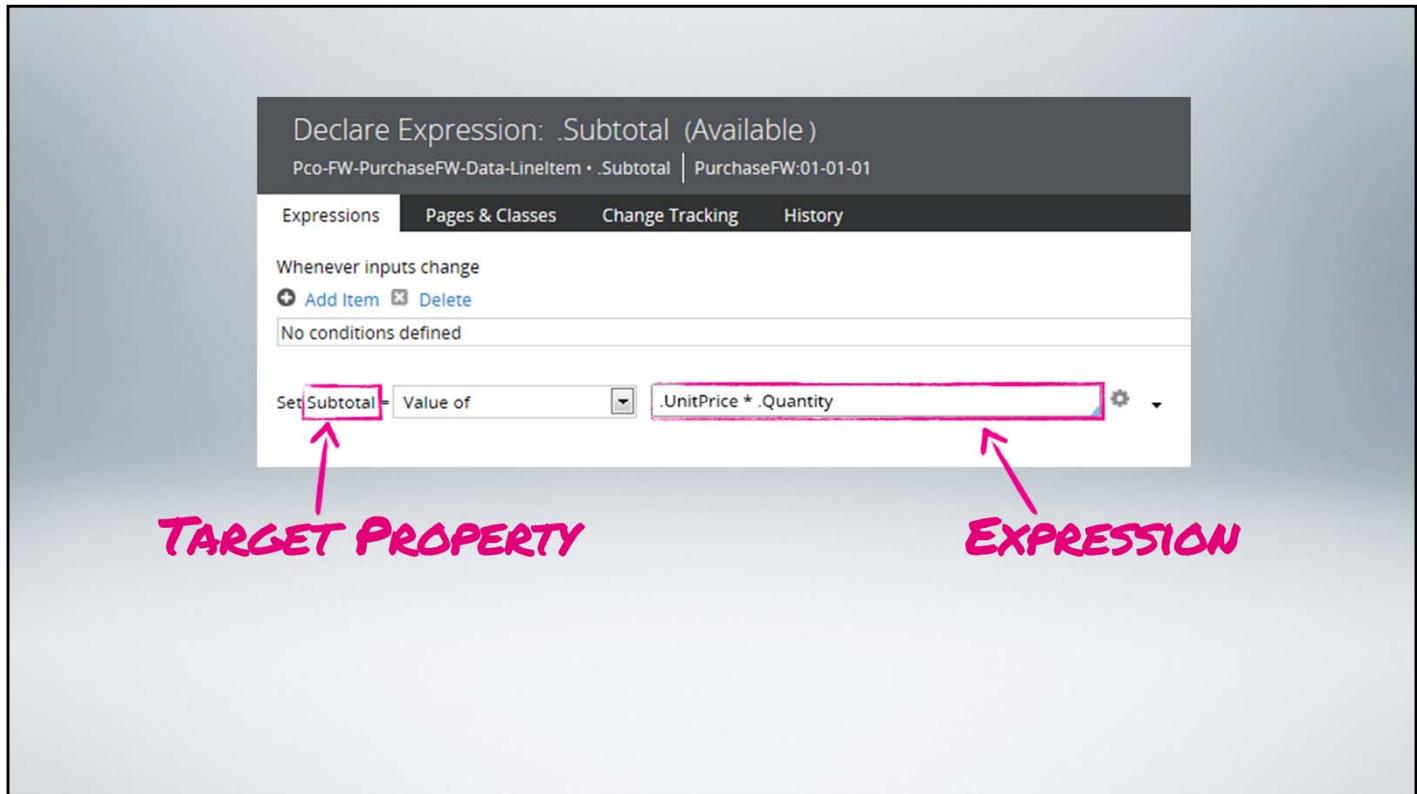
Form submission

Each step in activity
or data transfer

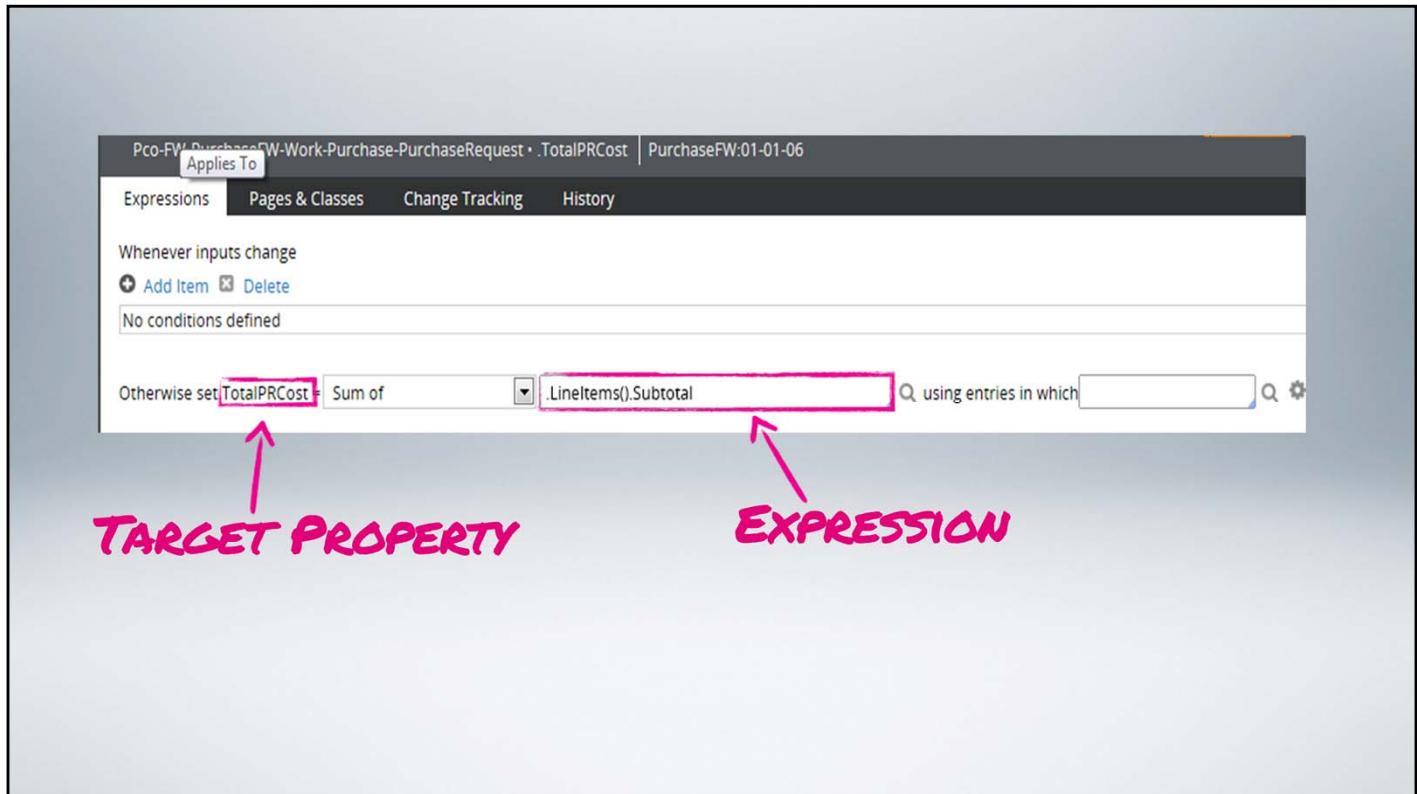
Flow execution

While there are some specific nuances with different declarative rules, in general they are triggered when one or more property values change or as is the case with data pages, are simply referenced. Below are some events that may trigger the declarative rules.

- When a user submits a flow action, the evaluation occurs at the conclusion of input processing.
- As each step in a data transform or activity runs.
- During flow execution, the evaluation occurs as control advances from one task (one shape on the flow diagram) to another.



Declare Expressions contain a target property and an expression. Any change to input values in the expression causes PPRC to automatically run the expression in the background and update the target property. This occurs independently of any user interaction and is also independent of specific calls made in an application. Here the input values are unit price and quantity and the target property is sub-total for each line item.



A target property can be an input property for another declare expression. For example, a Purchase Request's total cost is the total of all the line item's sub-totals. Here, each line item's sub total property is the input for the target property, purchase request's total cost.

Demo



Declare Expressions

Let's take a look at this in action. When we add a new line item with quantity and unit price, the subtotal of the line item gets updated. Also the total purchase request cost gets updated when a new line item is added and its sub-total is updated.

Also, when we change the quantity, we can see that line item's subtotal and the total purchase request cost are updated again.

Let's see how we can add a declare expression rule for a target property. Let's create a new Declare Expression rule to calculate the purchase order total cost, which is the sum of the line item total cost, the tax and the shipping cost. From the Case Designer, open the outline view by clicking the "Configure Process Details" link under the Order Placement Stage of Purchase Order Case. For any assignment, we can add an action that can be taken by the assignee by clicking the Add Action Link. We can also add relevant fields that we want to display in the section by clicking Add Fields. This section will be included in the flow action.

Since a default action is already in place for this assignment, we don't need to click the "Add Action" link to add more actions, unless we need more flow actions. Edit the default flow action and change the name to "PlaceAnOrder." Tab out of this text box and we see that the "Add Fields" link is now active.

Click the Add Fields link to add new properties. This opens a dialog box where we can add/edit properties that we want to display in the section which will be included in the flow action rule, as mentioned before. We can mark the added field as calculated, required and read only. We can also add more fields by clicking the Add Fields link.

Add a new field "PurchaseOrderTotal" by typing that name in the text box and change the data type to decimal and then select the "Calculated" check box. As soon as this check box is selected, we see the "Define Calculation" link. We can use this link to create the declare expression for the property we added.

Click the Define Calculation link to create the declare expression for this property. This opens the rule in a new tab. The target property where we want to store the total purchase order cost is, PurchaseOrderTotal and is being set to the value of the pyLabel by default.

If the rule is not already checked out, check it out to edit the expression. Enter the expression to be calculated as

the value of adding the line item total cost, tax and shipping cost, then click Save to save the rule.

Click the “Change Tracking” tab. Note, that the target value is calculated whenever the input value changes. So, whenever any one of the three input values change, the total purchase order cost is calculated automatically by the system. This default technique of calculating the value of the target property is called “Forward Chaining.” Forward chaining means that PRPC computes the target property value automatically each time when any of the input values in the expression change. To reiterate, the Forward chaining technique is set in the “Change Tracking” tab by using the default “Whenever inputs Change” option.

Close this rule to return to the Case Designer tab. We see the link next to the “Calculated” check box is changed from “Define Calculation” to “Open Expression.” We can click on “Open Expression” if we need to make changes to the calculation.

Click OK to close the “adding fields” dialog box. Click Save in the outline view. This saves the changes to the flow as well as the newly created flow action and the section rule with the field(s) added. Click the Open Section link to open the section rule. This section has only one property, “Purchase Order Total” to display. This rule can be checked out and modified to display more properties and modify the layout, if necessary.

As we have learned, declare expressions update a calculated value (the target property) when any value in its expression changes. To automatically update the UI, an Enable Expression Calculation checkbox is selected on the Advanced tab of a harness or HTML tab of a flow action. Enabling it in the harness such as Perform automatically updates all the user interfaces involved in the flow irrespective of the setting in the flow action. Disabling that check box in the harness and enabling that in a specific flow action updates that flow action user interface only.

During the development phase, if any PRPC architect wants to troubleshoot and know what properties are set through declarative rules, they can use Declarative Rules Inspector. The Declarative Rules Inspector shows us what properties are set declaratively. From the PRPC Developer Portal footer, select Inspection Prefs > Declaratives.

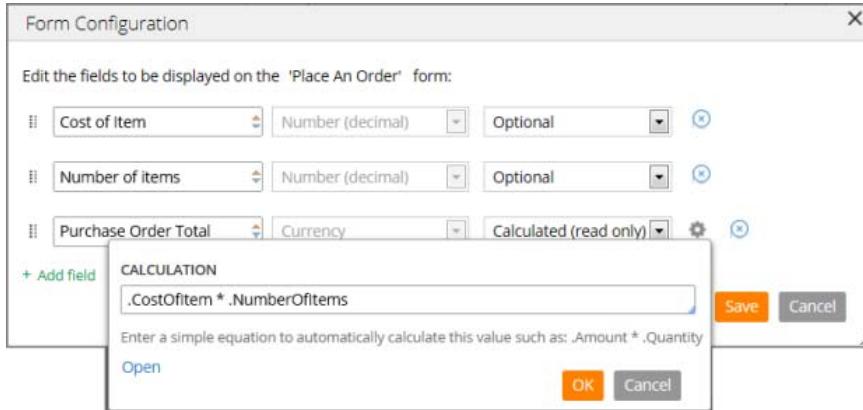
When the inspector is enabled, a checkmark appears next to the Declaratives entry on the Inspection Prefs menu. When we create a new purchase request case, a red D appears next to the “sub total” field and the “total purchase request cost” field to indicate the use of a declarative rule for those fields. Hovering over the D shows the target property used in the declarative rule.

We have learned how declarative processing can be useful with its automatic computations and actions, and different declarative rule types and their purpose. In addition, we learned how to create a Declare Expression rule with the Forward Chaining configuration, and how they can be used to calculate values automatically.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- Update to form configuration dialog calculated field
 - Create simple Declare Expression directly from the mode field



- Declaratives indicator is now included with UI Tree

Previously, we selected **Declaratives** from the Inspection Preferences menu in the Developer toolbar to see a "D" indicator next to fields that are calculated declaratively. This is now done through the new UI Tree developer tool. When the UI Tree is toggled on, and we hover over a calculated field, a D icon indicates it is the result of a Declare Expression.

The screenshot shows the Pega UI Tree developer tool interface. At the top, there is a form with fields: 'Expense Purpose*' (New Computer), 'Currency - .TotalExpense' (with a small 'D' icon indicating it's a declarative calculation), and 'Total Expense*' (\$2,814.00). Below this, there is a table with rows for 'Transaction Date' (10/6/2014), 'Item Description' (Monitor), and 'Line Cost' (214). Further down, another table row is shown for a second item: 'Transaction Date' (10/6/2014), 'Item Description' (Desktop), and 'Line Cost' (2600). At the bottom of the UI Tree panel, there are tabs for 'Tracer', 'Clipboard', 'UI Inspector', 'UI Tree' (which is currently selected), and 'Performance'.

Exercise: Determine Offered Salary



Conditional Processing

Conditional processing of a case is processing of a stage or a step or an alternate path in a flow based on certain condition(s) being true. This lesson provides an introduction to conditional processing and how When rules can be created, configured and used for conditional processing.

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

- Understand conditional processing and When rules
- Use and create When rules

Conditional Processing

Conditional Processing	Condition
Get approval from IT department manager, if a hardware or software item is being purchased.	Is the item to be purchased hardware or software not currently in stock?
Skip the quoting process if the purchase request is made from an existing preferred vendor	Is quoting process required?
Management approval is not required if the cost of the purchase request is less than \$100.	Is the total cost of the purchase request less than \$100?

Conditional processing helps to process or skip a step, process or skip a stage or take an alternate path in a flow based on certain conditions. This might occur when we:

- Need to get approval from an IT department manager, if the purchase request consists of a hardware item or a software item. They may know whether they have the item in stock or if they need to order the item from a vendor and hence we want to get their approval.
- Want to skip the quoting processes because the vendor we are going to use is a preferred vendor with a negotiated price.
- Do not need to get a manager's approval because the purchase request's total cost is less than \$100 and those cases can be processed without an approval.

In each of these scenarios, further processing is based on a condition. In the first scenario, the condition is based on whether the request includes a hardware or software item.

In the second scenario, the condition is based on whether or not a quote is required.

And, in the third scenario, the condition is based on whether the total cost of the purchase request is less than \$100 .

The business requests...	We configure our when rule
Does the purchase request consist of IT line items and hence require an IT Manager approval? •Yes (true) •No (false)	When LineItem Category = Software or Hardware •True = Consists one or more software or hardware line items. •False = Does NOT consist any software or hardware line item.

These types of conditions can be checked using a When rule. A When rule allows us to define a True-False condition based on comparing one or more properties with any other value or with other properties.

The return result is either True or False.

Now let's see how we can create a When rule and how to use them.

Usage of When Rules



Alter Stage behavior of a Case.

Alter the Step behavior of a Stage.

Alter the Flow behavior listed in a Step.

When rules can be referenced in several other Rule types. Here we are going to look at how they are being used for conditional processing of cases. We can also use a When rule to conditionally display UI elements, which will be covered in some other lessons in this course.

We use skip a stage if the When rule returns True or process the stage if the When rule returns False.

DEMO CLIP - Click on the Quote stage and select “Configure Stage Behavior.” We see a When rule “No Quote Requested” has already been entered for the skip stage When field. If this When rule returns True, the entire Quote stage and all the steps in that stage will be skipped.

We use process a step if the When rule returns True or skip the step if the When rule returns False.

DEMO CLIP - Get IT Approval is a process and it is listed as a step in the Approval Stage, for the Purchase request Case. This process is required only if the item purchased is a hardware or software item. We see the When rule “ITApprovalRequired” has already been entered in the step configuration. If this When rule returns True, this step is processed, or else it is skipped.

This When rule returns True if the category of the item purchased is HW or SW, indicating it's a Hardware or Software item respectively.

We have looked at how the stage and step behaviors are altered with When rules. We can also use a When rule to take alternate paths in a flow listed in a step.

Demo



Conditional Processing

The Program manager approval assignment can be bypassed in the program manager approval flow, as the business has indicated that there is no need to get this approval when the total cost of the purchase request is less than \$100.

Click on “Configure Process Detail” under the Approval Stage to open the flows in the outline editor. Add a decision shape in “Get Program Approval” flow. Change the decision shape type to fork so that we can enter or select a When rule on the connector. Enter the step name “Is PR Cost LT 100?”

Move the existing connector and add a connector, so that the decision shape of fork type is in the middle of the start shape and the assignment shape. Add another connector from the decision shape to the end shape, edit the connector and change the condition type to When. Enter an appropriate likelihood and enter “IsPRCostLT100” for the When rule field. Click on the magnifying glass to create this When rule.

The record identifier, that is the rule name is, “IsPRCostLT100.” In the new rule dialog for Short Description, we enter “Is PR Cost LT 100”.

Our When rule only contains one condition, therefore, we’ll use the Quick Create feature to create this When rule.

In the When Expression field, we select the TotalPRCost property and enter “Less Than 100.” We then click Quick Create.

The When rule is created and we are back at the Connector properties dialog box. Click OK to return to outline view. Edit the connector between the decision shape and the assignment shape and change the condition type to Else. If the purchase request cost is not less than \$100, this Else condition flow path is processed. Click OK and click Save to save the changes. Click “Back to Stages” to return to stages.

With the decision shape and When rule, we see how we can take alternate paths in a flow. The bypassing

condition can be for an assignment or any number of shapes in the flow for that matter.

Open the flow in outline view and open the When rule we just created by clicking on the magnifying glass from the connector properties. This When rule will return a value of True if the total purchase request cost is less than \$100 and a value of False if the total purchase request cost is more than \$100.

To create a When rule that evaluates multiple conditions, it's typically easier to use the Conditions Tab. For example, if we want to create a When rule that returns True if the total purchase request is within a range, say in between \$100 and \$5000, we can create a new When rule with the name "IsPRCostWithinRange" and click on the magnifying glass. Enter a short description and click Create.

In the Conditions Tab, double click to add the first condition. For the first value, select "TotalPRCost" from the list of smart prompt values. For the relational value, select "greater than or equal to" sign. For the second value, enter 100. Click OK to finish adding the first condition.

To add the second condition, click on Actions and then Insert Condition.

For the first value, select "TotalPRCost" from the list of smart prompt values. For the relational value, select "less than or equal to" sign. For the second value, enter 5000. Click OK to finish adding the second condition and click save to save this When rule.

We have created a When rule that returns True if the total purchase request cost is within the range of \$100 and \$5000. If it does not fall within that range, the When rule returns False.

We have learned how we can use a When rule to alter Stage behavior, Step behavior, and Flow behavior. In addition, we learned how to create a When rule using the quick create option for simple conditions and how to use the condition tab to set multiple conditions.

Exercise: Skip Screening Process



Optional Exercise*
Request Technical Sample for Engineering Positions
**Consider doing if time permits*



Automating Business Processes using Decision Rules

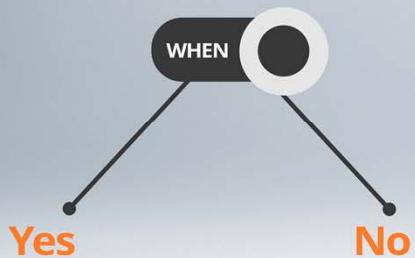
Two other types of decision rules, decision tables and decision trees, are used to automate decisions to return any number of results. This lesson will cover the details of both decision tables and decision trees and their usage.

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

- Create a decision table
- Create a decision tree
- Use a decision table or tree in a flow
- Use a decision table or tree declaratively

When Rules

↔
Yes/No Logic



Do we require additional approval?

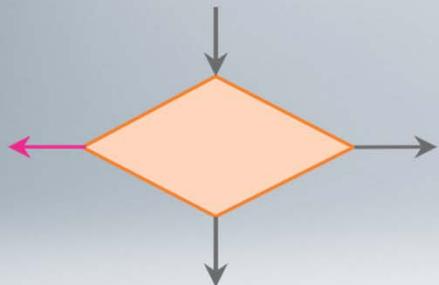
Is free shipping available for this offer?

Can I update my selections now?

In the previous lesson, we learned about basic decision logic with a when rule. When rules are well-suited to answering questions with a true/false answer, such as "Does this purchase order require additional approval?" and "Is this order eligible for free shipping?"

