Pega Platform '24.2

3 September 2025

CONTENTS

DevOps

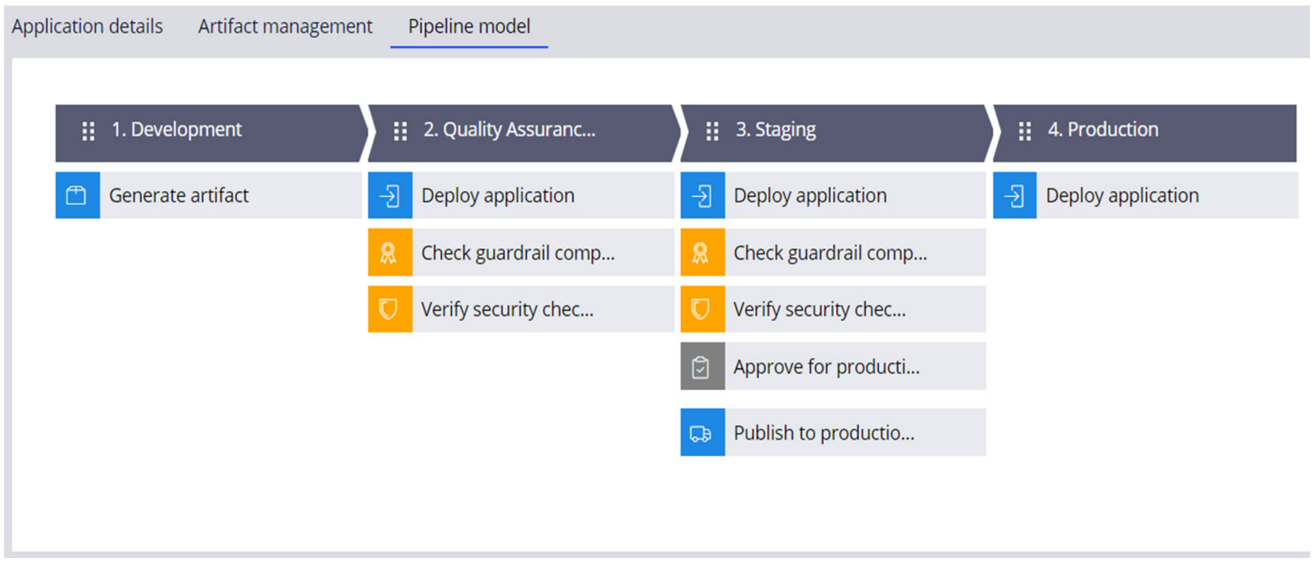
DevOps is a set of practices that bridge application development and operational behavior to reduce time to market without compromising on quality and operational effectiveness. It allows application developers and business owners to quickly respond to customer needs, develop a quicker feedback cycle, and ultimately achieve business value faster.

DevOps encourages a culture of collaboration between development, quality, and operations teams to reduce or eliminate barriers through fundamental practices such as continuous integration, continuous delivery, and continuous deployment. Adopting these practices and the tools to support them creates a standardized deployment process so that you can deploy predictable, high-quality releases.

Pega Platform™ provides the tools necessary to support continuous integration, delivery, and deployment through Deployment Manager, which provides a low-code, model-driven experience to configure and run continuous integration and delivery (CI/CD) workflows or deployment pipelines for your application. Deployment Manager provides out-of-the-box tools to enforce best CI/CD practices for your application. You can fully automate deployment pipelines, starting with automated integration of developer changes through branch merging and validation, application packaging, artifact repository management, deployments, test execution, guardrail compliance, and test coverage enforcement.

NOTE: Deployment Manager 5.5.6 or later are the stand-alone versions that support Pega Infinity™ '23 or later. Deployment Manager service supports Pega Infinity '23 or later.

Defining a CI/CD pipeline using Deployment Manager is quick and easy with customizable Stages and Tasks, or using default recommendations for a standard pipeline, as shown in the following pipeline model example:



Pipeline model

Pega Platform also includes support for open DevOps integration using popular third party tools such as Jenkins and Microsoft Azure DevOps by providing an open platform, with all the necessary hooks and services. With open DevOps integration, you can build a deployment pipeline using third-party tools to automate branch merging, application packaging and deployment, test execution, and quality metric enforcement.

For more information about configuring DevOps workflows, see the following topics:

- **Understanding DevOps concepts**
- **Applying DevOps concepts in Pega Platform**
- **Automating deployment pipelines using Open DevOps solutions**
- **Testing Pega applications**

Understanding DevOps concepts

The software industry evolves quickly and organizations are forced to rapidly respond to their changing markets. DevOps is a set of industry practices that have evolved to support this demand for rapid change. The primarily goal of DevOps is to reduce an organization's time-to-market for new features without compromising quality or end-user-experience. DevOps, like the Pega Platform, allows organizations to focus their efforts on adding customer value.

DevOps components

Adopting DevOps is an ongoing journey as opposed to a limited engagement project. There should be no point when you should stop seeking to improve efficiency. An evaluation of three main components is required when embarking on the DevOps journey:

- **People:** Before embarking on a DevOps journey, it's important to get buy-in from the entire organization. The cultural shift to embrace change and shared-responsibility is one of the first challenges which needs to be overcome before auditing process and adopting new technologies.
- **Organizational process:** Process is what makes each organization's software delivery process unique. Processes should be regularly reviewed for bottlenecks and Automation opportunities.
- **Technology:** Technology makes DevOps transformations possible and widely available. Technology enables Automation to many aspects of the software lifecycle from planning, to production, to monitoring.

- **Understanding development**
- **Understanding continuous integration**
- **Understanding continuous delivery**
- **Understanding deployment**

Understanding development

Pega Platform developers use Agile practices to create applications and commit the changes into branches in a shared development environment. Automated and manual testing provides rapid feedback to developers so that they can improve the application.

Follow these best practices to optimize the development process:

- Leverage multiple built-on applications to develop and process smaller component applications. Smaller applications move through the pipeline faster and are easier to develop, test, and maintain.
- Create one Pega Platform instance as a source environment that acts as a single source of truth for the application. This introduces stability into the developer environment and ensures that a problem in one developer environment does not affect other environments.
- Use Pega Platform developer tools, for example:
    - The rule compare feature allows you to see the differences between two versions of a specific rule.
    - The rule form search tool allows you to find a specific rule in your application.
- Follow branch-based development practices:
    - Developers can work on a shared development environment or local environments.
    - Content in branches migrates from the development environments to merge into the source environment.
    - Create an archive by exporting and storing backup versions of each branch in a separate location in the application repository. If a corrupted system state requires you to restore the source environment to a previous known good application version, the branches can be down-merged to reapply the changes in those branches that were lost as part of the restore.
    - Use unit tests to ensure quality.
- Ensure that the work on a Ruleset is reviewed and that the changes are validated. Lock every complete and validated Ruleset.

- Regularly synchronize the development environments with the source environment.

For more information, see the following articles:

- Application development
  - Searching for a Rule
  - Checking out a Rule
  - Checking in a Rule
  - Comparing Rule versions
  - Understanding best practices for version control in the DevOps pipeline
- Branching
  - Final Rules and utility Functions in branched development
  - Merging Branches into target Rulesets
  - Using the RuleSet Stack Landing Page Lock and Roll features for managing Ruleset versions
  - Adding a branch from a repository
  - Publishing a branch to a repository
- Testing
  - PegaUnit testing

Understanding continuous integration

With continuous integration, application developers frequently check in their changes to the source environment and use an automated build process to automatically verify these changes. Continuous integration identifies issues and pinpoints them early in the cycle. Use Jenkins with the prpcServiceUtils tool and the execute test service to automatically generate a potentially deployable application and export the application archive to a binary repository such as JFrog Artifactory.

During continuous integration, maintain the following best practices:

- To automatically generate a valid application, properly define the application Rule-Admin-Product rule and update the rule whenever the application changes. The prpcServiceUtils tool requires a predefined Rule-Admin-Product rule.
- To identify issues early, run unit tests and critical integration tests before packaging the application. If any one of these tests fails, stop the release pipeline until the issue is fixed.
- Publish the exported application archives into a repository such as JFrog Artifactory to maintain a version history of deployable applications.

For more information, see the following articles and help topics:

- PegaUnit tests
    - Running test Cases and suites with the Execute Tests service
- Application packaging
    - Packaging a release on your development environment
    - Using prpcServiceUtils and Jenkins for automated application deployment

Understanding continuous delivery

With continuous delivery, application changes run through rigorous automated regression testing and are deployed to a staging environment for further testing to ensure that there is a high confidence the application is ready to deploy on the production system.

Use Jenkins with the prpcServiceUtils tool to deploy the packaged application to test environments for regression testing or for other testing such as performance testing, compatibility testing, acceptance testing, and so on. At the end of the continuous delivery Stage, the application is declared ready to deploy to the production environment. Follow these best practices to ensure quality:

- Use Docker or a similar tool to create test environments for user acceptance tests (UAT) and exploratory tests.
- Create a wide variety of regression tests through the user interface and the service layer.

- Check the tests into a separate version control system such as Git.
- If a test fails, roll back the latest import.
- If all the tests pass, annotate the application package to indicate that it is ready to be deployed. Deployment can be done either automatically with Jenkins and JFrog Artifactory or manually.

Related concepts

- Deploying application changes to your staging or production environment
- Using prpcServiceUtils and Jenkins for automated application deployment
- Using restore points to enable error recovery

Understanding deployment

After an application change passes the testing requirements, use Jenkins and the prpcServiceUtils tools to migrate the changes into production after complete validation through automated testing on the staging system. Use application release guidelines to deploy with minimal downtime.

For more information, see the following articles and help topics:

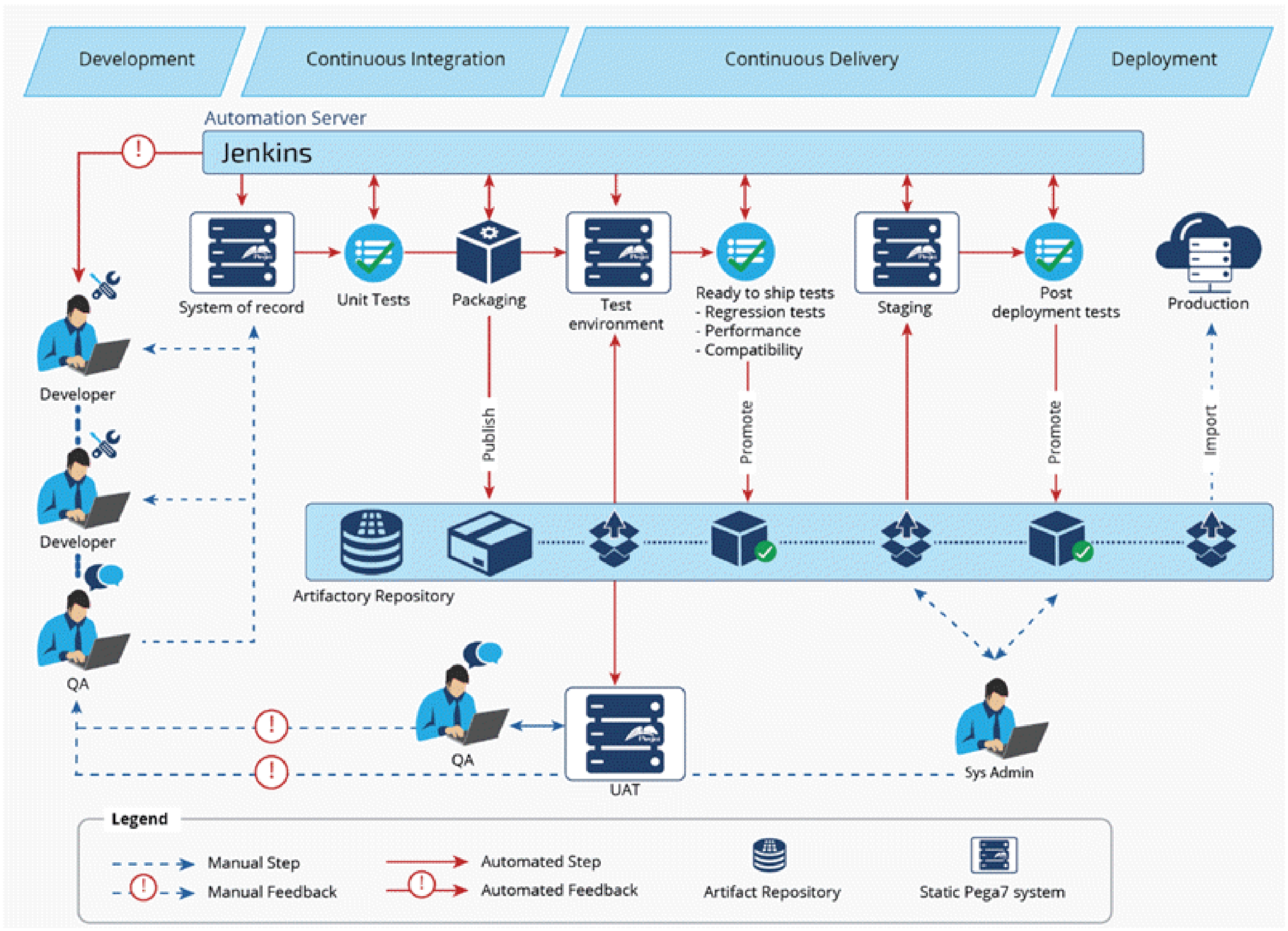
- Deploying to the production system
  - Understanding best practices for version control in the DevOps pipeline
  - Deploying application changes to your staging or production environment
  - Using prpcServiceUtils and Jenkins for automated application deployment
  - Migrating application changes
  - Understanding application release changes, types, and processes
- Enabling changes to the production system
  - Updating Access Groups by submitting a request to an active instance

Applying DevOps concepts in Pega Platform

Use DevOps practices such as continuous integration and continuous delivery to quickly move application changes from development through testing to deployment on your production system. Use Pega Platform tools and common third-party tools to implement DevOps.

The release pipeline in the following diagram illustrates the best practices for using Pega Platform for DevOps. At each Stage in the pipeline, a continuous loop presents the development team with feedback on testing results. This example includes the following assumptions:

- Pega Platform manages all schema changes.
- Jenkins is the Automation server that helps to coordinate the release pipeline, and JFrog Artifactory is the application repository; however, other equivalent tools could be used for both.



Open DevOps release pipeline overview

Developing Pega applications

Pega Platform developers use Agile practices to create applications and commit the changes into branches in a shared development environment. Automated and manual testing provides rapid feedback to developers so that they can improve the application.

Follow these best practices to optimize the development process:

- Leverage multiple built-on applications to develop and process smaller component applications. Smaller applications move through the pipeline faster and are easier to develop, test, and maintain.
- Create one Pega Platform instance as a source environment that acts as a single source of truth for the application. This introduces stability into the developer environment and ensures that a problem in one developer environment does not affect other environments.
- Use Pega Platform developer tools, for example:
    - The rule compare feature allows you to see the differences between two versions of a specific rule.
    - The rule form search tool allows you to find a specific rule in your application.

- Follow branch-based development practices:
    - Developers can work on a shared development environment or local environments.
    - Content in branches migrates from the development environments to merge into the source environment.
    - Create an archive by exporting and storing backup versions of each branch in a separate location in the application repository. If a corrupted system state requires you to restore the source environment to a previous known good application version, the branches can be down-merged to reapply the changes in those branches that were lost as part of the restore.
    - Use unit tests to ensure quality.
- Ensure that the work on a Ruleset is reviewed and that the changes are validated. Lock every complete and validated Ruleset.
- Regularly synchronize the development environments with the source environment.

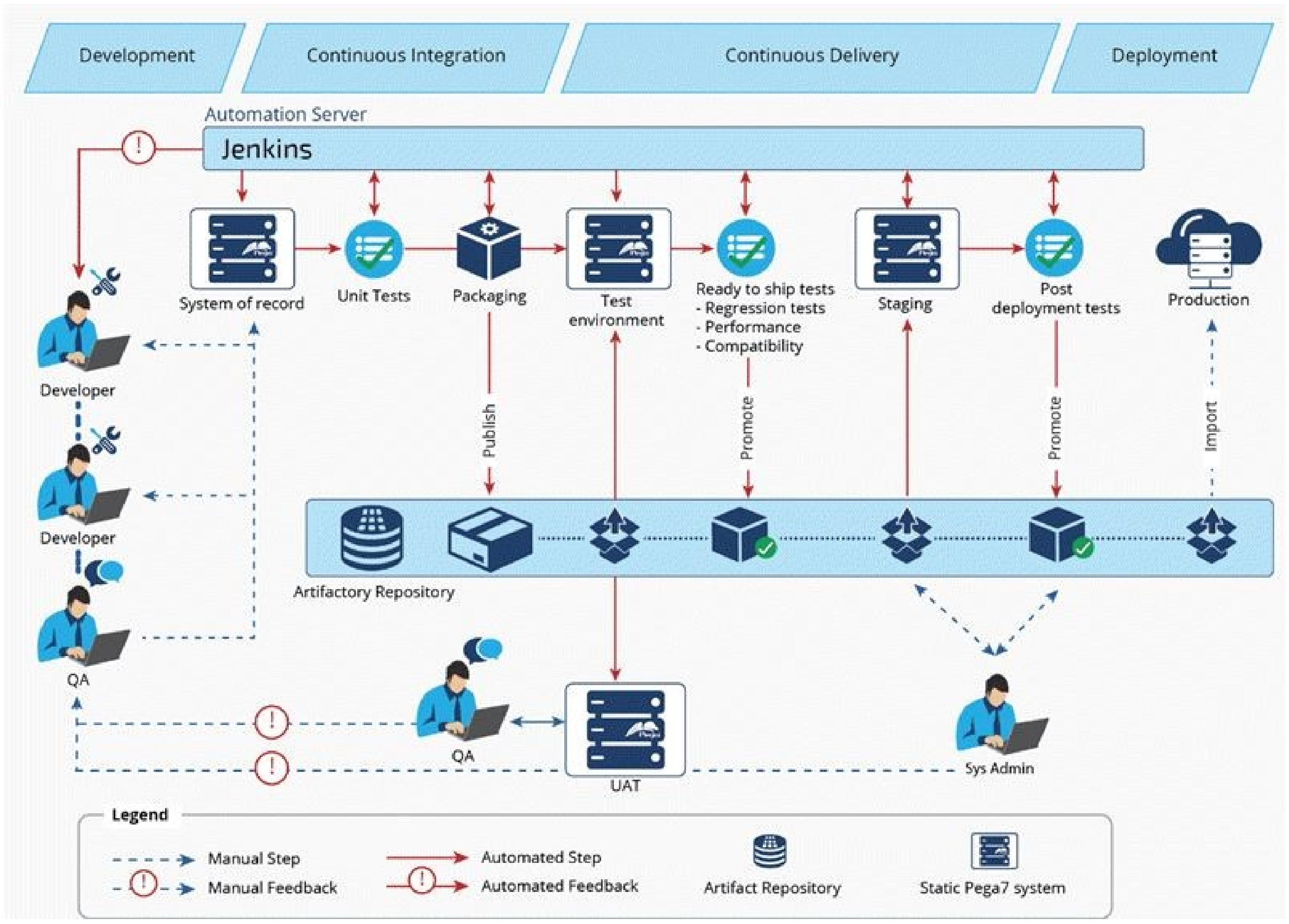
Understanding the DevOps release pipeline

Use DevOps practices such as continuous integration and continuous delivery to quickly move application changes from development through testing to deployment on your production system. Use Pega Platform tools and common third-party tools to implement DevOps.

