Developer Coding Guidelines and Standards SOP

Created by Jen Wisniewski, last modified just a moment ago

Purpose

The DTC Pega Center of Excellence (CoE) Application Development Guidelines and Standards SOP is designed to define standards, guidelines and best practices for Rule configuration and application development on the DTC PEGA PaaS.

- The intended audience for this SOP includes:
  Citizen Developers
- System Integrators
- Internal COE developer

Roles and Responsibilities

Role	Responsibilities
SI	• Develop modules using coding standards and best practices found below • Rectify and any deviations found with code quality
COE Team	• Review code if needed • Rectify guardrail warnings for Citizen developer led modules • Review code quality and notify SI's of any deviations • Field questions and offer support to SIs and CD
System Owner (SO)	• Have awareness of standards and best practices • Ensure SI's are following coding standards

Measurements

Task	SLA
SI Development	No SLA; dependent on SI project plan
CD Development	No SLA; dependent on SI project plan
Code Review from COE Team	Dependent on project timeline and sizing; amount of time required to review will be determined prior to project kick off

COE Team to rectify guardrail warnings for Citizen Developer modules	Dependent on project timeline and sizing; amount of time required to review will be determined prior to project kick off

Developer Coding Standards for Pega at the VA

This section provides the required coding standards for Pega at the VA. Failure to meet these standards will cause Pega Guardrails to be flagged as needing correction prior to release.

Rule Type: Access Roles

Rule Type	Impact Category	Rule	Severity
Access Roles	Security	Implement custom security roles to maintain security requirements. Do not grant the PegaRULES:SysAdm4 security role to end-user access groups. Roles should default to having the least privileges to end-user access groups.	High

Rule Type: Activity

Rule Type	Impact Category	Rule	Severity
Activity	Functionality	When you use the History-Add method, use field values to support localizations.	Medium
Activity	Functionality	Use @equals functions instead of '==' for string comparisons.	Medium
Activity	Functionality	Do not use the Obj-Save method to save rules. Instead, use call pxUpdateRecord/pxCreateRecord or Call Save.	Medium
Activity	Functionality	In unavoidable scenarios, if you use the *Page-Change-Class* method, use the dynamic class referencing (DCR) pattern rather than a hard-coded class.	Low
Activity	Maintainability	Do not use the Page-New method to create a page that you plan to use as a step page for the *Obj-Open, Obj-Browse, RDB-List,* or *RDB-Open* methods. These methods create the specified page themselves.	Medium
Activity	Maintainability	Do not use hard-coded reusable values in properties such as URL or classes (such as the activity class) in activities. Instead, use data pages, decision tables/map values.	Medium
Activity	Maintainability	The *Page-New* method with an empty class initializes the page with the class that is defined on the **Pages and Classes** tab. Consider using dynamic class referencing for easy maintenance and extensibility.	Medium
Activity	Maintainability	Use automation rules instead of activity rules when you create an API.	Medium
Activity	Maintainability	If scope is defined only within an activity, give precedence to a local variable over a property. If local variables are defined but are not used in the activity, consider removing them.	Low

Activity	Maintainability	Provide comments to describe the context of significant steps to improve the readability and maintainability of an activity.	Low
Activity	Maintainability	Comment out activity step lines to prevent them from running at run time during testing. Commented lines should usually be removed before merging your rules. However, you might want to keep commented lines if they are helpful in debugging or if they help someone understand the activity.	Low
Activity	Maintainability	Start activity names with a verb or noun.	Low
Activity	Maintainability	Activities should not have more than 20 steps	Low
Activity	Maintainability	If an Activity rule is required, set the activity type on the **Security** tab to remove warnings. For example, set **Utility (default)**, **Validate**, or **Load Data Page**.	Low
Activity	Performance	Use data pages backup by RD or lookup rather than using the *Obj-Browse* method. Do not use the *Obj-Browse* method for lists of objects. If you use the *Obj-Browse* method, fetch only the column that you need and not the entire BLOB. If some BLOB data is used frequently, consider exposing that column.	High
Activity	Performance	The MaxRecords parameter in the RDB-List method is not automatically specified. If you leave the MaxRecords parameter blank, it defaults to 10,000. Consider specifying a maximum limit for improved query performance.	Medium
Activity	Performance	If you use a parameter with the *Obj-Browse* method (to/from/like), define a pre-condition on the activity step to verify that all required parameters exist before the *Obj-Browse* method is invoked.	Medium
Activity	Performance	Avoid using the Info forced logging level in the Log-Message method. Remove unnecessary log messages before check-in. Change the logging level to prevent cluttering of log files and avoid potential performance impact.	Low
Activity	Security	The **Allow direct invocation from the client or a service** or **Allow invocation from browser** checkbox is cleared by default. If you select the **Allow direct invocation from the client or a service**" checkbox, you must add a privilege to the activity.	High

Rule Type: Agent

Rule Type	Impact Category	Rule	Severity
Agent	Performance	Use queue processors or job schedulers instead of agents for better performance, resiliency, fault tolerance, and flexibility.	Medium