Complex Decisions

When "Yes" and "No" don't suffice



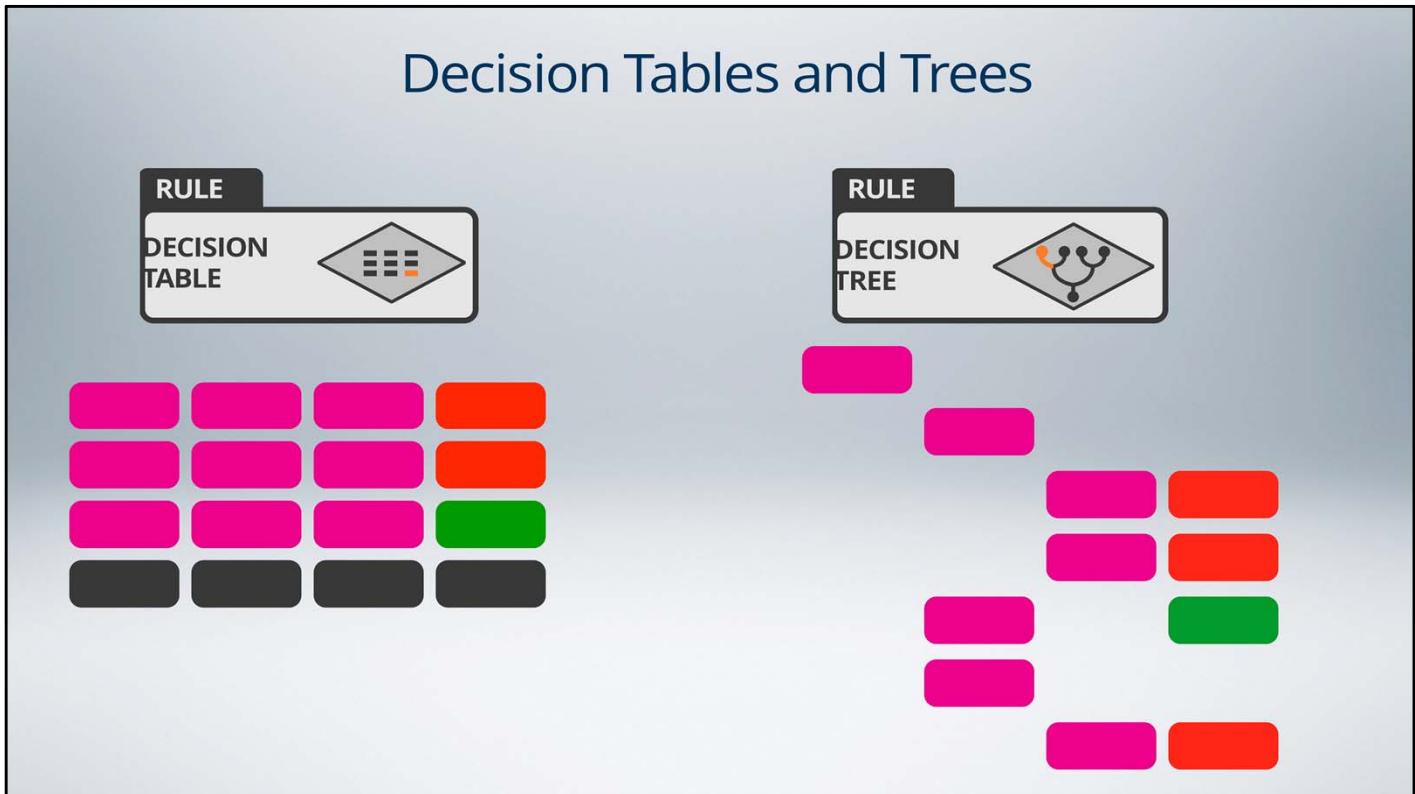
Which promotion should we offer?

How many claims have they filed?

Which tax rate should we use?

But, if we're not asking a yes/no question, then a when rule doesn't really help us. For example, if we need to answer the question "What promotional offer should we provide to the customer?" we aren't looking for a yes or no answer.

Decision Tables and Trees



To answer questions such as this, we need to use a different type of decision rule – and we have two options to choose from: decision tables and decision trees. As their names suggest, decision tables are formatted as a table of conditions and results, while decision trees are organized like a tree, starting from common conditions and extending outward to more-specific conditions and their results. Despite their differences in format and evaluation methods, decision tables and decision trees are fairly similar – both evaluate a series of conditions, and return the result that corresponds to the set of conditions that are true.

Uses of Decision Tables and Trees



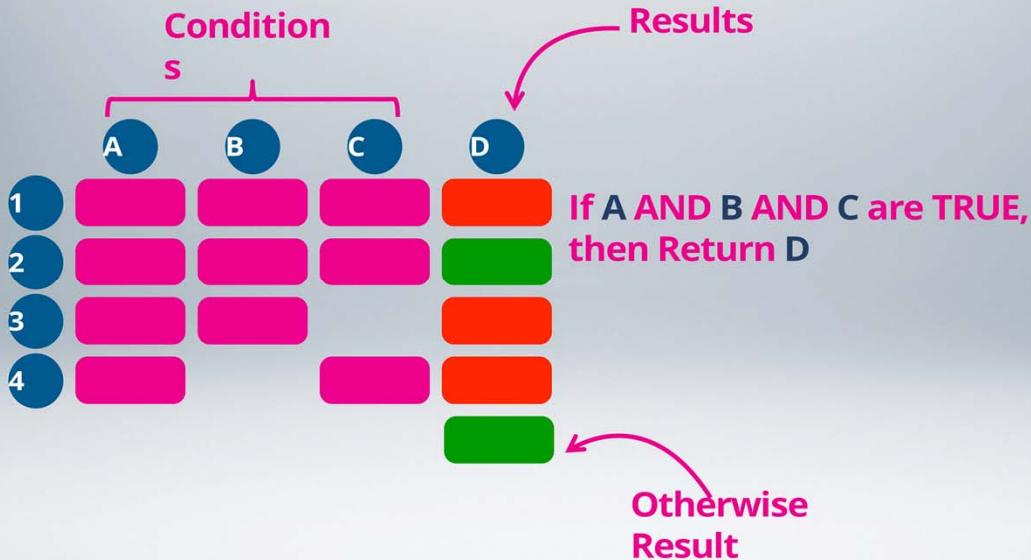
Decision tables and decision trees are frequently used in flows to direct processing of a case. We can use these rules with a Decision shape, to define the conditions under which PRPC selects the connector to advance the case.

Decision tables and trees can also be used declaratively. We can define a Declare Expressions rule to evaluate a decision table or tree, and use the result to define a property value.

In fact, we can use decision rules – tables, trees, and when rules – almost anywhere in a case where we need to enforce a policy decision. We can even chain decision rules to model complex decision processes.

For now, however, we'll focus on the simplest use of a decision table and decision tree – directing case processing in a flow.

Decision Table Evaluation



Let's first learn how to create a decision table.

A decision table consists of a table of conditions and results. We define a set of conditions – for example, property values that must fall within a certain range – and the results to return when the conditions are true. Each set of conditions contains a corresponding result. We can add columns for each condition we want to test against – either a property reference or an expression – and rows for each combination of conditions we want to test.

When PRPC evaluates a decision table, it starts with the top row and evaluates each condition in the row. If all of the conditions are true, PRPC returns the result for that row. If not, it advances to the next row, and evaluates the conditions in that row. And if none of our combinations returns a result, PRPC returns the otherwise result at the bottom of the table. This allows us to ensure that our decision always returns a result.

We can also set property values based upon our results, though we won't use this in our solution.

Demo



Creating a Decision Table

Let's see how we can enforce a policy with a decision table.

For our Purchase Request case, the behavior of our solution depends upon whether we approve a vendor for the request in the Wait for Approvals process – a decision that results in one of three possible outcomes:

- The vendor has been rejected, and the requestor must return to the Request Entry stage to select a different vendor.
- The vendor has been approved, and the request proceeds to the Quote stage.
- The vendor is under review, and the case must be routed to another stage, called Vendor Review. These requests may or may not advance after additional evaluation of the vendor.

Since we already modeled this process – with decision outcomes, but no decision logic – we don't need to add a Decision shape to the process. So let's configure our decision table and add it to the existing decision shape.

We need to enter a description of the table. We also need to specify the class and RuleSet for the table. Since we haven't really discussed these concepts, we'll accept the default values.

Once we create our decision table, we can specify the conditions we want to evaluate. So, our first step is to add the Has Rejected Vendors property to the table.

Now, we know we need to return one of three results. We already have two result fields, so we need to add

one more row to the table. We can add as many rows and columns as we need to model our decision, but in this case, one additional row is all we need to add.

Now that we have enough rows, let's create our conditions. The Has Rejected Vendors property can have three possible values: "Approved", "Rejected", and "On Review". So, let's select Approved in the first row, and Rejected in the second. The otherwise row catches any remaining results – in this case "On Review".

Lastly, we need to specify the result for each row. Our possible results are Approved, Rejected, and On Review, and they correspond to the allowed values for the Has Rejected Vendor property.

Now, when our solution evaluates a purchase request, it directs the request to the corresponding processing step.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

Rule forms have been updated to provide better east-of-use

- Identifier label reflects rule name (Decision) in place of generic text (Step Name)
- Vertically stacked sections on the form replace tabbed sections in previous form

Decision Properties

Decision: Approved Vendor ?
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*: VendorApproved

Result in:

Audit note:

Enable link if using breadcrumb trail

Specification
How should this decision should behave?

Specification: Approved Vendor ? Specification Actions ▾ .

B I U Font Size A A ☰ Tx

Please enter a description for the step or click 'Choose Another Specification' from Specification Actions to reuse a specification

Advanced
What else can I do?

► STATUS

► TICKETS

Decision Table/Tree creation form has been updated

- Context can now be selected

Create Decision Table

DECISION TABLE RECORD CONFIGURATION

Label*: VendorApproved
Identifier: VendorApproved Edit
A short description or title for this record

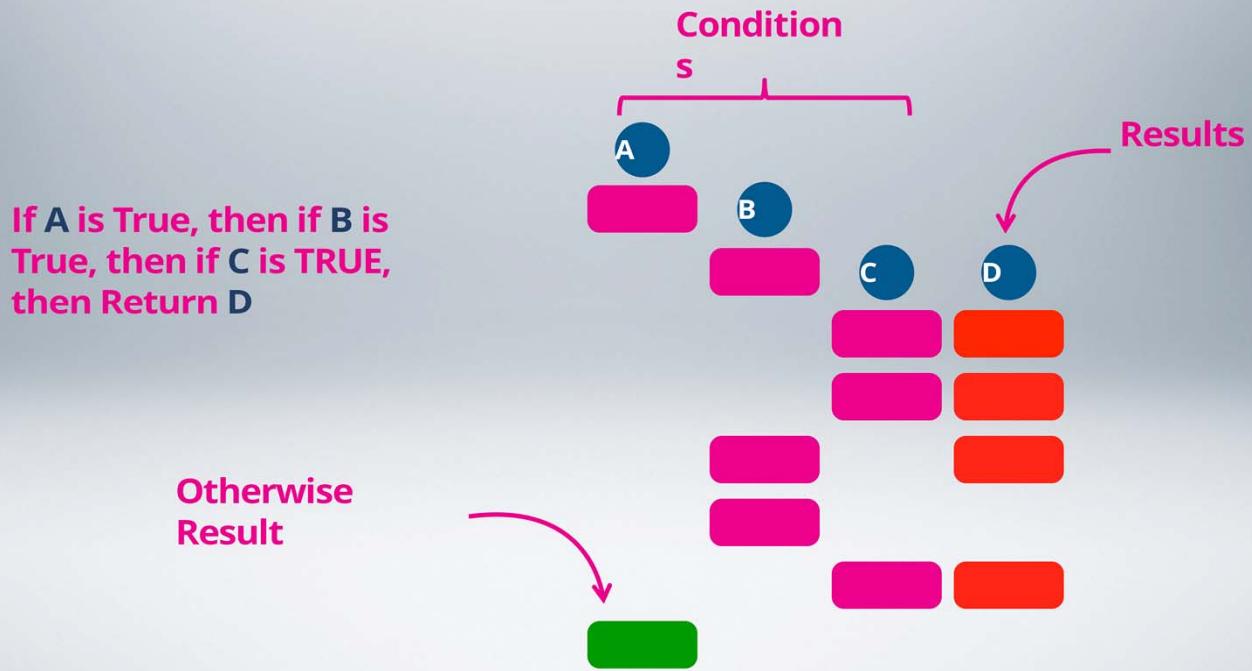
CONTEXT

Choose app layer: HR Services PegaRULES

Apply to*: SAE-HRServices-Work-Candidate
View all

Add to ruleset*: HRServices 01-01-01

Decision Tree Evaluation



Now, let's learn how to create a decision tree.

Like a decision table, a decision tree consists of a set of conditions to evaluate, and results to return. Unlike a decision table, however, a decision tree consists of a series of branches that contain more- and more-specific conditions, ultimately leading to a return value.

Demo



Creating a Decision Tree

For our Purchase Request case, we need to ensure that each request is reviewed by the appropriate individual. This decision is based on two factors: the total amount of the request, and the department against which it is logged:

- Company policy dictates that purchase requests of \$100 or less do not need any approval.
- The VP of Sales has requested that all purchase requests logged by members of the sales department be approved by him, and then the program owner.
- All remaining purchase requests must be approved by the owner of the program.

Let's see how we can enforce this policy with a decision tree.

Again, since we initially modeled our process with the decision, we don't need to add it to the flow. Instead, we can configure the existing decision and create the decision tree.

On the decision tree rule form, we want to start with the most-general condition. This is the \$100 threshold for requiring any approval, so let's add it to the decision tree.

Next, we need to add two sub-conditions: one to check if the requestor is part of the Sales department. If they are, then we want to return the result Sales, which corresponds to the Sales connector in our flow.

If the requestor is not a member of the Sales department, their request only requires the approval of the program owner, in which case we want to return the result Not Sales, which corresponds to the connector of the same name. Since, this result is returned for any department other than Sales, we can use an otherwise

condition. Since our decision involves multiple branches, we can add an otherwise result to each branch. We'll explore how these multiple "otherwise" results work shortly. For now, let's continue to implement the decision tree.

Finally, we need to return a result for purchase requests of \$100 or less. Since these requests do not need any approval, we can return a result of "Not needed", which corresponds to the connector of the same name. And since these are the only remaining purchase requests – we've returned results for all of the other requests – we can use the bottom-most otherwise branch to return this result.

Demo



Testing Decision Tables and Trees

So, our tree includes two “otherwise” conditions. Do we need both of them?

Well, if we were to remove the lower one, our tree would fail to return a result for any purchase request of up to \$100, which will return an error. Luckily, we can't remove it – the decision tree rule requires an otherwise condition at the end of the tree.

But what if we remove the upper one? What happens then?

In this case, any purchase request of more than \$100 that is not logged by a member of the sales department would not require approval. Rather than triggering a result in the correct branch – purchase requests that exceed \$100 – it would trigger then otherwise at the bottom, for which the result is “Not needed”.

This example illustrates the importance of testing decision rules. Since decisions are evaluated automatically – and can have a significant impact on case processing – it's important to ensure that the decisions we implement return the results that we expect.

Testing Decision Rules

Show Conflicts

Verify that all the rows/branches can be tested

Show Completeness

Verify that all possible permutations of conditions can be tested

Run the Rule

Test the decision with realistic data to verify the result

In PRPC, we can test decision rules several ways.

First, we can check for conflicts in our logic. This allows us to determine if we have configured our decision in a manner that prevents one or more of its rows or branches from ever being used. For example, if our decision table contains a row that returns a result if purchase requests exceed \$300, and another row below it that returns a result if purchase requests exceed \$500 for consulting services, the later row may never be evaluated, because the upper row includes that condition.

We can also check our decision for completeness. Similar to checking for conflicts, we want to identify conditions that won't be evaluated – but, in this case, PRPC tells us what conditions are missing from the decision, and adds them to the rule for us.

Finally, we can also test our decision. The Run option in the Actions menu of the decision rule form provides us with a form that allows us to enter values for the properties tested by our conditions. After we enter each property value, the form returns the corresponding result from the decision table or tree.

With these three options, we can verify our decision log before we add the rule to a flow, a declarative expression, or any other rule in our solution.

Exercise: Direct Flow Processing with a Decision Table

Exercise: Direct Flow Processing with a Decision Tree



Review of Automating Business Processes

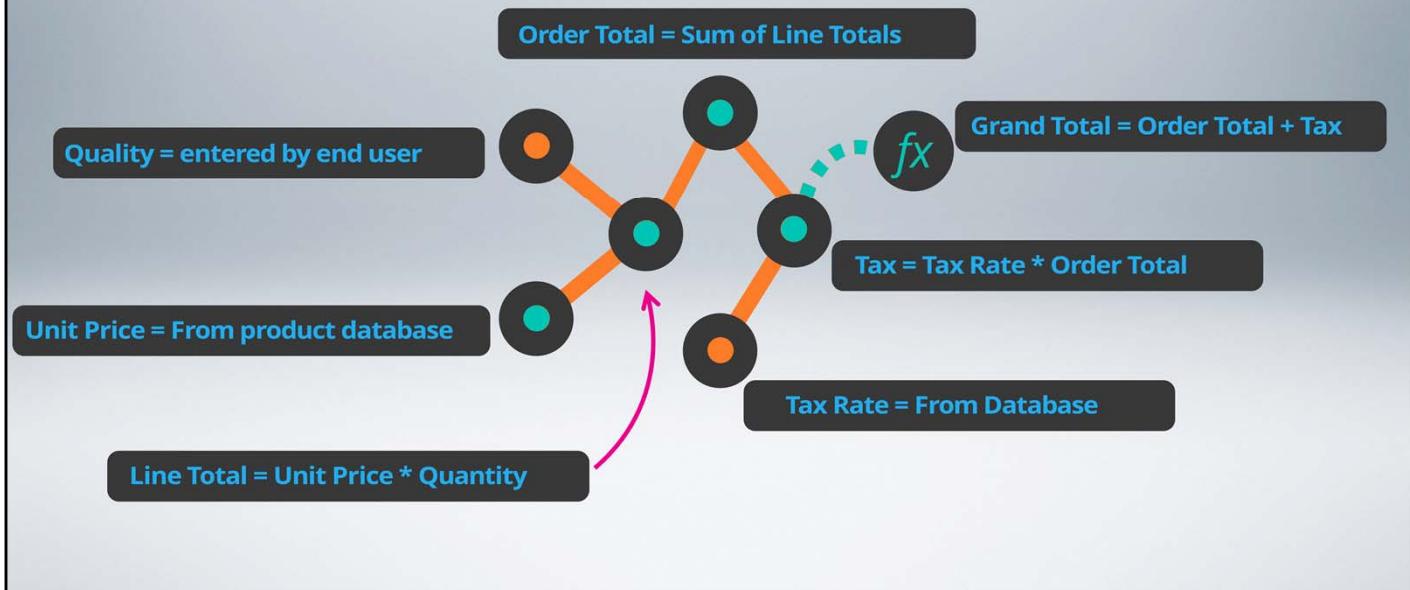
Use declarative and decision rules to help the business eliminate manual tasks and improve the efficiency of the case lifecycle

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

- Calculate property values declaratively
- Test binary conditions with a When rule
- Model complex business logic with decision tables and decision trees

Declarative Processing

Calculate - and Update - Property Values Immediately



In the past few lessons, we discussed methods for enforcing business policies – specifically declarative calculation of property values and automated decision logic.

Declarative rules evaluate automatically whenever a trigger value changes. We can delegate the responsibility for updating the value to PRPC. Through a mechanism known as a declarative network, PRPC can identify dependent properties and trace the dependency back to their source. And through another mechanism known as forward chaining, PRPC can automatically push changes to a source value all the way through the declarative network to any dependent properties. All of this happens automatically, freeing us from the need to anticipate changes and update values manually.

We explored these concepts with the Declare Expressions rule. Declare Expressions set the value of a target property according to a specified formula. We can add, subtract, multiply, and divide combinations of property values to model any calculations. We can even use declare expressions to evaluate other rules – such as decision tables and decision trees – as part of our expression.

Declarative Processing

Other forms of Declarative Processing



Constraints

Enforce relationships between properties



Declare Trigger

Run an activity when a class instance is created, updated, or deleted



Declare OnChange

Run an activity when a property value meets a specific condition



Declare Index

Index properties on pages to improve performance



Data Pages

Create pages of persistent data on the clipboard

We can also use declarative rules to validate property values, update database records, and perform other actions. Although these capabilities are more sophisticated, and best left to more advanced users – in some cases Senior System Architects (SSAs) and Lead System Architects (LSAs) – the principles behind them are the same as those for declare expressions.

Decision Rules

Codify Business Logic

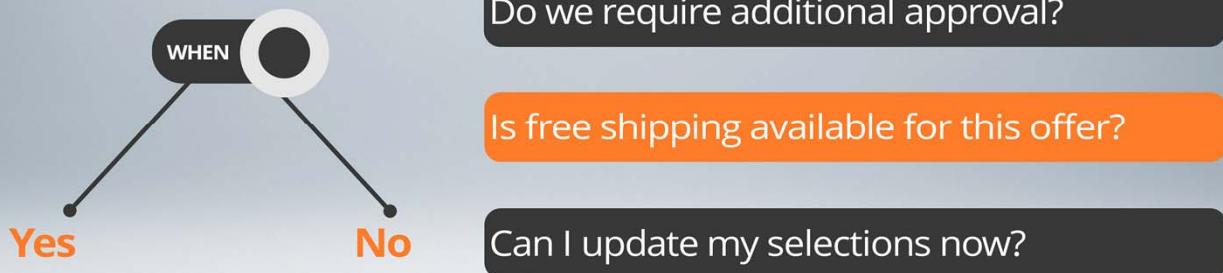
Model business decisions and apply them automatically



We also explored rules that allow PRPC to make decisions on behalf of users. Rather than creating assignments for end users to evaluate data and choose the appropriate response, we can model the decision logic with PRPC rules – When rules, decision tables, and decision trees.

When Rules

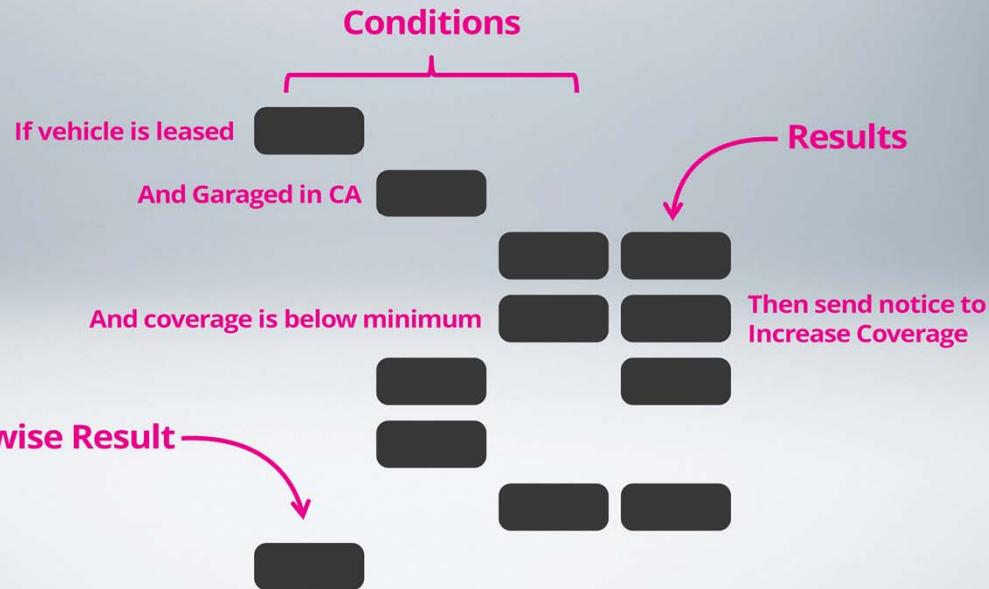
Yes/No Logic



When rules allow PRPC to answer Boolean-style questions – questions that require a yes/no, true/false, or on/off response. We used a when rule to skip a portion of our candidate process under specific circumstances. We can also use when rules to direct cases through a process flow, and even enable and disable options within our UI.