The release pipeline in the following diagram illustrates the best practices for using Pega Platform for DevOps. At each Stage in the pipeline, a continuous loop presents the development team with feedback on testing results. This example includes the following assumptions:

- Pega Platform manages all schema changes.
- Jenkins is the Automation server that helps to coordinate the release pipeline, and JFrog Artifactory is the application repository; however, other equivalent tools could be used for both.

Open DevOps release pipeline overview



Development workflow

In a DevOps workflow, the most important best practice for application developers to adopt is continuous integration. Continuous integration is the process by which development changes to an application are integrated as frequently as possible, at least once a day and preferably multiple times a day, every time developers complete a meaningful unit of work.

To enforce best practices when developing an application and to ensure that application changes are of high quality, developers should use Pega Platform features such as branches. Before merging branches and integrating changes, developers should also verify that the application meets guardrail compliance and that unit tests pass. If the validation of development changes passes, the branch is merged into the application Ruleset.

However, if validation fails, then the merge is rejected, and developers should be notified so that they can address the failure and resubmit their changes. The feedback cycle of validating and integrating development changes should be as fast as possible, preferably 15 minutes or less, because it increases productivity in the following ways:

- Developers do not spend unnecessary time to see that their changes are valid, which enables them to make incremental changes.
- Incremental changes tend to be easier to validate, debug, and integrate.
- Other developers spend reduced time coordinating making changes and can be confident that they are building on validated functionality.

How you implement best practices for continuous integration depends on whether you have a smaller scale development with one or two scrum teams using a shared development environment or multiple distributed development teams. See the following topics for more information:

- **Understanding development best practices working in a shared environment**
- **Understanding development best practices in a distributed development environment with multiple teams**
- **Adding a branch from a repository**
- **Publishing a branch to a repository**
- **Understanding rule rebasing**

Understanding development best practices working in a shared environment

Development environments can be shared by one or more teams collaborating on the production application. To practice continuous integration, use a team Application Layer, branches, and release toggles.

- Build a team Application Layer that is built on top of the main production application. The team Application Layer contains branches, tests, and other development Rulesets that are not intended to go into production.
- Create a branch of your production Ruleset in the team application. For more information, see Adding development Branches to your application.
- Use release toggles to disable features that are not ready for general use. Using toggles allows you to merge branch content frequently even if some content is not final. For more information, see Toggling features on and off.
- Create formal review tasks for other members of the development team to review your content. For more information, see Creating a branch review.
- Use the branch developer tools to review the content and quality of your branch. For more information, see Reviewing Branches.
- Lock the branch. For more information, see Locking a Branch.
- Frequently merge the branch from the team Application Layer to the production Rulesets. For more information, see Merging Branches into target Rulesets.

NOTE: No more than two or three scrum teams should share a development environment to improve version control stability.

Understanding development best practices in a distributed development environment with multiple teams

If you have multiple teams working on the same application, each team should have a separate, remote development server on which developers work. A central Pega

Platform server acts as a source development system, which allows teams to integrate features into the application in a controlled manner and avoid unexpected conflicts between teams working in the same Rulesets.

Development systems

Follow these best practices on the remote development systems:

- Multiple teams can share development systems, which can depend upon geographical distribution of teams, system load, risk of teams making system-wide changes, and demand for system restarts.
- Build a team Application Layer that is built on top of the main production application. The team Application Layer contains branches, tests, and other development Rulesets that are not intended to go into production.
- Put all necessary configuration information for the development server in a development application that you can maintain, package, and deploy on demand so that you can quickly start up new remote development systems.
- Create a branch of your production Ruleset in the team application.
- Name the branch with the feature or bug ID from your project management tool so that you can associate changes with a corresponding feature or bug.
- Perform all development work in the branch in versioned Rules. Use branches for targeted collaboration and so that you can use development best practices such as conducting branch reviews and monitoring application quality metrics. You can also quickly test and roll back changes before you merge branches.
- Do not do branch development on the source development system to avoid having multiple versions of the same branch on the both the source development system and remote development system. The same branch might contain different contents that conflict with each other.
- Avoid developing Rules in unlocked Rulesets. Lock Rulesets to ensure that Rules are not accidentally and directly changed, because changes should be introduced only when branches are merged. Use a continuous integration server such as Deployment Manager to ensure that passwords to locked Rulesets are not shared publicly. For more information, see Configuring Ruleset version settings.

- Use release toggles to disable features that are not ready for general use. Using toggles allows you to merge branch content frequently even if some content is not final. For more information, see Toggling features on and off.
- Create formal review tasks for other members of the development team to review your content. For more information, see Creating a Branch review.
- Use the branch developer tools to review the content and quality of your branch. For more information, see Reviewing Branches.
- Lock the branch before you migrate it to the source development system. For more information, see Locking a Branch.
- Avoid deleting a branch before you migrate it into the main development system.
- Delete a branch after you import it into the main development system so that you do not import older data or rule instances and unintentionally merge them into the main application.
- Maintain a branch only as long as you need it. The longer that you keep a branch, the likelihood increases that the branch will have conflicts, which can be difficult to resolve.
- Be aware that you cannot make some rule updates in branches, such as updates to application records, classes, libraries, and schema. Senior application architects on the team should make these changes directly on the main development system.

Source development system

Follow these best practices on the source development system:

- Use an established and reliable backup and restore process.
- Maintain high availability on the source development system so that development teams are not affected by extended periods of downtime.
- Limit and restrict developer access to the main development system so that developers cannot make impromptu application changes without going through the DevOps workflow.

Adding a branch from a repository

If you are working in a continuous integration and delivery (CI/CD) pipeline, you can add a branch from a repository to your development application. You cannot add a branch that contains branched versions of a Ruleset that is not in your application stack.

1. In the navigation panel, click **App**, and then click **Branches**.
2. Right-click the application into which you want to import a branch and select **Add branch from repository**.
3. In the **Add branch from repository** dialog box, from the **Repository** list, select the repository that contains the branch that you want to import.
4. In the **Branch name** field, press the Down Arrow key and select the branch that you want to import.
5. Click **Import**.
6. Click **OK**.
7. If you are using multiple Branches, reorder the list of Branches so that it matches the order in which Rules should be resolved.
   For more information, see Reordering Branches.
8. Create Rules and add them to your Branch.
   When you create Rules, you can select the Branch and Ruleset into which you want to save them. Rulesets are automatically created. For more information, see Final Rules and utility Functions in branched development.

- Adding development Branches to your application

Publishing a branch to a repository

If you are using a continuous integration and delivery (CI/CD) pipeline with third-party tools such as Jenkins, you can publish a branch from your development application to a Pega repository on the main development system (remote system of record) to start a merge.

1. In the navigation panel, click **App**, and then click **Branches**.

2. Right-click the branch that you want to push to a repository and click **Publish to repository**.
3. In the **Push branch to repository** dialog box, select the repository from the **Repository** list.
4. Click **Publish**.

   Result:

   You receive a message if the repository is not configured properly or is down, and you cannot push the branch to the repository.
5. Click **Close**.

- Branches and Branch Rulesets
- Integrating with file and content management systems

Understanding rule rebasing

If you are using continuous integration in a CI/CD pipeline with either Deployment Manager or third-party Automation servers such as Jenkins, after you merge branches, you can rebase your development application to obtain the most recently committed Rulesets. Rebasing allows development teams working in separate development environments to share their changes and keep their local rule bases synchronized. Having the most recently committed Rules on your development system decreases the probability of conflicts with other development teams.

For example, you can publish a branch from your development application to a Pega repository on a source development system, which starts a job on Jenkins as your Automation server and merges branches. You can also use the Merge branches wizard to start a Deployment Manager build by first merging branches in a distributed or nondistributed, branch-based environment. After the merge is completed, you can rebase the Rulesets on your development application to obtain the merged Rulesets.

- **Configuring settings for rebasing**

- **Enabling the Pega repository type**
- **Enabling Ruleset versions for Pega repositories for rebasing**
- **Rebasing Rules to obtain latest versions**

Configuring settings for rebasing

Before you can rebase your development system, you must first configure a Pega repository and then enable Ruleset versions for them. You must also have the appropriate permissions for rebasing.

1. If you are using Pega repositories with third-party Automation servers such as Jenkins, enable the Pega repository type.
   You do not need to enable Pega repositories if you are using Deployment Manager. For more information, see Enabling the Pega repository type.
2. Create a connection to a Pega type repository that supports Ruleset version artifacts.
3. Enable Ruleset versions for Pega repositories by configuring the *HostedRulesetsList* dynamic system setting on the system of record. For more information, see Enabling Ruleset versions for Pega repositories for rebasing.
4. Ensure that you the *pxCanRebase* privilege so that you can rebase Rules. This privilege is associated with the sysadmin4 role.
   If you do not have this privilege, you can add it to your role. For more information, see Specifying privileges for an Access of Role to Object rule.

Enabling the Pega repository type

When you use continuous integration and delivery (CI/CD pipelines) third-party Automation servers, you use as a binary repository for rule artifacts during development. You also use Pega repositories when you are rebasing your development application when you are using third-party Automation servers or Deployment Manager.

If you are using Deployment Manager, the Pega repository type is already enabled; otherwise, you must first enable it for your application by completing the following steps:

1. Open the *pyCanRebase* rule:
    a. In the navigation pane of , click **Records**.
    b. Expand the **Decision** records, and then click **When**.
    c. Open the *pyCanRebase* rule that applies to *@baseclass*.
2. In the rule form header, click the Down arrow next to the **Save** button, and then click **Specialize by class or Ruleset**.
3. Choose a Ruleset in your application, then click **Create and open**.
4. On the **Conditions** tab, click **Actions** > **Edit** and change the Condition to `true`.
5. Click **Submit**.
6. Click **Save**.

What to do next:

If you are rebasing Rules to refresh your development system with the latest Rulesets that are hosted on a remote development system, enable Ruleset versions for Pega repositories. For more information, see Enabling Ruleset versions for Pega repositories for rebasing

- Creating a repository
- Integrating with file and content management systems

Enabling Ruleset versions for Pega repositories for rebasing

When you rebase Rules, you must enable Ruleset versions for Pega repositories so that they can host Ruleset versions. To enable Ruleset versions, configure the *HostedRulesetsList* dynamic system setting on the remote development system on which you are merging branches.

1. Complete one of the following tasks:

- Open the *HostedRulesetsList* dynamic system setting if it exists.
    - Click **Records** > **SysAdmin** > **Dynamic System Settings.**
    - Click the record with the HostedRulesetsList Setting Purpose and the Pega-ImportExport Owning Ruleset.
- Create this record if it does not exist.
    - Click Create > **SysAdmin** > **Dynamic System Settings.**
    - Enter a short description.
    - In the **Owning Ruleset** field, enter `Pega-ImportExport`.
    - In the **Setting Purpose** field, enter `HostedRulesetsList`.
    - Click **Create and open.**

2. On the **Settings** tab, in the **Value** field, enter a comma-separated list of the Rulesets on the remote development system. Enclose each Ruleset value within quotation marks, for example, “HRApp.”
3. Click **Save.**

- Enabling the Pega repository type
- Understanding rule rebasing

Rebasing Rules to obtain latest versions

If you are using a continuous integration and continuous delivery (CI/CD) pipeline with Deployment Manager or third-party Automation servers such as Jenkins, you can rebase your development application to obtain the most recently committed Rulesets through Pega repositories after you merge branches on the source development system.

Before you begin:

To rebase Rules, you must first merge branches to make changes to Rules. Changes made to Rules in an unlocked Ruleset version will not be visible by the rebase functionality.

NOTE: Only one rebase event at a time is supported per development system to prevent accidentally overriding a rebase event that is in progress.

NOTE: You can improve rebase performance by frequently incrementing the application patch version.

To rebase Rules, complete the following steps:

1. If you are migrating and merging branches separately, manually migrate branches for an application that has a new major or minor version. Rebase only pulls the Ruleset version that is visible to your current application.

   For example:

   For example, if you previously migrated a branch for an application of version 1.x but are now working on a 2.x application version, migrate the 2.x branch Ruleset to the main development system before rebasing. Otherwise, rebase refreshes your development system with the 1.x Ruleset versions.

2. In the header of Dev Studio click the name of your application, and then click **Definition**.
3. On the **Definition** tab, click **Get latest Ruleset versions**.
4. In the **Select repository** list, select the repository from which to retrieve Rules to see a list of Ruleset versions that will be rebased.
5. Click **Rebase**.
   - If there are no import conflicts, your development application is refreshed with the Rules.
   - If there are import conflicts, the system displays them. For example, a conflict can occur if you made a change to the same Ruleset version on your

local development system or if you modified a non-resolved rule in the Ruleset, such as the Application record. To resolve a conflict, complete the following step.

6. If you have conflicts, you must resolve them before rebasing continues, either by overwriting or rejecting the changes on your development system. Complete the following steps to import the Ruleset and either overwrite or reject the changes that you made to the Ruleset on the development system:
   a. For each Ruleset, click the **Download Archive** link and save the .zip file.
   b. Click the **Click here to launch the Import wizard** link at the top of the Rebase rule form to open the Import wizard, which you use to import the .zip files. For more information, see Importing Rules and data by using the Import wizard.
   c. In the wizard, specify whether to use the older version of the Ruleset or overwrite the older version with the newer version.
   d. After you resolve all conflicts, restart the rebase process by starting from step 1.

- Understanding rule rebasing
- Integrating with file and content management systems

Understanding best practices for version control in the DevOps pipeline

As a best practice, use semantic versioning when changing the version number, because it offers a logical set of Rules about when to increase each version number.

When you use semantic versioning, the part of the version number that is incremented communicates the significance of the change. Additional information about semantic versioning is available on the web.

The version number, in the format NN-NN-NN, defines the major version (first two digits), minor version (middle digits), and patch version (last digits), for example, 03-01-15.

- Major versions include significant features that might cause compatibility issues with earlier releases.
- Minor versions include enhancements or incremental updates.
- Patch versions include small changes such as bug fixes.

Rulesets include all versions of each rule. Skimming reduces the number of Rules by collecting the highest version of Rules in the Ruleset and copying them to a new major or minor version of that Ruleset, with patch version 01. For more information about skimming, see Rule skimming for higher Ruleset versions.

Best practices for development

Follow these best practices for version control in development:

- Work in branches.
- Consider creating a major version of your application if you update your application server or database server to a major new version.
- For small single scrum teams:
    - Increment both the patch and the minor version during every merge.
    - Developers merge into the next incremented patch version.
- For multiple scrum teams:
    - Assign a user the role of a release manager, who determines the application version and release strategy. This user should be familiar with concepts and features such as Rulesets, Ruleset versioning, application records, and application migration strategies.
    - The release manager selects a development Ruleset version number that includes a patch version number.
    - Developers merge into the highest available Ruleset version.
    - Frequently increment Ruleset versions to easily track updates to your application over time.
    - Maintain an application record that is capped at major and minor versions of its component Rulesets.

Best practices for deployment

Follow these best practices when you deploy your application to production:

- Define target Ruleset versions for production deployment.
- Use lock and roll to password-protect versions and roll changes to higher versions. For more information, see Ruleset Stack tab.
- Increment the Ruleset version every time you migrate your application to production, unless the application is likely to reach the patch version limit of 99.
- Create restore points before each deployment. For more information about restore points, see Using restore points to enable error recovery.

- **Using the RuleSet Stack Landing Page Lock and Roll features for managing Ruleset versions**

Using the RuleSet Stack Landing Page Lock and Roll features for managing Ruleset versions

The RuleSet Stack landing page simplifies following the best practices of Ruleset locking and application versioning. The landing page provides a summary of the locking status of the Ruleset versions in your application, as well as the Lock and Roll button.

Best practices for Ruleset versioning include:

- When possible, all versions in a Ruleset should remain locked. Use branch development instead of updating unlocked Rulesets directly.
- Cases where unlocked Rulesets may be required:
  - Reports
  - Delegated Rules
  - System Configurations stored in Rules
- If a Ruleset requires unlocked versions, then only the highest version of a Ruleset should be unlocked.
- Once locked, a Ruleset version should not later be unlocked.

1. In Dev Studio click**Configure** > **Application** > **Structure** > **RuleSet Stack**.

2. Observe the icons in the Current Application section.
    - If a column contains an unlocked warning icon, then a version lower than the highest is unlocked. You will be able to use the Lock and Roll Function on this lower Ruleset version to lock, but not to roll (increment). Click the unlock warning icon to view a list detailing the currently unlocked versions. Once the lower versions are locked, you can again use the roll portion of the Lock and Roll Function on the highest Ruleset version.
    - If the column contains a warning icon, then the Ruleset version does not exist, but is included in your application. Check your application to determine if the version of the Ruleset was incorrectly entered, or if the Ruleset was incorrectly entered. Once this error is corrected, then you can run the Lock and Roll Function on this Ruleset version, if needed.
3. Click the Lock and Roll button.
4. In the **Application Lock and Roll** window, in the Lock column, select the check box next to the Rulesets that you want to lock. If a Ruleset is currently locked, has a lower unlocked version, or does not exist, the corresponding icon will appear instead of a check box.
5. In each unlocked Ruleset that you selected for locking, a text box appears under the Password column. Enter a password for the Ruleset version.