Rule Type: All Rules

Rule Type	Impact Category	Rule	Severity

All Rules	Functionality	Do not assume that a clipboard page is available. Instead, perform the required verification checks before running a data transform or activity.	Medium
All Rules	Functionality	Add Pega units for all the supported rule types. Run the units during the development in branches and during the merge process. Look for 100% Unit Test passed before merging the branches.	Medium
All Rules	Maintainability	Reduce or eliminate the use of duplicate loops in rules such as activities and data transforms, which can have a significant performance impact.	High
		Whenever possible, do as much processing as possible in one pass to avoid looping over the same data multiple times.	
		When looping over a data structure, be aware that the larger the list, the more times processing occurs. For example, you can use a report definition Class Join feature to gather data in one database call rather than making a separate database call for every result returned. If services are called within a loop, use data pages to cache results to improve overall performance.	
All Rules	Maintainability	Consider deprecating rules to post warnings on usage. Provide documentation on the deprecated rules.	High
All Rules	Maintainability	Avoid creating custom-coded rules. If no better option is available, then ensure custom-coded rules, such as activities with Java steps, non-autogenerated UI rules, and functions, are final.	High
All Rules	Maintainability	When you create a rule such as an activity or data transform, use parameters to clearly define the inputs and outputs so that system architects understand how the rule is used. If the rule needs a specific value to process, rather than passing a complete page reference, pass a scalar parameter.	Medium
All Rules	Maintainability	When you create a new rule that is similar to an existing rule, minimize code duplication. If you change only values but do not change the code in a rule, refactor the rule to use parameters instead of creating multiple versions of the same rule. If the same code is used multiple times, create well-documented, reusable assets that can be called from different places rather than copying the code. Determine the reusability of the rule, as this determines where to place the rule in the class and ruleset hierarchy.	Medium
All Rules	Maintainability	Carefully consider the Applies To class of your rules. Put rules in a higher parent class if you anticipate reusing them across all children or if you want to avoid duplicate rules. If a rule is not reusable, put it in the class where it is used. Do not flood abstract classes with rules that can be used only in particular concrete classes. If a rule is needed in multiple concrete classes, do not copy the rule to the sibling. Instead, promote the rule to the child class and withdraw the original rule to avoid duplicating code.	Medium
All Rules	Maintainability	Ensure that the name of a workbasket clearly identifies the purpose of the workbasket.	Medium
All Rules	Maintainability	Add test rules to a separate test application that is built on the target application.	Medium
All Rules	Maintainability	Remove all unused rules before you move to production.	Medium
All Rules	Maintainability	Document your rules when the purpose or use of a rule is not clear to improve maintainability. Document what the rule does in the description	Low

		and how the rule is used in Usage. Ensure that the rule label indicates the purpose of the rule in human-readable format.	
All Rules	Maintainability	When defining parameters, select the correct data type and provide a meaningful description and purpose.	Low
All Rules	Maintainability	Provide appropriate comments when checking in a rule after modification. Include a work item, such as a change request or error number, at the beginning of the description of the change. For example: US-00001 – Removed pyWorkPage Page Delete Step from Activity.	Low
All Rules	Performance	Use Alert logs, PAL, and Log Usage to monitor performance during development, UAT, and performance testing. If you observe performance issues, collaborate with performance tuning experts to resolve those issues. Assess performance load and verify the capacity of your production system.	High
All Rules	Performance	Implement Debug Log (oLog) in activities to debug and add warnings or errors to log files if an exception occurs. When writing log information externally, verify that logged information provides sufficient detail to assist in debugging an issue. By default, the application logs class name, activity name, step, date and time, requestor, and error type. Additional log items might include case ID, case status, and operator. Make sure that you do not log sensitive information. Specify the correct log level settings so that production logs and PDC are not cluttered. Remove or comment out System.out.print after development.	Medium
All Rules	Performance	Avoid using large text values in top-level pages. Elasticsearch indexes all top-level properties when it is enabled on a class. Consider keeping large text values in embed page properties.	Medium
All Rules	Performance	Use a wrapper activity to consolidate actions and reduce the number of requests to the server.	Medium
All Rules	Update	Do not block the availability rules in the framework layer	High
All Rules	Update	Replace deprecated rule references with recommended, available rules. Do not use deprecated activity methods.	High
All Rules	Update	Do not reference internal Pega rules in implementations.	High
All Rules	Update	The use of locatable pages was deprecated in Pega Platform version 7.3.1. Locatable pages are a legacy feature that is not supported by the Internal Language (IL) assembly. Using locatable pages would require falling back to legacy assemblers that do not utilize the IL assembly optimizations.	High
All Rules	Update	Use extensions wherever available, instead of overriding rules. Use Pega rules as-is and append new functionality in your application rules.	High
All Rules	Update	Pega Infinity has been enhanced with more guardrails and validations and with better coding standards. Some legacy features are no longer supported and must be modified to Pega Infinity standards. Update references such as $ANY or $NONE with appropriate class references.	Medium

Rule Type: Circumstance Definition

Rule Type	Impact Category	Rule	Severity
Circumstance-Definition	Functionality	List more likely outcomes in rows at the top of the table.	Medium
Circumstance-Definition	Maintainability	When circumstancing rules, use circumstance templates rather than circumstancing on properties to allow implementation layers to customize the templates.	Medium

Rule Type: Class

Rule Type	Impact Category	Rule	Severity
Class	Maintainability	Consider enabling **Propagate schema changes to child classes** for work classes to avoid explicit optimization at the customer end.	Medium
Class	Performance	Ensure that classes are not mapped to more than one purpose database table, such as *pr_data* or *pr_other*. Maintenance of exposed properties for more than one type of data prevents using the correct indexes.	High
Class	Performance	Consider not storing the history of classes in environments in which changes are frequent and for which you have no business reason to track updates.	Medium

Rule Type: Clipboard

Rule Type	Impact Category	Rule	Severity
Clipboard viewer	Performance	Identify and remove unnecessary clipboard pages at the end of each process. Redundant pages use JVM memory.	High