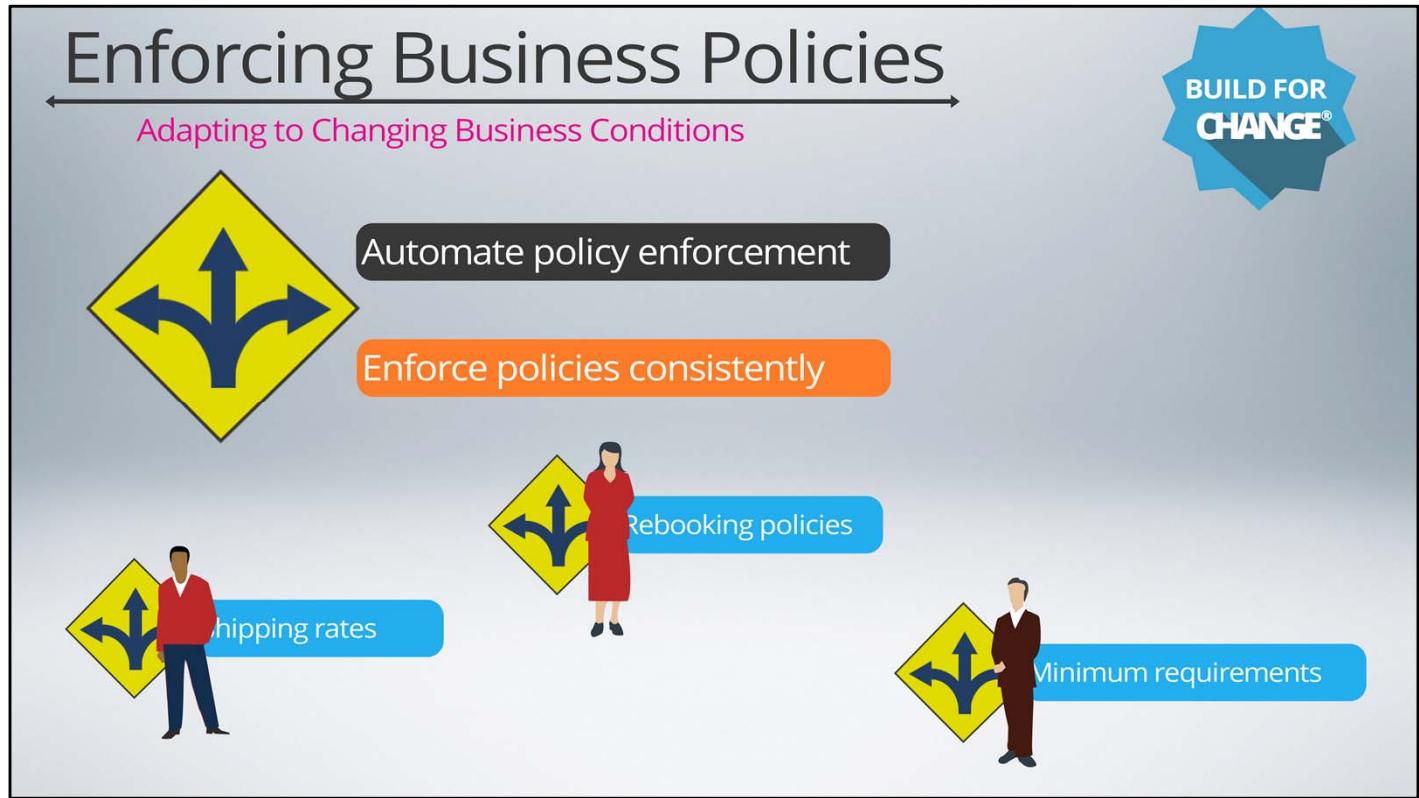
Decision Tables and Decision Trees

Business-friendly policy enforcement



And we also learned about decision tables and decision trees. These rules allow us to assemble combinations of conditions to test – either in a grid or a branching structure – and allow us to designate values to return for each combination of conditions. Decision tables and decision trees allow us to model more complex decisions, such as “If a vehicle is leased, and garaged in a particular location, and the driver does have an insurance policy with a satisfactory level of coverage for damage to other people’s property, then they must increase the coverage on their insurance policy.”

We used decision tables and trees in process flows to direct cases to specific assignments. We can also use decision tables and trees to determine case routing. The ToDecisionTable and ToDecisionTree routers allow us to use decision rules to route assignments.



Unlike the other rules we've covered in this course – properties, UI rules, and flows – policy rules allow us to model policy logic in our solution. By doing so, we can automate policy enforcement, allowing business users to process cases more quickly and consistently than if these policies were enforced manually.

Policies are the most reactive elements of our solutions, as they need to adapt to changing business conditions – often with little or no warning. As a result, we want the business to remain engaged in the maintenance of policy rules. One way to do this is through delegation. We can delegate these rules to end users – generally line managers – allowing them to update the policy logic in response to changes in business conditions, rather than wait for an updated solution to be developed, tested, and released. This is just one of the ways in which Pega solutions allow us to Build for Change®. Delegation should be performed by SSAs and LSAs, and rules should only be delegated to a small, approved group of trusted users.

Module 09: Advanced Case Processing

This lesson group includes the following lessons:

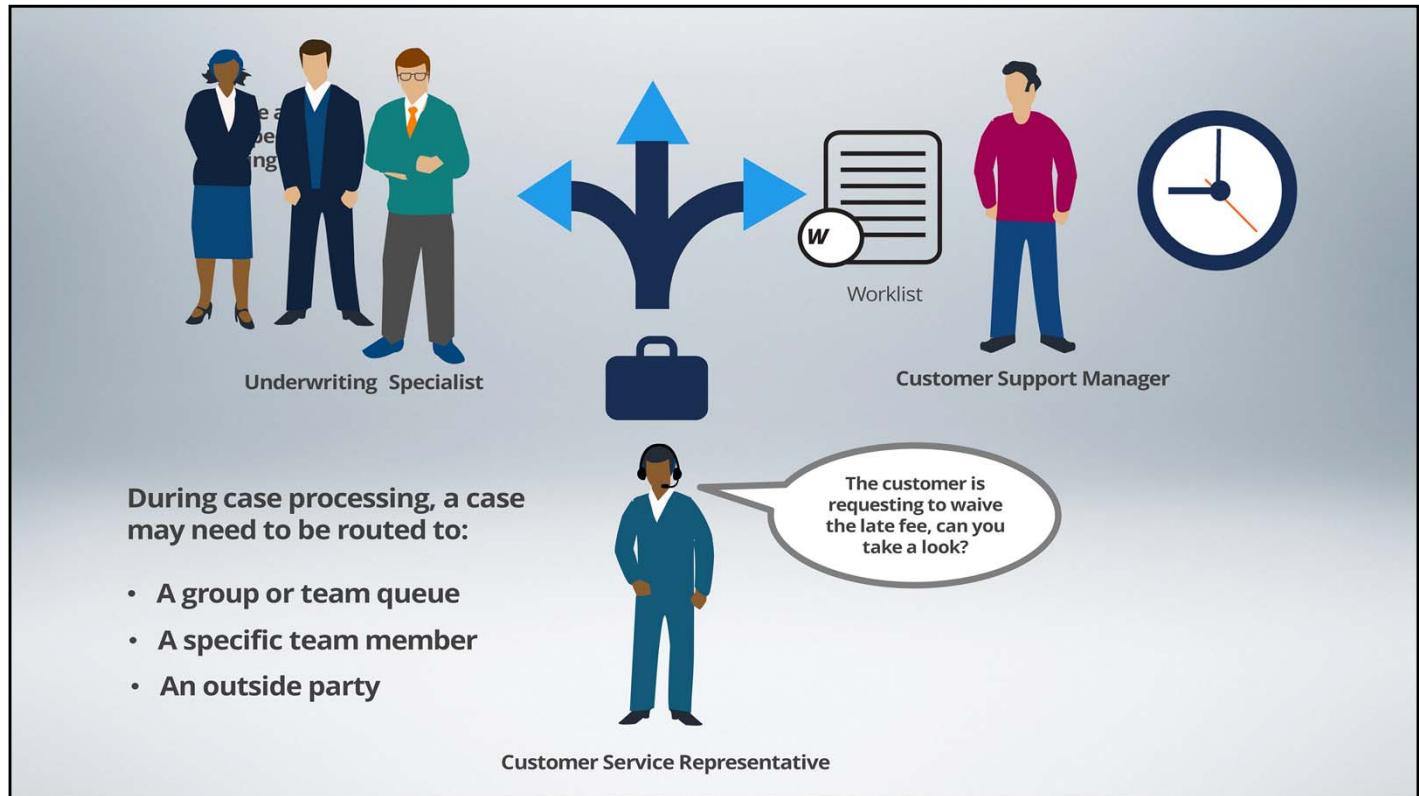
- Routing Cases
- Case Stage Configuration
- Creating an Optional Process
- Introduction to Task Based Smart Shapes
- Creating a Case Level SLA
- Review of Advanced Case Processing

Routing Cases

When a case is being processed, various actors perform specific tasks, which necessitate us to route cases to an user or a work queue that can assign to a group of users.

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

- Understand the importance of assignment
- Define the terms – work list and workbasket
- Know how to route cases to other operators or queues
- Leverage standard router rules
- Apply decision based routing



Routing cases is a key concept in case processing.

In most enterprise applications, a case is worked on by more than one operator until it gets resolved - First a customer service representative (CSR) receives the call from a customer, he enters the information collected from the customer and then figures out that he wants his manager to work on the customer's request.

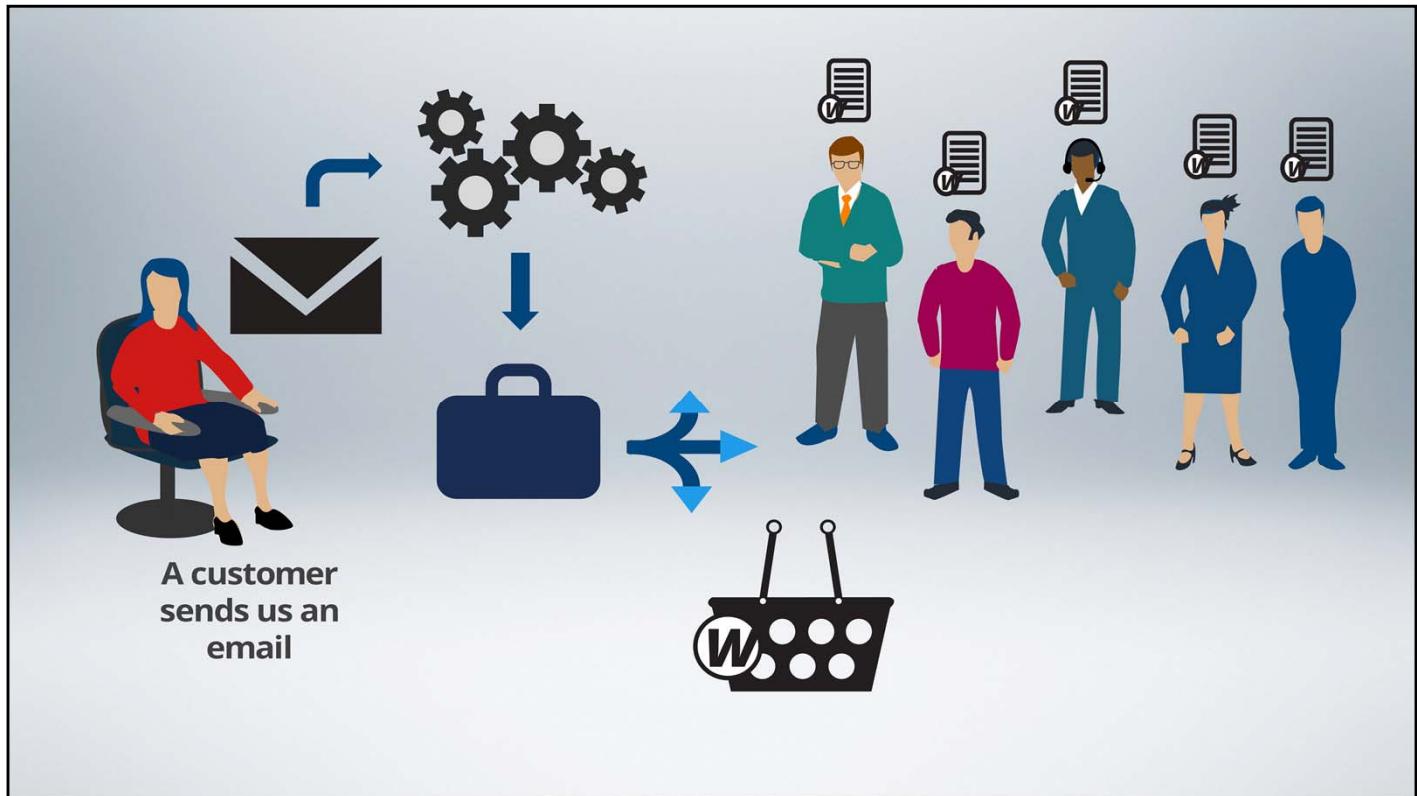
The case is then routed to the manager who is now tasked to perform an action on the case.

When processing a case it has to be routed to the manager or a specialist like an underwriter for performing a specific action.



Specialists such as an underwriter are usually not a single person but a group of people who are trained to perform the task. In that scenario the case gets assigned to a routing queue instead of one person. The case is then moved from the queue to one of those specialists.

Routing queues in PRPC are known as workbaskets, we will learn more about this in a little bit.



Let's look at another example - when a customer sends an email to the customer service department, the system parses the email and creates a case for a person to work on it.

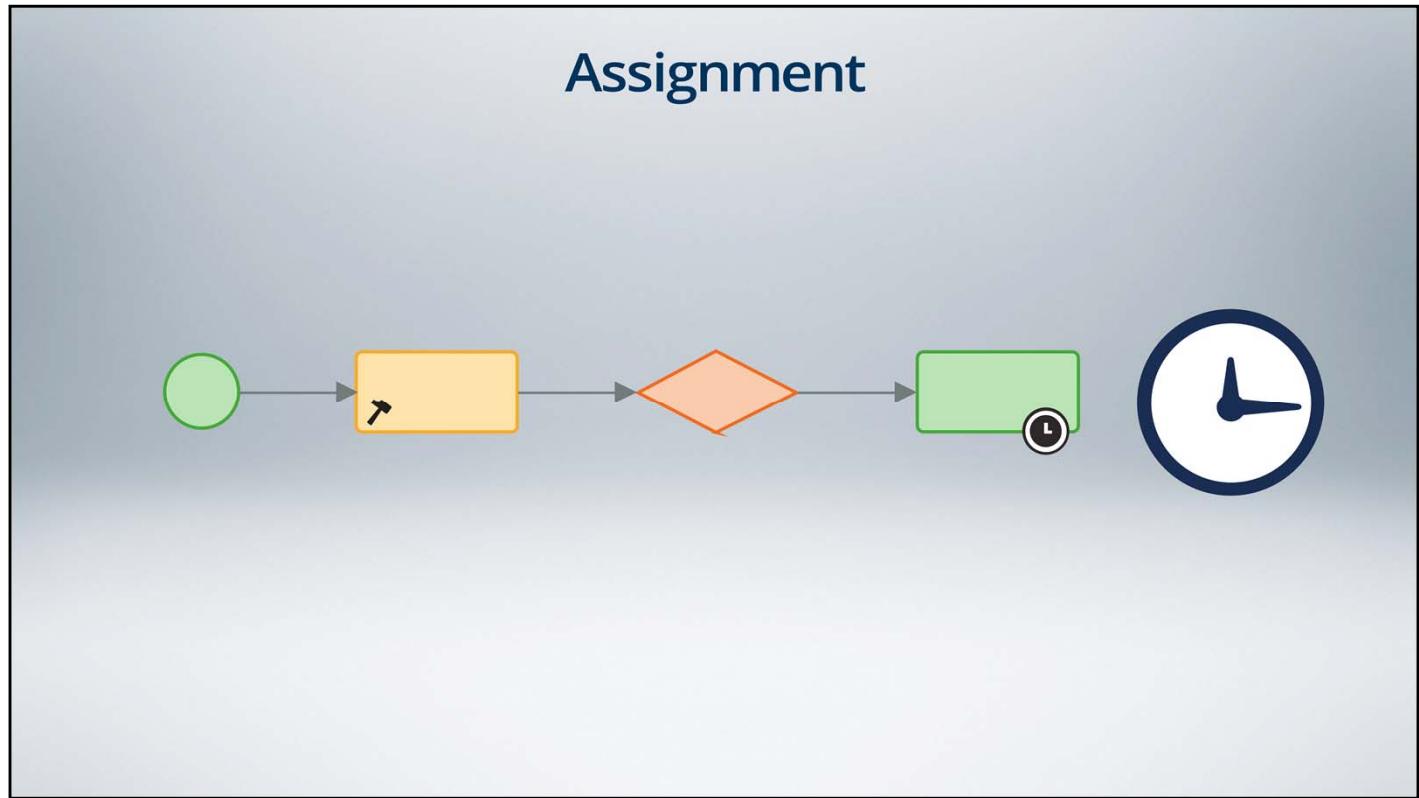
The system then assigns it to a specialist or a single person or it can go to a routing queue where multiple people can access it.

Now that we have reviewed routing requirements let's learn more about how we can implement routing in PRPC.

Assignment



Let's start by learning about assignments which play a key role in routing. Assignment, the term sounds familiar, that's right, we have talked about this term when we were using process modeler and also while working in stage designer to define processes.



When a case is created it runs the starter flow that is defined for that case. The starter flow moves from one shape to another until it reaches an assignment shape. When it reaches the assignment shape, it stops there and does not continue forward in the flow unless a user action is performed to complete the assignment task.

Assignment

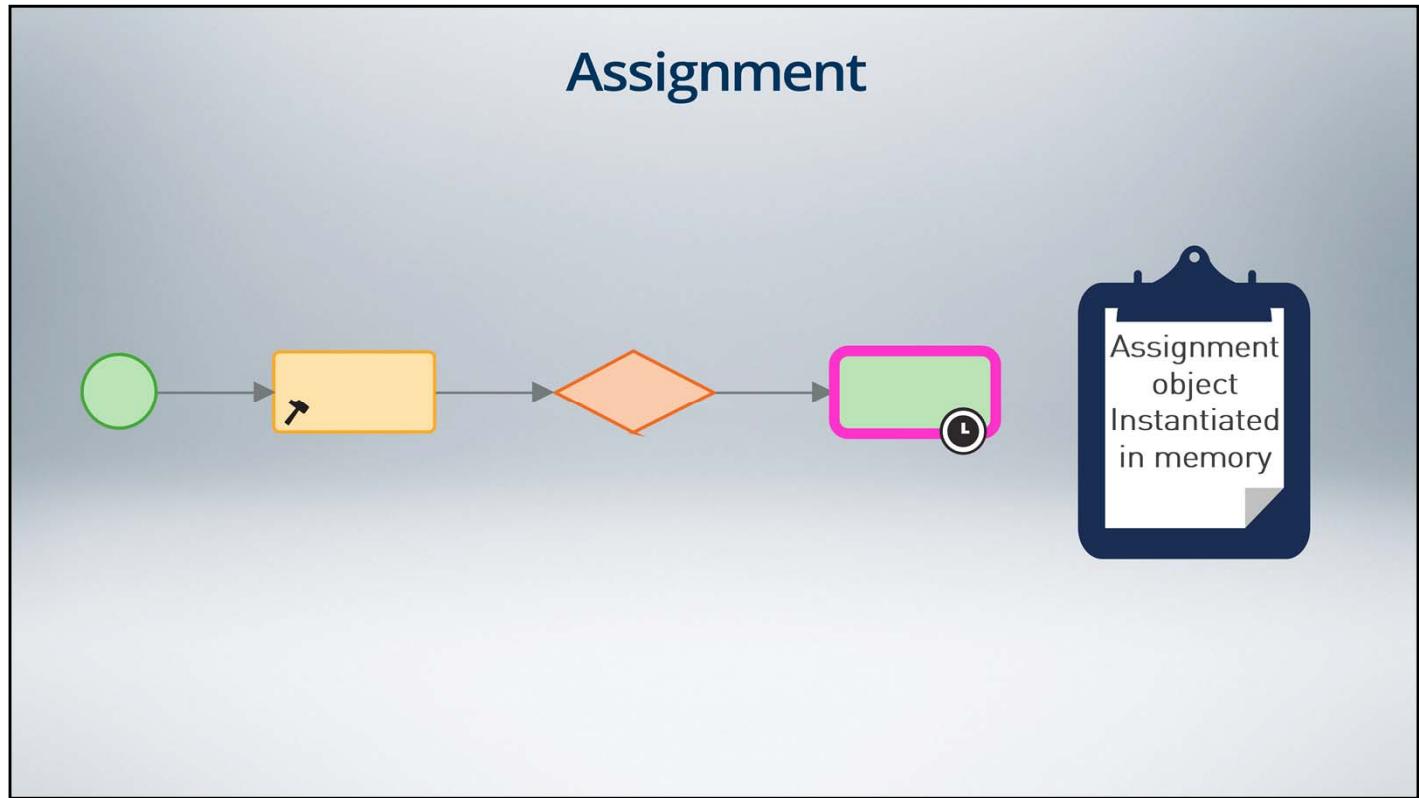
The diagram illustrates the assignment shape and its configuration. On the left, there is a green rounded rectangle representing the assignment shape, with a black circle containing a white letter 'L' positioned below it. A pink arrow points from this shape towards a white box containing two questions. To the right of the white box is a small screenshot of a 'Flow Action' interface.

1. Who works on this ?

2. What do they see?

Flow Action

The assignment shape has two key configurations – who works on this and what do they see? The connector coming out of assignment shape references a flow action rule that controls what content displays to users when they open the assignment.