    NOTE: The password that you enter appears in plain text to ensure that the password is entered correctly.

6. In the Roll column, check the box next to the Rulesets which are to be rolled to a higher version. If the Ruleset is not locked or selected to be locked, the check box for this Ruleset will not be available.
7. Observe the value in the Roll to Version column. By default, the value in this column will be the next highest patch version from the current Ruleset. If you are rolling a selected Ruleset is to a higher minor or major version, change this value to the appropriate version number.

8. Select the **Application update option** from the list below the Rulesets appropriate for the environment in which your development is taking place. For more information on these options, see Application Structure landing page.
9. Click **Run**.

   You might need to log out and back in to the application before the new version of any incremented Rulesets become visible.

Migrating application changes

With minimal disruption, you can safely migrate your application changes throughout the application development Lifecycle, from development to deployment on your staging and production environments. In the event of any issues, you can roll back the deployment and restore your system to a state that was previously known to be working.

The process that you use to release changes to your application is different depending on the types of changes that you are making. This topic describes the Standard Release process that you can use to deploy changes to Rules, data instances, and dynamic system settings. The Standard Release process is a self-service way to deploy changes without downtime. Other methods for releasing changes to your application are not covered in this article. For more information, see Understanding application release changes, types, and processes.

This Standard Release process applies to both on-premises and Pega Cloud Services environments. As a Pega Cloud Services customer, if you use this self-service process to release changes to your application, you are responsible for those changes. For more information, see Change Management process.

The Standard Release process includes the following steps and is scalable to the number and types of environments that you have:

1. Package the release on your shared development environment. For more information, see Packaging a release on your development environment.

2. Deploy the changes to your staging or production environment. For more information, see Deploying application changes to your staging or production environment.

- **Understanding application migration requirements**
- **Understanding application migration scenarios**

Understanding application migration requirements

Before you migrate your application, both your environment and application must meet certain requirements. For example, you must be able to download a RAP archive to a file system location with the required available space.

Your environments and applications must meet the following requirements:

- You have at least two existing Pega Platform environments. These environments can be any combination of sandbox and production environments, and can be on-premises or in the Pega Cloud virtual private cloud (VPC).
- You can log in to a machine from which you will complete the release process, such as your organization's laptop, workstation, or server.
- You have a location on a file system with enough available space to store the RAP archives. You must be able to download the RAP archive to this location and upload a RAP archive to another system from this location.
- You have complied with stated application guideline and guardrail requirements.
- Your application rule must specify Rulesets that have a patch version. Most application Rules have the Ruleset version in the stack set to 01-01, such as "Ruleset: 01-01". You must change this to specify your Rulesets to the patch level of their version, which is "Ruleset: 01-01-01". This is required for your top-level application rule and all built-on applications, except the base PegaRULES built-on application. This application structure gives you greater control over the release of your software, minimizes the impact of the release on application users, and provides for the smoothest recovery path in case of a troubled deployment.

Understanding application migration scenarios

The Standard Release migration process supports the following scenarios:

- All developers in the organization use a single shared development environment (recommended by Pegasystems).
- The organization follows a distributed development model, where individual developers or development teams have their own isolated development environments.

The release process works for either development scenario, because it begins after changes have been merged into the appropriate Ruleset versions. Regardless of development scenario or team size, development teams must use branching and merging for releasing applications. Otherwise, you cannot take full advantage of the tools and capabilities of the Pega Platform. For more information, see Understanding application release changes, types, and processes.

Deploying application changes to your staging or production environment

As part of the Standard Release process, after you set up and package a release on your shared development environment, you can deploy your application changes to your staging or production environment.

This Standard Release process applies to both on-premises and Pega Cloud Services environments. As a Pega Cloud Services customer, if you use this self-service process to release changes to your application, you are responsible for those changes. For more information, see Change Management process.

This process involves completing the following steps:

1. Deploying the application archives
2. Testing the deployment
3. Activating the release for all users

In the event of any issues, you can roll back the deployment and restore your system to a state that was previously known to be working.

Before you deploy application changes, you must know about the types of changes that you can make within a release, the release types, and the release management process to follow based on the changes you want to deploy. For example, SQL changes that remove columns from database tables or remove data types can interrupt service for users of the application. You must deploy these types of changes during scheduled downtime when users are offline. For more information, see Understanding application release changes, types, and processes.

- **Deploying the application archives**
- **Testing the deployment**
- **Activating the release for all users**
- **Rolling back a problematic deployment**

Deploying the application archives

After you create the application archives, deploy them to your target system. This process is the best way to deploy changes into your staging or production environment, control their activation, and recover from problematic deployments.

The user who imports the archives must have the zipMoveImport and SchemaImport privileges on the target system.

1. Ensure that you have connectivity to both the target system and to the location where the archives are stored.
2. Use the prpcServiceUtils command line utility to import the archives to the target system:
    - Deploy the Rules archive by following the steps in Importing Rules and data by using a direct connection to the database. Your changes are sent to the system, imported to the database, and ready for activation.

- Deploy the Data archive. When you deploy the Data archive, you use the same tool that you used to deploy the Rule archive but with different properties. You can roll back these changes if required.

Testing the deployment

After you deploy the changes, the release engineer and specified users can test the changes. For the staging environment, test the performance and the user interface, run automated tests, and do acceptance testing. For the production environment, perform validation tests.

Do the following steps:

1. On the target system, create a copy of the Access Group for your application. This step is a one-time process, because now this Access Group is available anytime you deploy changes.
2. Update the copied Access Group so that it references the new application version.
3. Find the operator ID record for a test user and give that operator ID record access to the Access Group that you just created.

Result:

You can now safely test your changes in the system at the same time as other users who are running on the previous version.

When you are satisfied that your release was deployed successfully, Release Engineering can activate the release for all users in the production environment.

If you experience any issues, see Rolling back a problematic deployment.

Activating the release for all users

After your Rules archive and Data archive are successfully deployed, changes are activated in various ways. Activation is the process by which a category of changes becomes usable by appropriate users of the system, if they have access.

Data changes, including schema changes, take effect immediately after being imported to the system. Your application might be able to access these fields immediately. After testing and when you are sufficiently comfortable, you should commit these changes.

To activate rule changes, you need to update the Access Groups that point to the prior version of your application rule:

1. In Dev Studio, click **Records**.
2. Click **Security** > **Access Group**.
3. Search for the Access Groups to be updated by specifying your application name in the search box and filtering the list.
4. After you locate the Access Group, open the record and increment the version number for your application to the new release version.
5. Click **Save**.

Result:

If you deploy code changes that need to be compiled, you must restart the system. Code changes cannot be made without downtime, and your System Administrator must perform a system restart. For information about the types of changes that you can make within a release, the release types, and the release management process to follow, see Understanding application release changes, types, and processes.

Rolling back a problematic deployment

In the event of a problematic deployment, the first goal is to prevent further issues from occurring. Then you can roll back the deployment and restore your system to a state that was previously known to be working.

Do the following steps:

1. Ensure that no new operators can access the problematic application. You can temporarily disable access to the entire system.
2. Roll back the problematic Rules changes. You can roll back changes by updating the Access Group for your application and specifying the previous version of your application.

Result:

Roll back the data instances that you changed to their previous version. To roll back data changes, use the prpcServiceUtils command line utility. This process replaces those modified data instances with their prior definition, rolling back your data changes to the last known, good state.

Packaging a release on your development environment

As part of the Standard Release process for migrating your application changes from development to production, you set up and package the release on your shared development environment.

This Standard Release process applies to both on-premises and Pega Cloud Services environments. As a Pega Cloud Services customer, if you use this self-service process to release changes to your application, you are responsible for those changes. For more information, see Change Management process.

This process involves completing the following steps:

1. Creating the release target (Ruleset version)
2. Locking the release
3. Creating the application archives

After you set up and package the release, you are ready to deploy the changes to your staging or production environment.

- **Creating the release target**
- **Locking the release**
- **Creating the application archives**

Creating the release target

When developers merge changes by using the Merge Wizard, they must select the Ruleset version to which to merge them. The release engineer is responsible for ensuring that each release has an unlocked Ruleset version that acts as the release target and into which these merges can be performed. Developers are responsible for merging their branches into the correct, unlocked Ruleset version and addressing any conflicts.

Locking the release

After all merges are completed, the release engineer locks the applications and Rulesets to be released. They are also responsible for creating the new, higher-level Ruleset versions and higher-level application Rules for the next release.

To lock the release, do the following steps:

1. In Dev Studio, click **Application** > **Structure** > **Ruleset Stack**.
2. Click **Lock & Roll**.

NOTE: As a best practice, lock your built-on applications first, and then work your way up the stack to your top-level application. This way, as

> each higher version application rule is created, you can open that rule and update the version of the built-on application.

3. For each Ruleset:
    a. Click **Lock** and provide a password.
    b. Click **Roll**.
4. Click **Create a new version of my Application**.
5. Click **Run**.

> **Result:**
>
> The application Rules and Ruleset versions for the current release are locked and require passwords to make changes. Also, you will have created the higher-level Ruleset versions and application Rules that will be used for the next release.

> **What to do next:**
>
> Configure higher environments: Staging, BOE, and Production.

Creating the application archives

For each release, you create one or two RAP archives, depending on the changes you made to your application. The user who exports the archives must have the zipMoveExport privilege.

These RAP archives include:

- The Rules RAP, which contains the Rules portion of your release, instances from Rules- types only, and all Rules changes.
- The Data RAP, which contains the Data portion of your release, instances from Data classes only, and all data changes.

Splitting the release into a Rules RAP and a Data RAP provides you with more control over the deployment and activation of your changes on other systems.

More RAP archives might be created during the development process. Import these RAP archives to a single system from which the Rules RAP and Data RAP will be created. This method provides the greatest level of control over the release by separating the release process from the development process.

1. Define the RAPs by using the Application Packaging wizard or by copying an existing RAP rule.
2. Export each RAP as an archive:
    a. Export the Rules. For more information, see Exporting an application, product Rule, or Ruleset to an archive or repository by using the Export wizard
    b. Provide a standard name for the archive, such as **Application-01-01-02.zip**.
    c. Store these archives in a standard location to which you have access, and that will be accessible during deployment.

Result:

A Rules archive and a Data archive are created as the result of this process.

Understanding application release changes, types, and processes

The following tables provide information about the types of changes that you can make within a release, the release types, and the release management process to follow based on the types of changes that you want to deploy.

Change type	Technical changes	Activates by Access Group	Activates immediately	Activation requires restart	Release frequency	Requires a Support Request (Pega Cloud Services only)
Rules (including non-rule-resolved Rules)	Rule- Rule-Application-Rule Rule-Obj-Class Rule-Ruleset-Name Rule-Ruleset-Version	Yes	Yes	No	Daily/ Weekly	No

Change type	Technical changes	Activates by Access Group	Activates immediately	Activation requires restart	Release frequency	Requires a Support Request (Pega Cloud Services only)
	Rule-Access-Role-Obj Rule-Access-Deny-Obj					
Data instances	Data-	No	Yes	No	Weekly	No
Dynamic system settings	Data-Admin-System-Settings	No	Yes[1]	No[2]	Monthly[3]	No
Functional	Rule-Utility-Function	Yes	Yes[4]	No[5]	Monthly	Yes

Change type	Technical changes	Activates by Access Group	Activates immediately	Activation requires restart	Release frequency	Requires a Support Request (Pega Cloud Services only)
	Rule-Utility-Library					
Data Model	SQL	No	Yes	No	Monthly	Yes
Code	Java JAR file Java .class file	No	No	Yes	Monthly	Yes
Environment	Changes outside of Pega	No	No	Yes	Quarterly	Yes

Change type	Technical changes	Activates by Access Group	Activates immediately	Activation requires restart	Release frequency	Requires a Support Request (Pega Cloud Services only)
	(JVM, XML configuration)					

Release type	Activates for users	Application users affected	Release frequency	New application version	Self-service	Requires a Support Request (Pega Cloud only)
Bug fix	Immediately	All	Daily	No	Yes	No
Standard release	On Access Group update	By Access Group	Weekly	Yes	Yes	No

Release type	Activates for users	Application users affected	Release frequency	New application version	Self-service	Requires a Support Request (Pega Cloud only)
Database release	Immediately	All	Monthly	Yes	No	Yes
Code release	After restart	All	Monthly	Yes	No	Yes
Environment release	After restart	All	Quarterly	Yes	No	Yes
Major release	Per change type[6]	Per change type[7]	Quarterly	Yes	No	Yes

Does your release contain the following changes?								Follow this release process	Requires a Support Request (Pega Cloud Services only)
Rules	Data instances	Dynamic system settings	Functional	Data Model	Code	Environment	Significant UX impact		
X	X	-	-	-	-	-	-	Bug fix	No
X	X	X	X	-	-	-	-	Standard release	No
-	X	-	-	X	-	-	-	Database release	Yes

Does your release contain the following changes?								Follow this release process	Requires a Support Request (Pega Cloud Services only)
-	-	-	-[8]	-	X	-	-	Code release	Yes
-	-	-[9]	-	-	-	X	-	Environment release	Yes
X	X	X	X	X	X	X	X	Major release	Yes

[1] Certain dynamic system settings activate only on system restart and require you to follow the Environment release process.

[2] Certain dynamic system settings activate only on system restart and require you to follow the Environment release process.

[3] Certain dynamic system settings activate only on system restart and require you to follow the Environment release process.

[4]

Treat functional changes that reference code as a Code release, which requires a system restart to activate if you are making code changes.

- **Change type** – This column lists the high-level category of changes that you can make in a release.
- **Technical changes** – Technical changes describe the Rule Types or artifacts for a change type. Rule- and Data- include all subtypes under that parent type, unless specifically identified for a different change type.
- **Activates by AccessGroup** – Rule resolution for this change type is controlled by the Access Groups of an operator.
- **Activates immediately** – Rule resolution uses this change type immediately after deployment.
- **Activation requires restart** – This change type requires a system restart before it is available to the rule resolution process.
- **Release frequency** – Release frequency indicates the period in which you can deploy this type of change to production.
- **Requires a Support Request (Pega Cloud only)** – As a Pega Cloud customer, you are responsible for any application changes that you make; however, as a best practice, inform and engage Pega Support before releasing application changes. You can open a Support Request on My Support Portal. For more information, see My Support Portal FAQ.

[5]

Treat functional changes that reference code as a Code release, which requires a system restart to activate if you are making code changes.

- **Change type** – This column lists the high-level category of changes that you can make in a release.
- **Technical changes** – Technical changes describe the Rule Types or artifacts for a change type. Rule- and Data- include all subtypes under that parent type, unless specifically identified for a different change type.
- **Activates by Access Group** – Rule resolution for this change type is controlled by the Access Groups of an operator.
- **Activates immediately** – Rule resolution uses this change type immediately after deployment.
- **Activation requires restart** – This change type requires a system restart before it is available to the rule resolution process.
- **Release frequency** – Release frequency indicates the period in which you can deploy this type of change to production.
- **Requires a Support Request (Pega Cloud only)** – As a Pega Cloud customer, you are responsible for any application changes that you make; however, as a best practice, inform and engage Pega Support before releasing application changes. You can open a Support Request on My Support Portal. For more information, see My Support Portal FAQ.

6

Activation of a Major release occurs based on the change types that the release contains. For information about how each change type is activated, see Table 1.

- **Release type** – This column lists the high-level category of releases that you can deploy.
- **Activates for users** – This column indicates when this release type takes effect for users.
- **Application users affected** – This column provides the scope of application users that see the effect of this release type.
- **Significant UX impact** – This release type might require users to significantly relearn a process or has significant layout changes.

- **Release frequency** – This column provides the frequency of this type of release.
- **New application version** – This column indicates whether you must create a new application version for this release.
- **Self-service** – A user with appropriate permissions can execute this release type using the Pega Platform, and a Pega System Administrator is not required to roll back changes.
- **Requires a Support Request (Pega Cloud Services only)** – As a Pega Cloud Services customer, you are responsible for any application changes that you make; however, as a best practice, inform and engage Pega Support before releasing application changes. You can open a Support Request on My Support Portal. For more information, see My Support Portal FAQ.

7

Activation of a Major release occurs based on the change types that the release contains. For information about how each change type is activated, see Table 1.

- **Release type** – This column lists the high-level category of releases that you can deploy.
- **Activates for users** – This column indicates when this release type takes effect for users.
- **Application users affected** – This column provides the scope of application users that see the effect of this release type.
- **Significant UX impact** – This release type might require users to significantly relearn a process or has significant layout changes.
- **Release frequency** – This column provides the frequency of this type of release.
- **New application version** – This column indicates whether you must create a new application version for this release.
- **Self-service** – A user with appropriate permissions can execute this release type using the Pega Platform, and a Pega System Administrator is not required to roll back changes.

- **Requires a Support Request (Pega Cloud Services only)** – As a Pega Cloud Services customer, you are responsible for any application changes that you make; however, as a best practice, inform and engage Pega Support before releasing application changes. You can open a Support Request on My Support Portal. For more information, see My Support Portal FAQ.

8

Treat functional changes that reference code as a Code release, which requires a system restart to activate if you are making code changes.

- **Rules** – Are you deploying Rules- records in this release?
- **Data instances** – Are you deploying Data- records in this release?
- **Dynamic System Settings** – Are you loading Data-Admin-System-Settings records in this release?
- **Significant UX impact** – Will users need to significantly relearn a process, or are there significant layout changes?
- **Code** – Are you loading JAR files as part of this release?
- **Data Model** – Are there changes to your Data Model in this release (SQL)?
- **Environment changes** – Will there be operating system or application server changes in this release?
- **Follow this release process** – Based on your answers to these questions, follow this release process.
- **Requires a Support Request (Pega Cloud Services only)** – As a Pega Cloud Services customer, you are responsible for any application changes that you make; however, as a best practice, inform and engage Pega Support before releasing application changes. You can open a Support Request on My Support Portal. For more information, see My Support Portal FAQ.