Rule Type: Data Pages

Rule Type	Impact Category	Rule	Severity
Data Pages	Performance	Running the load mechanism on every interaction for a data page can impact performance. Select this option only when the business use case requires the freshest information.	Medium
Data Pages	Security	Select the correct user-level access group for all node-level data pages. Create an access group used specifically for batch processing (data page loading and agent activity processing).	Medium
Data Pages	Update	Do not remove parameters for a data page. Never remove parameters from shipped data pages in the framework layer. The implementation layer might use your data pages with Key,Value; removing a parameter might make referenced rules invalid, causing exceptions in flows.	High

Data Pages	Update	Avoid adding required parameters to old rules. Avoid adding new required parameters to shipped data pages in the framework layer. The implementation layer might use these data pages, and adding new parameters might make their referenced rules invalid, causing exceptions in flows.	High

Rule Type: Data Transformation

Rule Type	Impact Category	Rule	Severity
Data Transformation	Maintainability	Ensure that data transforms do not use the *pxExecuteAnActivity* function to call an activity. Evaluate whether the called activity can be transformed to a data transform.	High
Data Transformation	Maintainability	Consider modularizing a data transform so that it does not require more than 20 steps. This improves reusability and unit-level quality.	Medium
Data Transformation	Maintainability	Inline Java is always overhead in terms of maintenance. Consequently, consider using a data page or data transform. If unavoidable, RUF is recommended.	Medium
Data Transformation	Maintainability	Do not use more than five levels of nested steps in data transforms.	Medium
Data Transformation	Maintainability	Consider adding comments to data transforms for easier readability, maintenance, and potential reuse	Low
Data Transformation	Maintainability	Under most circumstances, do not leave disabled steps in data transforms. Review and remove disabled steps if they are not required.	Low
Data Transformation	Maintainability	To improve readability, avoid using () () Instead, consider using **Append and map to.**	Low
Data Transformation	Performance	Enable call super-class data transform checks only if you have a data transform with the same name in the class inheritance tree. Doing so avoids the overhead of looking up to parent classes.	Low

Rule Type: Data Types

Rule Type	Impact Category	Rule	Severity
Data Types	Update	Pega Infinity has updated data management and modeling features that provide better design and run-time performance. Update these components to new features. There are no auto-update utilities for most of the features in this area. Update old data tables to the new local storage data types to comply with Pega Infinity standards.	Medium

Rule Type: Database

Rule Type	Impact Category	Rule	Severity
Database	Maintainability	Do not use custom store procedures to accomplish a task.	High

Rule Type: Decision Tables

Rule Type	Impact Category	Rule	Severity
Decision tables	Functionality	Specify the most likely outcome rows before you specify less likely outcome rows.	Medium
Decision tables	Performance	To avoid slow performance, limit your decision tables to no more than 300-500 rows.	High

Rule Type: Decision Tree

Rule Type	Impact Category	Rule	Severity
Decision tree	Functionality	Do not redirect a rule to itself under **Circumstance Value** in the **Options** field of the **Results** tab. Avoid creating a circular set of redirections (A to B and B to A) because this causes an exception at run time.	High
Decision tree	Performance	To avoid slow performance, limit your decision trees to no more than 300-500 rows.	High

Rule Type: Declare Expressions

Rule Type	Impact Category	Rule	Severity
Declare Expressions	Maintainability	Adopt a naming convention for properties designed to be computed in Declare Expression rules. Create Declare Expression rules early, using stub or dummy expressions. By following naming conventions, developers can easily identify declare expression properties and can avoid setting values procedurally.	Low
Declare Expressions		The use of reference properties in target or page context of an expression can result in unpredictable behavior. Consider using non-referenced properties instead.	
Declare Expressions	Functionality	If there are multiple triggers on the same class (user-defined or inherited), then the order in which the triggers fire is not guaranteed. Implement trigger activities in such a way that they are mutually exclusive and independent of each other in terms of functionality.	High
Declare Expressions	Functionality	Do not create declarative indexes for top-level Single Value properties.	Medium
Declare	Maintainability	Do not use a **Complex Java Expression** under **When condition** in	Medium

Expressions		the **Conditions** field of the On Change **Properties** tab.	

Rule Type: Delegated Rules

Rule Type	Impact Category	Rule	Severity
Delegated Rules	Security	Delegated rules continue to work after you update to Pega Infinity, but they might need updates to address changes to referencing rules form changes. Skimming is recommended during update, which might require delegation re-mapping. Changes to rule forms require business retraining. Update delegated rules wherever required	Low

Rule Type: Exception Handling

Rule Type	Impact Category	Rule	Severity
Exception handling	Functionality	Handle database calls and other integration calls by incorporating preconditions and transitions that are related to errors and exceptions. Use Jump To Later Step and Exit Activity effectively in activities to handle exceptions. Consider raising the correct PDC alerts when errors occur.	Medium
		Using the *Obj-open*, *Obj-refresh-and-lock*, *Obj-open-by-handle*, and *Obj-Delete-by-handle* methods requires the handling of lock acquisition failures.	
		Use the Obj-save and Commit methods to handle potential DB transaction errors.	
		Use Connect- methods to handle the potential for connector failure.	
		Not handling an error might result in ungraceful errors.	
Exception handling	Functionality	Implement the Error flow in all critical use cases.	