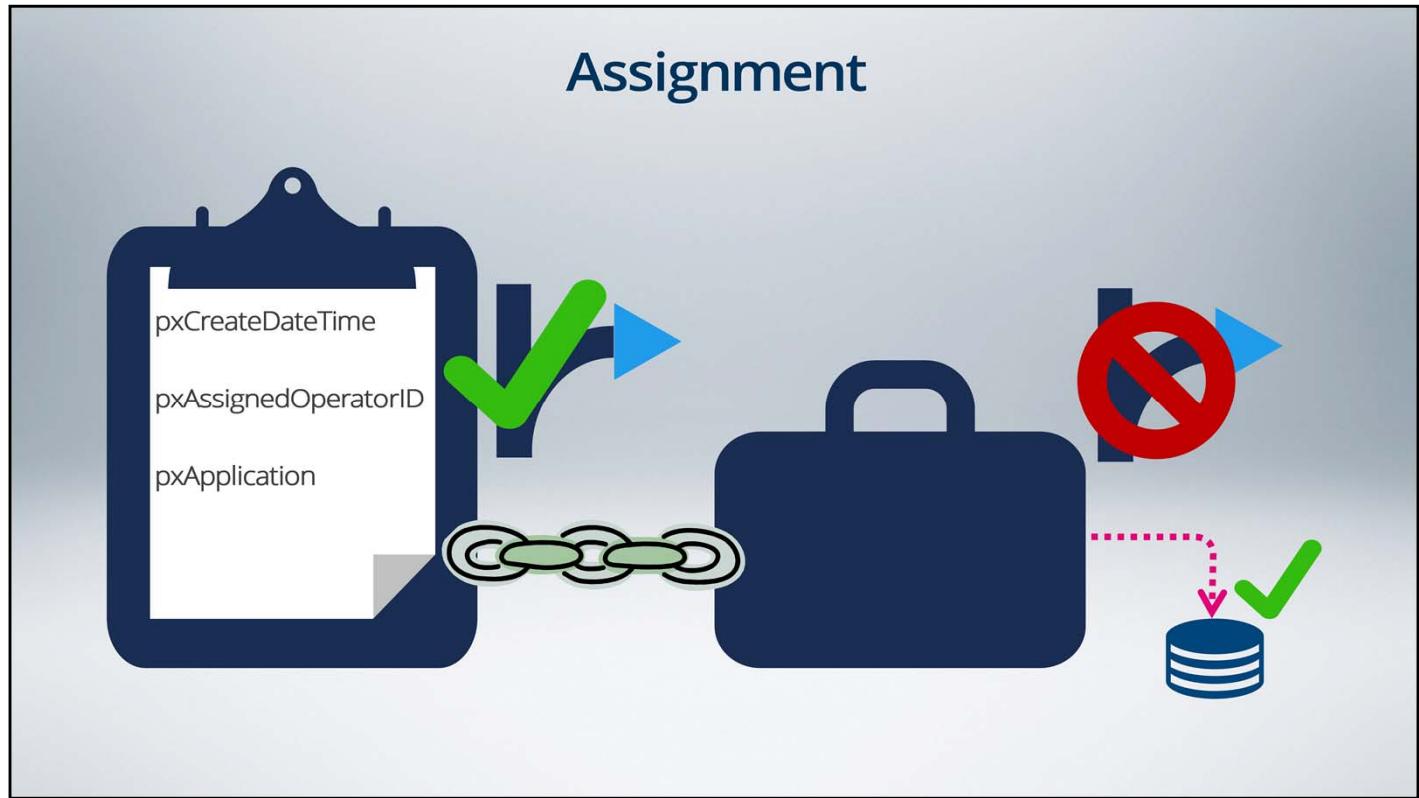
When the system reaches an assignment shape, it creates an assignment object in memory and in the database. The assignment object can be seen in the clipboard when the operator opens the assignment.

Assignment



Assignment Object

The assignment object stores additional details such as when the assignment is created, who is working on this assignment, the application name, and so on.



The assignment object is linked to the case so we can keep track of where the case is currently being worked on. So in essence, routing a case means the assignment object is routed and not the case.

Assignment



- Assignment service levels



- Timeliness Reports



- Parallel Assignments

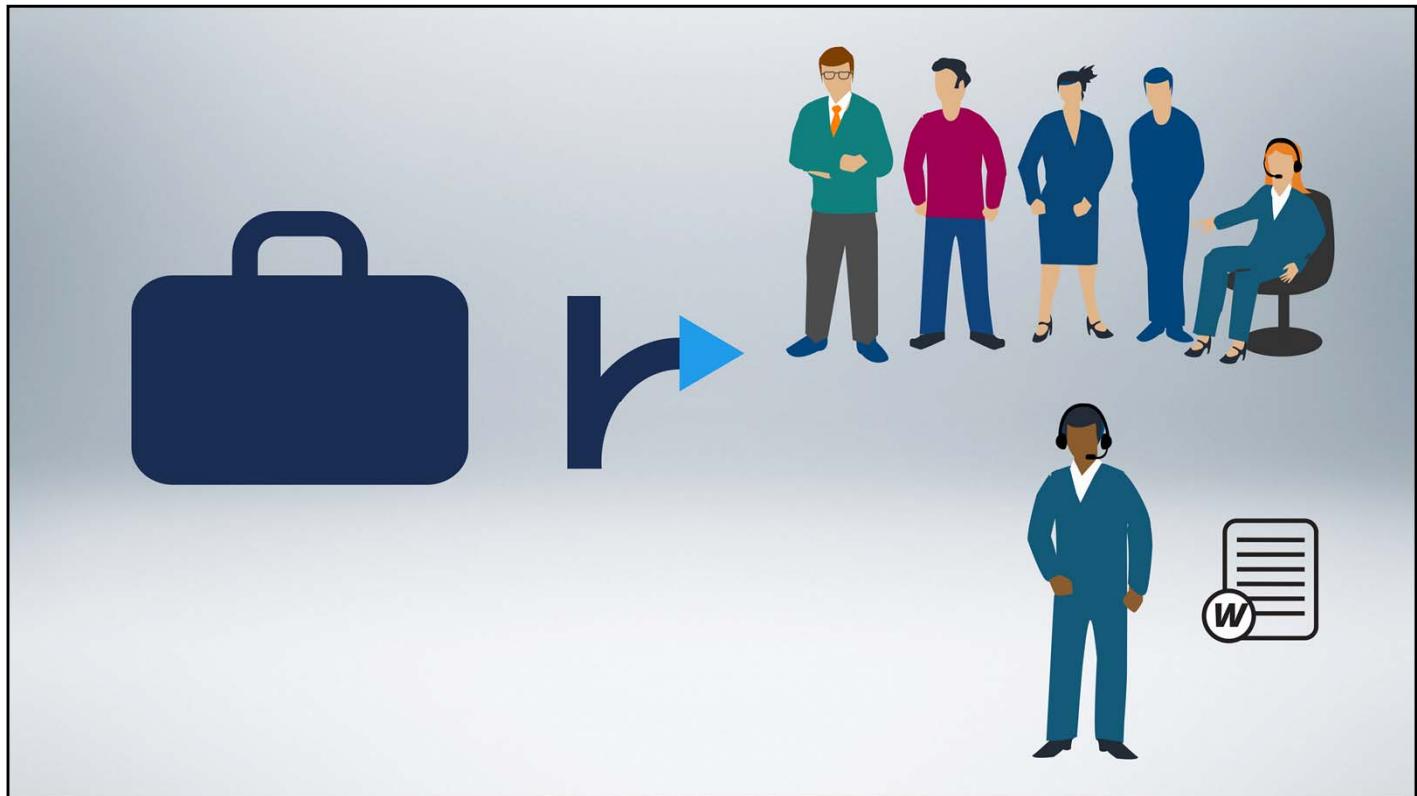


Why does the system create an assignment object?

The first reason is to establish service levels (SLA) which we will learn about in detail in the System Architect Essentials II course. When the operator is assigned a task they are expected to finish it within a specific period of time. Service levels govern this and they are defined on assignments.

Managers in most times want to know the effectiveness in which cases are being processed, how different operators perform their assignments, whether someone requires additional training to quicken the case processing and so on. There are various reports on these assignment objects to provide insight into how cases are processed.

The last reason to create assignment objects is to support the functionality where we want more than one operator to work in a case at the same time. In such cases, the system generates multiple assignment shapes and assigns them to different operators. The operators can work in parallel and the system generates separate assignment objects for each operator.



We learned that cases can be routed either to an operator or to a routing queue that is accessible to a group of operators. When the case is routed to an operator, in PRPC we refer to it as routed to their worklist.



Worklist

- List of cases the Operator is currently working on
- List of cases assigned to the Operator
- List of cases waiting for the user to perform an action
- Can view work list in portals

What is a worklist? – A worklist is a list of all the assignments for a particular operator.. In other words, the list of cases that are waiting for the operator to perform an action. Each Operator in PRPC has their own worklist and it is accessible from their portals.

The screenshot shows a software interface titled "My WorkList". On the left, there is a icon of a document with a "W" on it. The main area displays a table with columns: URGGENCY, DUE, ID, DESCRIPTION, CATEGORY, and OWNER. Two rows are visible:

URGENCY	DUE	ID	DESCRIPTION	CATEGORY	OWNER
10		PF-2	Enter Program Fund	Program Fund	Senior Vice President
10		VM-3	Add/Edit Vendor	Vendor Maintenance	Senior Vice President

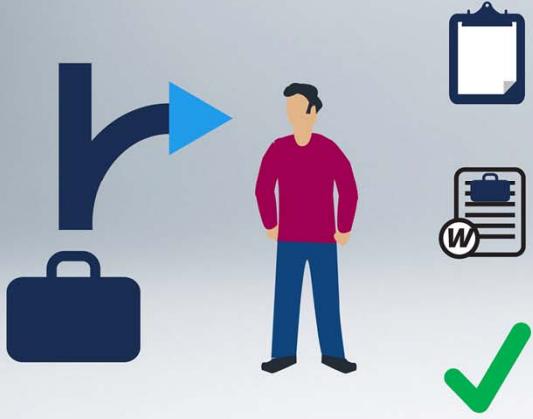
Below the table, there is a section titled "TEAM MEMBERS" with a list of users and their counts:

- SAE1 Architect: 3
- Karthikkeyan Ramachandran: 3
- External User: 0
- Domenic Giancola: 0
- Product Manager: 0
- RequestorAPAC: 0
- RequestorEU: 0
- RequestorNA: 0
- Senior Vice President: 0
- Vice President: 0

MANAGER'S WORKLIST (pink handwritten text) is written next to the icon on the left. **CAN ACCESS WORKLISTS FOR ALL OPERATORS WORKING** (pink handwritten text) is written below the table, with two pink arrows pointing from this text towards the "TEAM MEMBERS" section.

The manager has his own worklist and in addition to it they also can access the worklists of all the operators reporting to him.

Case Routing to Operator



- A case is routed to a User
- Creates an assignment object
- The assignment appears in the Operator's work list
- When the Operator completes the action, the assignment is resolved

So let's sum up what happens when a case is routed to an operator – the system creates an assignment object, the assignment shows up in the operator's worklist and when the operator finishes processing the assignment by submitting the flow action, the assignment object gets resolved and the flow continues processing.

Workbasket

- Workbasket = queue for storing cases
- Operators are assigned to workbaskets based on workgroup
- Can access all workbaskets belonging to that workgroup



Now, let's see what happens when the case is routed to a queue. PRPC uses work baskets in cases when routing queues are needed.

What is a workbasket? A workbasket is the routing queue where assignments are stored and multiple operators can access them.

Workbaskets are tied to another record named workgroup – workgroup is the record that groups a set of operators.

The Operator can access all workbaskets that are part of the workgroup to which they belong.

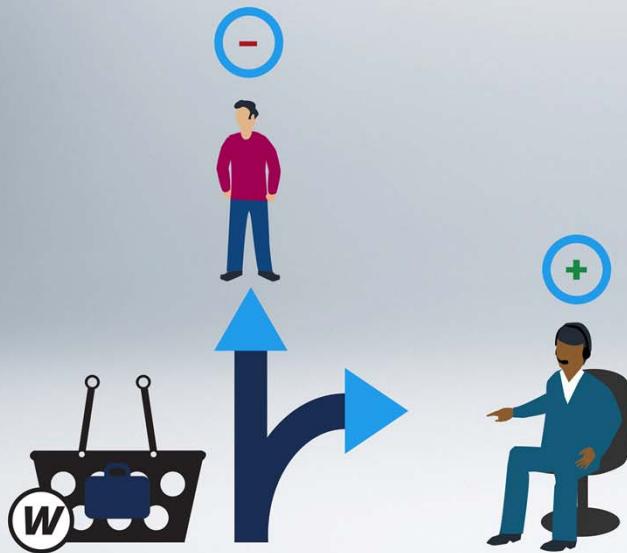
Case Routed to a Workbasket



- A case is routed to a Workbasket
- Creates an assignment object
- The assignment appears in the workbasket list
- It stays there until it gets assigned to an operator

So let's see what happens when the case is routed to a queue – the system creates an assignment object, the assignment shows up in the workbasket list and it stays in the workbasket until it gets assigned to an operator.

Routed from Workbasket to an Operator



Push routing

- System decides based on the rule configuration

How do cases get routed from workbaskets to operators?

The system pushes the cases to operators using a set of configured rules that are known as PUSH routing. These rules can be customized by a senior system architect.

Routed from Workbasket to an Operator

Pull routing

- User picks from the list



The second option is called “PULL Routing” – here users access the workbasket list and pull a case from it.

Routed from Workbasket to an Operator



The assignment moves to the Operator's work list

When the Operator completes the action, the assignment is resolved

It then follows the same steps – where the assignment object does not exist in the workbasket list, but is moved to the operator's worklist.

The operator sees it in their worklist until they perform an action.

Demo



Routing in Case Processing

Routing in Case Processing

Now that we have learned a lot about assignments and routing let's see it in the application. Let's create a new purchase request to see how the case gets transferred to different actors or operators. This case is now being worked on by the current operator or in this case, it's the SAE1Architect. Let's create a sample request by filling in one line item

and then continue to the next step. Clicking Finish completes the creation of request.

Now we see that the case is moved to the Approval step and it does not show up as an assignment for this operator because it's assigned to a different operator. Notice the Assigned to field- it has the Sales Manager as its entry. Let's open the Where Am I icon to open the flow. There is only one assignment shape in this flow. Let's open the properties panel to see the configuration. Take a look at the Assignment type field – it has Worklist as its entry. The Routing area has an entry for the Router field- ToWorklist. Let's come back to what that is in the next video.

For now, notice that there is a parameter named operator which has param.CurrentApprover. Let's close the window and go back to the previous screen where the case is being processed. Since we are logged in as a developer we have access to perform the action assigned to other operators. This privilege can be changed but by default it is set this way so that developers can perform unit testing.

Let's go ahead and approve the purchase request. The case now moves to Procurement stage. Notice that the assignment is now routed to PegaPurchase-Purchasing.

If we look at the flow by clicking Where Am I icon and open the properties panel on the assignment shape, we can see that the Assignment type is selected as Workbasket.

The router field uses ToWorkbasket so that the case can be routed appropriately and the parameter field now

show the name of the workbasket.

So when the purchase request required approval it was routed to a manager, and when it required that the generate purchase order be generated it is routed to a workbasket so a group of operators having access to that workbasket can create the purchase order quickly.

The assignment shape identifies whether the assignment is routed to a worklist or a workbasket. In both cases the router uses ToWorklist or ToWorkbasket and the parameters can be configured to identify who is working on it. Let's continue along after the operator working on the queue has entered the information. The Purchase Order is generated and is assigned to PegaPurchase-Purchasing the Place Order assignment step has the configuration set - most likely to workbasket, let's see.

The assignment type is a workbasket, just as we would expect. However, the Router is using a new entry ToDecisionTable. The parameter now displays DecisionTable and it references an actual decision table rule. So what is this entry in Routing field? We saw ToWorkList, ToWorkbasket and now ToDecisionTable. What are they?

Routing cases and Standard Routing rules

We saw that routing is configured in the assignment shape. Let's open up a purchase request case that we have configured for demo purposes. To configure an assignment we need to open the case in the outline view. Once the process loads, let's open the properties panel for the assignment shape. We see the choices for Assignment type, and it's set to the default which is Worklist. Now let's look up router field and notice it says "Select Router Activity" Pega ships many standard router activities that we can use for routing.

Similarly, if we want to configure the assignment to be routed to a workbasket we will see a list of standard router activities in this field. An activity is a rule type in Pega that we use very rarely. When custom activity rules are needed they are written by senior system architects.

Now let's select ToDecisionTable. It shows the parameter for selecting a decision table. We can use ToDecisiontable for both worklist and workbasket. Let's select another standard router rule-ToCostCenterManager. CostCenterManagers are configured while setting up the system, so there is no additional parameter to set the name here. In most cases, the workbasket is always assigned to a specific named workbasket and we use ToWorkbasket.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- Routing section in Assignment Properties form has been updated
 - Visible routing parameters depend upon Route to field
 - Route to operator

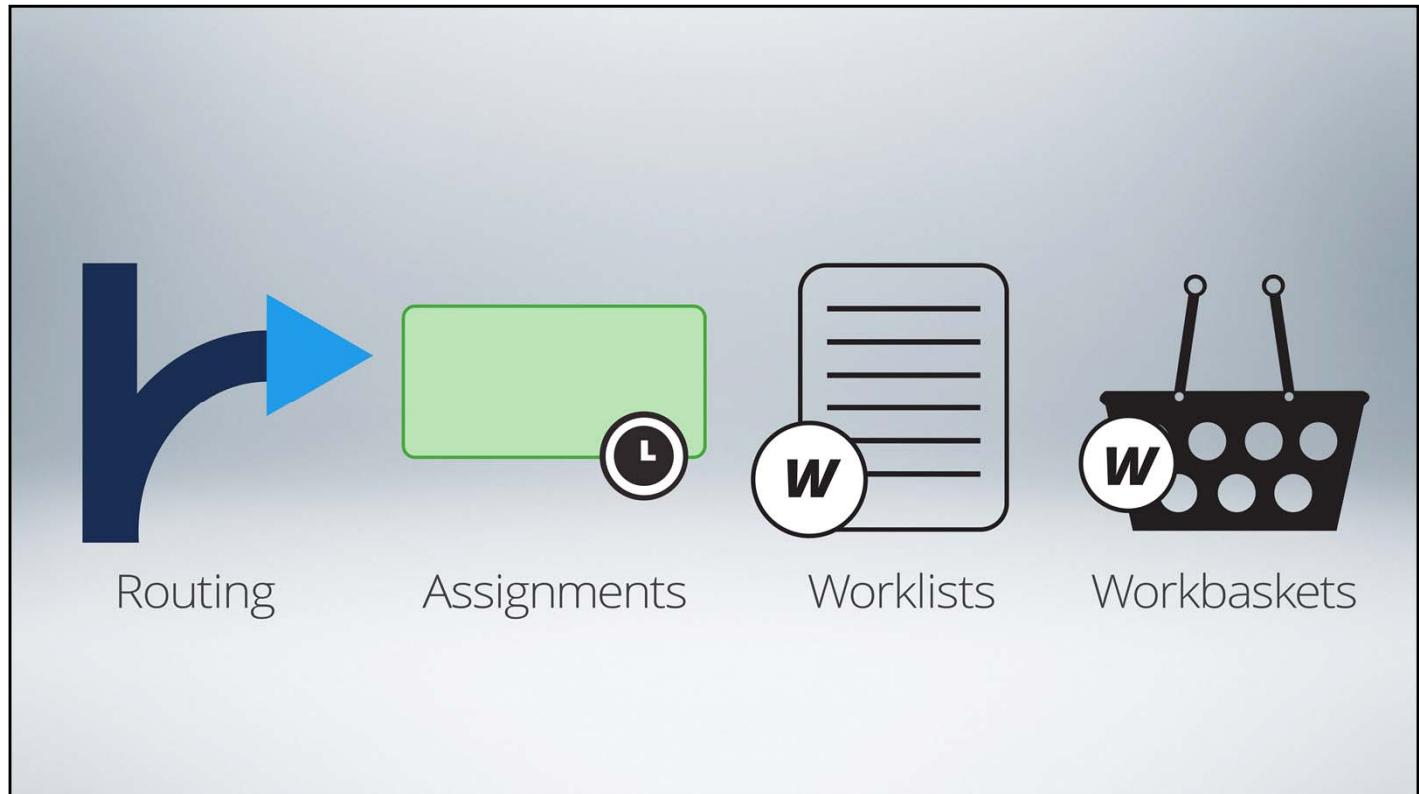
The screenshot shows the 'Assignment Properties' dialog. In the 'Assignment' field, it says 'Select New Position'. Below it, a note says 'A step in your process that requires human judgement and input.' Under the 'Routing' section, there is a question 'Who should complete this step?'. A dropdown menu labeled 'Route to:' is set to 'Operator', and a text input field next to it contains 'Manager@SAE'.

- Route to WorkBasket

The screenshot shows the 'Assignment Properties' dialog. In the 'Assignment' field, it says 'Screen Candidate'. Below it, a note says 'A step in your process that requires human judgement and input.' Under the 'Routing' section, the 'Route to:' dropdown is set to 'Custom' and the 'Assignment type' dropdown is set to 'WorkBasket'. The 'Router' field contains 'ToDecisionTable'. Under the 'Parameters' section, 'DecisionTable*' is set to 'PurchasingRouting'.

- Route to Other

The screenshot shows the 'Assignment Properties' dialog. In the 'Assignment' field, it says 'Select New Position'. Below it, a note says 'A step in your process that requires human judgement and input.' Under the 'Routing' section, the 'Route to:' dropdown is set to 'Custom' and the 'Assignment type' dropdown is set to 'WorkBasket'. The 'ROUTING' section contains a 'Router' field with 'ToDecisionTree'. Under the 'PARAMETERS' section, 'DecisionTree*' and 'InputProperty*' both have empty text input fields.



That sums up this lesson – we learned about routing, assignments, worklists and workbaskets.

Standard Router Activities

ToWorklist
ToWorkbasket
ToOrgUnitManager
ToDecisionTable
ToDecisionTree
ToCostCenterManager
ToCreateOperator
ToDefaultWorkbasket
ToSkilledGroup

We then looked at how the assignment shape can be configured to do all this by using standard router rules, rather activities. These activities are shipped in the product and in rare cases the senior system architect will write some custom routing activities.