9

Certain Dynamic System Settings activate only on system restart and require you to follow the Environment release process.

Automating deployment pipelines using Open DevOps solutions

Use DevOps practices such as continuous integration and continuous delivery to quickly move application changes from development, through testing, and to deployment. Use Pega Platform tools and common third-party tools to implement DevOps.

You can set up a continuous integration and delivery (CI/CD) pipeline that uses a Pega repository in which you can store and test software and a third-party Automation server such as Jenkins that starts jobs and performs operations on your software. Use a CI/CD pipeline to quickly detect and resolve issues before deploying your application to a production environment.

For example, you can configure an Automation server with REST services to automatically merge branches after you publish them to a Pega repository. You can also configure Jenkins to create branch reviews, run PegaUnit tests, and return the status of a merge.

- **Downloading and installing prpcServiceUtils**
- **Using prpcServiceUtils and Jenkins for automated application deployment**
- **Running test Cases and suites with the Execute Tests service**

Downloading and installing prpcServiceUtils

Download and install prpcServiceUtils so that you can use it with Jenkins deploy applications.

To download and install prpcServiceUtils, do the following steps:

1. Download the **prpcServiceUtils.zip** file onto the Jenkins server. The package is available for download on the Marketplace here.

2. Extract the files onto any location to which Jenkins has access.

Using prpcServiceUtils and Jenkins for automated application deployment

You can use Jenkins to automate exporting and importing Pega Platform applications. Download the prpcServiceUtils command-line tool and configure Jenkins to export or import archives. You can use a single Jenkins build job to both export and import an application archive, or you can create separate jobs for each task.

For more information about prpcServiceUtils for service-enabled scripting, see Using service-enabled scripting and the prpcServiceUtils tool.

Ensure that your system includes the following items:

- Jenkins 1.651.1 or later
- Jenkins Plugins:
  - Ant Plugin
  - Environment Injector Plugin
  - Build with Parameters Plugin
- Ant version 1.9 or later
- JDK version 1.7 or later

- **Configuring the Jenkins build environment**
- **Configuring the Jenkins project**
- **Configuring Jenkins in 5.1.x**
- **Adding build steps to import or export the archive**
- **Importing or exporting the archive by running the Jenkins job**

Configuring the Jenkins build environment

Configure your build environment to call the prpcServiceUtils.bat or prpcServiceUtils.sh script and pass parameters to import or export the RAP.

To configure the Jenkins build environment, do the following steps:

1. Verify that the following Jenkins plugins are installed:
    - Ant Plugin
    - Environment Injector Plugin
    - Build with Parameters Plugin
2. Open a web browser and navigate to the Jenkins server.
3. Click **Manage Jenkins**.
4. Click **Configure System**.
5. Configure the PEGA_HOME environment variable:
    a. In the **Global properties** section, select **Environmental variables**.
    b. In the name field, enter `PEGA_HOME`.
    c. In the **value** field, enter the location where you extracted the prpcServiceUtils.zip file.
    d. Click **Add**.
6. Click **Apply**, and then click **Save**.

Configuring the Jenkins project

Configure your build environment to call the prpcServiceUtils.bat or prpcServiceUtils.sh script and pass parameters to import or export the RAP:

1. Complete one of the following Actions:
    - Create a project if you have not already done so.
    - Open an existing project.
2. Click **Configure**.
3. Select **This build is parameterized**.

4. Click **Add Parameter** and create the parameters that Jenkins passes to the prpcServiceUtils tool:
    - To import or export a RAP, create the following parameters:

Parameter name	Type	Default value
productName	String	The name of the RAP rule used to generate the archive
productVersion	String	The version number of the RAP rule

    - To import or export an application, create the following parameters:

Parameter name	Type	Default value
applicationName	String	The name of the application
applicationVersion	String	The version number of the application.

5. Select **Prepare an environment for the run.**
6. In the **Properties Content** section, set the following properties:
    - `SystemName=$BUILD_TAG`
    - ` Source Code Management=None`
    - `Inject environment variables to the build process=Properties Content`
7. In the **Properties Content** section, set the *ImportExistingInstances* property to one of the following values. The default is unset:
    - override

      For Rules - If a rule with the same key exists in the system, but the rule resolution properties differ (for example, Ruleset or version), replace the existing rule with the imported rule.

For work - If a work object with the same key exists but belongs to a different application (for example, it has a different class hierarchy but same classgroup name and same ID prefix), replace the existing work object with the imported work object.

- skip

  For Rules - If a rule with the same key exists in the system, and the rule resolution properties differ, do not replace the existing rule.

  For work - If a work object with the same key exists but belongs to a different application, do not replace the existing work object.

- unset: The import will fail if keys already exist either for rule instances that have different rule resolution properties or for work objects that belong to a different applications that use the same classgroup name.

8. Set the artifact directory where exported logs and files are downloaded In the following format: `ARTIFACTS_DIR=<var>path to artifact directory</var>`
   The default is the logs directory.

   NOTE: You can also set the directory later by specifying `-artifactsDir` when you run the batch file.

9. In the **Properties Content** section, enter the static properties in the format `ParameterName=Value`.
   - Source properties for export:

Parameter name	Default value
SourceHost	The host name and port number of the Pega Platform server from which to export the archive file.

Parameter name	Default value
SourceUser	The operator name. This operator must have export privileges.
SourcePassword	The operator password.

- Target properties for import:

Parameter name	Default value
TargetHost	The host name and port number of the Pega Platform server that contains the archive file to import.
TargetUser	The operator name. This operator must have import privileges.
TargetPassword	The operator password.

10. Click **Save**.

Configuring Jenkins in 5.1.x

If you are using a Run Jenkins step task in your pipeline, configure Jenkins so that it can communicate with the orchestration server.

1. On the orchestration server, create an authentication profile that uses Jenkins credentials.
    - If you are using a version of Jenkins earlier than 2.17.6, create an authentication profile on the orchestration server that specifies the credentials to use.
        - Click **Create** > **Security** > **Authentication Profile**.
        - Enter a name, and then click **Create and open**.
        - In the **User name** field, enter Jenkins user ID.
        - Click **Set password**, enter the Jenkins password, and then click **Submit**.
        - Click the **Preemptive authentication** check box.
        - Click **Save**.

        - Go to step 4.

          For more information about configuring Authentication Profiles, see Creating an authentication profile.

    - If you are using Jenkins 2.17.6 or later and want to use an API token for authentication, go to step 2.
    - If you are using Jenkins 2.17.6 or later and want to use a Crumb Issuer for authentication, go to step 3.
2. If you are using Jenkins version 2.17.6 or later and want to use an API token for authentication, do the following steps:
    a. Log in to the Jenkins server.
    b. Click **People**, click the user who is running the Jenkins job, and then click **Configure** > **API token**.
    c. Generate the API token.
    d. Create an authentication profile on the orchestration server by clicking **Create** > **Security** > **Authentication Profile**.
    e. In the **User name** field, enter the Jenkins user ID.
    f. Click **Set password**, enter the API token that you generated, and then click **Submit**.
    g. Click the **Preemptive authentication** check box.
    h. Click **Save**.
    i. Go to step 4.

    For more information about configuring Authentication Profiles, see Creating an authentication profile.

3. If you are using Jenkins version 2.17.6 or later and want to use a Crumb Issuer for authentication, do the following steps:
    a. Log in to the Jenkins server.
    b. Click **Manage Jenkins** > **Manage Plugins** and select the check box for the **Strict Crumb Issuer** plug-in.
    c. Click **Manage Jenkins** > **Configure Global Security**.

   d. In the **CSRF protection** section, in the **Crumb Issuer** list, select **Strict Crumb Issuer**.
   e. Click **Advanced**, and then clear the **Check the session ID** check box.
   f. Click **Save**.
   g. Create an authentication profile on the orchestration server by clicking **Create** > **Security** > **Authentication Profile**.
   h. In the **User name** field, enter the Jenkins user ID.
   i. Click **Set password**, enter the Jenkins password, and then click **Submit**.
   j. Click the **Preemptive authentication** check box.
   k. Click **Save**.
   l. Go to step 4.

      For more information about configuring Authentication Profiles, see Creating an authentication profile.

   The following steps describe one of the ways to configure Jenkins jobs for the integration with Deployment Manager to work (using Jenkins Free Style Project as the option). This can be done in multiple different ways (e.g. with workflow plugin and groovy, or scripts in Ant, Gradle, etc.).

4. Install the Post build task plug-in.
5. Install the curl and jq commands on the Jenkins server. As mentioned above, these can be replaced with any other scripts as well.
6. Create a new freestyle project.
7. On the **General** tab, select the **This project is parameterized** check box.
8. Add the DeploymentID and CallBackURL parameters.
   a. Click **Add parameter**, and then select **String parameter**.
   b. In the **String** field, enter `DeploymentID`.
   c. Click **Add parameter**, and then select **String parameter**.
   d. In the **String** field, enter `CallBackURL`.
9. To add parameters that you can use in Run Jenkins step tasks in the pipeline, click **Add parameter**, select **String parameter**, and enter the string of the parameter.

The system automatically populates these values in Jenkins tasks. You can add any of the following strings:

a. If you are configuring Jenkins tasks in a merge pipeline, add any of the following strings:
    - DeploymentID: Deployment ID of the pipeline on which the task is triggered.
    - DeploymentNumber: Sequence number of the deployment.
    - TaskID: TaskID of the Jenkins task.
    - CallBackURL: URL to post the status of task.
    - OrchestratorURL: URL on which the Jenkins task is configured.
    - PipelineName: Pipeline name on which the Jenkins task is configured.
    - PipelineID: ID of the pipeline on which the Jenkins task is configured.
    - ApplicationName: Application name for which the pipeline is configured.
    - ApplicationVersion: Application version for which the pipeline is configured.
    - RepositoryName: Repository to publish the merged branch.
    - BranchName: Name of the branch for which the merge Jenkins task is configured.
    - BranchFilePath: File path to the branch artifact.
b. If you are configuring Jenkins tasks in a deployment pipeline, add any of the following strings:
    - DeploymentID: Deployment ID of the pipeline on which the task is triggered.
    - DeploymentNumber: Sequence number of the deployment.
    - TaskID: TaskID of the Jenkins task.
    - CallBackURL: URL to post the status of task.
    - OrchestratorURL: URL on which the Jenkins task is configured.
    - PipelineName: Pipeline name on which the Jenkins task is configured.
    - PipelineID: ID of the pipeline on which the Jenkins task is configured.
    - RepositoryName: Repository to publish the merged branch.

- **DeploymentArtifactName**: Artifact name that the Deploy task uses on the Stage on which the Jenkins task is configured.
- **ArtifactPath**: Full path to the artifact that the Deploy task uses.
- **CurrentStage**: Name of the Stage on which the Jenkins task is configured.
- **CurrentStageURL**: URL of the system on which the Jenkins task is configured.

10. In the **Build Triggers** section, select the **Trigger builds** remotely check box.
11. In the **Authentication Token** field, select the token that you want to use when you start Jenkins jobs remotely.
12. In the **Build Environment** section, select the **Use Secret text(s) or file(s)** check box.
13. In the **Bindings** section, do the following actions:
    a. Click **Add**, and then select **User name and password (separated)**.
    b. In the **Username Variable** field, enter `client_id`.
    c. In the **Password Variable** field, enter `client_secret`.
    d. In the **Credentials** field, click **Specific credentials**.
    e. Click **Add**, and then select **Jenkins**.
    f. In the **Add credentials** dialog box, in the **Username** field, enter the client id configured in the oAuth client credentials on Deployment Manager.
    g. In the **Password** field, enter the client secret.
    h. Optionally enter a description, and click **Save**.
14. Add post-build tasks by doing one of the following actions:
    a. If Jenkins is running on Microsoft Windows, go to step 15.
    b. If Jenkins is running on Linux, go to step 16.
15. If Jenkins is running on Microsoft Windows, add the following post-build tasks:
    a. Click **Add post-build** action, and then select **Post build task**.
    b. In the **Post-Build Actions** section, in the **Log text** field, enter a unique string for the message that is displayed in the build console output when a build fails, for example `BUILD FAILURE`.
    c. In the **Script** field, enter:

```
@echo off
for /F %%I in ('curl --insecure -d "client_id=%%client_id%%&client_secret=%
%client_secret%%&grant_type=client_credentials" https://10.224.203.11:8
443/prweb/PRRestService/oauth2/v1/token ^| jq -r .access_token') do set t
oken=%%~I
curl --insecure -H "Content-Type: application/json" -H "Accept: application/j
son" -H "Authorization:Bearer %token%" -k -X PUT --data "{\"taskStatus\":\"
Resolved-Rejected\",\"errors\":[{\"errorMessage\":\"Jenkins job failed. Chec
k here for logs: %BUILD_URL%\"}],\"taskInfo\":{\"outputParameters\":[{\"n
ame\": \"BuildNumber\",\"type\": \"Text\",\"value\": \"%BUILD_NUMBER%\"
},{\"name\": \"JenkinsBuildURL\",\"type\": \"Text\",\"value\": \"%BUILD_URL
%\"}]}}" %CallBackURL%.
```

d. Click **Add another task**.
e. In the **Post-Build Actions** section, in the **Log text** field, enter a unique string for the message that is displayed in the build console output when a build is successful, for example `BUILD SUCCESS`.
f. In the **Script** field, enter:

```
 @echo off
for /F %%I in ('curl --insecure -d "client_id=%%client_id%%&client_secret=%
%client_secret%%&grant_type=client_credentials" https://10.224.203.11:8
443/prweb/PRRestService/oauth2/v1/token ^| jq -r .access_token') do set t
oken=%%~I
curl --insecure -H "Content-Type: application/json" -H "Accept: application/j
son" -H "Authorization:Bearer %token%" -k -X PUT --data "{\"taskStatus\":\"
Resolved-Completed\",\"taskInfo\":{\"outputParameters\":[{\"name\": \"Bu
ildNumber\",\"type\": \"Text\",\"value\": \"%BUILD_NUMBER%\"},{\"name\"
: \"JenkinsBuildURL\",\"type\": \"Text\",\"value\": \"%BUILD_URL%\"}]}}" %C
allBackURL%
```

g. Click **Save**.
h. Go to step 17.

16. If Jenkins is running on Linux, add the following post-build tasks. Use the dollar sign ($) instead of the percent sign (%) to access the environment variables:
    a. Click **Add post-build action**, and then select **Post build task**.
    b. In the **Log text** field, enter a unique string that for the message that is displayed in the build console output when a build fails, for example `BUILD FAILURE`.
    c. In the **Script** field, enter:

    ```
    export token=`curl --insecure -d "client_id=${client_id}&client_secret=${cli
    ent_secret}&grant_type=client_credentials" https://10.224.203.11:8443/pr
    web/PRRestService/oauth2/v1/token | jq -r .access_token`
    curl --insecure -H "Content-Type: application/json" -H "Accept: application/j
    son" -H "Authorization:Bearer ${token}" -k -X PUT --data "{\"taskStatus\":\"
    Resolved-Rejected\",\"errors\":[{\"errorMessage\":\"Jenkins job failed. Chec
    k here for logs: ${BUILD_URL}\"}],\"taskInfo\":{\"outputParameters\":[{\"na
    me\": \"BuildNumber\",\"type\": \"Text\",\"value\": \"${BUILD_NUMBER}\"},
    {\"name\": \"JenkinsBuildURL\",\"type\": \"Text\",\"value\": \"${BUILD_URL}\
    "}]}}" $CallBackURL
    ```

    d. Click **Add another task**.
    e. In the **Log text** field, enter a unique string that for the message that is displayed in the build console output when a build is successful, for example BUILD SUCCESS.
    f. In the **Script** field, enter:

    ```
    export token=`curl --insecure -d "client_id=${client_id}&client_secret=${cli
    ent_secret}&grant_type=client_credentials" https://10.224.203.11:8443/pr
    web/PRRestService/oauth2/v1/token | jq -r .access_token`
    curl --insecure -H "Content-Type: application/json" -H "Accept: application/j
    son" -H "Authorization:Bearer ${token}" -k -X PUT --data "{\"taskStatus\":\"
    Resolved-Completed\",\"taskInfo\":{\"outputParameters\":[{\"name\": \"Bu
    ildNumber\",\"type\": \"Text\",\"value\": \"${BUILD_NUMBER}\"},{\"name\":
    ```

```
\"JenkinsBuildURL\",\"type\": \"Text\",\"value\": \"${BUILD_URL}\"}]}}" $Call
BackURL
```

g. Click **Save**.
h. Go to step 17.

17. To stop a pipeline deployment if a Jenkins build fails, add a post-build script:
    a. Click **Add post-build action**, and then select **Post build task**.
    b. In the **Log text** field, enter a unique string for the message that is displayed in the build console output when a build fails, for example `JENKINS BUILD FAILURE`.
    c. In the **Script** field, enter:

    ```
     (for Windows) for /F %%I in ('curl --insecure -d "client_id=%%client_id%%&
    client_secret=%%client_secret%%&grant_type=client_credentials" https://1
    0.224.203.11:8443/prweb/PRRestService/oauth2/v1/token ^| jq -r .access_
    token') do set token=%%~I
    curl --insecure -H "Content-Type: application/json" -H "Accept: application/j
    son" -H "Authorization:Bearer %token%" --data "{\"reasonForAbort\":\"Jen
    kins task failed\"}" -k -X PUT %OrchestratorURL%/PRRestService/Deploym
    entManager/v1/deployments/%DeploymentID%/abort
    ```

    d. In the **Script** field, enter:

    ```
    (for Linux) export token=`curl --insecure -d "client_id=${client_id}&client_s
    ecret=${client_secret}&grant_type=client_credentials" https://10.224.203.1
    1:8443/prweb/PRRestService/oauth2/v1/token | jq -r .access_token`
    curl --insecure -H "Content-Type: application/json" -H "Accept: application/j
    son" -H "Authorization:Bearer ${token}" --data "{\"reasonForAbort\":\"Jenki
    ns task failed\"}" -k -X PUT ${OrchestratorURL}/PRRestService/Deployment
    Manager/v1/deployments/${DeploymentID}/abort
    ```

    e. Click **Save**.