Rule Type: Flow

Rule Type	Impact Category	Rule	Severity
Flow	Functionality	Always provide the status of an assignment and the status of work in the parameters of an assignment task.	Medium
Flow	Functionality	Do not hardcode the workbasket or router user name in the routing unless it is important. Use a parameter page instead.	Medium
Flow	Functionality	Provide meaningful and readable audit information in a flow. For example, enter a correct action description in the assignment task. Remember that whatever action comes in an assignment is described in the work item history as Assignment To and the added text.	Medium
		As a best practice, use one of the following prefixes to support standard reports and maintain continuity in between your custom status	

Flow	Functionality	and standard status values. New- Open- Pending- Resolved-	Low
Flow	Maintainability	To create the implementation layer case type, select the **Other** option and use DCR to specify the class name rather than selecting the case type in the Create case smart shape.	Medium
Flow	Maintainability	Ensure that a Business rule flow is neat and readable, as it can be exposed to business users. Improve readability by adding annotation.	Medium
Flow	Maintainability	Ensure that utilities serve a business purpose. Instead of adding one utility that performs an entire step, separate the utility into separate smaller utilities, and use these smaller utilities as individual identity in the flows. This makes the flows descriptive and readable.	Medium
Flow	Maintainability	Ensure that the number of smart shapes (excluding connectors) is not higher than 15 per work flow.	Medium
Flow	Maintainability	Flow implementation must be model-driven. For example, identifying and using decision rules in a flow rather than an activity makes the logic of the flow clearer.	Medium
Flow actions	Maintainability	Give flow actions names that describe the functions of the flow actions. For example: **AttachAFile**, **Notify External**.	Medium
Flow actions	Maintainability	Use a validate rule to perform data validations, and invoke the same validate rule from the flow action. Validating data with an activity in post-processing using the Obj-Validate or Property-Set-Messages methods does not support extensibility.	Medium

Rule Type: Map Values

Rule Type	Impact Category	Rule	Severity
Map Values	Functionality	Do not redirect a rule to itself under **Circumstance Value** in the **Options** field of the **Results** tab. Avoid creating a circular set of redirections (A to B and B to A) because this causes an exception at run time.	High
Map Values	Performance	To avoid slow performance, limit your Decision Map rules to no more than 300-500 rows.	High

Rule Type: Navigation

Rule Type	Impact Category	Rule	Severity
Navigation	Maintainability	Do not configure the Control action by using a hard-coded class. Consider using dynamic class referencing (DCR) for easy maintenance and extensibility.	Medium

Rule Type: Portal

Rule Type	Impact Category	Rule	Severity
Portal	Update	Legacy fixed or custom portals are no longer recommended in Pega Infinity, because they do not support cross-browser or responsive design. Debugging tools such as Live UI might not work for these legacy portals. Replace fixed or custom portals that lack responsiveness with Pega Infinity composite portals.	Low

Rule Type: Property

Rule Type	Impact Category	Rule	Severity
Property	Functionality	Use **Date Time** rather than the **Date** or **TimeOfDay** types for properties that are part of persistent objects in your application. Avoid using **Date** property types in persistent objects even if your application is used in only a single time zone, unless a time zone value is also stored in another property of the object.	
Property	Maintainability	Do not duplicate properties if you can avoid it.	Medium
		If multiple classes use a property, define the property once where it is accessible to all classes that need it.	
		If one of two classes that need to use the property do not share a parent (for instance, one is Embed- and the other is not), you can copy the property. Ensure that you use the exact same name.	
Property	Maintainability	Avoid copying data.	Medium
		During processing, data should exist in only one place. In Pega Infinity, that means in a single property.	
		If user input has a cascading effect (setting a value in one place creates defaults for a number of other values), then you can copy data if the effect is one-directional. In other words, you can copy data only when you can define it using something like a declare expression. Verify that you do not create circular dependencies when you copy data.	
Property	Maintainability	Select property types from the Type drop-down list rather than specifying them in text.	Medium
Property	Maintainability	Configure property names that are based on nouns or noun phrases and describe the property.	Low

Rule Type: Report

Rule Type	Impact Category	Rule	Severity
Report-Definition	Functionality	Consider using the SmartInfo capability to enrich the value of reports.	Low

Report	Performance	Fetch only the required columns and not all columns from the database.	High
Report	Performance	Avoid unnecessary Join filters while fetching database records.	High
Report	Performance	Verify that the report definition filters load only necessary data to the clipboard. Configuring filters correctly is better than using row visibility conditions to hide rows in a grid.	High
Report	Security	Add the correct privileges and restrictions to reports to control which user groups can run a report.	High
Report	Update	Do not remove properties from report definitions if they are declared as available, especially in the framework layer. Never remove properties from report definitions that are shipped, even if those properties are not used. If the implementation layer attempts to reuse a report definition for which properties have been removed, it affects the referencing rules.	High

Rule Type: Rule

Rule Type	Impact Category	Rule	Severity
Rule-Obj-Activity	Performance	Always use Obj- methods (rather than RDB-methods) to ensure that the BLOB and columns stay in sync. Use RDB- methods only for Always use Obj- methods (rather than RDB-methods) to ensure that the BLOB and columns stay in sync. Use RDB-methods only for.	Medium
Rule-Utility-Activity	Functionality	Use the String.Equals () or String.equalsIgnoreCase () Java method instead of the == or != operators, to compare the value of a property to a literal string or to the value of another property.	Medium
Ruleset	Update	Only application ruleset validation (ABV) is recommended for ruleset setting.	Medium