Exercise: Routing to get Approval for Offer

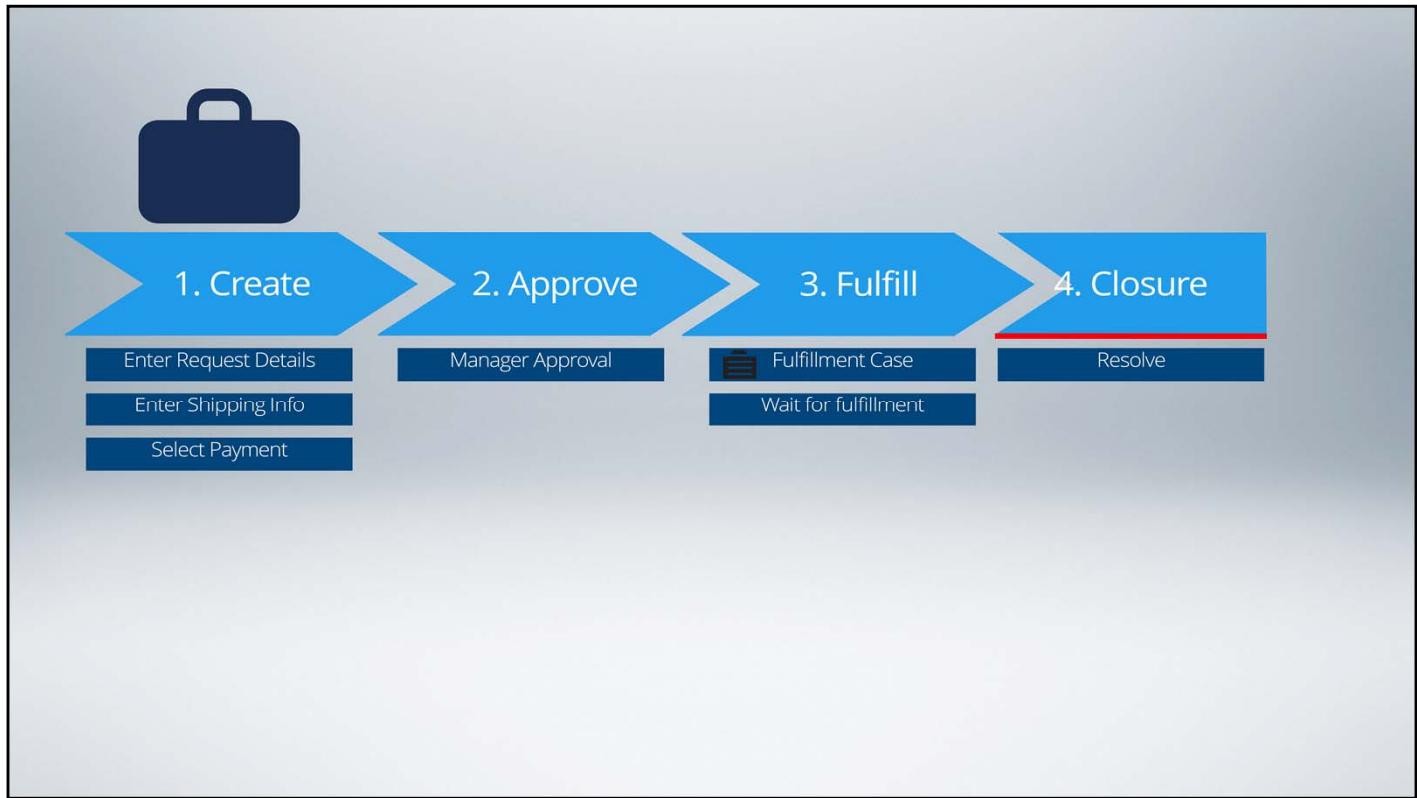


Case Stage Configuration

We need to configure case stages to skip them on a condition, to start them after validation, start an optional action in any step on that stage.

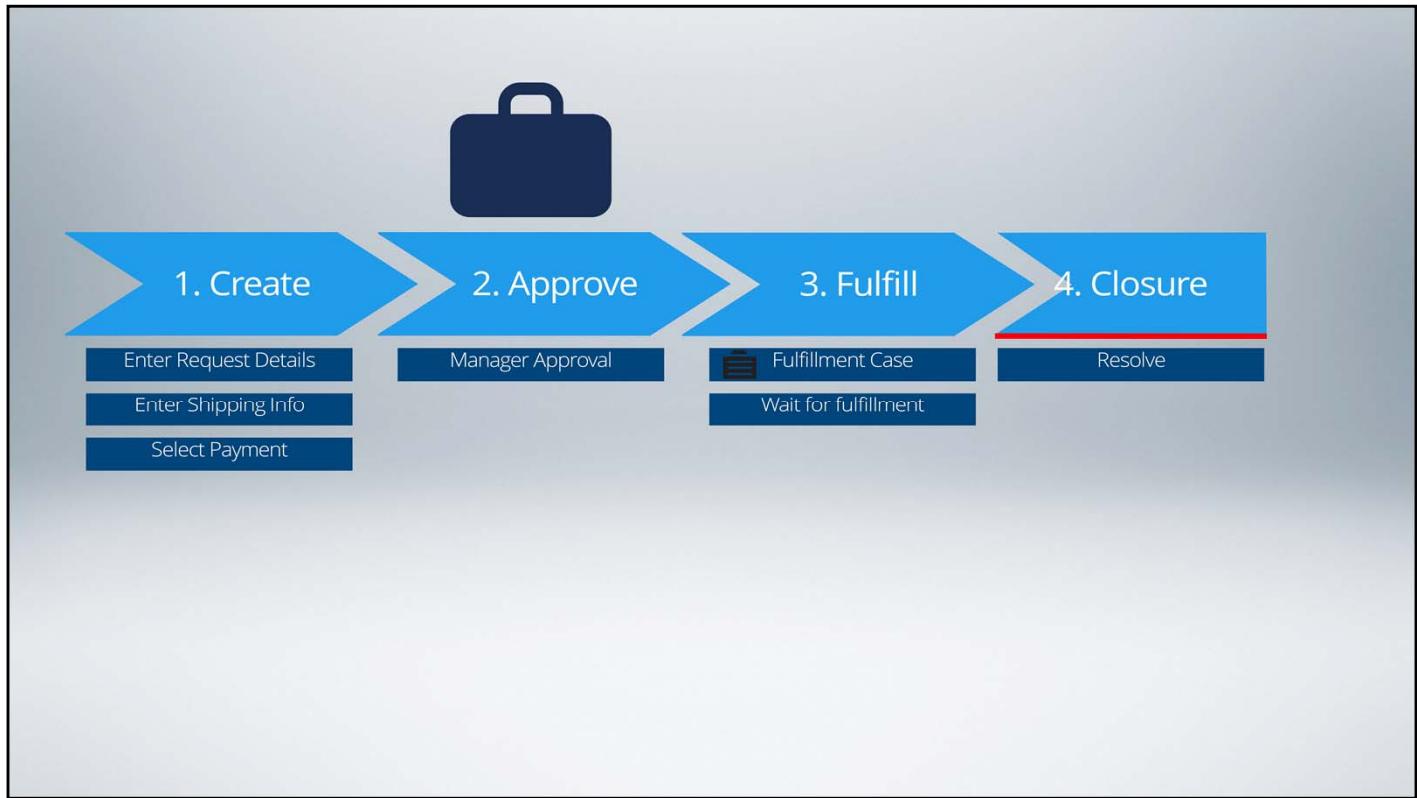
At the end of this lesson, you should be able to:

- Configure the stage to render on a condition
- Configure how the next stage is started after the current stage is completed
- Configure to mark a resolution stage
- Configure to add optional actions in the stage

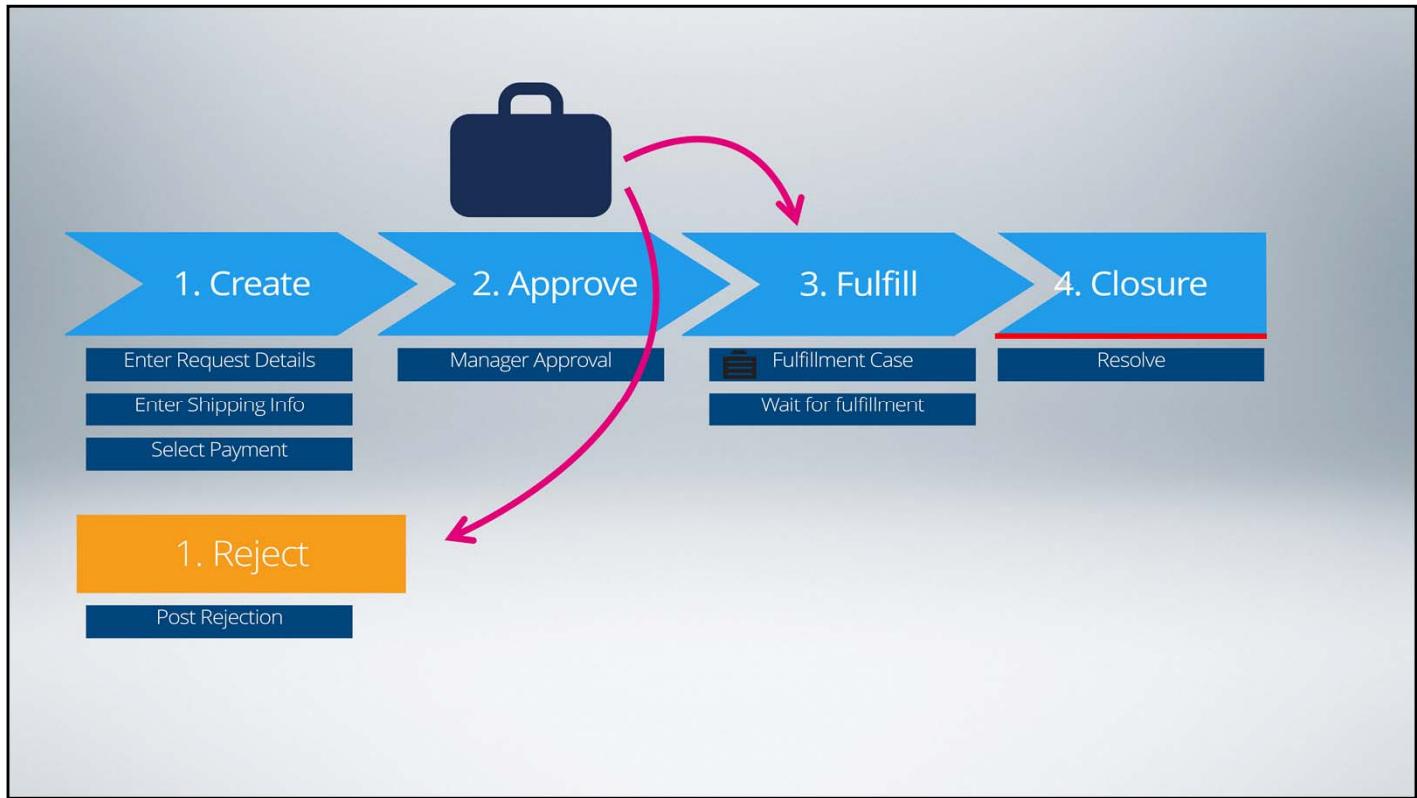


A Case moves through stages. When defining the case we identify the stages and the processes that must be completed in each stage.

In most situations, a case moves from one stage to the next stage automatically until its resolution.



There are exceptions – sometimes we may need to skip a stage based on a specific condition. For example, suppose an external system submits a request to a PRPC service with all the information that is required to create a purchase request. In that scenario we would move the case directly to the approve stage and skip the Create Stage since the case already has a purchase request defined in it.



Also there are other scenarios which would require us to jump to a different stage instead of going to the next stage. For example, when the case is in the Approve stage,

the approver makes a decision which requires us to move the case according to the decision that is made – if the request is approved we want the case to move to the Fulfill stage which is the next stage,

and if the request is rejected, we want the case to move to the Rejection stage.

When we design case stages, it's highly recommended that we identify the resolution stage. Doing this will add an indication in the stage designer on the resolution stages.

Demo



Configuring Case Stages

Now let's switch to the product to see how we can configure the case stage to meet these requirements. Here we are in the stage designer with all the stages and processes identified for a case.

Some of the stage configurations can be done directly in the stage designer- the Edit stages button allows us to rename the stage names for all the stages.

To make configurations on a specific stage, we hover over the stage name which displays the arrow and then click the arrow to open the menu.

In the menu, we select Configure stage behaviors to open the dialog that allows us to make various configuration options that we can apply on a case stage. The first field we see here is the stage name- we can change it to a different name if we want. The Stage Entry validation field references a validate rule and when used, the stage cannot be started unless the validate rule is successful. The next field allows us to conditionally skip the stage. In the requirements we saw before, we have to skip the create stage if the case details were supplied by an external system. To configure this, we can reference a when rule in this field which checks if the case details exist. If the when condition returns true, the stage is skipped. The resolution stage field is the one which helps in identifying a resolution stage, this is set to no since this is the first stage. Let's go back to the stage designer to see the configuration on a resolution stage.

The resolution stages are the ones that are underlined in red, here there are three of them - closure, rejection and withdrawal. The stage names indicate how this stage is reached-Closure, means the purchase request is approved and fulfilled by the purchase order, so the case can be closed as complete. Rejection, means the purchase request is rejected by an approver and withdrawal means the requestor withdrew the purchase request.

Let's open one of them – notice the field is set to yes. If we switch the selection to no another entry appears which asks us to make a selection on what to do after the stage is completed. All stages other than resolution stages require us to configure where the case should go after the stage is completed. By default, when all the processes of a stage are completed, the case moves to the next stage by default.

As we saw in the requirements earlier, we want the navigation from the Approval stage to be based on the approval decision.

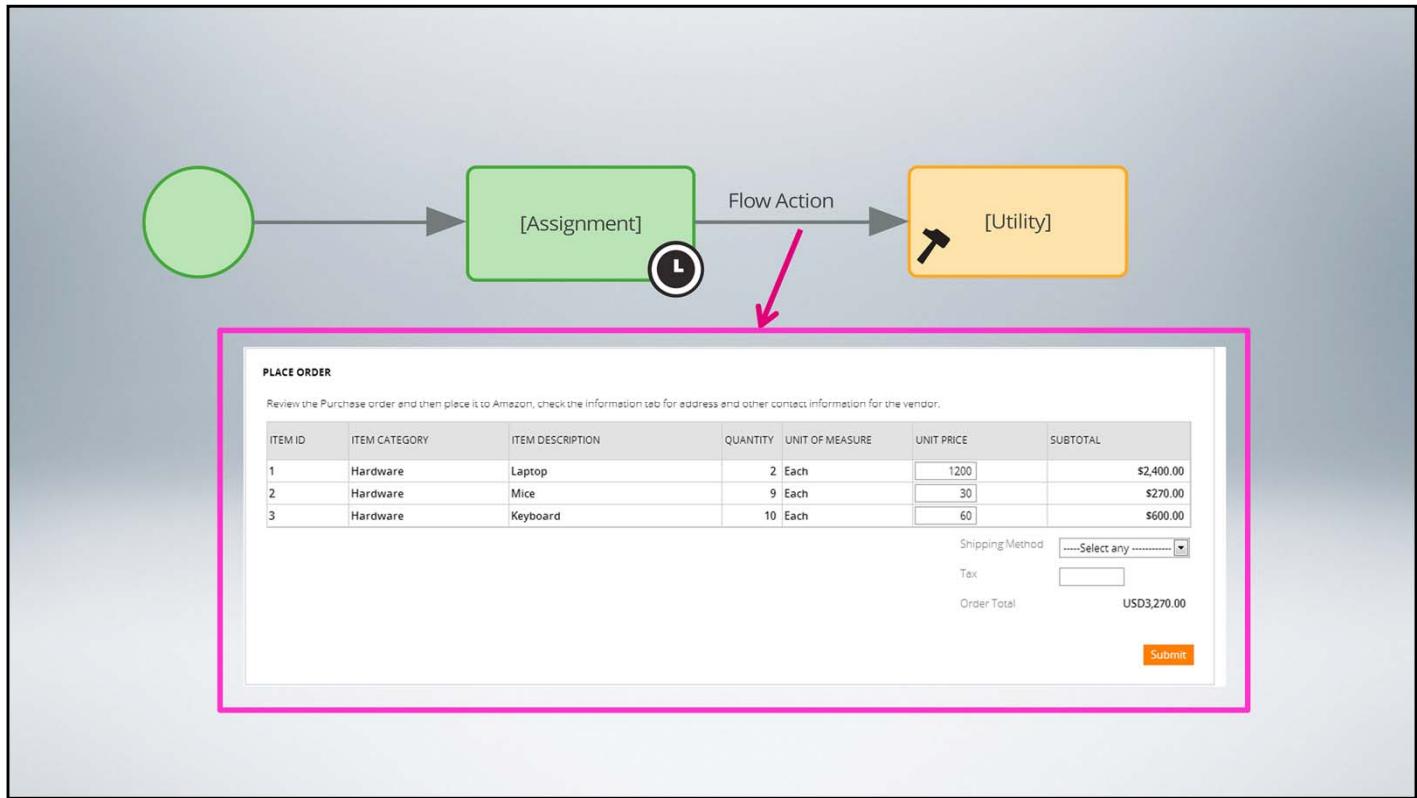
Let's start by making the case stop at this stage and not move to the next stage automatically.

Notice now that the stage does not have the arrow going from approve to procurement. When this field is enabled, the processing stops at this stage and it becomes our responsibility to make sure the case moves to other stages. We will learn about configuring the jump to a stage in a later lesson. When configuring stages, we can also add an optional process which uses a flow rule, or an optional action which is a flow action rule. We will learn about optional actions in this lesson and we'll learn about optional processes in a later lesson.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- Configure stage entry validation is an open on the Stage menu
 - Was previously on the Stage Configuration form
 - New form allows attachments to be included as a requirement for stage entry



When we learned about flow actions in the Introduction to UI lesson, we saw that they are referenced in the connector coming out of an assignment shape.

A Flow Action is the UI rule that is used in the flow to render a UI for a specific user action.

When the user submits the action, it progresses the flow to the next step in that flow.

Optional Actions

When added on a stage, it can be accessed on all processes of that stage



When we add an optional action in the stage, we see the optional action on all steps for all of the processes in that stage as an option for the end user to select.

Optional Actions

Display a screen

Submission does not progress to the next step in the flow

Not mandatory



ADD ATTACHMENTS

Drag and Drop Files Select a File

FILES	DESCRIPTION	CATEGORY

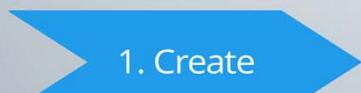
Submit

When the user selects this optional action, it displays a screen and the user can review and make any selection if necessary and submit the action. When the action is submitted, the changes get processed, however the case still remains in the same step and does not move along.

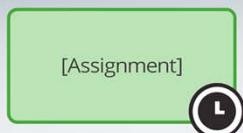
If we add flow actions in the connector shape of a flow, the case moves from one step to another, however the optional action does not move to the next step in the process.

When an optional action is added in a step, it is not mandatory for users to execute that action on that step.

Optional Actions



Stage Specific



Single Assignment



Case wide

Optional actions can be added in stages,
processes,
on a specific assignment
or even across the whole case.

Demo



Configuring Optional Actions

Let's look at an example where an optional action is added in a case stage. In the quote stage, we have a flow action named transfer that is added in the optional actions area. We can add any of the standard flow action or ones we have defined as part of our application.

Let's see how this looks at runtime.

When the user reaches that stage, they see the flow action that is referenced in the connector shape coming out of the assignment here, and the optional actions appear under other actions.

When we click the other actions, we notice that the optional action we added appears below a line, the line indicates that all actions underneath the line are optional, submitting these does not progress the flow.

When we are in the complete quote action, we see the screen that is part of the flow action and when we open the other actions menu, it shows only the optional action and not the complete quote action. When we click transfer, the system now displays the screen that is part of the Transfer optional action and we see the complete quote action in the other actions menu.

This action allows transferring the case to another operator. The new operator, when they open the case will see the action to perform the complete quote. Since the assignment is not moved, the newly assigned operator can either complete the action or again select the same optional action to transfer it back. The process does not move forward unless someone performs the complete quote action, and even if we perform the optional action any number of times the case stays in the same step.

Exercise: Identifying Resolution Stages



Exercise: Schedule Company Tour

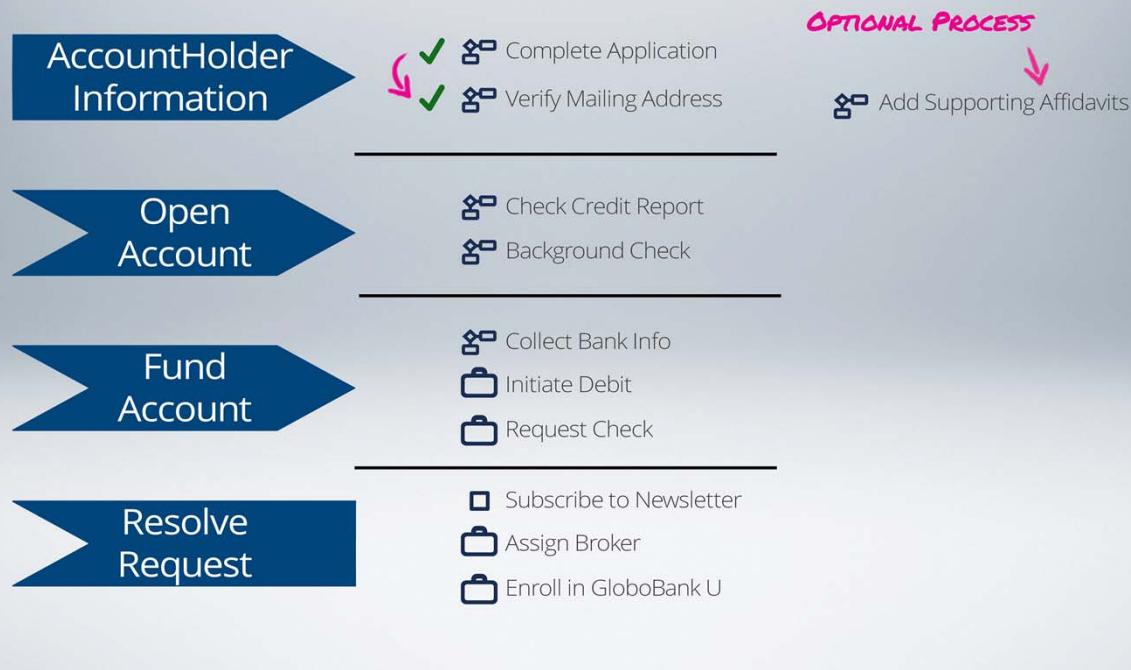
Creating Optional Process

When working on the case, special circumstances might warrant users to create a special process only when they deem it is required.

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

- Create an optional process which can be instantiated manually
- Determine when to create an optional process

Stages and Processes: Reviewing the New Account Application Case



Earlier we learned about case stages and processes.

Shown here is the list of stages for a new account opening case.

Each stage has processes that perform tasks required to complete that stage.

So when a case gets created it is in the AccountHolder information stage and the Complete application process is started.

When this process completes, it progresses to the next process, verify mailing address.

Once that finishes the stage is considered complete and the case moves to the open account stage where it starts the check credit report process.

Generally, these stages and processes are initiated automatically one after the other as the user works on the case.

But, what if we need to perform a process only when required by the user? This is where we would use the Optional process.

For example, when collecting account holder information –special circumstances might require the agent to collect supporting affidavits. This can be triggered by the agent on an as needed basis.

Optional Process



AccountHolder
Information



Starts when the operator performs it
The process can be started from the other actions menu

Usually tied to a specific stage
but can also be added across a case

This process does not start unless the agent or the operator working on the case starts it.

The operator sees an option to start the process in the other actions menu while working on the case in their portal.

The optional process can be tied to a stage as in the example described above where it can only be initiated in the Account holder information stage. Alternatively an optional process can be made available across the entire case.

Now let's see how we can configure an optional process in the system.

Demo



Configuring an optional process

Configuring an Optional Process Demo

As mentioned we can add optional processes on a stage or across the whole case. Let's see how we can add one to the entire case.