Adding build steps to import or export the archive

You can enter build steps to import an archive or export an archive, or you can do both in one job.

- **Adding export build steps**
- **Adding import build steps**

Related tasks

- Adding export build steps
- Adding import build steps

Adding export build steps

To add export steps to your build job, do the following steps:

1. Add an Invoke Ant build step:
   a. In the **Build** Section, click **Add build step** and select **Invoke Ant**.
   b. n the **Targets** field, enter `exportprops`.
   c. In the **Build File** field, enter the path to the build file:
   - On Windows, enter the following path: `$PEGA_HOME\samples\Jenkins-build.xml`
   - On UNIX, enter the following path: `$PEGA_HOME/scripts/samples/jenkins/Jenkins-build.xml`
2. Add a build step to run either prpcServiceUtils.bat or prpcServiceUtils.sh.
   - If you are on Windows, go to step 3.
   - If you are on UNIX, go to step 4.
3. On Windows, create an Execute Windows batch command build step:
   a. In the **Build** Section, click **Add build step** and select **Execute Windows batch command**.
   b. In the **Command** field, enter the following command: `%PEGA_HOME%\scripts\utils\prpcServiceUtils.bat export --connPropFile`

`%WORKSPACE%\%SystemName%_export.properties --artifactsDir %WORKSPACE%`

4. On UNIX, create an Execute Shell batch command build step:
   a. In the **Build** Section, click **Add build step** and select **Execute Shell batch command**.
   b. In the **Command** field, enter the following command: `$PEGA_HOME/scripts/utils/prpcServiceUtils.sh export --connPropFile $WORKSPACE/${SystemName}_export.properties --artifactsDir $WORKSPACE`

Related concepts

- Adding build steps to import or export the archive

Related tasks

- Adding import build steps

Adding import build steps

To add import build steps to your build job, do the following steps:

1. Add an Invoke Ant build step:
   a. In the **Build** Section, click **Add build step** and select **Invoke Ant**.
   b. n the **Targets** field, enter `importprops`.
   c. In the **Build File** field, enter the path to the build file:
   - On Windows, enter the following path: `$PEGA_HOME\samples\Jenkins-build.xml`
   - On UNIX, enter the following path: `$PEGA_HOME/scripts/samples/jenkins/Jenkins-build.xml`
2. Add a build step to run either prpcServiceUtils.bat or prpcServiceUtils.sh.
   - If you are on Windows, go to step 3.
   - If you are on UNIX, go to step 4.

3. On Windows, create an Execute Windows batch command build step:
    a. In the **Build** Section, click **Add build step** and select **Execute Windows batch command**.
    b. In the **Command** field, enter the following command:

    ```
    %PEGA_HOME%\scripts\utils\prpcServiceUtils.bat import --
    connPropFile %WORKSPACE%\%SystemName%_import.properties --
    artifactsDir %WORKSPACE%
    ```
4. On UNIX, create an Execute Shell batch command build step:
    a. In the **Build** Section, click **Add build step** and select **Execute Shell batch command**.
    b. In the **Command** field, enter the following command:

    ```
    $PEGA_HOME/scripts/utils/prpcServiceUtils.sh import --
    connPropFile $WORKSPACE/${SystemName}_import.properties --
    artifactsDir $WORKSPACE
    ```

Related concepts

- Adding build steps to import or export the archive

Related tasks

- Adding export build steps

Importing or exporting the archive by running the Jenkins job

Run a Jenkins job to import or export the application archive.

Do the following steps:

1. In Jenkins, click **Build with Parameters**.
2. When the parameter values are displayed, verify the default settings and edit any values.

3. Set the artifact directory where exported logs and files are downloaded in the following format: `-artifactsDir=<var>path to artifact directory</var>`
   The default directory is the logs directory.
4. Click **Build**.

Running test Cases and suites with the Execute Tests service

You can use the Execute Tests service (REST API) to validate the quality of your code after every build is created by running unit test Cases that are configured for the application.

A continuous integration (CI) tool, such as Jenkins, calls the service, which runs all the unit test Ccases or test suites in your application and returns the results in xUnit format. The continuous integration tool interprets the results and, if the tests are not successful, you can correct errors before you deploy your application.

When you use Jenkins, you can also use the Execute Tests service to run unit tests after you merge a branch on a remote system of record and start a job.

The service comprises the following information:

- Service name: Pega unit Rule-Test-Unit-Case pzExecuteTests
- Service package: Pega unit
- End point: http://<yourapplicationURL>/prweb/PRRestService/Pega unit/Rule-Test-Unit-Case/pzExecuteTests

You can quarantine a test Case by marking it **Disabled**. A disabled test Case is not run by the Execute Tests service. Test Case quarantines prevent noncritical tests from running if they are causing failures so that the service can continue to run.

- **Request parameters**
- **Response**

- **Configuring your default Access Group**
- **Configuring your build environment**
- **Running tests and verifying results**
- **Test failures**

Request parameters

The Execute Tests service takes certain string request parameters.

The strings are:

- ApplicationInformation – Optional. The name and version of the application for which you want to run Pega unit test Cases. You can pass it instead of the AccessGroup parameter.
    - If you pass only this parameter, the service runs all the test Cases in the application.
    - If you do not pass this parameter, the service runs all the test Cases in the application that are associated with the default Access Group that is configured for your operator.

Use the format ApplicationInformation=<application_name:application_version>.

- AccessGroup – Optional. The Access Group that is associated with the application for which you want to run Pega unit test Cases. You can pass it instead of the ApplicationInformation parameter.
    - If you pass this parameter, the service runs all the test Cases in the application that are associated with this Access Group.
    - If you do not pass this parameter, the service runs all the test Cases in the application that are associated with the default Access Group that is configured for your operator.

Use the format AccessGroup=<access_group_name:access_group_user>.

- TestSuiteID – The *pxInsName* of the test suite that you want to run. You can find this value in the XML document that comprises the test suite by clicking Actions > XML on the Edit Test Suite form. You can run one test suite at a time. When you use this parameter, all the test Cases in the test suite are run, but no other test Cases in your application are run. This parameter is required for Pega unit test suites. If test suites share the same name among applications:
    - If you pass the ApplicationInformation or AccessGroup parameter with the TestSuiteID parameter, the service runs the test suite in the application that you specified.
    - If you do not pass the ApplicationInformation parameter or the AccessGroup parameter with the TestSuiteID parameter, the system runs the test suite in the application that is associated with the default Access Group.

Use the format TestSuiteID=<*pxInsName*>.

- LocationOfResults – The location where the service stores the XML file that contains the test results. This parameter is optional for test Cases and test suites.
- RunWithCoverage – Determines whether the application-level test coverage report is generated after the Execute Tests service runs all relevant test Cases or the selected test suite. For more information, see Generating an application-level test coverage report.
    - If you set the parameter to False, the application-level test coverage report is not generated. This is the default behavior.
    - If you set the parameter to True, and application-level coverage is not running, the Execute Tests service starts application-level coverage mode, runs all unit tests, stops coverage mode, and generates the application-level coverage report. This report is displayed on the test coverage landing page in the Application level section.
    - If you set the parameter to True, and application-level coverage is already running, the Execute Tests service returns an error.

Response

The service returns the test results in an XML file in xUnit format and stores them in the location that you specified in the LocationOfResults request parameter.

The output is similar to the following example:

```
<test-case errors="2" failures="0" label="Purchase order transformation with a bad
element in the output expected" name="report-bad-element-name" skip="0" tests="7">
<nodes expected="/" result="/"><nodes xmlns:purchase="urn:acme-purchase-order"
expected="/purchase:order[1]" result="/purchase-order[1]"><error type="Local name
comparison">Expected "order" but was "purchase-order"</error><error
type="Namespace URI comparison">Expected "urn:acme-purchase-order" but was ""</
error></nodes></nodes><sysout>This text is captured by the report</
sysout><syserr/></test-case>
```

Configuring your default Access Group

When you run the Execute Tests service, you can specify the Access Group that is associated with the application for which you want to run all unit test Cases or a test suite. If you do not specify an Access Group or application name and version, the service runs the unit test Cases or test suite for the default Access Group that is configured for your Pega Platform operator ID.

1. In the navigation pane of Dev Studio, click the **Operator** menu, and then click Operator.
2. In the Application Access Section, select your default Access Group.
3. Selecting default Access Group configuration
4. Click Save.

Configuring your build environment

Configure your build environment so that it can call the Execute Tests service and run all the unit test Cases or a test suite in your application. Your configuration depends on the external validation engine that you use.

For example, the following procedure describes how to configure the Jenkins server to call the service.

1. Open a web browser and go to the location of the Jenkins server.
2. Install the HTTP request plug-in for Jenkins to call the service and the JUnit plug-in so that you can view reports in xUnit format.
   a. Click **Manage Jenkins**.
   b. Click **Manage Plugins**.
   c. On the **Available** tab, select the HTTP Request Plugin and the JUnit Plugin check boxes.
   d. Specify whether to install the plug-in without restarting Jenkins or to download the plug-in and install it after restarting Jenkins.
3. Configure the Pega Platform credentials for the operator who authenticates the Execute Tests service.
   a. Click Credentials, and then click System.
   b. Click the drop-down arrow next to the domain to which you want to add credentials, and click Add credentials.
   c. In the Username field, enter the operator ID that is used to authenticate the service. This operator should belong to the Access Group that is associated with the application for which you want to run test Cases and test suites.
   d. In the Password field, enter the password.
   e. Click OK.
4. Configure the Jenkins URL that runs the service.
   a. Click Manage Jenkins, and then click **Configure System**.
   b. In the **Jenkins Location** section, in the Jenkins URL field, enter the URL of the Jenkins server.
   c. Click Apply, and then click Save.
5. Add a build step to be run after the project is built.
   a. Open an existing project or create a project.
   b. Click **Configure**.
   c. In the Build Section, click Add build step, and select HTTP Request from the list.

d. In the **HTTP Request** Section, in the URL field, enter the endpoint of the service. Use one of the following formats:
    - http://<*your application URL*>/prweb/PRRestService/Pega unit/Rule-Test-Unit-Case/pzExecuteTests
    - http://<*your application URL*>/prweb/PRRestService/Pega unit/Rule-Test-Unit-Case/pzExecuteTests?AccessGroup=<access_group_name:accessgroup_group_users>
    - http://<*your application URL*>/prweb/PRRestService/Pega unit/Rule-Test-Unit-Case/pzExecuteTests?TestSuiteID=<pxInsName>
    - http://<*your application URL*>/prweb/PRRestService/Pega unit/Rule-Test-Unit-Case/pzExecuteTests?ApplicationInformation?=ApplicationInformation:<application_name:application_version>

    If you are using multiple parameters, separate them with the ampersand (&) character, for example, http://<your application URL>/prweb/PRRestService/Pega unit/Rule-Test-Unit-Case/pzExecuteTests?ApplicationInformation?=ApplicationInformation:<application_name:application_version>&TestSuiteID=<pxInsNa

6. From the **HTTP mode** list, select **POST**.
7. Click **Advanced**.
8. In the **Authorization** section, from the **Authenticate** list, select the Pega Platform operator ID that authenticates the service that you configured in step 3.
9. In the Response Section, in the Output response to file field, enter the name of the XML file where Jenkins stores the output that it receives from the service. This field corresponds to the LocationOfResults request parameter. In the Post-build Actions Section, from the Add post build Section list, select Publish Junit test result report and enter **/*.xml in the Test Report XML field. This setting configures the results in xUnit format, which provides information about test results, such as a graph of test results trends. These results are displayed on your project page in Jenkins.
10. Click **Apply**, and then click **Save**.

Running tests and verifying results

After you configure your validation engine, run the service and verify the test results. Your test suites and test Cases must be checked in so that you can run them.

For example, in Jenkins, complete the following steps.

1. Open the project and click Build Now.
2. In the Build History pane, click the build that you ran.
3. On the next page, click Test Result.
4. In the All Tests section, click root. The results of all tests are displayed.
5. **Optional:** Expand a test result in the All Failed Tests section and view details about why the test was not successful.

Test failures

Test Cases and suites that are run using Execute tests services can fail for a few reasons.

Reasons for failed tests:

- The operator does not have access to the location of the results.
- The Access Group that is passed by the service either does not exist or no Access Group is associated with the operator ID.
- The application name and version that are passed do not exist.
- An application is not associated with the Access Group that is passed by the service.
- No Pega unit test Cases or test suites are in the application.
- The test suite *pxInsName* does not exist for the application name and version or for the Access Group that is passed by the service.

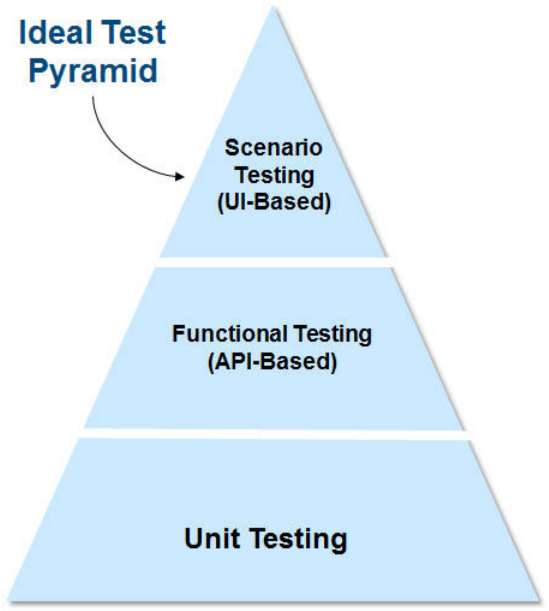
Testing Pega applications

Having an effective Automation test suite for your application in your continuous delivery DevOps pipeline ensures that the features and changes that you deliver to your customers are of high-quality and do not introduce regressions.

At a high level, the recommended test Automation strategy for testing your Pega applications is as follows:

- Create your Automation test suite based on industry best practices for test Automation
- Build up your Automation test suite by using Pega Platform capabilities and industry test solutions
- Run the right set of tests at different Stages of your delivery pipeline
- Test early and test often

Industry best practices for test Automation can be graphically shown as a test pyramid. Test types at the bottom of the pyramid are the least expensive to run, easiest to maintain, take the least amount of time to run, and should represent the greatest number of tests in the test suite. Test types at the top of the pyramid are the most expensive to run, hardest to maintain, take the most time to run, and should represent the least number of tests in the test suite. The higher up the pyramid you go, the higher the overall cost and the lower the benefits.



Ideal test pyramid

- **PegaUnit testing**
- **Running scenario tests against a user interface**
- **Analyzing application quality metrics**
- **Setting up for test automation**

PegaUnit testing

Automated unit testing is a key Stage of a continuous development and continuous integration model of application development. With continuous and thorough testing, issues are identified and fixed prior to releasing an application, which improves the application quality.

Automated unit testing involves creating unit test Cases for tests that are run against individual Rules, grouping multiple test Cases into test suites, running the tests, and viewing the results. When the tests run, the results are compared to the expected results that are defined in assertions.

- **Understanding unit test Cases**

- **Grouping test Cases into suites**
- **Setting up and cleaning the context for a test Case or test suite**
- **Viewing unit test reports**
- **Viewing unit tests without Rules**
- **Unit testing individual Rules**
- **Understanding Pega Platform 7.2.2 and later behavior when switching between Pega unit testing and Automated Unit Testing features**
- **Working with the deprecated AUT tool**

Understanding unit test Cases

A test Case identifies one or more testable Conditions (assertions) that are used to determine whether a Rule returns an expected result. Reusable test Cases support the continuous delivery model, providing a way to test Rules on a recurring basis to identify the effects of new or modified Rules.

You can run test Cases whenever code changes are made that might affect existing functionality. For example, an account executive wants to ensure that a 10% discount is applied to all preferred customers. You create a test Case that verifies that this discount is applied to all preferred customers in the database. The test Case test fails if there are any preferred customers for which the 10% discount is not applied. You then add a new preferred customer to the database and run the test Case to make sure that the customer is correctly configured to receive the discount and that the discount for other preferred customers is not affected.

Additionally, you can group related unit test Cases into a test suite so that you can run multiple test Cases and suites in a specified order. For example, you can run related test Cases in a regression test suite when changes are made to application functionality. For more information about test suites, see Grouping test Cases into suites.

After you create test Cases and test suites, you can run them in a CI/CD pipeline for your application by using Deployment Manager or a third-party Automation server such as Jenkins.

You can use unit test the following types of Rules:

• Activities • Case types • Collections • Data Pages • Data Transforms • Decision Tables • Decision Trees	• Declare Expressions • Flows • Map values • Report definitions • Strategies • When

Typically, you unit test a Rule, and then convert it to a test Case. For flow and Case Type Rules, you record the test Case.

- **Creating unit test Cases for Rules**
- **Creating unit test Cases for processes and Case types**
- **Defining expected test results with assertions**
- **Opening a unit test Case**
- **Running a unit test Case**
- **Viewing test Case results**
- **Exporting a list of test Cases**
- **Managing unit tests and test suites**
- **Best practices for designing Pega unit tests**

Creating unit test Cases for Rules

For most Rules, you can create a reusable test Case by converting a unit test to a test Case, configuring Case details, and then defining expected test results with assertions (test Conditions). When the test Case runs, the test results are compared to the expected results defined for the Rule's assertions. If the test results do not meet the defined assertions, then the test fails.

Before you begin:

Unit test a rule and convert the test run into a test Case. For more information, see Unit testing individual Rules.

1. **Optional:** To modify the rule or class that is used for the test, in the upper-right corner of the Definition tab, click the Gear icon, select the rule or class and then click Submit.

   If you are testing a strategy rule, then the *componentName* and *pzRandomSeed* parameters are also displayed. If you change either of these parameters, the test Case does not return the expected results.

   - *componentName* – The name of the component (for example, Switch) that you are testing.
   - *pzRandomSeed* – An internal parameter, which is the random seed for the Split and Champion Challenger shapes.
2. **Optional:** To prevent the test from being run as part of a test suite or from a REST service, on the Definition tab, select the Disable check box.
   The test Case will be run only when you click Actions > Run.
3. In the **Expected results** section, add assertions that define the expected results of the test. For more information about creating assertions, see Assertions.
4. On the **Setup & Cleanup** tab, configure the Actions to perform and the objects and Clipboard Pages to be available before and after the test runs. You can also clean up the Clipboard after the test is run by applying additional Data Transforms or activities. For more information, see Setting up your test environment.