Rule Type: Section

Rule Type	Impact Category	Rule	Severity
Section	Functionality	Use Refresh When and Refresh Other Section.	Medium
		Use Refresh When whenever possible, especially for read-only references that need to remain synchronized with data on the server, as it declares dependencies (This section needs to be updated when .x changes). When this is not possible, for example, if you have editable references that are updated based on other input or based on untrackable inputs such as button clicks, use a Refresh Other Section targeted refresh.	
		Avoid Refresh section.	
		Refreshes interrupt the user experience. The user must wait, the screen seems to freeze until the section refreshes, and the user might lose	

Section	Functionality	focus and manually return to the original screen. Manage client-side interactions by using the change tracker whenever possible. For example, instead of refreshing a section with a pre-processing data transform, call a data transform or use set value and set your visibility conditionals to evaluate on the client. **Refresh a section in the following circumstances**: Property values are updated on the server and the new values must be reflected in the UI. If you must submit an action that causes a change to more than one property and that occurs only on the client, such as deleting a row. If you must remove parts of the UI must be removed from the DOM because of other input. Do not refresh a section in the following circumstances: If you need to submit user input. Instead, use post value. If you need to call a data transform or an activity after a user action. If you need to recalculate visibility, the enabled/disabled state, or the read-only state. Select the Evaluate on client checkbox next to the Expression field Make a judgment call regarding refreshing a section in the following circumstances: If you have a complex expression to set visibility, enabled/disabled, or read-only state that requires a When condition. Consider using a declare expression to provide a value for a property and conditionalize on changes to that declare expression. If you must handle style changes based on value changes. If you must validate input and potentially show messages related to various properties. Note that validation provided by the controls, combined with a validate rule on the flow action, should cover most situations.	Medium
Section	Functionality	Use set value to set flags. Consider the following use case: you have a property *.pyIsThingVisible*. Put the property in a hidden input field in the UI and use set value *'pyIsThingVisible* = true (or false). This is a more efficient method than using Run data transform pzSetIsThingVisible (and creating and maintaining the data transform).	Medium
Section	Functionality	Follow the general design and development approach that read-only display of a particular section is controlled outside of the section through the harness in which it is included.	Low
Section	Maintainability	Do not hard-code the create work action class input parameters. Consider using dynamic class referencing (DCR) to improve maintainability and extensibility	High
		When deciding whether to use a section or to nest a dynamic layout, minimize the number of sections that you create to improve performance and maintenance. Use a section in the following circumstances:	

Section	Maintainability	You plan to use the section in multiple places.	Medium
		The contents of the section must be refreshed from the server more often than the content around it.	
		The data requires a table/grid layout but you want to include it in the same layout as the rest of the UI to share a container format, style, or header.	
		The content of the section uses a different step page than the UI around it.	
		In other circumstances, nest a dynamic layout.	

Rule Type: UI Rules

Rule Type	Impact Category	Rule	Severity
UI rules	Functionality	Do not use horizontal scrolling. Minimize vertical scrolling.	High
UI rules	Functionality	Do not use fixed widths. For table-style layouts such as grids and tree grids, always set the width to 100%. This allows the display to adapt to the device that the viewer uses.	Medium
UI rules	Functionality	Ensure that autogenerated controls have validation options.	Medium
UI rules	Functionality	Improve the user experience by removing unneeded clicks, especially for tasks that are simple or frequently repeated.	Medium
UI rules	Functionality	To support localization, use field values for text that is displayed to users.	Medium
UI rules	Functionality	To ensure that client validation works, enable client validation on a harness.	Medium
UI rules	Functionality	Select Localize on Section rules.	Medium
UI rules	Functionality	Avoid using icons for controls. Use link text or a button for clarity and ease of localization.	Low
UI rules	Functionality	Bundle your transactions in minimum calls. For example, call activity and refresh section can be bundled.	Low
UI rules	Functionality	Configure input validation on both the client and the server	Low
UI rules	Maintainability	Whenever possible, set the label value for a control from the short description for its property rather than from inline labels, which increase maintenance costs.	Medium
UI rules	Maintainability	Use When rules rather than expressions in section/layout/control visibility conditions if the same condition is used in multiple places for easy maintenance and extensibility.	Medium
UI rules	Maintainability	Consider using pagination of records for memory and performance optimization based on your specific use case requirements. Enable pagination when fetching records from the database.	High
UI rules	Maintainability	Records that are fetched in memory should be just-in-time rather than persisted throughout the user session. Use the correct data page	Medium

		scoping.	
UI rules	Performance	Enable defer load for tabs/section for faster screen loads. Load data just in time for when the end user needs it.	Medium
UI rules	Performance	Use JPEG and PNG images only if the image size is greater than 100x100 px. For smaller images, consider using the Icon font so that you can avoid time-consuming requests to the server.	Medium
UI rules	Update	Non-autogenerated, custom, or hand-coded rules are not likely to be cross-browser-compliant or responsive. For better performance, security, ease of maintenance, and support for cross-browser consistency and responsiveness, move to autogenerated UI components. Replace custom JavaScript, HTML, JSP, and other code with standard features that are available in Pega Infinity. Do not create non-autogenerated records, such as sections, HTML rules, HTML fragments, and text files. Generate all screens for JSP (including *Rule-HTML-Property*, *Rule-HTML-Fragment*, and *Rule-Obj-HTML.*	High
UI rules	Update	Pega Infinity has controls that support new features, such as offline capability, responsiveness, and cross-browser support. Deprecated or custom controls do not have these capabilities, so you should update to controls that are shipped with Pega Infinity. Replace custom or deprecated controls with the latest Pega Infinity controls. If no updated control is available, make custom controls cross-browser and responsive.	High
UI rules	Update	Use the skin rule for all styling needs in the application. This reduces maintenance cost and provides flexibility to style consistently, regardless of browser or device. Make applications cross-browser and cross-device-compliant. Inline styles and custom CSS must be re-engineered using Pega Infinity skin, mixins, and style formats.	High
UI rules	Security	Add the correct roles to workbasket definitions to control who can process assignments in the workbasket.	Medium