In the Details tab, we have the option to specify Case –wide supporting process. Adding it here makes the process accessible on all stages irrespective of the process running.

We can add any process that is part of the application here. But let's close this screen.

For our demonstration let's go back to the Stage Designer and add it in a stage.

To add it on the stage we click on the stage name and in the menu that appears we select Configure stage behaviors.

The optional processes can be added here in the optional section..

To add a row, we click the icon and then we select a flow . The flow can either be a starter or a sub flow.

In this example, the operator wants some feedback on his entries, this is optional by definition since the operator can start the flow at their own discretion, however we can supplement this by using a when rule. Specifying a when rule makes the option accessible only when the condition is satisfied.

For this example, let's not specify a when rule. So we can just save the changes in the Stage Designer view.

After saving let's see how it looks at runtime by creating a new case.

The optional process should appear under other actions menu, so let's open it.

Optional processes are grouped under Add Work.

We see the request feedback is here, so let's start the flow by clicking it.

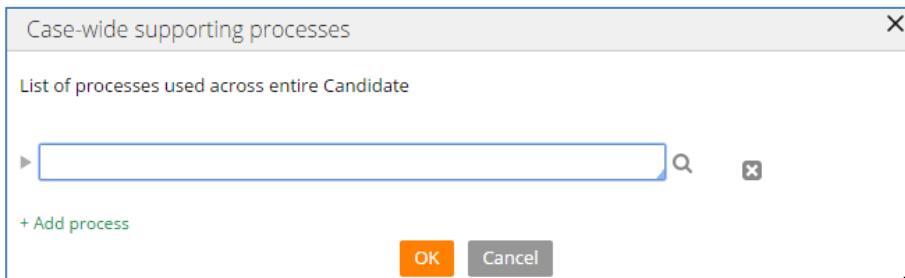
Now the requested feedback appears. Notice the other process is still running, so we can see both of these processes are assigned to the same operator.

To sum up this lesson, use optional processes to allow operators to have the option to manually initiate processes either within a particular stage or for the entire lifecycle of the case.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- Updated features seen in this lesson:
- Case-wide supporting processes form has been updated
 - Format updates only – no feature changes



Exercise: Schedule Company Tour



Exercise: Schedule Company Tour

Introduction to Task Based Smart Shapes

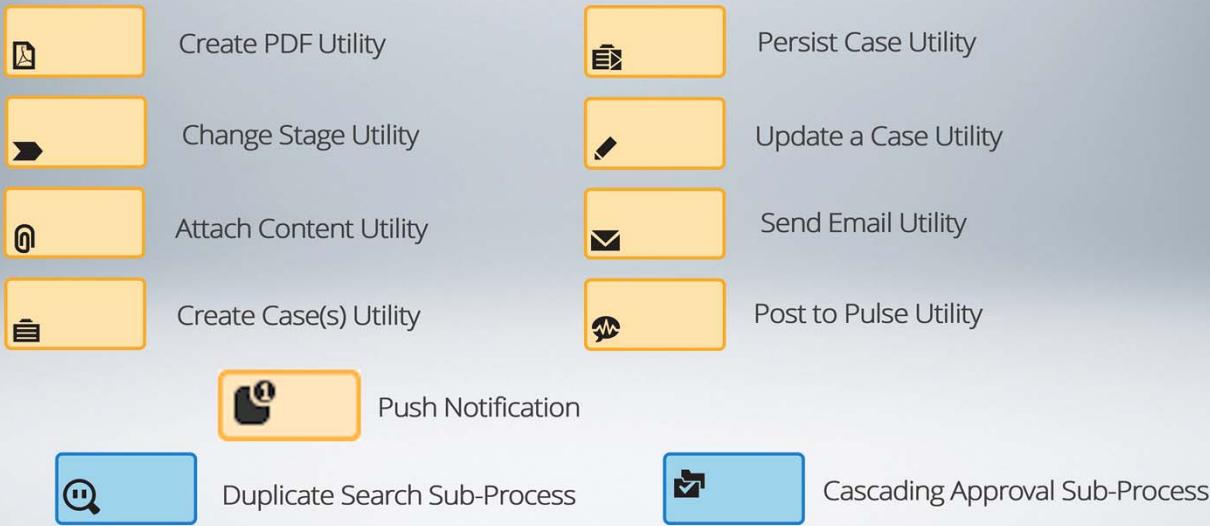
Smart shapes are shapes that are pre-configured utilities and sub-processes and are directly accessible through the process modeler. This lesson provides an introduction to smart shapes, and how to use couple of task oriented Smart Shapes such as sending an E-mail and changing to a Stage.

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

- Understand what Smart Shapes are
- Use Smart Shapes to send Email and change the Stage

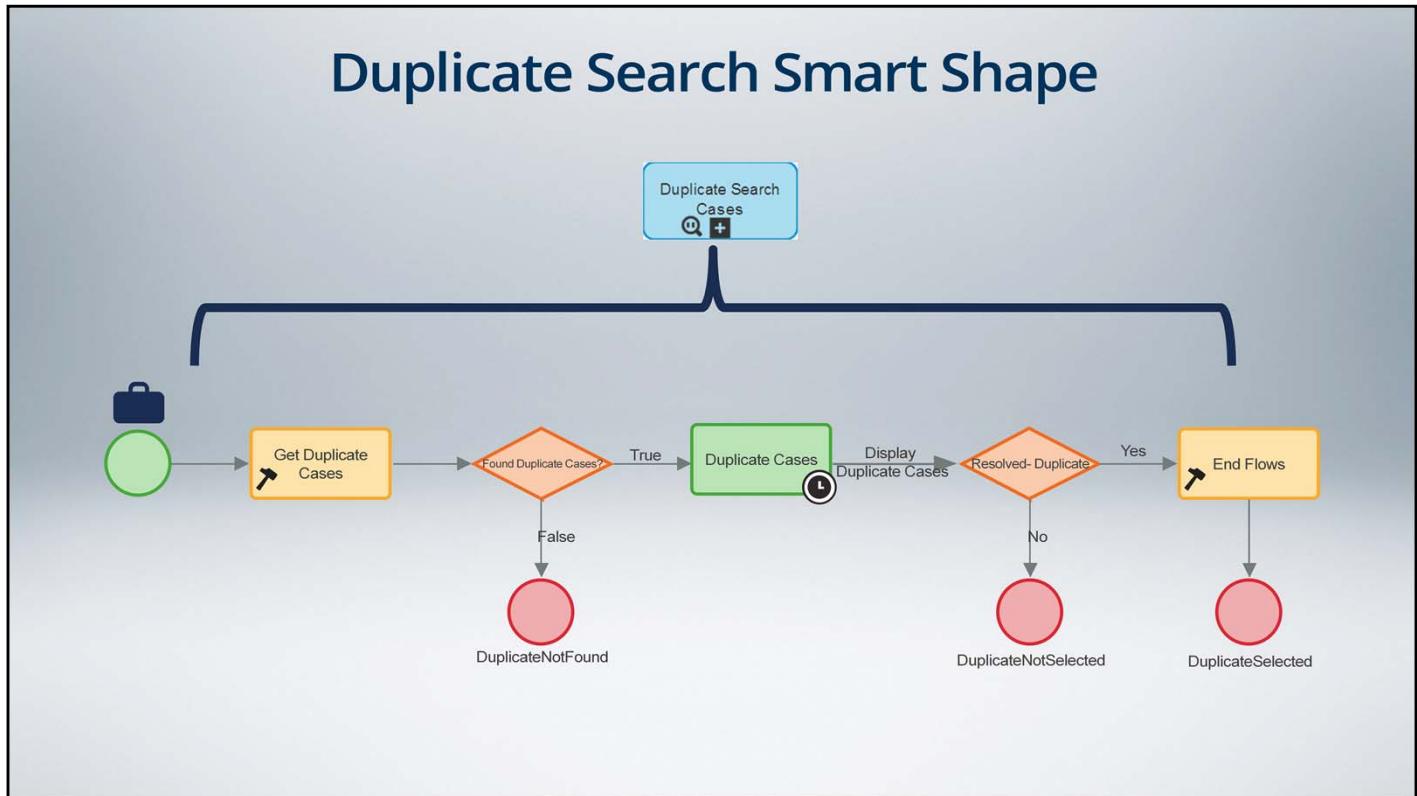
Smart Shapes:

- flow shapes preconfigured to perform specific tasks



Smart Shapes are pre-configured utilities or sub-processes that facilitate specific actions – such as creating a PDF, changing to a different case stage, or attaching a file – similar to macros or scripts found in various software packages. Smart Shapes are directly accessible through the process modeler with easy to configure parameters. There are two types of Smart Shapes, utilities and sub-processes. Utility Smart Shapes are orange in color and provide an easy way to accomplish common actions. Sub-process Smart Shapes which are blue in color also provide common actions but are typically more complex in nature and therefore are best accomplished via a sub-process.

Even though most of these shapes are self-explanatory from the names, we will discuss a few Smart Shapes in detail that will enable us to do basic tasks, such as send email and change a stage. Details of other Smart Shapes will be covered in other advanced courses.



Now let's take a look at the duplicate search sub-process smart shape. Remember these are blue and encapsulate a complex sub-process. If we expand the sub-process via the plus sign, we can see the process in detail.

Notice there are multiple steps including decisions, assignments, and utilities.

Without getting into too much detail, this sub-process searches for duplicate cases based on criteria entered on the case designer. The process ends if no results are found.

"Send Email" Smart Shape

To configure:

-Email Address



-Subject

-Message or Correspondence Template with dynamic fields such as <>pyID>>, <>FirstName>>, <>LastName>>

To configure the "Send Email" Smart Shape, requires us to use the following properties.

Email address or a work party of where to send the email. A Work Party is a person, organization, or business involved or interested in some way in a case. A Work Party can be added or edited through details tab of the Case Designer. Work Party is covered in detail in some other advanced course.

Subject of the email and

Message body of the email or a correspondence template where the message content can be filled dynamically with various properties such as pyID, First Name, and Last Name. These property values are filled in at runtime.

Demo



Email Smart Shape

Let's see how to add the "Send Email" Smart Shape and configure it now. The case reaches the Procurement Stage after all the approvals are done. So, in the Procurement Stage, we want to notify the requestor if their purchase request has been approved.

Let's add a step called "Send Email Note" and let's make it the first step in that stage.

Let's check the configuration of the step added by clicking on "Configure Step Behavior." When a step is added for first time, the default step type is "Single Step Assignment."

Since this is not a single step assignment and this involves sending an email to notify the requestor, let's change it from the "Single Step Assignment" to "Multi Step Process."

Click OK for the warning note and click OK again to return to the stages. Click Save to save the stages.

Let's edit the flow of this newly created step. Click on the "Configure Process Detail" link under all the steps of the "Procurement" stage to open the flows in outline view. By default, the "multi step process" step type has two assignment shapes.

Since we are looking to send an email, delete both assignment shapes and add the "Send Email" Smart Shape and connect it between the start and end shapes.

Configure the shape by right clicking on the shape and selecting the View Properties menu option. There are three fields which are required when the Email Address option is selected. These are: To, Subject and Message. Fill in the appropriate values for these three fields and click OK to return to the flow diagram. Click Save to save the flow.

With the shape is properly configured, we can remove the draft mode option by clicking the "Draft on" button.

Let's edit the properties of the Smart Shape again. This time, let's change the "send to" option from Email Address to Party. This option allows us to send email to any work party. "RequestedBy" is a work party already configured for the requestor of the purchase request. We can leave the subject and message as they are and click OK and then save the flow.

Let's edit the properties of the Smart Shape again. This time, let's change the "Message" option from Rich Text to Correspondence. This option allows us to choose a correspondence template that can be reused for other email notifications and can also have dynamic fields. Type in "PurchaseRequestConfirmation" for the correspondence template.

Click OK, save the flow, return to the stages and save the stages.

Here is the "PurchaseRequestConfirmation" correspondence template with a message and a dynamic field. The sample message is "Your purchase request <>pyID<> has been approved." pyID within angle brackets is the dynamic field which is replaced by the case ID at runtime. We can have more than one dynamic field such as "requested for" name, line item description, if we want to.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- New Push Notification smart shape
- Push notification to mobile device indicating an action is required

- Send Email rule form has been updated to provide better ease-of-use

Send Email

Utility: **Send Email**

Perform automated processing in your case.

Automation Details
Define automation

Send to Email address Party

To *

Subject *

Message Rich text Correspondence

Font **Size** **A-** **A+** **Align** **Italics**

Audit note

Enable link if using breadcrumb trail

Specification
How should this utility should behave?

- Change Stage rule form has been updated

Change Stage Rejection

Utility: **Change Stage Rejection**

Perform automated processing in your case.

Automation Details
Define automation

Go to next stage Select a stage

Stage

Close open assignments from current stage on change

Audit note

Enable link if using breadcrumb trail

Specification
How should this utility should behave?

“Change Stage” Smart Shape



To Configure “Change Stage” smart shape:

- Stage Name

Changing to a stage that has been already been processed:

- Steps configured with “Launch on re-entry”

Change Stage will be required for us whenever we want to return back to a stage that has been already processed or jump to another primary stage or to an alternate stage.

To configure the “Change Stage” Smart Shape, all we require is the name of the stage to be selected from the dropdown values of the stage names for the “Select a stage” option.

If we re-enter a stage that we have previously accessed, using the change stage Smart Shape or by any other means, only the steps that are marked “Launch on re-entry” are kicked off again and the other steps that are not marked as such are ignored.

Demo



Change Stage Smart Shape

Open the Quote stage configuration by right clicking and selecting “configure stage behavior.” Here we see the behavior is configured to “Transition to the next stage” when all stage processes are complete and skipped. This means that the case automatically moves to the next stage when all the steps in quote stage are completed or skipped.

This behavior of the automatic transition to next stage can be changed using the “change stage” Smart Shape for certain situations. Let’s open the quote flows in outline view by clicking on the Configure Process Details link. To bring the “Review Request” flow into the process modeler, click on the flow name in the tree view. We see that this flow has an assignment “Purchase Review Request” and three flow actions, “Withdraw,” “Back to Data Entry” and “Review and Submit for Approval.”

The “Review and Submit for Approval” flow action is connected to the end shape. So, if the end user selects this action, the case automatically moves to the next stage as configured in the stage behavior configuration.

The Withdraw and Back to Data Entry flow actions are connected to the “change stage” Smart Shape. So, if the end user selects these actions, the flow changes to the stage identified in the shape.

Let’s view the properties configured for the “Change Stage” Smart Shape, named “Change Stage: Request Entry.” We see that the name of the shape is configured to be “Change Stage: Request Entry,” the “Select a stage” option is selected, and the “Request Entry” stage name is selected in the Stage dropdown. So, if the “Back to Data Entry” flow action is selected by the end user, the case moves back to the request entry stage.

Let’s view the properties configured for the “Change Stage” Smart Shape, named “Change Stage: Withdrawal.” We see that the name of the shape is configured to be “Change Stage: Withdrawal,” the “Select a stage” option is selected, and the “Withdrawal” stage name is selected in the Stage dropdown. So, if the “Withdraw” flow action is selected by the end user, the case moves to the withdrawal alternate stage.

Return to the stages by clicking on “Back to Stages.” The Request Entry stage would have been already processed once, when the end user selected the “Back to Data Entry” flow action and returned to the “Request Entry” stage through the change stage Smart Shape. Open the step behavior configuration on the only step “Enter Request Details” of the “Request Entry” stage. We see “Launch on re-entry” is selected for this step. So, this step is processed when we re-enter this stage and step. If not marked, this step is ignored when we re-enter this stage through the change stage Smart Shape or by any other means.

We have learned that Smart Shapes are available for some common utilities and common subprocesses and we saw how the Smart Shapes can be configured and used for basic tasks, such as sending an email and changing to a stage.

Exercise: Add and Configure Smart Shapes

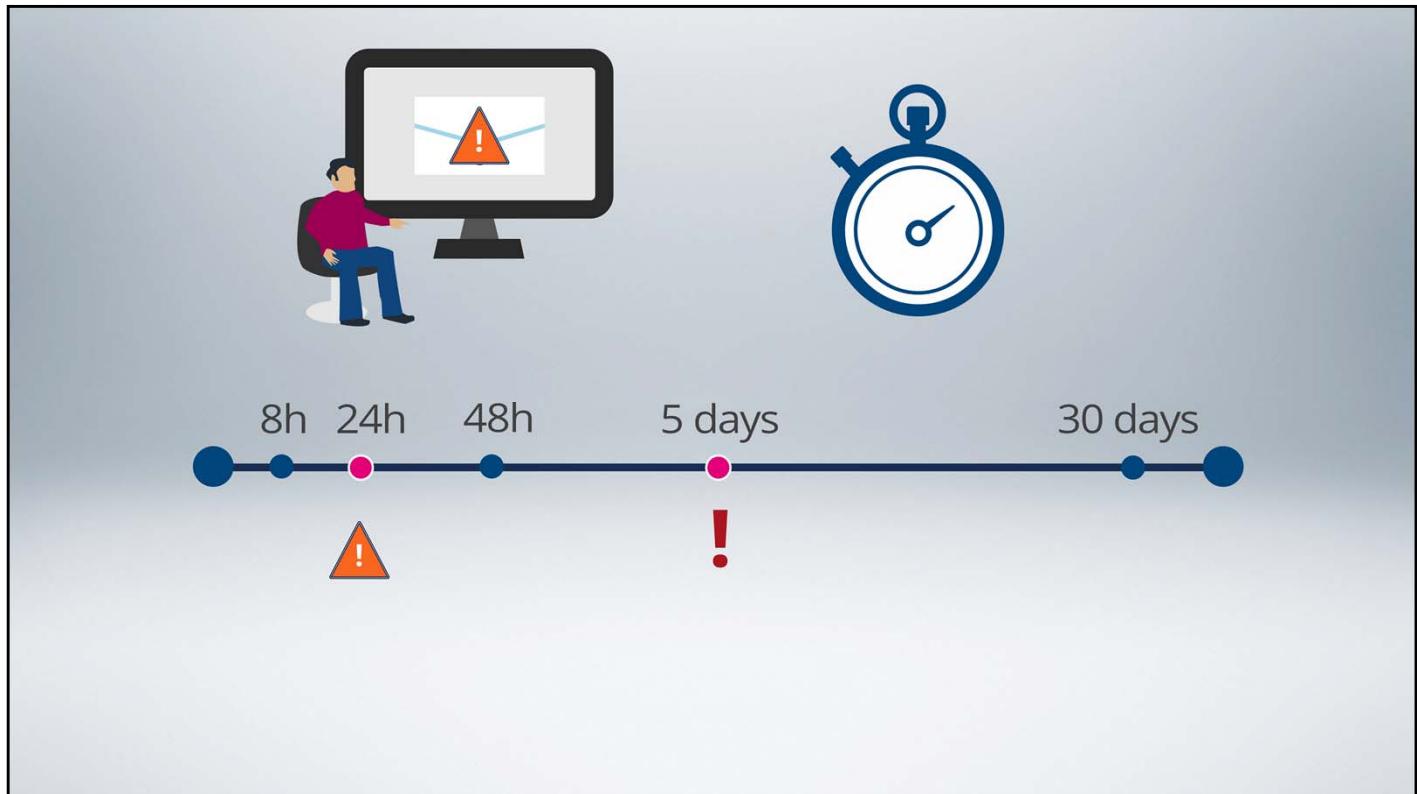


Creating a Case Level SLA

To ensure that cases are processed in a timely manner, we can establish and enforce goals and deadlines.

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

- Describe how service levels facilitate timely processing of assignments and cases
- Establish a goal and deadline for a case



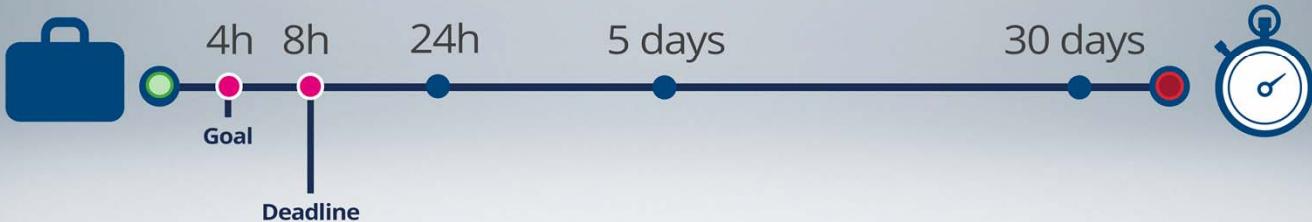
As end users perform their work – completing assignments and resolving cases – they often need to achieve certain performance milestones. For example, they may be required to respond to a customer complaint within 24 hours. Or they may need to resolve a support issue within 5 business days.



These obligations are called service-level agreements (SLAs), and they outline an agreed-to expectation of service to be provided by a business to their customers. They stipulate the amount of time in which the business intends to respond to the issue. As an item of work – either an individual assignment or an entire case – approaches its established deadline, it becomes increasingly urgent that it be completed. To help communicate this increasing urgency – as a way of enforcing the deadline – we can create a service level.

Case Service Level Intervals:

- Goal – the preferred time to complete the work
- Deadline – the maximum time allotted to compete the work



A service level consists of a goal interval – how much time we prefer end users complete the work – and a deadline interval – the maximum amount of time the end user can use. Ideally, the business wants its users to complete the work within the goal interval, and expects them to complete the work within the deadline interval. For example, an online retailer might set a goal of 4 hours to respond to a customer complaint, and a deadline of 8 hours.

In this course, we learn how to enforce an SLA for an entire case. In System Architect Essentials II, we'll learn how to configure a service level rule to enforce an SLA for an assignment, including the addition of escalation actions and urgency adjustments to goal and deadline intervals.

Demo



Establish a Goal and Deadline for a Case

Let's learn how to establish a goal and a deadline for the resolution of our purchase requests.

The Design tab – which we mentioned in passing near the start of this course – provides us with options to configure our case. Two of the options we can configure here are the goal interval and deadline interval for the purchase request.

We can establish a goal and deadline for either the current case, its parent, or the top-most case in the hierarchy. Since we're concerned with the purchase request, we'll keep the default setting.