5. Click Save.
6. In the **Details** dialog box, enter a label that identifies the test Case. The test Case identifier is generated based on the label and cannot be modified after it is saved.

Creating unit test Cases for processes and Case types

When you create a unit test Case for a process or Case Type, you run the process or Case Type and enter data for assignments and Decisions. The system records the data that you enter in a Data Transform, which is created after you save the test form. You can start recording at any time.

Certain Conditions apply on the data that you can record for processes and Case Types. For information, see Data that you can record for flows and Case types about the data that you can record.

RESTRICTION: From Pega Platform™ version 8.5, running and recording a test Case for a process that has a Create Stage is no longer possible for new Case Types. If you have migrated from a version lower than 8.5, you can include the process in any Stage of a migrated Case Type, and then run the test on the process.

Before you begin:

Exclude properties in your work class from the test by modifying the *pxCapturePropertyIgnore* Data Transform. For more information, see Data that you can record for flows and Case typesabout the data that you can record.

1. Open the flow or case type for which you want to record a test.
2. On the toolbar, click Actions > Record test Case.
   The system starts running the flow or Case Type.
3. Enter input as you step through the flow or Case Type.

4. Click Create test Case in the lower-right of the screen to save the recording as a test Case.
5. Click Save, enter a label that identifies the test case, and then click Submit.
6. **Optional:** To modify the rule or class that is used for the test, in the upper-right corner of the Definition tab, click the Gear icon, select the rule or class, and then click Submit.
7. **Optional:** To prevent the test from being run as a part of a test suite or from a REST service, on the Definition tab, select the Disable check box.

   The test Case will be run only when you manually click Actions > Run.
8. In the **Expected results** section, add assertions that define the expected results of the test. For more information about creating assertions, see Defining expected test results with assertions.
9. On the **Setup & Cleanup** tab, configure the Actions to perform and the objects and Clipboard Pages to be available before and after the test runs. You can also clean up the Clipboard after the test is run by applying additional Data Transforms or activities. For more information, see Setting up and cleaning the context for a test Case or test suite.
10. Click Save.
11. Configure the unit test Case.

Result:

After you save the test Case, a Data Transform, which captures the input that you entered, is created and associated with the test Case. You can edit this Data Transform to modify the test Case input. The Edit test Case form also displays the path of the flow or Case type.

- **Data that you can record for flows and Case types**
- **Excluding work Class properties from flows and Case Type tests**
- Creating unit test Cases for Rules

- Running a unit test Case
- Viewing test Case results
- Exporting a list of test Cases

Data that you can record for flows and Case types

When you create a unit test Case for a flow or Case type, the system records the data that you enter.

You can record the following type of information:

- Starter flows. Non-starter flows may be tested from a starter flow that calls on the non-starter flow.
- Subprocesses that are configured as part of a flow.
- The Assignment, Utility, and Approval shapes. For flows, assignments must be routed to the current operator so that the recording of the flow continues and the system captures data as part of the test Case.
- Data that is captured on the *pyWorkPage*.

  When a flow or Case Type runs, a *pyWorkPage* is created on the Clipboard and captures information such as data that you enter for assignments. It also captures information such as Case ID, date and time that the Case was created, and the latest Case Status.

  There are additional assertions that you can configure for flows and Case Types, including Case Status, assigned to, and attachment exists. For these assertions, the system compares expected values to the value that is recorded on the *pyWorkPage*.

  If you refresh or cancel recording the flow or Case Type, data that is on the *pyWorkPage* might not be accurate.
- Local Actions and Flow Actions that are configured as part of the flow or Case Type.

- Child Cases that are created and finish running before the flow or test Case resumes running.
- All properties, excluding properties that begin with either `px` or `pz`.
- Creating unit test Cases for processes and Case types

Excluding work Class properties from flows and Case Type tests

Exclude properties in your work Class from the test by modifying the *pyDataCapturePropertyIgnores* Data Transform.

Some properties, like *.pyID*, are not processed when a unit test Case is run. These properties vary for every test run. The *pxDataCapturePropertyIgnore* Data Transform displays the properties that unit tests do not process.

1. Open the Data Transform:
   a. In the navigation pane of Dev Studio, click **App** > **Classes**, and enter `Work-` in the **Search** field.
   b. Expand **Data Model** > **Data Transform** and then click **pyDataCapturePropertyIgnores.**
2. Save the Data Transform to your *Work-* Class and in your test Ruleset.
3. For each property that you want to exclude in the Data Transform, do the following steps:
   a. On the **Definition** tab, click the **Add** icon.
   b. From the **Action** list, select **Set**.
   c. In the **Target** field, enter the property that you want to exclude.
   d. In the **Source** field, enter two double quotation marks, separated by a space: `" "`.
4. Save the Data Transform.

Defining expected test results with assertions

Use unit test Cases to compare the expected output of a rule to the actual results returned by running the Rule. To define the expected output, you configure assertions (test Conditions) on the test Cases that the test, when run, compares to the results returned by the Rule.

When a test runs, it applies assertions in the order that you define them on the **Definition** tab of the test Case. All assertions, except for run time assertions, must pass for the test to be successful.

For example, an account executive wants to ensure that a 10% discount is applied to all preferred customers. You can create a test Case that verifies that this discount is applied to all preferred customers in the database. If the test does not pass, the results indicate where the 10% discount is not applied.

NOTE: On Decision Trees and Decision Rules, you cannot configure properties from a read-only Data Page or a Data Page that is a declarative target.

- **Configuring activity status assertions**
- **Configuring assigned to assertions**
- **Configuring attachment exists assertions**
- **Configuring Case instance count assertions**
- **Configuring Case Status assertions**
- **Configuring Decision result assertions**
- **Configuring expected run-time assertions**
- **Configuring list assertions**

- **Configuring page assertions**
- **Configuring property assertions**
- **Configuring result count assertions**

Configuring activity status assertions

You can verify that an activity returns the correct status when it runs by configuring an activity status assertion. You can also assert if an activity has an error and, if it does, what the message is so that you can validate that the message is correct.

Before you begin:

Open the unit test case. For more information, see Opening a unit test Case.

1. On the bottom of the Definition tab, click Add expected result.
2. In the Assertion type list, click Activity status.
3. In the Value list, click the status that you expect the activity to return when the test runs.
4. To validate the message that displays for the activity, select Validate message, select a Comparator, and then enter the message that you want to validate in the Value box.
5. **Optional:** To add a comment, click the Add comment icon, enter a comment, and then click OK.
6. Click Save.

- Defining expected test results with assertions

Configuring assigned to assertions

For flows and Case Types, you can use the assigned to assertion to verify that an assignment is routed to the appropriate Work Queue or operator.

If you have multiple assignments on a flow or test Case, you can route each assignment to an operator ID or Work Queue. Clipboard Pages are created for each assignment under the **pyWorkPage** page and capture the assignment details, including the operator ID or Work Queue to which the assignment was routed. The assigned to assertion compares the operator ID or Work Queue to the last assignment that is configured on the flow or Case type, which depends on where you stop recording the flow or Case Type.

For example, your flow has a `Customer Details` assignment, which is routed to the operator ID `johnsmith`. It also has a subprocess with an `Account Information` assignment, which is routed to the `account_processing Work Queue.`

If you record only the `Customer Details` assignment, the assigned to value is `johnsmith`. If you also record the `Account Information` assignment, the assigned to value is `account_processing`.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. On the bottom of the Definition tab, click Add expected result.
2. From the Assertion type list, select Assigned to.
3. From the Assigned to list, select Operator or Work queue.
4. Select a comparator from the Comparator list.
5. In the Value field, press the Down Arrow key and select the operator ID or Work Queue.
6. **Optional:** To add a comment, click the Add comment icon, enter a comment, and click OK.
7. Click Save.

- Defining expected test results with assertions

Configuring attachment exists assertions

For flows and Case Types, you can verify that the flow or Case Type has an attachment of type file or note (attached using the Attach Content shape) or email (attached using the Send Email shape) attached.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. On the bottom of the **Definition** tab, click **Add expected result**.
2. From the **Assertion type** list, select **Attachment exists**.
3. From the **Attachment type** list, select one of the following options, and then provide the value for each field:
    - **File**: Select to specify that the attachment type is file, and then enter the following values:
        - **Description**: Enter the text that was provided as the description in the Attach Content shape.
        - **Name**: Enter the name of the file that was provided in the Attach Content shape.
    - **Note**: Select to specify that the attachment type is note, and then enter text that was entered as the note description in the Attach Content shape.
    - **Email**: Select to specify that the attachment type is an email, and then enter the email subject that was provided in the Send Email shape.
4. Repeat steps 1 through 3 to add additional attachment assertions.
5. **Optional:** To add a comment, click the **Add comment** icon, enter a comment, and click **OK**.
6. Click **Save**.

- **Attachment exists assertions**

- Defining expected test results with assertions
- Attaching content to a Case
- Sending automatic emails from Cases

Attachment exists assertions

On Case types and flows, you can test whether an attachment of type file or note, which were attached in the Attach Content shape, or email, which was attached using the Send Email shape, exists.

If you have multiple attachments on a flow or test Case that match the expected value of the assertion, the assertion runs for every attachment that exists. If the system finds an attachment that matches the assertion value, the assertion passes and iterates over all the attachments on the flow or Case Type. If no attachment exists, the assertion fails

The system compares the expected output on attachments that are recorded on the *pyWorkPage* page. For example, if a Case type has a parent Case that spins off a child Case, and you record just the child Case, the **pyWorkPage** page records attachments for only the child Case and not the parent Case, which is recorded on the *pyWorkCover* page.

In addition, if you create a test Case from a parent Case that generates a child Case that is returned to the parent Case after the child Case runs, the *pyWorkPage* page records the attachments only on the parent Case.

For example, your Case has an Attach Content shape that attaches a `Process immediately` note in the first Stage of the Case type. In the third Stage, your Case has a Send Email shape that attaches an email with the subject `Request approved`. The assertion passes if you searched for either the `Process immediately` note or `Request approved` email subject.

- Configuring attachment exists assertions
- Defining expected test results with assertions
- Attaching content to a Case

- Sending automatic emails from Cases

Configuring Case instance count assertions

For flows and Case Types, you can verify the number of Cases that were created when the Case Type or flow was run.

For example, a Job Application Case Type runs a child Case that processes background checks. If you record the entire Job Applicant case type and the child Case Type, the number of Case instances for Job Application Case Type is one, and the number of Case instances of Background Check child Case Type is one.

If you do not run the run the Background Check child Case Type when you create the test Case, the number of Background Check Case instances is zero.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. On the bottom of the **Definition** tab, click **Add expected result**.
2. From the **Assertion type** list, select **Case instance count**.
3. In the **Of Case Type** field, do one of the following:
    - To select a Case Type from your work pool, press the Down Arrow key and select the Case Type.
    - Enter a Case Type that is not part of your work pool.
4. Select a comparator from the **Comparator** list.
5. In the **Value** field, enter the number of Cases to compare against the output.
6. **Optional:** Click **Add** to add another Case instance count assertion and repeat steps 4 through 6.
7. **Optional:** To add a comment, click the **Add comment** icon, enter a comment, and click **OK**.
8. Click **Save**.

- Defining expected test results with assertions

Configuring Case Status assertions

You can configure a Case Status assertion on a flow or Case Type to verify the status of the Case.

If you have multiple assignments on a flow or Case Type, you can configure a Case Status on each assignment. The *pyWorkPage* on the Clipboard captures the latest Case Status, which depends on where you stop recording the flow or Case Type.

For example, your flow has a `Customer Details` assignment, with the Case Status set as `New`. It also has a subflow with an `Account Information` assignment, with the Case Status set as `Pending`.

If you record only the `Customer Details` assignment, the Case Status, which is captured in the *.pyStatusWork* property on the *pyWorkPage*, is set to `New`. If you also record the `Account Information` assignment, the Case Status is set to `Completed`.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. On the bottom of the Definition tab, click Add expected result.
2. From the Assertion type list, select Case status.
3. Select the comparator from the Comparator list.
4. In the Value field, press the Down Arrow key and select the Case Status.
5. **Optional:** To add a comment, click the Add comment icon, enter a comment, and click OK.
6. Click Save.

- Defining expected test results with assertions

Configuring Decision result assertions

After you create a unit test Case for a Decision table or Decision tree, the system generates a Decision result assertion. This assertion displays the input values for testing the rule, and the result that is generated by the rule.

You can manually update the input values, add properties, remove properties, and modify the default Decision result if the test is modified.

NOTE: This assertion is supported on when Rules, Decision tables, and Decision trees only.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. Click the **Definition** tab.
2. To add multiple input values and results to the assertion, or add other assertions, perform one of the following actions:

   NOTE: You can add multiple input values and results to this assertion but cannot add other assertion types to this test Case. You can add other assertion types to this test Case only if you have a single input and result entry for the assertion.

   - To add multiple input values and results to the assertion:
   a. Select the **Multiple input combinations** check box.
   b. Enter values for the input and result that you expect the assertion to generate when the test stops running.
   c. Click **Add** and enter values for each additional input and result that you want to test.

   - To use one input value and result, enter the values that you expect the assertion to generate when the test stops running. You can then add additional assertions to the test Case.
3. To update the assertion to reflect properties that were added to the rule, click **Refresh**.

   NOTE: Refresh updates the assertion with properties that are added to the rule. If properties have been removed from the rule, then you need to manually remove the properties from the assertion.

4. Add or remove properties by clicking **Manage properties** and then entering the changes. You need to enter data for properties that were added to the rule.

   Result:

   The properties are reflected as unexpected results in test Case results.
5. In the rule form, click **Save**.

   Result:

   The test Case runs the Decision tree or Decision table with each input combination and compares the result with the expected Decision result for that combination.

   Other Decision result combinations or other configured assertions then run. If the expected result of any of the input combinations in the Decision result assertion does not match the result that the rule returns, the assertion fails.

- Defining expected test results with assertions

Configuring expected run-time assertions

You can create an assertion for the expected run time of the rule. The expected run-time assertion is less than or equal to an amount of time that you specify, in seconds.

An actual run time that is significantly longer than the expected run time can indicate an issue. For example, if you are using a report definition to obtain initial values for a data page from a database, there might be a connectivity issue between the application and the database.

By default, after you create a Pega unit test Case for a Data Page, the system generates the expected run-time assertion. The default value of the expected run time is the time that is taken by the rule to fetch results when the test was first run. The system compares that time against future run-time tests.

You can change the default value and configure expected run time assertions for all rule types.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. On the bottom of the Definition tab, click Add expected result.
2. From the Assertion type list, select Expected run time.
3. In the Value field, enter a value, in seconds, that specifies the amount of time within which the execution of the rule should be completed.
4. **Optional:** If you want the test Case to fail when the rule is not run within the specified time, select the Fail the test Case when this validation fails check box.
5. **Optional:** To add a comment, click the Add comment icon, enter a comment, and click OK.
6. Click Save.

- Defining expected test results with assertions

Configuring list assertions

You can create list assertions for page lists on a rule to determine if either the expected result is anywhere in the list of results returned by the rule. Even if the order of results changes, the test will continue to work.

For example, you can verify if a product is present in a product list in a Data Page, regardless of where the product appears in the list results. You can also verify if there is at least one employee with the name John in the results of the Employee list Data Page.

You can also configure assertions for page lists to apply assertions to all the results that are returned by a rule so that you do not have to manually create assertions for each result in the list.

For example, you can verify that a department name is Sales and that a department ID starts with SL for each department in the list of results in the Sales department Data Page. You can also verify if a discount of 10% is applied to each customer in the list of results of the VIP customers Data Page.

You can configure list assertions for page lists on a rule to apply assertions to all the results that are returned by the rule. Configure an ordered list assertion so that you do not have to manually create assertions for each result in the list.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. On the bottom of the **Definition** tab, click **Add expected result**.
2. From the **Assertion type** list, select **List**.
3. Add properties to the assertion.
    a. Click **Add properties**.
    b. If you are adding properties for flows, Case Types, Decision trees, Decision tables, or Data Pages:

        1. In the **of object** field, enter the path of the object with which the properties are compared during the assertion.
        2. Proceed to step d.
    c. If you are adding properties for Data Transforms or activities, complete the following tasks:
        1. From the **Thread** list in the **Actual results** section, select the thread that contains the page whose properties or pages you want to add.
        2. In the **Page** field, enter the page whose properties or pages you want to add.
        3. In the **of object** field, enter the path of the object with which the properties are compared during the assertion.
        4. Proceed to step d.
    d. Select the properties or pages that you want to add. You can search for a property or its value by entering text in the search bar and pressing **Enter**.

        If you select a page, all embedded pages and properties from the page are added. Added properties are displayed in the right pane.

        When you add multiple properties, the assertion passes if the expected output and results match for all properties.

4. **Optional:** In the **Filter** field, enter a property and value on which to filter results or open the Expression Builder by clicking the **Gear** icon to provide an Expression that is used to filter results. The list assertion applies only to the page list entries that are specified for this filter value.
5. From the **Comparator** list, select the comparator that you want to use to compare the property with a specified value.

    Select the `is in` comparator to compare a text, integer, or decimal property to multiple values. The assertion passes if the property matches any of the values that you specify.

6. In the **Value** field, either enter a value with which to compare the property or open the Expression Builder by clicking the **Gear** icon to enter an Expression that is used to provide the value.

   > **NOTE:** The **Gear** icon is not displayed until after you have saved the rule form.

7. To add a comment, click the **Add comment** icon, enter a comment, and click **OK**.
8. Click **Done**.
9. Click **Save**.

> **Result:**
>
> When you run the test Case, the system searches for the specified properties in the page list. One of the following occurs:
>
> - If you selected **In ANY Instance**, the assertion passes if all the properties in the set match the expected values in the page list. If none of the properties match any of the values in the page list, the assertion does not pass.
>
> - If you selected **In ALL instances**, the assertion passes if all the properties in the set match the expected values in every entry in the page list. If any of the properties do not match any entry in the page list, the assertion does not pass.