Pega Guardrails to Success

1. What is the definition of Guardrails?

The Ten Guardrails to Success are design guidelines and recommended best practices for PRPC and Pega Platform implementations. Following the fundamental principles promoted in the Ten Guardrails to Success leads to rules-based applications that are well designed, straightforward to maintain, and architected to Build for Change. These principles are your keys to success with PRPC and the Pega Platform. As development is in flight, Pega will flag items that do not meet the required coding/development standards based on these Guardrails.

2. Pega's 10 Guardrails to Success

Guardrail	Description
1. Adopt an iterative approach	• Define an initial project scope that can be delivered and provide business benefit within 60 to 90 days from design to implementation.

	• Document five concrete use case scenarios up front and evaluate them at the end to calibrate benefits. • Use your scenarios as storyboards and ensure that each delivers a measurable business benefit.
2. Establish a robust foundation	• Design your class structure complying with the recommended class pattern. It should be understandable, be easy to extend, and utilize the standard work and data classes appropriately. • Use your organization entities as a starting pattern, and then proceed with class groups. • Lead with work objects. Create the class structure and completed work objects early. • Position rules correctly by class and ruleset. • Actively use inheritance to prevent rule redundancy.
3. Do nothing that is hard	• Use out-of-the-box functionality as much as possible, especially in the initial project release. • Avoid creating custom HTML screens or adding buttons. • Always use the “Auto Generated HTML” feature for harness sections and flow actions. • Always use the standard rules, objects, and properties. Reporting, urgency, work status, and other built-in behaviors rely on standard properties. • Never add a property to control typical work or to manage the status or timing of work.
4. Limit custom Java	• Avoid Java steps in activities when standard rule types, library functions, or activity methods are available. • Reserve your valuable time and Java skills for implementing things that do not already exist.
5. Build for Change	• Identify and define 10 to 100 specific rules that business users own and will maintain. • Do not include activities on this list. Use other rule types for business-maintained logic.
6. Design intent-driven processes	• Design your application control structure to consist of flows and declarative rules, calling activities only as needed. • Use flow actions to prompt a user for input. • Present fewer than five connector flow actions for any individual assignment. If you need more than that, you need to redesign the process. • Create activity rules that implement only a single purpose to maximize reuse.
7. Create easy-to-read flow	• Design your flows to fit on one page. Flows must not contain more than 15 smart shapes per page, excluding routers, notify shapes, and connectors. • Redesign a flow if it has more than 15 smart shapes. You can: ◦ Create a subflow. ◦ Use parallel flows to perform additional functions.
8. Monitor performance regularly	• Evaluate and tune application performance at least weekly by using the Performance tool to check rule and activity efficiency. • Use PAL early to establish benchmarks. Compare these readings to later readings and correct the application as required.
9. Calculate and edit declaratively, not procedurally	• Whenever the value of a property is calculated or validated, use declarative rules wherever appropriate. • Create a Declare Expressions rule instead of using a Property-Set method in an activity. • Use a Declare Constraints rule instead of a Validation rule.
10. Keep security object-oriented, too	• Implement a security design that is rule-based and role-driven, based on who should have access to each type of work. • Never code security controls in an activity. • Use the standard access roles that are provided with PRPC or the Pega 7 Platform only as a starting point.

	• Use rulesets to segment related work for the purpose of introducing rule changes to the business, not as a security measure.

3. Coding Best Practices for Pega

The section provides some useful information and tips/general best practices regarding the guidance provided above. This section will be expanded as new content is found to be valuable and available but is meant to be informational only. This content is created and provided by Pega and can be located here: https://docs.pega.com/bundle/platform88/page/platform/app-dev/best-practices-for-pega-infinity-application-development.html

USER INTERFACE	
Best Practice	**Description**
Reuse and alter UI elements that already exist	Borrow, adapt, convert, and enhance from what is already available, rather than creating new UI elements. We want every part of our application to behave and look like all the other parts. When deciding on style, look at styles already in use in your application and adopt what you can; adapt what you must.
Avoid using icons for controls	Use link text or a button for clarity and ease of localization.
Always use dynamic layouts	Dynamic layouts separate content from presentation and make possible a highly flexible display of content across computers and mobile devices. Consider custom layouts and smart layouts as deprecated.
Do not create new legacy (non-autogenerated) records	Controls, sections, HTML rules, HTML fragments, text files, and so on. This includes specializing existing records, like breadcrumbs, to use in a class in your application.
Do not use inline styles at all	The Pega 7 Platform provides flexible tools for styling and branding that simplify creating and maintaining your application's look.
Do not use fixed widths	For table-style layouts that you must use, such as grids or tree grids, always set the width to 100%. This setting allows the display to adapt to the device on which the user is viewing it.
Use control validation to make sure entries and selections are valid.	Check that a required field has a value, that a text entry has more than the minimum number of characters and fewer than the maximum, and so on. Autogenerated controls have validation options: make use of them so that you do not have to do your validation in a post-submit activity.
Select the Use property default checkbox.	When possible, set the label value for a control from the short description for its property, rather than using inline labels, which increase maintenance costs.
Choose the right control for each list type:	**Simple lists**. The same distinction (do I need a header row?) applies when you choose between column repeat and dynamic column repeat layouts. If you need a header row, use a grid with a width of 100%. Do not set a fixed width in pixels. If you want columns and automatic filtering options or you want the data sourced directly from a report definition, use a grid. Otherwise, a repeating dynamic layout is preferable. **Nesting lists** If you need a table with multiple header cells, use a tree grid. But note that tree grids that display large amounts of data can have display and performance issues. For other nesting lists, use a tree, which is much simpler and can display huge amounts of data with no performance issues.