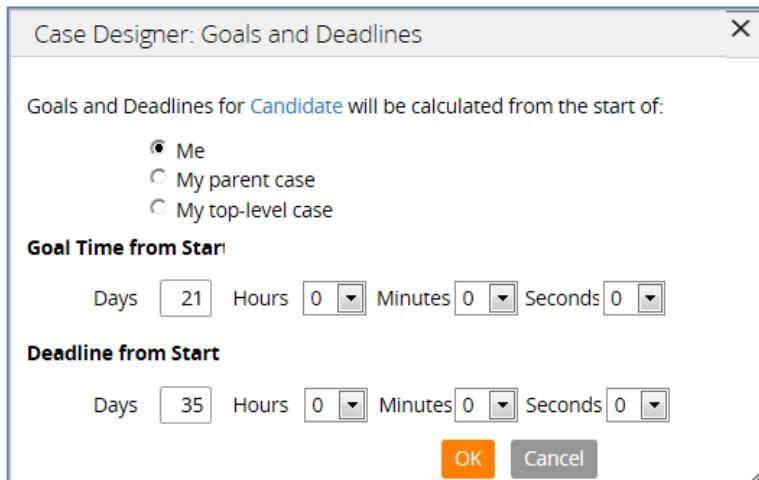
Each interval begins when the selected object is created. Let's configure a goal of 7 days, and a deadline of 14 days. This means that when we create a purchase request, we would like someone to resolve it in 7 days, and expect that someone will resolve it in 14 days.

Now, when an end user creates a purchase request, our goal and deadline intervals appear on the request. This lets the end user know how much time remains in each interval.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- Updated Goals and Deadlines form
 - Form instructions updated to provide better ease-of-use



Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- Stage Configuration now includes a Stage service level agreement field

- See the related article: Using service levels (SLAs) in Pega 7

<https://pdn.pega.com/case-management/using-service-levels-slas-in-pega-7>

You can add a stage SLA by doing the following:

1. Optionally, create a condition. For example, create a when condition using: +Create > Decision > When

Example: When Total Expense > 500

2. Create a service level record using: +Create > Process > Service Level

3. Add the service level agreement to the Stage Configuration rule form

Exercise: Set Goal and Deadline Intervals for a Case



Review of Advanced Case Processing

Case stages can be configured to perform tasks like skipping a stage, adding an optional process or action, identifying a resolution stage. Case can be routed to different operators, have SLA's defined on them. Task based smart shapes allow us to configure common tasks quicker.

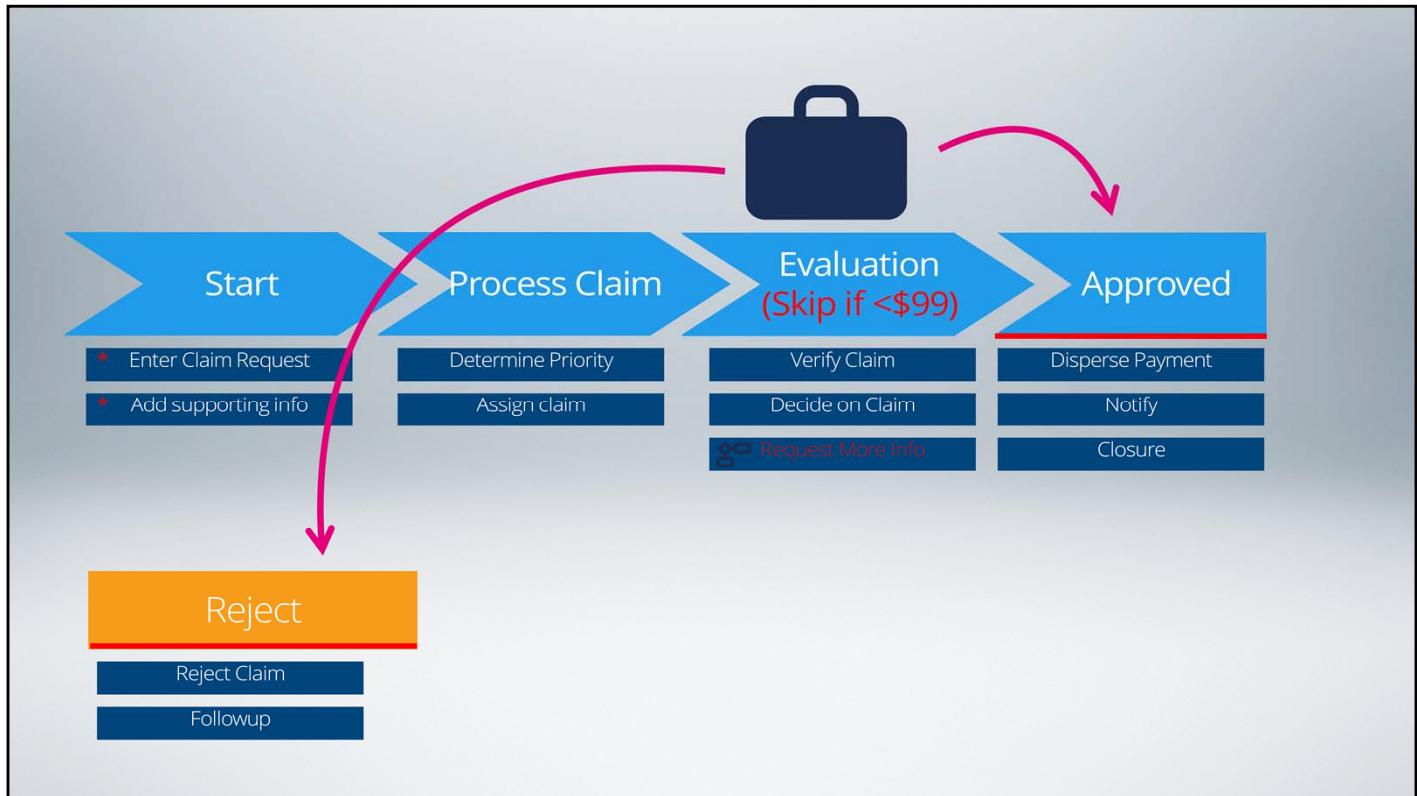
At the end of this lesson, you should be able to:

- Route cases to a work list or a workbasket
- Review the standard routing rules used
- Review the configuration that can be done on case stages
- Configure optional actions
- Configure optional processes
- Configure a case level SLA
- Use smart shapes in process to perform a specific task



A case moves through stages and in each stage, there are processes which are executed one after another.

Once all processes in a stage are complete, the case moves to the next stage.



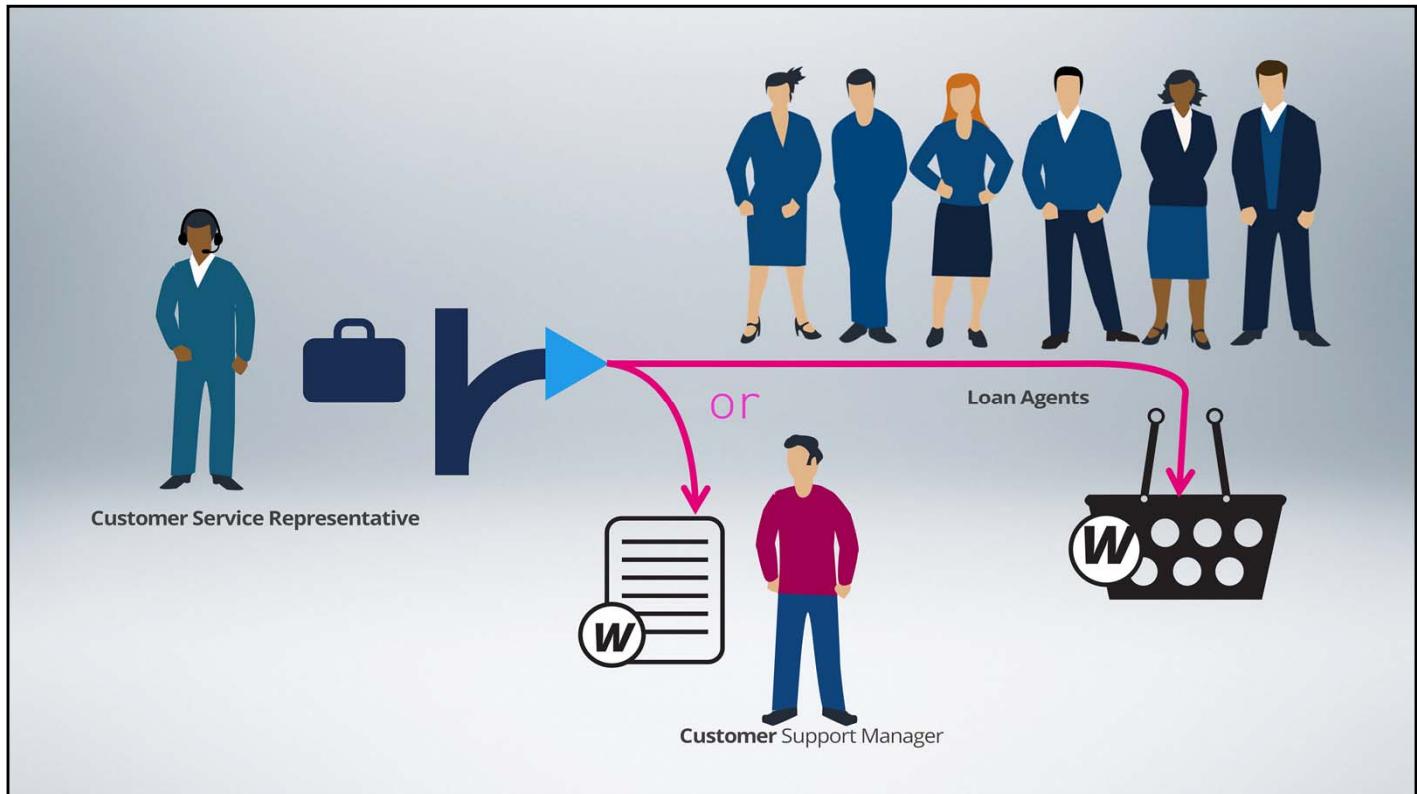
In certain scenarios, case processing is not as simple as we just described - we learned about how to configure case stages to address those scenarios. We learned about configuring case stages to handle specific scenarios such as renaming the stage,

skipping a stage based on a specific condition, such as if the claim amount is small, making sure the case has valid information required in the stage controlling the navigation from one stage to another and performing optional processes or actions during a stage.

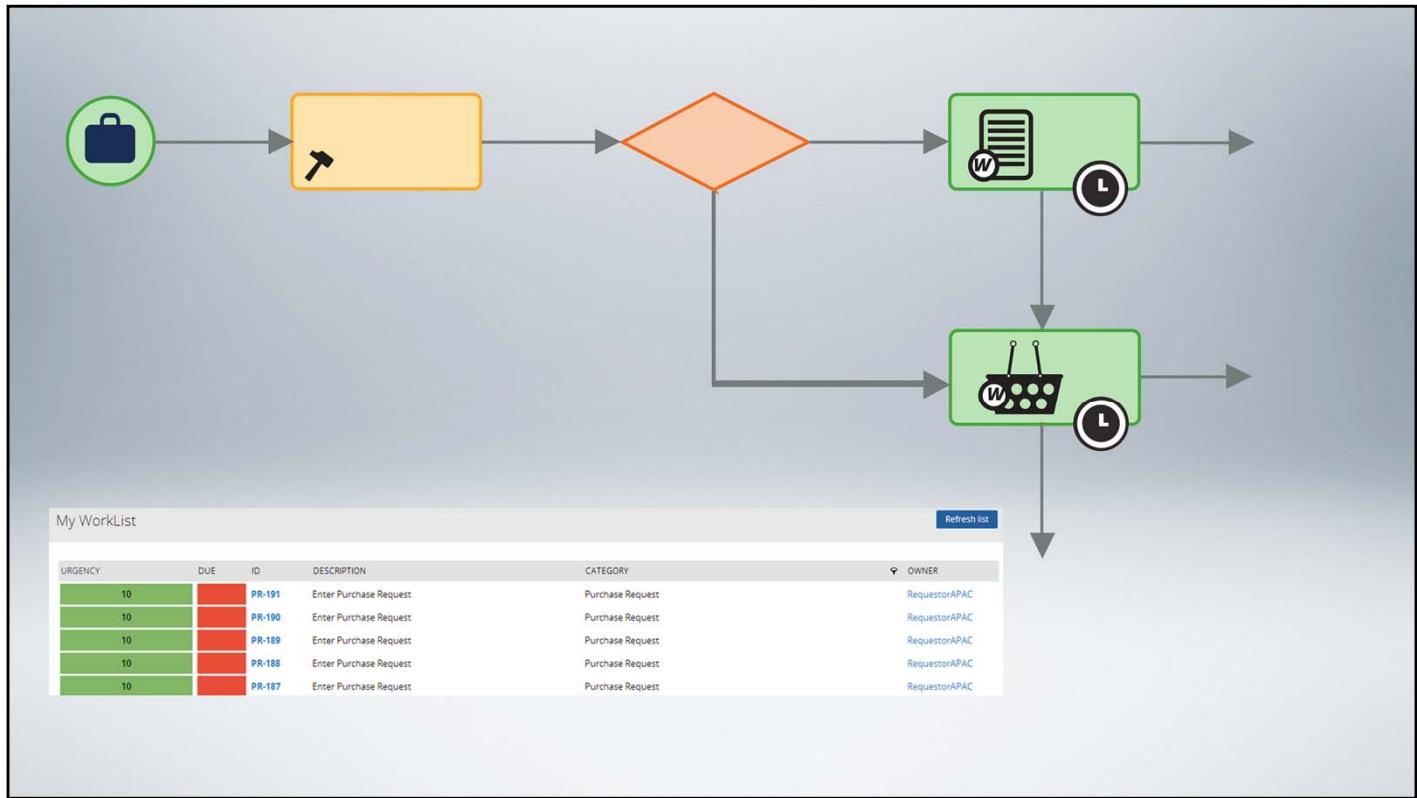
A stage can also be configured to identify when the case is resolved.

After the case stage has been completed, it can be configured to automatically move to the next stage or stay in the same stage. The latter might occur during claim processing where we wait until all the documents are received from the claimant and verified for accuracy before deciding to move the case to the processing stage or the rejection stage.

If the case stage is configured to not move to the next stage automatically, then we typically need to ensure that the flow rule must be configured by using appropriate rules such as adding the Change Case smart shape to move the case to a different stage.



Cases are usually worked on by more than one operator, so the system needs to be configured to route the cases to another operator to get an approval - or to a queue accessible to a group of users who are specialized in performing a specific task such as issuing loans.



A user creates a case, performs some actions and then the case might move to another user for further processing.

When defining processes, the assignment shape is the only step which requires user action.

The assignment shape can be configured to be assigned to a user's work list or to a specific work basket.

Routing Activities



ToWorklist



ToWorkbasket



ToDecisionTree

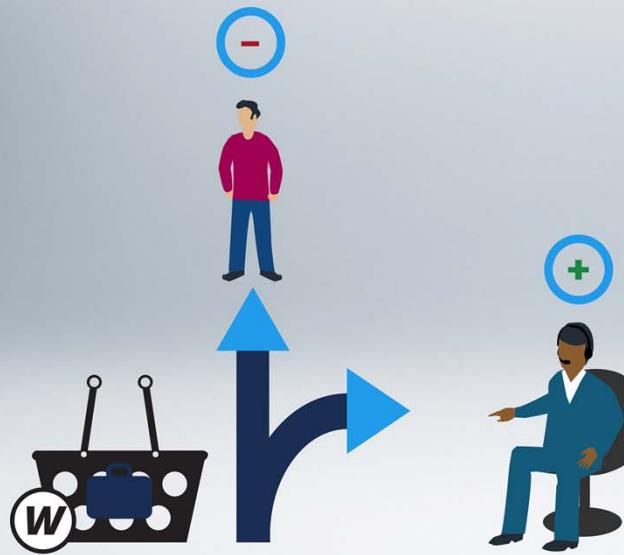
Routing uses routing activities that come with PRPC to configure the routing.

Some specific examples are – Toworklist when we want to assign the case to a specific user's worklist

ToWorkbasket when we want to assign the case to a specific workbasket,

ToDecisionTree to use a decision tree which has the set of conditions to decide where to route the case based on conditions.

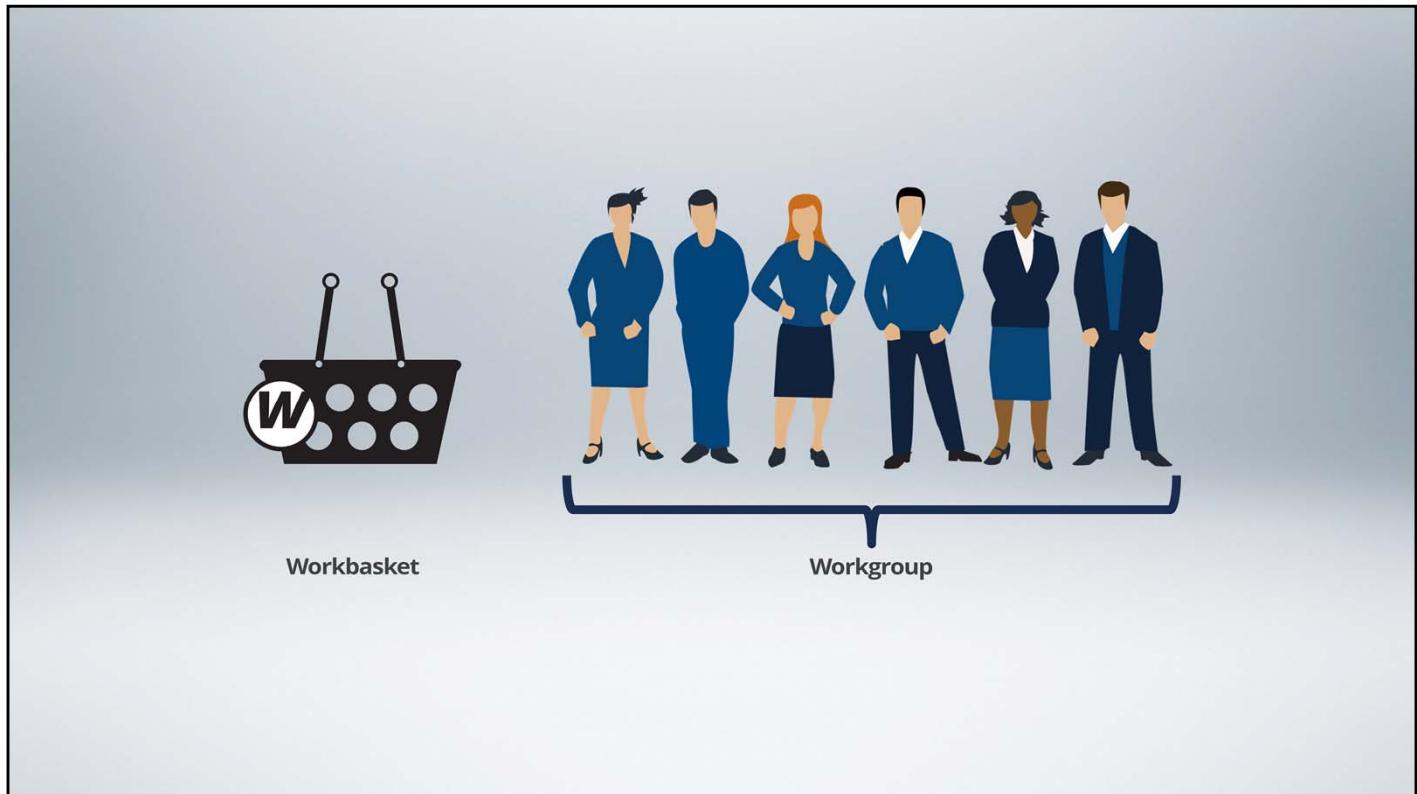
Routing Strategies



Push routing

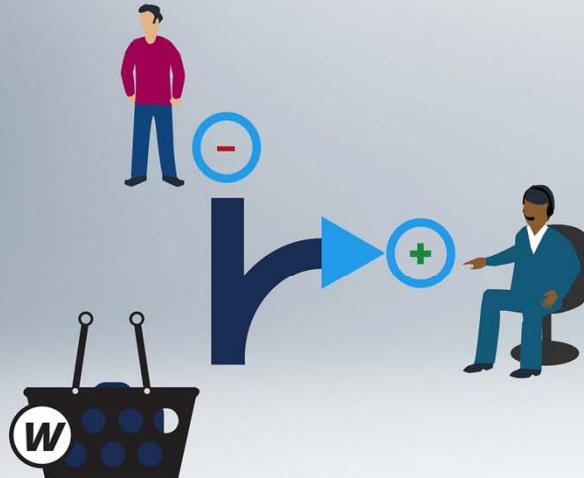
Pull routing

When routed to a workbasket, the system supports both Push and Pull routing to reassign the case to a specific user.



The workbasket is a queue shared by a set of users, usually belonging to the same workgroup. The work group is configured in the OperatorID record.