- Building Expressions with the Expression Builder

Configuring page assertions

Some Rules, such as activities and Data Transforms, can create or remove pages from the system. You can create page assertions to determine whether or not a page exists after a unit test Case runs. You can also assert if a property has an error and, if it does, what the message is so that you can validate that the message is correct.

You can configure page assertions for embedded pages, Data Pages, Data Pages with parameters, and embedded pages within Data Pages that either have or do not have parameter stop-level pages.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test case.

1. On the bottom of the **Definition** tab, click **Add expected result**.
2. In the **Assertion type** list, select **Page**.
3. In the **Page** field, enter the name of the page.
4. In the **Comparator** list, select the comparator that you want to use to compare the property with a specified value:
    - To ensure that the page is created after the unit test runs, select **exists**. The assertion passes if the system does not find the page.
    - To ensure that the page is removed after the unit test runs, select **does not exist**. The assertion passes if the system does not find the page.
    - To ensure that the page has an error after the unit test runs, select **has errors**. The assertion passes if the system finds errors on the page.
    - To ensure that the page is free of errors after the unit test runs, select **has no errors**. The assertion passes if the system finds no errors on the page.
    - To ensure that the page has a specific error message after the unit test runs, select **has error with message** and then enter the message in the **Value** box or click the **Gear** icon to build an Expression. The assertion passes if the page contains the complete error message.
    - To ensure that the page has a portion of an error message after the unit test runs, select **has error message that contains** and then enter the message in the **Value** box or click the **Gear** icon to build an Expression. The assertion passes if the page contains the words or phrases in the error message.

5. **Optional:** To add another page to the assertion, click **Add pages**, and then perform steps 3 through 4.
6. **Optional:** To add a comment, click the **Add comment** icon, enter a comment, and then click **OK**.
7. Click **Save**.

For example:

An activity runs every week to check the last login time of all operators and deletes any operator record (page) from the system if the last login was six months ago. When you test this activity, you can:

1. Set up the Clipboard to load an operator page that has the last login time as six months ago.
2. Create a page assertion that ensures that the page no longer exists after the activity runs.

- **Page assertions**
- Defining expected test results with assertions

Page assertions

You can configure page assertions to determine if a page exists on the Clipboard or if a page has errors.

You can configure page assertions on the following types of pages:

- Embedded pages
- Data Pages
- Data Pages with parameters
- Embedded pages within Data Pages that either have or do not have parameters
- Top-level pages

For example, an activity runs every week to check the last login time of all operators and deletes any operator record (page) from the system if the last login was six months ago. When you test this activity:

- Set up the Clipboard to load an operator page that has the last login time as six months ago.
- Create a page assertion that ensures that the page no longer exists after the activity runs.

- Configuring page assertions
- Defining expected test results with assertions

Configuring property assertions

You can configure property assertions to validate that the actual values of properties returned by a rule are the expected values. You can also assert if a property has an error and, if it does, what the message is so that you can validate that the message is correct.

For example, you can create an assertion that verifies that a customer ID, which appears only once on a Data Page, is equal to 834234.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. On the bottom of the **Definition** tab, click **Add expected result**.
2. In the **Assertion type** list, select **Property**, and then click **Add properties**.
3. Select the properties to add by doing one of the following.
    - For Data Transforms, activities, flows, or Case Types, in the **Actual results** section, select the page containing the properties to add.
    - For other Rules, select the property or page that you want to add.

> **Result:**
>
> Properties are displayed in the right pane. If you selected a page, then all embedded pages and properties from the page are added.

4. To add another property or page, click **Add row**, and then repeat step 3.

> **Result:**
>
> When you add multiple properties, the assertion passes if the expected output and results match for all properties.

5. In the **Comparator** list, select the comparator that you want to use to compare the property with a specified value. Do one of the following:

    - Select the `is in` comparator to compare a text, integer, or decimal property to multiple values. The assertion passes if the property matches any of the values that you specify.
    - Select the `is not in` comparator. The assertion passes if the property does not match any of the values that you specify.
    - Select the `has error with message` comparator to verify that the property has the exact message that you specify in the **Value** box.
    - Select the `has error message that contains` comparator to verify that the property has a portion of the message that you specify in the **Value** box.

6. In the **Value** field, enter a value with which to compare the property. Separate values for the comparators by using the pipe (|) character. For text properties, use double quotation marks at the beginning and end of the value, for example, `"23|15|88"` .

    For example, if you want the assertion to pass when Age property matches either the 5 or 7 values, configure the assertion as `.Age is in 5|7` .

7. **Optional:** To add a comment, click the **Add comment** icon, enter a comment, and then click **OK**.
8. Click **Save**.

- Defining expected test results with assertions
- Building Expressions with the Expression Builder

Configuring result count assertions

You can configure assertions to compare the number of items returned in a page list, page group, value list, or value group on the rule to the result that you expect to see on the Clipboard.

For example, you can create an assertion that verifies that the number of returned results for the number of employees is greater than X number of employees, less than Y number of employees, or equal to Z number of employees.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. On the bottom of the **Definition** tab, click **Add expected result**.
2. For activities and Data Transforms, complete the following tasks:
   a. In the **Page** field, enter the page that contains the property for which you want to test the result count.
   b. In the **Page class** field, select the class to which the page belongs.
3. In the **of object** field, enter the path of the object with which the results are compared or counted against during the assertion.
   - For Data Pages, this value is usually .pxResults.
   - For Data Transforms and activities, you can use any page list on a page.
4. **Optional:** In the **Filter** field, enter a property and value on which to filter results or open the Expression Builder by clicking the **Gear** icon to provide an Expression

that is used to filter results. The list assertion applies only to the page list entries that are specified for this filter value.

5. Select the appropriate comparator from the **Comparator** list.
6. In the **Value** field, enter the value that you want to compare with the object.
7. **Optional:** To add a comment, click the **Add comment** icon, enter a comment, and click **OK**.
8. Click **Save**.

Result:

When you run the test Case, the assertion fails if the expected value does not match the result count returned from the page list, page group, value list, or value group.

- Defining expected test results with assertions

Opening a unit test Case

You can view a list of the unit test Cases that have been created for your application and select the one that you want to open.

1. Open the test Case. Complete one of the following Actions:
   - In the navigation pane of Dev Studio, click **Configure** > **Application** > **Quality** > **Automated Testing** > **Unit Testing** > **Test Cases**.
   - Click the **Test Cases** tab on the rule form.
2. In the **Test Case name** column, click the test Case that you want to open.

- Running a unit test Case
- PegaUnit testing
- Managing unit tests and test suites

Running a unit test Case

Run a unit test Case to validate rule functionality.

1. Open the test case.
2. Complete one of the following Actions:
   - To run multiple test Cases, select the test Cases that you want to run, and then click **Run selected**.
   - To run a disabled test Case or a single test Case, click the test Case to open it, and then click **Actions** > **Run**.

- PegaUnit testing
- Managing unit tests and test suites

Viewing test Case results

After you run a unit test Case, you can view the results of the test run.

The following information is displayed:

- When the test was last run and the user who ran it.
- The rule associated with the test.
- The parameters sent.
- Errors for failed tests.

- Unexpected results for failed tests. This information also includes the run time of the test and the expected run time of the test if the expected run time assertion fails.

1. Open the test case.
2. In the **Run history** column, click **View** for the test Case that you want to view.
3. In the **Test Runs Log** dialog box, click the row for the instance of the test Case that you want to view to open the test results in a new tab in Dev Studio.

You can also view test Case results in the Edit Test Case form after you immediately run the test, in the **Test Case** tab of the Rule form or, for Data Pages, in the **Data Page testing** landing page.

- PegaUnit testing
- Managing unit tests and test suites

Exporting a list of test Cases

You can export a list of all the unit test Cases that are in your application or configured on a Rule form.

1. Export a list of all the Pega unit test Cases that are in your application or for a Rule Type.
   Complete one of the following Actions:
   - To export a list of all the unit test Cases that are in your application, in the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Automated Testing** > **Unit Testing** > **Test Cases**.
   - To export a list of Pega unit test Cases that are configured on a Rule form, click **Test Cases** in the Rule form.
2. Click **Export to Excel**.

- PegaUnit testing
- Managing unit tests and test suites

Managing unit tests and test suites

On the Application: Unit testing landing page you can run unit test Cases and test suites that are not marked as disabled, and view reports to check the quality of your application to identify Rules that did not pass unit testing.

From the Application: Unit testing landing page, you can do the following tasks:

- View, open, and run unit test Cases in your application, export all test Cases in Excel format, and create test suites from selected test Cases on the **Test Cases** tab and from existing test suites on the **Test Suites** tab. To access all test Cases from your application stack, not just from the currently active application, select the **Include built-on applications** option on the Application: Quality Settings landing page.

> **NOTE:** If a test Case is marked as disabled, you cannot run it from the landing page or from a test suite. To run a disabled test Case, open it in Dev Studio and click **Actions** > **Run**.

- View a list of test Cases with no associated Rules in the **Test Cases** tab by clicking **Tests without Rules**.
- Create, view, open, and run Pega unit test suites in the **Test Suites** tab.
- View the test pass trend graph, the percentage of Rules on which unit tests are configured, and see the run results of unit tests on the **Reports** tab.

> **TIP:** To filter information by multiple criteria, click the **Advanced filter** icon.

- Changing application quality metrics settings
- Working with the deprecated AUT tool
- Viewing unit test reports
- Viewing unit tests without Rules

Best practices for designing Pega unit tests

A Pega unit test Case identifies one or more testable Conditions (assertions) that are used to determine whether a rule returns an expected result. Reusable test Cases support the continuous delivery model, providing a way to test Rules on a recurring basis to identify the effects of new or modified Rules.

You can run Pega unit test Cases whenever you make code changes that might affect existing functionality. Additionally, unit test Cases can be grouped together into a test suite so that multiple test Cases and suites can be run in a specified order.

Pega units provide support for the following Rule Types:

- Activity
- Case type
- Collection
- Data Page
- Data Transform
- Decision table
- Decision tree
- Declare Expression
- Flow
- Map value
- Report definition
- Strategy
- When

Identify the right test to automate

Ensure that the following tests are targeted to automate on priority:

- Tests that have predictable results.
- Tests that you might require to run frequently.
- Tests that are easy to automate.
- Tests that help to reduce manual effort while testing complex logic.
- Tests to be aimed at higher test coverage during new functionality development.
- Rules that are isolated and are not dependent on complex predata setup.
- Rules that have wide usage across the application.
- Tests with high return on investment in terms of effort and complexity of creating a Pega unit compared to the coverage that it provides. Assess this profitability before creating a test.

Consider tests that are related to the following categories as low priority for Automation:

- Tests that undergo very frequent changes that require frequent maintenance of test Cases.
- Tests that are easy to test manually with no effort and are complex to automate.
- Tests that need persistence of data to database because they might interfere with existing use Cases in some scenarios. Exercise due diligence in such scenarios.

Keep tests independent and deliver consistent results

Develop tests in line with rule development.

Use the following guidelines to leep each test Case as independent as possible:

- A test must not be dependent on any other test Case.
- Create all the prerequisite data that is required to execute a test by using the **Setup** tab of test Case, so that the test can be run in any environment and produce the same results.
- While configuring a test, consider all data that is necessary for the rule under test, as well as any Rules from which it is sourced or to which it refers.
- Author tests in such a way that they do not interfere with any tests that run after them. Use the Cleanup feature to restore any Clipboard system pages and changes to data instances or work instances that occur during testing.
- The choice of data setup plays a crucial role in developing a test Case quickly and in giving proper coverage for the rule under test. For example, if a test Case requires complex data with embedded pages and page lists for its evaluation, take a snapshot of the Clipboard page instead of setting all the data in a Data Transform.

Keep tests portable

Store all test Case Rules in a separate test Ruleset. Follow these guidelines for the test Ruleset:

- You do not need to package the test Ruleset for production.
- You can package the test Ruleset to any environment that you want to have the tests.
- Configure the test Ruleset to store test Case Rules.
- Add the test Ruleset as the last Ruleset in the order for an application to avoid storing other application Rules in this Ruleset.
- Include the Pega units in the daily CI execution to identify any issues on a daily basis, for example due to new code merges.

If you have a test application that is built on your actual application, add all the test Rulesets that have your Pega units, Pega scenario tests, and related test data, to this test application. For more information, see the following discussion: How to maintain a test application for storing your test cases and related artefacts of an actual Pega application.

Ensure correct test coverage

- Design possible Cases and scenarios manually before attempting to write automated tests.
- Ensure that you cover different possible paths of execution for the rule, and not just the happy path.
- Consider all positive and negative scenarios.
- Consider boundary Cases for the tests.
- Cover exceptions and error messages.
- Add correct and sufficient validations to ensure thorough testing of a functionality. Add enough tests to cover all input and output combinations.
- Create as many small unit tests per rule as necessary to cover all the above recomendations. Smaller unit tests can help quickly identify when rule functionality does not work.
- Keep the test Case logic short, crisp, and visible, covering only what is required.

Ensure ease of maintenance

Following these guidelines helps make each test Case easy to read and understand by any person:

- Ensure that test Case name and description are relevant and explain the purpose of the test Case.
- Follow a naming convention that helps filter test Cases for execution and modification.
- Add comments for every step for better readability. Specifically, user comments for assertions.
- Maintain the history of changes by using the **History** tab.
- Test only one functionality in one test, instead of adding all assertions in one test Case, so that in the Case of a failure, you can quickly identify the root cause.
- Use limited assertions in one test Case and keep only the relevant ones together.
- Keep together a ll relevant assertions in a test Case.
- Modularize the preparation of test data as much as possible in Case of huge and complex structures. Modularization makes maintenance for updating any changes easy and quick in case of any changes in the future.

Ensure that Automation delivers results

- The time that you spend on running and maintaining Automation tests must be much less compared to the time that you spend on manual testing.
- Run tests on a regular basis as a daily CI, on every merge and check-in at a minimum.
- The majority of defects should be caught through automated tests. If not, it is time to refactor or change the strategy.

Limitations

- This framework does not yet follow the standard data-driven or keyword-driven approaches.

- Data is tightly coupled with the test Case and whenever you want to change the data for a test Case, you must edit the test Case.
- Test-driven development (TDD) approach is not incorporated yet.
- Case type and flow Rules have limited support. Non-starter flows are not supported.
- When you run test Cases or test suites in bulk, they run sequentially, in other words, one after the other but not in parallel.

Grouping test Cases into suites

You can group related unit test Cases or test suites into a test suite so that you can run multiple test Cases and suites in a specified order. For example, you can run related test Cases in a regression test suite when changes are made to application functionality.

- **Creating unit test suites**
- **Opening a unit test suite**
- **Running a unit test suite**
- **Viewing unit test suite run results**
- **Adding Cases to a test suite**

- Opening a unit test suite
- Running a unit test Case

Creating unit test suites

To create a unit test suite, add test Cases and test suites to the suite and then modify the order in which you want them to run. You can also modify the context in which to save the scenario test suite, such as the development Branch or the Ruleset.

1. In the header of Dev Studio, click Configure > Application > Quality > Automated Testing > Unit Testing > Test Suites.

2. Click Create new suite.
3. **Optional:** In Description, enter information that you want to include with the test suite. For example, enter information about when to run the test suite.
4. In the Category list, click the type of scenario test suite you are creating:
   - To informally test a feature, select Ad-hoc.
   - To verify critical application functionality, select Smoke.
   - To confirm that changes have not adversely affected other application functionality, select Regression.
5. **Optional:** Provide a value, in seconds, that specifies the length of time within which the run time of the suite should complete in the Expected max runtime field. If you want the test suite to fail when the expected run time has been exceeded, select the Fail the test suite when runtime validation fails check box.
6. Add unit tests Cases or other test suites to the test suite:
   - To add test Cases to the test suite, in the **Test Cases** section, click Add, select the test Cases to include in the suite, and then click Add.
   - To add test suites to the test suite, in the **Test suites** section, click Add, select the test suites to include in the suite, and then click Add.

   NOTE: To filter information by multiple criteria, click the Advanced filter icon.

7. **Optional:** To change the order in which the test Cases or test suites run, drag them to a different position in the sequence.
8. Save the test suite:
   a. Click Save and then enter a Label that describes the purpose of the test suite.

      NOTE: Pega Platform automatically generates the Identifier based on the label you provide. The identifier identifies the scenario test

> suite in the system. To change the identifier, click **Edit**. The identifier must be unique to the system.

   b. **Optional:** In the **Context** section, change details about the environment in which the test suite will run. You can:
      - Change the development branch in which to save the scenario test suite.
      - Select a different application for which to run the scenario test suite.
      - Select a different Ruleset in which to save the scenario test.

9. Click **Save**.
10. Complete any of the following Actions:
    - Remove test Cases or suites from the test suite by selecting them and clicking **Remove**.
    - Apply one or more Data Pages, Data Transforms, or activities to set up the Clipboard before running a test suite in the **Setup** section of the **Setup & Cleanup** tab. You can also create objects, load work and data objects, and add user pages from the Clipboard which will be available on the Clipboard when running the test suite. For more information, see Setting up your test environment.
    - Apply additional Data Transforms or activities to clean up the Clipboard in the **Cleanup** section of the **Setup & Cleanup** tab. You can also prevent the test data from being removed after the test suite runs. For more information, see Cleaning up your test environment.
    - Run a configured test suite by clicking **Actions** > **Run**.

> **NOTE:** If you made changes to the suite, such as adding or removing test Cases or test suites, save those changes before running the suite. Otherwise, the last saved version of the suite will run.

   - View more details about the latest result by clicking **View details** in the banner. Viewing details is possible after a test suite runs. For more information, see Viewing unit test suite run results.
   - To view historical information about previous test runs, such as test date, the run time, expected run time, and whether test passed or failed, click **View previous runs**.
11. Click **Save**. If you are saving the form for the first time, you can modify the Identifier. After you save the Rule form, you cannot modify this field.

- Creating unit test Cases for Rules

Opening a unit test suite

You can view a list of the unit test suites that have been created for your application and select the one that you want to open.

1. In the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Automated Testing** > **Unit Testing** > **Test Suites**.
2. In the **Test suite name** column, click the test suite that you want to open.

- Running a unit test Case
- Managing unit tests and test suites

Running a unit test suite

You can run a unit test suite to validate rule functionality by comparing the expected value to the output produced by running the Rule. Test Cases are run in the order in which they appear in the suite.