Using a section versus nesting a dynamic layout	For performance and maintenance reasons, minimize the number of sections that you create. **Use a section if:** • You plan to use it in multiple places. • Its contents need to be refreshed from the server more often than the content around it. • The data requires a table-based layout such as grid, tree grid, or column repeat, but you want to put it in same layout as the rest of the UI, perhaps to share a container format, style, or header. • Its content uses a different step page from the UI around it. Otherwise, nest a dynamic layout.
Only use a top-level clipboard page directly in your UI if:	It is a data page or system page, and you are displaying its data in read-only form. It is an editable data page that you are using to support a stand-alone user input form. All other references to top-level pages that are unrelated to the primary page (or undefined pages) should be considered technical debt. This is especially true in the UI regardless of whether the page is used as read-only or editable.
Do not overwrite secret UI extension points.	Do not overwrite Secret UI extension points at your class, which changes the default modal template for your class and all classes that inherit from it. Secret UI extension points include sections and activities such as pyModalTemplate, pyGridModalTemplate, pyPostGridUpdate, and pyOverlayTemplate. You can solve one use case by overwriting a Secret UI extension point but doing so can cause maintenance issues. In addition, your change affects every UI interaction of that type in your process. Changing the templates to fix a problem in a modal dialog box is like rewriting the browser because you do not like how a website is displayed. Instead, use built-in parameters to change the template or behavior for your particular use case.
Use set value to set flags	Use case: you have a property **.pyIsThingVisible**. Put the property in ahidden input field in the UI and use "set value 'pyIsThingVisible = true" (or false). This is lighter weight than using "Run data transform pzSetIsThingVisible" (and creating and maintaining the data transform)
Using Refresh When and Refresh Other Section	Use Refresh When whenever possible, especially for read-only references that need to stay in sync with data on the server, as it declares dependencies ("This section needs to be updated when .x changes"). When this is not possible (for example, you have editable references that get updated based on other input or based on untrackable input such as button clicks), use a Refresh Other Section targeted refresh.
Avoid Refresh Section	Refreshes interrupt the user experience. The user has to wait, the screen seems to hang until the section refreshes, and after the refresh is done, the user has lost focus and has to manually get back to the original screen. It is much better to use client-side interactions by using change tracker whenever possible. For example, rather than refresh a section with a pre-processing data transform, call a data transform or use set value and set your visibility conditionals to evaluate on client. **Do refresh a section when:** • Property values have been updated on the server and the new values need to be reflected in the UI. An action that causes a change to more than one property and occurs only on the client, such as deleting a row, needs to be submitted. • Parts of the UI need to be removed from the DOM because of other input. **Do not refresh a section when:** • You need to submit user input (use post value instead). You need to call a data transform or an activity after a user action. • You need to recalculate visibility, enabled/disabled state, or read-only state. Select the evaluate on client check box next to the expression field.
Judgment calls:	You have a complex expression to set visibility, enabled/disabled, or read-only state requiring a when condition. Consider using a declare expression to provide a value for a property and conditionalize on changes to that declare expression. You have to handle style changes based on value changes. You have to validate input and potentially show

	messages related to various properties. Note that validation provided by the controls, and a validate rule on the flow action, should cover most such situations.
Combine server actions	Every action in an action set that makes a call to the server (run a data transform, refresh a section, and so on) is a separate Ajax call that affects server resources.
DATA	
Use the tools at your disposal	There are two kinds of data in the application: user input and everything else. Any piece of information used in the process or on the primary page that the user did not give you is non-user input data.
Read-only data pages	These take a fixed set of scalar parameters and load any piece of complex data from them. **Obvious examples:** • Instance from a database based on keyed access. • Instance list from a parameterized report. • Connector to pull in information from a remote location. **Less obvious examples:** • Taking in a WSDL and parsing out important pieces. • Taking in XML or JSON data and creating a nested treeview list from it. • Taking in XML or JSON data and turning it into a matching clipboard page or putting it through an XML Parse. • Taking in a URL and parsing it into its significant components. • Taking in a generation request page and obtaining an evaluation of rules to be generated if the request is valid. • Creating a page of simulation data based on the parameters provided to simulate a connector. • Taking in the Applies To class and the type of the record that you want to create and returning a list of rulesets that are valid for that context. Basically, if you need a page or object loaded with data, use a data page. It will work for your use case.
Declare expressions	These define a relationship between the value of a single scalar property and a list of other scalar properties on the work page. When you specify an expression involving one or more properties on the top-level, parent, or primary page specified, the property always contains the value of that expression. You can set the value of a property on every page of a page list this way. Your expression can involve multiple when conditions and functions, giving you a powerful tool for populating property values. **Examples:** • The display version of a URL is always the normal URL without the query part. • The display version of a URL is always the normal URL without the query part. • The default name is always set from the service name without the namespace. • The branch ruleset is always + "_Branch_" + . • The authentication profile name for each method is equal to the authentication profile name selected for the workspace. • If @isAnIPAddress returns false, the data source name equals @getResourcePaths(.pySourceURL, 3); otherwise it equals the Integration name.
Auto-populated properties	These define a relationship between the value of a single page or page list property and a list of other scalar properties on the work page. You specify a data page to construct the page or page list and the properties to use as parameters to determine the data page's value. This is functionally the same as using a data page directly.
Maintainability	If you find yourself repeatedly referencing a data page and passing in the same values as parameters, make an auto populated property. That way, if later you change the data page or any of the properties, you will not have to hunt all over to find all the places to make the change.