Routing Strategies



Push routing

System decides based on the rule configuration

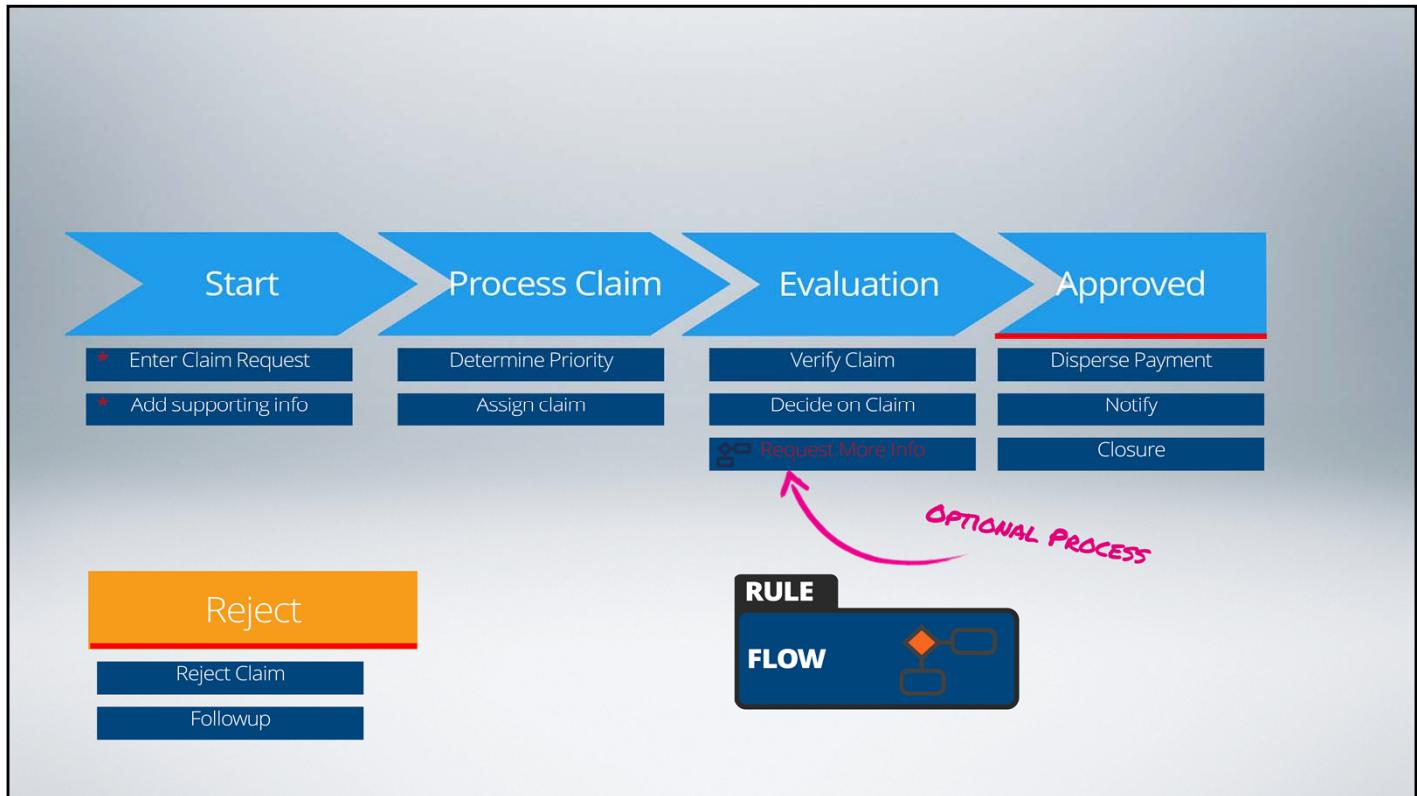
Pull routing

User picks from the list

Pega supports configuring both PUSH and PULL routing- When the case is assigned to workers either by using a ToWorklist activity or some other similar activity it is called PUSH routing,

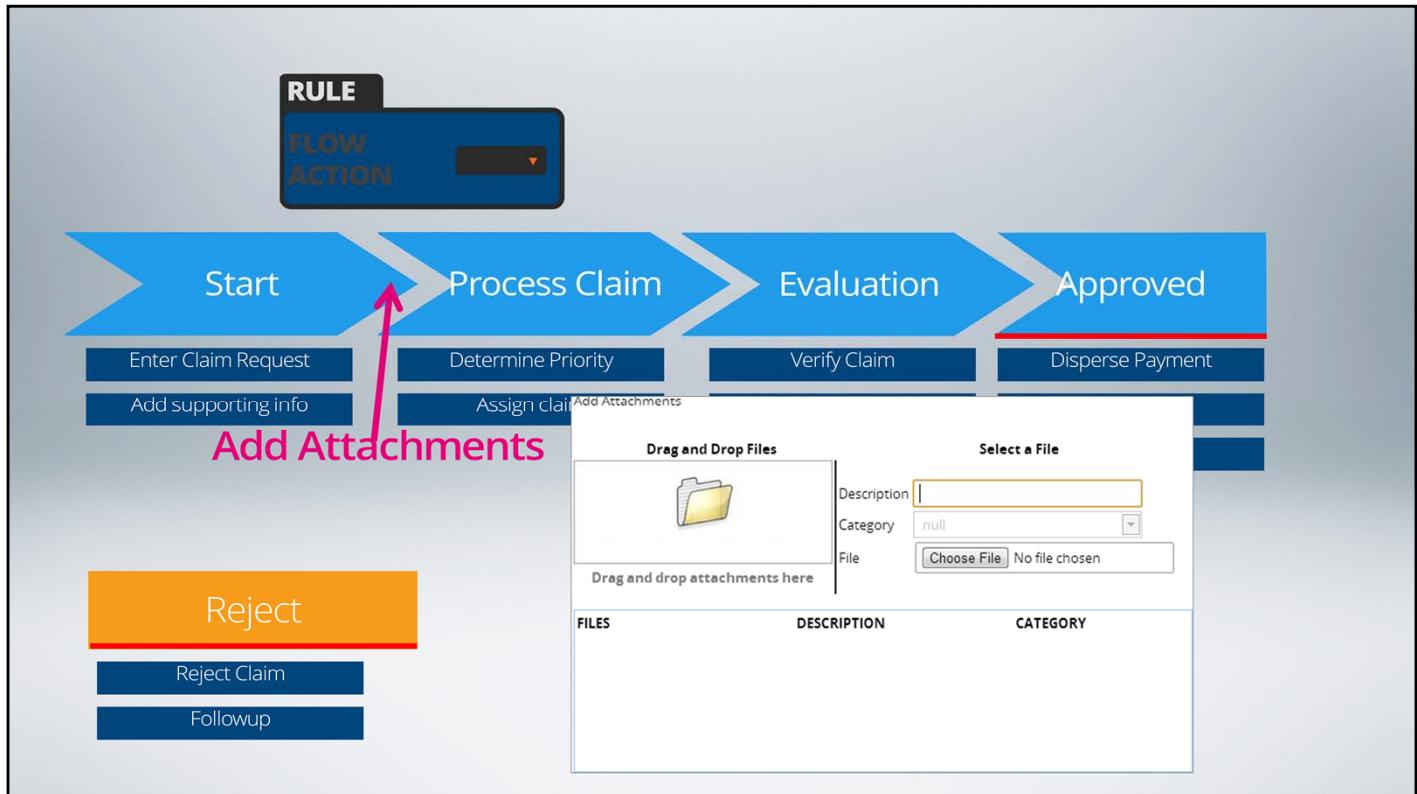
Alternatively, when the case is routed to a workbasket then the operator can access to the list of cases in the workbasket and pick a case, which is called PULL routing.

The decision of using PUSH vs. PULL is usually based on the application requirements.



Some specific circumstances might warrant the end user to start a process in addition to the ones that are automatically started. We define optional processes on a case stage to support this functionality. For example, when the claim is being evaluated, the processing agent would like to start a separate process to request more information. In most cases, the process we define here must be optional and the case should be able to complete processing without invoking it.

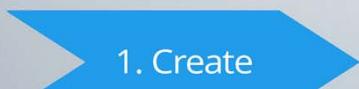
The optional processes use a flow rule and this can be any of the flow rules defined in the application or any standard flow that comes with the product.



In certain scenarios, we may require an optional flow action. This is configured by defining optional actions, for example we might need to provide the option to include an attachment to support the claim filed by the user.

The optional actions use a flow action rule and this can be any of the flow action rules defined in the application or a standard one that comes with the product.

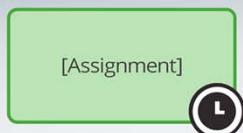
Optional Actions



Stage Specific



Process



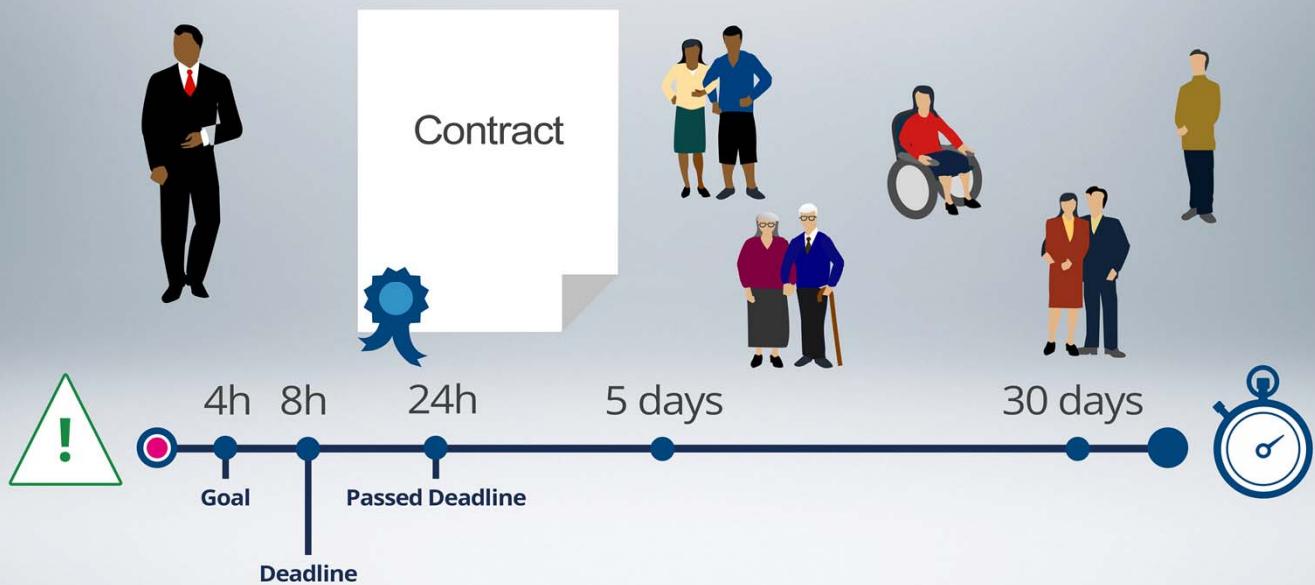
Single Assignment



Case wide

In addition to referencing an Optional action in a case stage they can also be added in a flow rule, a specific assignment or in the case definition. The scope of these optional actions is directly related to where they are added. This means that an optional action added at the case definition is available for all assignments in that entire case as opposed to adding it on a specific assignment where it would only be available on that one assignment.

Goals and Deadlines: Establish Performance Expectations



We also learned how to enforce service level agreements (SLAs) for our case. Service levels – expectations placed on the business to perform a task within a certain amount of time – ensure that work is performed in a timely manner. We can establish processing targets for our case to enforce SLAs, to define two thresholds for our case:

Goals and Deadlines: Establish Performance Expectations



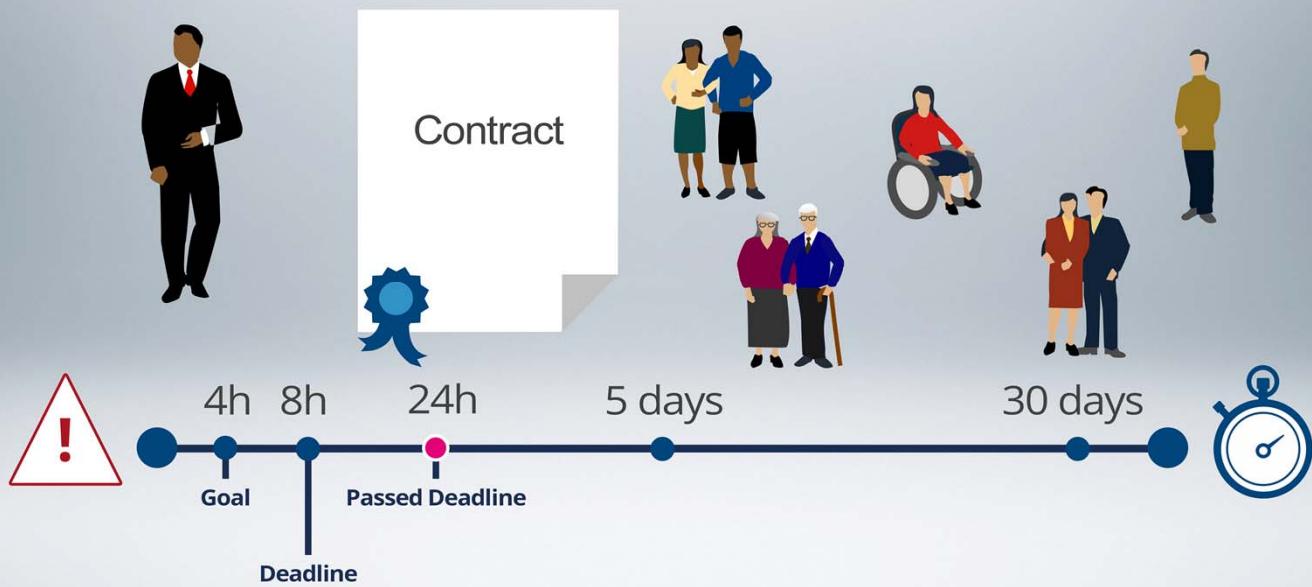
a goal – the time by which the business *wants* the work completed –

Goals and Deadlines: Establish Performance Expectations



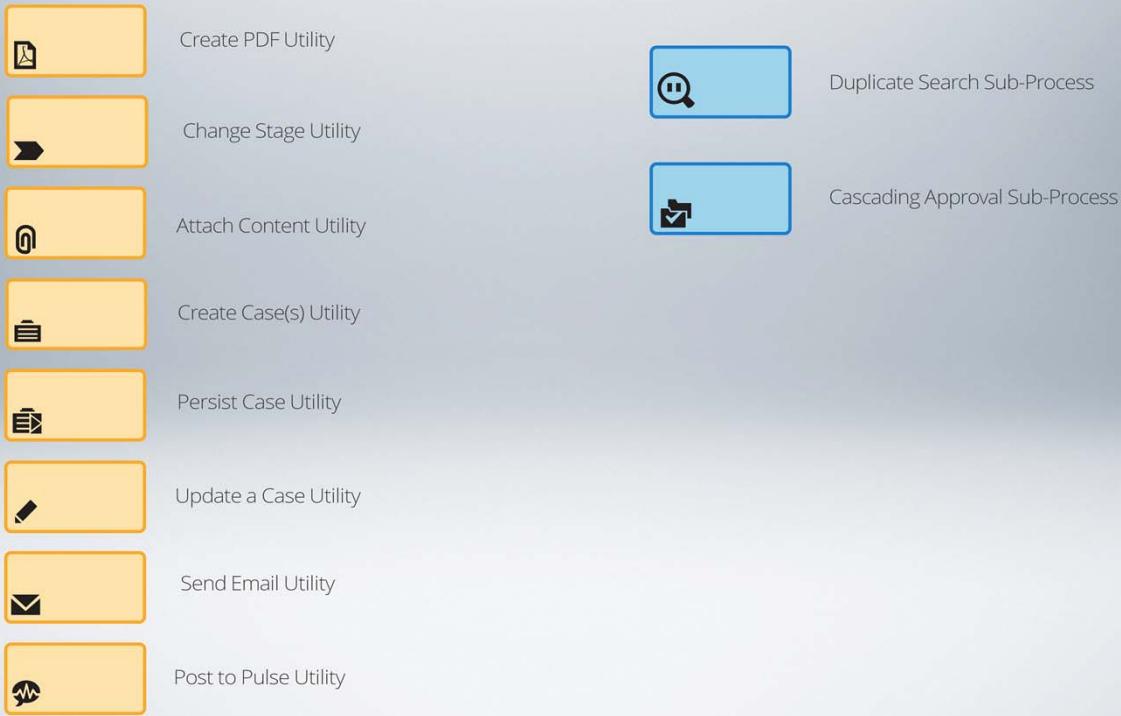
and a deadline – the time by which the work *must* be completed.

Goals and Deadlines: Establish Performance Expectations



As we approach each threshold, our case may become increasingly urgent, which end users can use to prioritize their work.

Smart Shapes: flow shapes preconfigured to perform specific tasks



Smart Shapes are pre-configured utilities or sub-processes that facilitate specific actions – such as creating a PDF, changing to a different case stage, or attaching a file. The Smart shapes are directly accessible through the process modeler for common utilities and common sub-processes with easy to configure parameters. We discussed a few of the smart shapes in detail that enabled us to do basic tasks, such as **send email** and **change stage** which can send an email and jump to a different stage respectively.

Module 10: Documenting the Application

This lesson group includes the following lessons:

- Documenting an Application

Documenting an Application

As we implement an application, we need to generate documentation so that other application designers can build off of our work.

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

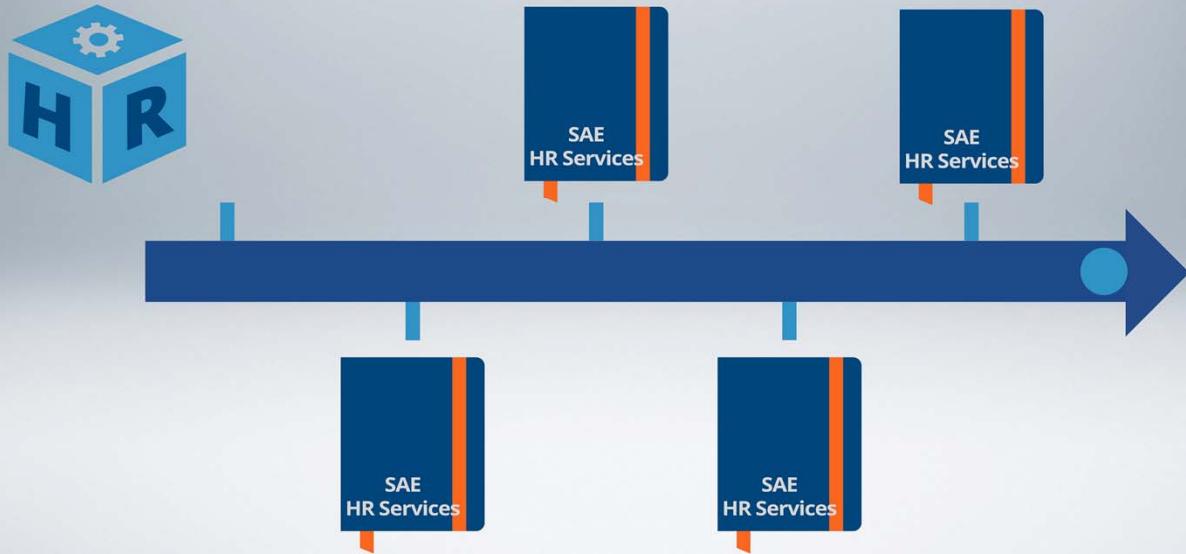
- Create application documentation throughout an implementation

Document a completed application



Now that we have completed our solution, we want to document it – not only for end users, but also for the next set of developers who will be responsible for maintaining and upgrading our solution.

Create documentation as needed



We don't have to wait until we finish our solution, however. We can also create documentation intermittently. One best practice is to document a solution at each meaningful milestone, as this helps to quantify the changes in our solution from one milestone to the next.

To help us document our solution, we can use the Document wizard. And, as we will see in this lesson, the wizard helps us to produce documentation for a solution at any point in its development, from inception to completion.

Document Wizard



Application Document: Documentation for developers and/or end users.



Application Profile: Proposal document for stakeholder review.



Specification Document: Specification sets for design review and elaboration.

Depending upon our needs, we can use the Document wizard to create one of the following types of documentation:

An **application document**. This is the type of documentation that we discuss in this lesson.

An **application profile**. The application profile is a proposal document, outlining what an implementation team intends to develop. The application profile is typically generated near the beginning of the implementation process, as a summary of the proposed solution, and includes business objectives, requirements, specifications, and project assumptions. Project managers and business analysts use an application profile as a solution proposal to submit to project ownership, to review and ultimately approve.

A **specification document**. We haven't discussed specifications much in this course, but they are an integral part of the development process. As part of this process, implementation teams will create, refine, finalize, and ultimately implement specifications as features within their solution. Business analysts and project managers define specification sets and can then publish a specification document for each set. Application designers can then use these documents as guides for drafting processes and user interfaces, and ultimately implementing properties, decisions, and other rules.

In this course, we focus on the application document, as this is the type we will be responsible for creating during an implementation. The last two options – application profile and specification document – are the responsibilities of other members of the implementation team, usually business analysts.

The Document Wizard steps

1

Select content: Select the contents of the document to be created.

2

Define data: Define data to use in screen screenshots.

3

Capture Screenshots: Take screenshots of the application, using the defined data.

4

Order content: Arrange the contents of the document, and add outside material.

The Document wizard guides us through a four-step process to generate documentation. Our choice of documentation to produce may eliminate steps from the wizard, so let's review the steps and the effect of our choice of documentation to produce.

In the first step of the wizard, we select the contents of our document. We can choose from three levels of documentation:

- A high-level overview that contains information about case types, flows, user interfaces, requirements, and specifications.
- Full documentation that contains everything in the high-level overview, plus information about secondary rules such as service levels and correspondence.
- A property reference that lists all of the data elements in our solution, organized by case type.

In the second step, we define the data to use in screenshots of our application. The wizard skips this step when generating an application profile or specification document.

In the third step, we capture screenshots of our application, using whatever example data we defined in the previous step. The wizard skips this step when generating an application profile or specification document.

In the last step, we can reorder the contents of our document, and even add outside materials, such as instructions or other documents. When we finish, we can generate our documentation as a Microsoft Word document.

Demo



Generating a Printable Application Document

Let's generate a printable application document for our sample Purchase Application.

First, let's check that the wizard will use the Application Document template, to ensure we generate the correct type of documentation.

We can choose from one of three levels of detail for our documentation, so let's select Full Application Documentation. For information about the contents generated for each detail level, hover over the help icon to the right of the drop-down list. We can opt to display the rule name or short descriptions when identifying each rule in the documentation. Since we've mostly dealt with the descriptions in this course, let's keep the default setting.

Our other options in this section are to suppress class and RuleSet references, and add links to the rules themselves. We haven't really discussed classes or RuleSets in this course, so let's enable this option. And since we don't really need links to rules in our printed document, we can disable that option.

The Document Table of Contents lists all of the sections of our document. Since we changed the detail level, we should refresh our contents. We can ignore the rest of the options on this step for now. By default, the wizard selects all of the RuleSets, case types, and document sections. Since we haven't really explored these concepts in this course, we can accept the default selections.

We can create documentation at any point of the wizard, but let's review the other steps before we generate our document. In the Define step, we can select sample data for screenshots, and select the criteria for our entity relationship diagrams. Entity Relationship Diagrams represent our application's data model in a visual

format, and are helpful for application designers to understand an application's data structure. We haven't really discussed RuleSets and classes in this course, and we need to understand these concepts to generate these diagrams, so let's de-select this option for now.

We can use an existing case to define our screenshot data. This option enables the wizard to populate our user interfaces with sample data when generating screenshots, which helps anyone who reads the documentation to understand the intent of each UI.

In the Capture step, we can capture screenshots from our application. If we decided to define sample data in the previous step, it will appear in our screenshots. Otherwise, the screenshots will contain no data. Any screenshots that have already been captured appear in the Screenshot Catalog, at the bottom of the form. If we were re-running the wizard, we could capture screenshots for only those user interfaces that have changed, which speeds up the capturing process.

Since this is our first attempt at documentation we need to capture all of our screenshots. When we do, the wizard displays all of our UI on-screen, one at a time, so it's important to not switch to another application until PRPC captures all of the UI used in our application. In the Order step, we can organize the contents of our document, and decide what elements to include or exclude. We haven't really used application requirements, so let's exclude the Requirement Matrix.

In addition, we can also add outside documents as either chapters or subchapters in our documentation. For example, if we wanted to add a summary of the application, we could upload that document as a new chapter, and move it to the desired spot in the chapter order. Since we don't need to add any contents to our document, we can skip this part of the process.

Now that we have completed the configuration steps in the wizard, we can generate our application document.

The Document wizard creates a new Microsoft Word document, and adds the appropriate content from our application. When the wizard completes, we can save the document and distribute it as we wish.

Pega 7.1.6 Update Notes

Updates for Pega 7.1.6:

- Application Documentation reminder
 - Documentation should be created using the 32-bit version of Internet Explorer
 - Refer to the following article:

<https://pdn.pega.com/deployment/issue-adobe-flash-and-activex-controls-do-not-work-in-64-bit-internet-explorer-browser>

Exercise: Documenting an Application