Understandability	Non-user input data loaded from user input can still be vital to a user's understanding of how the process works. The data model should lay out exactly what the process needs, and that includes loaded data.
Keyed page lists	This feature is extremely powerful, and in the Pega 7 Platform, the only way to leverage it is through auto-populated properties. See This feature is extremely powerful, and in the Pega 7 Platform, the only way to leverage it is through auto-populated properties. See Instantly access embedded pages in a data page. There are three types of data transforms: Data transforms support complex data manipulation. If you move data around in your application, you can do it with a data transform (in some cases you may also need to make and use functions).
Maps	Maps define a translation between two different pieces of data. For example, when mapping the work page to a more compact generation request object you should use a data transform.
Models	These are used to initialize an object based on its state. For example, it is a good idea to have a Default and a data transform in between the steps in a flow. The first pair sets the object's initial state, and the others set the object to its initial state for each step. Setting optional relationships is a replacement for declare expressions when they cannot be used. Declare expressions cannot be used to set editable fields. You can have a field set with a declare expression or set with user input, but not both. So if you have a field that is always initially set to .A + " " + .B when .A or .B is set, but the user has the option to customize the field's value, you can't use a declare expression. You need to create a data transform and call it as necessary, because a declare expression cannot do it for you. For data management, activity usage should be primarily restricted to persistence. Creating, updating, and deleting data require using an activity. Otherwise, use activities sparingly in your data layer. While a few situations require using an activity, such as trying to source a page or page list when there is no other way to define the parameters as a set of scalars, in most other cases, you can use specific rules. There are also methods to pass some types of complex objects as parameters. See D_EvalResults for an example with data integration.
Do not duplicate properties if you can avoid it.	If multiple classes use a property, define the property once where it is accessible to all classes that need to use it. If one of two classes that need to use the property do not share a parent (for example, one is Embed- and the other is not) you can copy the property; but make sure to use the exact same name.
Avoid copying data	Data the process uses should exist in only one place. In the Pega 7 Platform, that means in a single property. If the data appears in many places, use the same property in each of those places. Do not copy the data from one property to another property. If user input has a cascading effect (setting a value in one place defaults a number of other values), then it is all right to copy data if the effect is one-directional. In other words, it is **only** all right to copy if you can define using something like a declare expression. If two properties have a circular dependency then something is wrong
GENERAL RULES	
Do not duplicate logic	If the same set of steps is used in two places, capture the steps in a single data transform or activity and call that record from both places where the steps are needed. If the same UI elements appear in two places, capture them in a single section, and reference the section in both places where the UI elements are required. If the same Java code or complex expression is used in two places, capture the code or expression in a function and include the function in both places where it is required.
Use inheritance	If the same rule applies to two different classes, move it to a shared parent class so both child classes can inherit it. If the same record applies to two classes, with tiny differences, move the shared logic to a record in a shared parent and add pyextension points. Overwrite the extension points as needed in the subclass and include the subtle differences there.
Do not reinvent the wheel	Leverage the work of others whenever you can. If an API already exists for your task, and has been tested and proven reliable, use it. Time spent researching what is already available

	to you when you are in unfamiliar territory is almost never time wasted. If you don't know, ask around and find someone who knows the territory to advise you. The Designer Studio has APIs for creating rulesets, creating ruleset versions, creating branches, updating records, and thousands and thousands of other actions. Integration has APIs, records, and utilities for creating HTTP requests, parsing XML and JSON, making SOAP requests, authenticating requests, parsing and validating URLs, and thousands of other tasks.
Do not assume a top-level page is there	The golden rule: if another developer with no knowledge of your use case would not be able to tell immediately where the data you are using came from, you should not be using it. **Exceptions to this rule are:** • Data pages • System pages • Parameter pages • Pages created within that rule, and then removed at the end • Properties on the primary page that are: • Autopopulated • Linked • Set from user input • Set from declare expressions
Items to Avoid	Using non-parameter pages not created within that rule Properties on editable data pages that are not loaded by the definition. **Gray areas:** • Manually populated properties on the primary page are not loaded within that rule. This is sometimes required, but it still creates confusion for other developers. • Top or Parent references. This introduces a dependency in your rule: the rule not only requires a page of the Applies To class as primary, but the primary page must also be embedded in a page with a specific class. **Use validate rules.** • Only do validation in a post-load activity if validate rules do not support the case. They usually do. • **Do not use obj-save except for testing.** Use "call pxUpdateRecord" or "Call Save" instead.

Current SOP Owner

Name	Role	Email
Amanda Ross	PEGA COE	amanda.ross3@va.gov

Revision History

Date	Changed By	Notes
08 Sep 2023	Amanda Ross	PEGA Intake SOP Version 1.0
30 Sep 2023	Jen Wisniewski	Converted PDF to Confluence