1. In the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Automated Testing** > **Unit Testing** > **Test Suites**.
2. Select the check box for each test suite that you want to run.
3. Click **Run selected**. The test Cases run, and the **Result** column is updated with the result, which you can click to open test results.

You can stop the test run by clicking **Stop test execution**.

> **NOTE:** The test suite continues to run even if you close or log out of the Pega Platform, close the Automated Testing landing page, or switch to another Dev Studio tab.

- Managing unit tests and test suites

Viewing unit test suite run results

After you run a unit test suite, you can view the test run results. For example, you can view the expected and actual output for assertions that did not pass.

1. In the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Automated Testing** > **Unit Testing** > **Test Suites**.
2. In the **Run history** column, click **View** for the test suite that you want to view.

> **NOTE:** To quickly view results of the most recent run, click the result in the **Result** column.

3. In the **Test suite runs log** dialog box, click the row for the instance of the test suite run that you want to view to open the results of that run in a new tab in Dev Studio.

> **NOTE:** You can also view test results after you run the test in the Edit Test Suite rule form.

- Running a unit test suite
- Managing unit tests and test suites

Adding Cases to a test suite

You can add test Cases to a unit test suite. When you run a test suite, the test Cases are run in the order in which they appear in the suite.

1. **Optional:** Open the Pega unit test suite, if it is not already open.
2. Click Add test Cases.
3. In the **Add test Cases** dialog box, select the test Cases that you want to add to the test suite.

   > **NOTE:** You can click the Advanced filter icon to filter information by multiple criteria.

4. Click Add.
5. Save the rule form.

- Grouping test Cases into suites

Viewing unit test suite run results

After you run a unit test suite, you can view the test run results. For example, you can view the expected and actual output for assertions that did not pass.

1. In the header of Dev Studio, click Configure > Application > Quality > Automated Testing > Unit Testing > Test Suites.
2. In the Run history column, click View for the test suite that you want to view.

   > **NOTE:** To quickly view results of the most recent run, click the result in the Result column.

3. In the **Test suite runs log** dialog box, click the row for the instance of the test suite run that you want to view to open the results of that run in a new tab in Dev Studio.

   NOTE: You can also view test results after you run the test in the Edit Test Suite rule form.

- Running a unit test suite
- Managing unit tests and test suites

Setting up and cleaning the context for a test Case or test suite

You can set up the environment and Conditions required for running a test Case, determine how to clean up test data at the end of the test run, and set pages on which to automatically run Rules.

You can set Clipboard Pages, apply Data Transforms, load Data Pages, execute activities, create and load objects. All the referenced Data Pages, and Data Objects and user pages that were created during a test run will be automatically removed at the end of each run. To further clean up the Clipboard, add steps to apply additional Data Transforms and execute activities. You can set up or clean up the Clipboard if you are running a test for which the output or execution depends on other Data Pages or information.

For example, your application contains a Data Page *D_AccountTransactionsList*. This Data Page is sourced by a report definition or activity that loads the transactions of the logged-in customer, based on the account type for which the customer views transactions.

The customer number and account type that the customer selects are dynamic properties that are stored on the work page of the Case. The report definition or activity retrieves these properties as parameters from the work page and filters the results as it obtains the results for the Data Page.

When you create a test Case for *D_AccountTransactionsList*, ensure that one of the following Conditions is met:

- The parameter properties are on the work page of the Clipboard before running the data page test.
- Your Data Page has an activity or report definition that refers to the properties of another Data Page that is on the Clipboard to filter the results.

The system always runs Data Transforms, activities, and strategies on the *RunRecordPrimaryPage* page, regardless of which page you chose when you unit tested the Rule in the **Run** dialog box. The system also runs flows and Case Types on the *pyWorkPage*. To update the page with any information required to set up test data, click the Setup tab.

- **Setting up your test environment**

- **Cleaning up your test environment**

- Data Page definition
- Data Transforms
- Unit testing a Data Transform

Setting up your test environment

Configure which Actions you want to run and which objects and pages you want to view on the Clipboard before, during, and after the test is run.

To set up the environment and Conditions that are required before running this test Case, copy or create Clipboard Pages, apply Data Transforms, load Data Pages, execute activities, and create and load objects. Then, define the connections to Data Pages or third-party databases to simulate during the test. Finally, after running the test Case, set up the environment and Conditions that are required by applying Data Transforms, loading Data Pages, executing activities, and creating and loading objects.

Before you begin:

Open the test Case or test suite that you want to set up. For more information, see Opening a unit test Case.

1. Click the **Setup & Cleanup** tab.
2. **Optional:** To make specific Conditions available during test execution, expand the **Before rule execution** section, and then configure the Conditions:
   a. Copy or create Clipboard Pages.
      For more information, see Copy or create clipboard pages.
   b. Add additional Clipboard data.
      For more information, see Add additional clipboard data.
3. **Optional:** To define simulation settings for the test, expand the **Simulation** section, and then configure the simulations.
   For more information, see Simulating data pages and third-party connections.
4. **Optional:** To make specific Conditions available after test execution, expand the **After rule execution** section, and then add additional Clipboard data.
   For more information, see Adding additional Clipboard data.

   NOTE: You can set up Actions after rule execution for test Cases only.

5. To run the Rule under on a page and avoid copying the entire page to *RunRecordPrimaryPage*, in the **Advanced** section, enter the page under which you want to run the Rule.
6. Click **Save**.

- **Copying and creating Clipboard Pages in setup**
- **Adding additional Clipboard data**

- **Simulating Data Pages and third-party connections**

- Clipboard tool
- Data Page definition
- Data Transforms

Copying and creating Clipboard Pages in setup

When setting up your test environment, you can set to copy or create Clipboard Pages before the test runs.

1. On the Setup & Cleanup tab for the test Case or test suite for which you want to set up the context, expand the **Before rule execution** section, and then expand the Setup data section.
2. Click Add data.
3. In the Description box, enter a description of the Clipboard page you want to copy or create.
4. **Optional:** Copy a Clipboard page:
    a. In the Type section, select Copy page.
    b. To copy a Clipboard page from a different thread, click the current thread name, and then click desired thread name.
    c. Select the check box next to the page that you want to be available in the Clipboard during test execution, and then click Next.
    d. Edit the Clipboard page and then click OK. You can rename the parent page, modify the values of existing properties or add new properties and their values to the parent page and child pages.
5. **Optional:** Create a Clipboard page:
    a. In the Type section, select Create page.
    b. In the Page Name field, enter a name of the page you want to create.
    c. In the Class field, enter or select the class of the page you want to create.
    d. Click Next.

   e. Edit the Clipboard page and then click OK. You can rename the parent page, modify the values of existing properties or add new properties and their values to the parent page and child pages.
6. Save the test Case or test suite.

Adding additional Clipboard data

When setting up your test environment, you can add additional Clipboard data before or after the test runs. You can apply Data Transforms, load Data Pages, execute activities, load objects, create data objects, and create work objects.

1. On the Setup & Cleanup tab for the test Case or test suite for which you want to set up the context, choose whether to add the data before or after the Rule runs.
   - To add the data before the test Rule runs, select **Before rule execution** section, expand the **Additional Clipboard data** subsection.
   - To add the data after the test Rule runs, select **After rule execution**.
2. For each action you want to perform to the Clipboard data, click Add step and then select the action.
   - To apply a Data Transform:, select Apply data transform, and then, in the Name field, enter or select the name of the Data Transform to apply.
   - To load a Data Page, select Load Data Page, and then, in the Name field, enter or select the name of the data to apply.
   - To execute an activity, select Execute activity, and then, in the Name field, enter or select the name of the activity to apply.
   - To load an object, select Load object, enter or select the Class of the object in the Of Class field, and then enter a name for the page in the Load on page field.
   - To create a data object, select Create data object, enter or select the class of the object in the Of Class field, and then enter a name for the page in the Load on page field.
   - To create a work object, select Create work object, and then, in the Of Class field, enter or select the Class of the work object.

3. **Optional:** If parameters are configured on Rules, then you can modify them by clicking the gear icon, and providing values in the **Configure parameters** dialog box, and then clicking Submit.
4. **Optional:** If keys are configured on loaded or created objects, then you can define their values by clicking the With Keys gear icon, and providing values in the **Configure parameters** dialog box.
5. Save the test Case or test suite.

Simulating Data Pages and third-party connections

When setting up your test environment, you can simulate Data Pages and third-party connections. Such simulations let you run your tests without depending on the availability of third-party servers.

1. In Dev Studio, open a Pega unit test case.
2. Click the Setup & Cleanup tab.
3. In the **Setup** section, expand the **Simulation** section, and then click Add Rules.
4. To include a rule that the test rule directly references, on the Referenced Rules tab, select the Rules to simulate, and then click Add.
5. To include any rule that the test rule does not directly reference, do the following for each rule:
   a. Click the Other Rules tab and then click Add.
   b. In the Rule type list, click the type of rule that you want to simulate.
   c. In the Class box, enter the class of the rule that you want to simulate.
   d. In the Rules field, enter the rule that you want to simulate.
   e. Click the Add button.

   The selected Rules display in the **Simulation** section on the Setup & Cleanup tab.
6. In the Simulate with list for each rule, click a simulation method:
   - Select As defined in the rule to use the default simulation defined in the rule.
   - Select Select datatransform rule to define your own Data Transform rule. You can reuse this rule in other test Cases.

- Select Define data here to manually provide test data specific to this particular test Case. You can copy pages from the Clipboard or create new pages and populate them with required test data.
- Select None to disable the simulation.

- Understanding unit test Cases

Cleaning up your test environment

After you run a unit test Case or test suite, user and Data Pages used to set up the test environment and the parameter page are automatically removed by default. You can apply additional Data Transforms or activities to remove other pages or information on the Clipboard before you run more test Cases or suites.

Before you begin:

Open the unit test Case. For more information, see Opening a unit test Case.

1. Click the Setup & Cleanup tab.
2. To keep the test data on the Clipboard at the end of the test run, clear the Cleanup the test data at the end of run check box in the Cleanup Section.
3. In the Cleanup Section, click Add Step.
4. Select Apply Data Transform or Execute activity, and then provide the appropriate Data Transform or activity in the next field.
5. If parameters are configured on the rule, you can configure them by clicking the Parameters link and providing values in the **Configure parameters** dialog box.
6. **Optional:** Provide additional information in the Enter comments field.
7. Click Save.

- Creating unit test Cases for processes and Case types

Test environment cleanup

After you run a unit test Case or test suite, user and Data Pages used to set up the test environment and the parameter page are automatically removed by default.

NOTE: You can override this behavior if you want the data from the current test to be available to the subsequent tests.

You can also apply additional Data Transforms or activities to remove other pages or information on the Clipboard before you run more tests. Cleaning up the Clipboard ensures that Data Pages or properties on the Clipboard do not interfere with subsequent tests. For example, when you run a test Case, you can use a Data Transform to set the values of the pyWorkPage Data Page with the AvailableDate, ProductID, and ProductName properties.

You can use a Data Transform to clear these properties from the pyWorkPage. Clearing these values ensures that, if setup data changes on subsequent test runs, the test uses the latest information. For example, if you change the value of the AvailableDate property to May 2018, you ensure that the Data Page uses that value, not the older (December 2018) information.

- Cleaning up your test environment
- Setting up and cleaning the context for a test Case or test suite
- Creating an activity
- Data Transforms

Viewing unit test reports

View a graph with test pass rate trend data, a summary of Pega unit tests that were run, and an overview of Pega unit test compliance for currently included applications on the **Reports** tab on the Unit Testing landing page.

By default, a test Case is considered as executed if it ran in the last seven days. You can change the number of days for which a test can be considered executed on the Application: Quality Settings landing page. The overview also includes the percentage of Rules on which Pega unit test Cases are configured. View Pega unit test reports to check the quality of your application and identify Rules that did not pass Pega unit testing.

1. In the header of Dev Studio, click Configure > Application > Quality > Automated Testing > Unit Testing > Reports.
2. **Optional:** To filter information by more than one criterion, click the Advanced filter icon.
3. **Optional:** Generate and export a report for test coverage and test runs for a Rule Type.
    a. For the Rule Type for which you want to export a report, click a number in the column of the table for either pie chart.
    b. Click Actions.
    c. Click Export to PDF or Export to Excel.

- Changing application quality metrics settings
- Viewing application quality metrics
- Managing unit tests and test suites
- Viewing unit tests without Rules

Viewing unit tests without Rules

On the Application: Unit testing landing page you can display a list of unit tests that are not associated with any Rule and export this list to an XLS or a PDF file. You should deactivate these unit tests because they will always fail.

1. In the header of Dev Studio, click Configure > Application > Quality > Automated Testing > Unit Testing.
2. Click Tests without Rules.
3. **Optional:** Generate and export a report that contains a list of test Cases that are not associated with any Rules.

    - To export to PDF format, click **Actions** > **Export to PDF**
    - To export to XLS format, click **Actions** > **Export to Excel**

- Managing unit tests and test suites
- Viewing unit test reports

Unit testing individual Rules

An incorrect Rule configuration in an application can cause delays in Case processing. To avoid configuration errors such as incorrectly routed Assignments, unit test individual Rules as you develop them. To expedite future Rules testing, you can create reusable test Cases from the unit test.

You can test a Rule with test data that you provide by clicking **Actions** > **Run** on the Rule form toolbar. For some Rule Types, such as binary file Rules, Pega does not provide an option for unit testing. If the Rule cannot be unit-tested, the **Run** option is not available.

NOTE: Some Rules can be unit-tested without the **Run** option available. For example, to perform a unit test on a Case Type, you can use the **Record test Case** action. For more information, see Understanding unit test Cases

The appearance of the **Run Rule** window varies across Rule Types, so how you run a Rule varies by its type. In general, however, the Rules run by using data from a test page that you define for the test.

Tasks involved in defining the test page include the following:

1. Selecting a method for creating the test page – You can copy values from a thread of an existing Clipboard Page to the test page, create a new test page, or reset the values of an existing test page. For more information about Clipboard Pages, see Clipboard tool.
2. Applying Data Transforms – For a reusable and expedited method of making decisions and calculating values, you can apply Data Transforms to set values for

the test page. For example, to unit test a Decision table, you can create a Data Transform to provide values for the properties evaluated by the table, rather than manually entering values when you run the Rule. For more information about Data Transforms, see Data Transforms.

3. Manually entering test data – In some Cases, you can manually enter values to use. If you enter values for a test, the values that you enter override values on the test page.
4. Specifying how service Rules run – For services, you also specify whether the service Rule is to run in your session or is to run as a newly created service requestor. If the service is configured to run as an authenticated user, you are prompted for a user name and password.

NOTE: To test a circumstance Rule, ensure that the circumstances are correct for the Rule. Otherwise, the system tests the base Rule.

When you run the Rule, the system uses Rule resolution. If you unit test a Rule, and there is a higher version of the Rule, the system runs the higher version.

After you run the test, you can also convert the test to a reusable test Case that you can run at any time. For more information about using unit test Cases, see Understanding unit test Cases.

- **Unit testing service Rules**

Unit testing service Rules

Unit test service Rules such as, email, Java, MQ, SAP, and SOAP Rules.

- **Providing test data when unit testing service Rules**
- **Unit testing a Service EJB Rule**
- **Unit testing a Service File rule**

- **Unit testing a Service HTTP Rule**
- **Unit testing a Service dotNet Rule**
- **Unit testing a Service SAP Rule**
- **Unit testing a Service SAPJCo Rule**

Providing test data when unit testing service Rules

When unit testing service rules, you can provide some representative data for the service rule either by typing or pasting in text, or by identifying an activity that will generate the test data.

Determine the source of the test data

Before you begin, determine how you to provide the sample data for the rule to process. Each service Rule Type has different options.

Service Type	Test Data Options
EJB, HTTP, JMS, Java, JSR 94, and MQ	• Manually provide values for each parameter. • If the request data includes objects rather than scalar values, identify an activity that sets the values.
Email	Manually provide values for the subject and body of the email message.
File	• Manually provide content that is similar to file input. • Browse to and select a test input file for the service to process.

SOAP and .NET	Additionally, if the rule has data mappings for request headers, you must enter values for them, too.

Create activities that set up request data

If the EJB, HTTP, JMS, Java, JSR 94, or MQ service rule that you want to test receives objects rather than scalar values in the request, you cannot provide request values directly in the unit testing form. Instead, create and identify an activity that sets the request values.

The *Data-Admin-IS-ClientSimulation* class is a helper class for the unit testing feature. A page of class *Data-Admin-IS-ClientSimulation* serves as temporary storage location for the test data that the unit testing feature uses to test a service rule. Your activity must create a page named `pySimulationDataPage` for the *Data-Admin-IS-ClientSimulation* class and store the test request data on that page.

- For the test message data, configure the activity to put the values in the Java Object List property named *pyRequestObjectValues*.
- If the test message is for an HTTP, JMS, or MQ service, your simulation activity must also provide any information that is required for header fields or message properties.
-
    - Use the `Value Group` property named *pyRequestHeaderGroup* to store test data for HTTP, JMS, or MQ header fields.
    - Use the `Value Group` property named *pyRequestPropertyGroup* to store test data for JMS message properties.

For an example, locate and open one of the following standard activities:

- *Rule-Service-.SetRequestData*
- *Data-Admin-IS-ClientSimulation.SetRequestData*

You can use one of these activities as the template for yours: save it into the appropriate RuleSet and specify the same Applies To class as that specified as the Primary Page class on the Service tab of the service rule that you want to test.

- Unit testing individual Rules

Unit testing a Service EJB Rule

Use the unit testing feature to verify that the operations of a Service EJB rule Function correctly before you add the external client to your testing process.

> **NOTE:** Service EJB Rules are no longer being actively developed, and are being considered for deprecation in upcoming releases. Using Service EJB Rules does not follow Pega development best practices. Consider other implementation options instead.

Unit testing provides only partial evidence of a correct implementation. For more comprehensive information on testing services, see the Pega Community article *Testing Services and Connectors*.

Before you begin, see How to provide test data when testing service rules.

To run a unit test, complete the following steps:

1. Save the Rules form.
2. Start the Tracer by clicking **Actions** > **Trace**. For more information, see Tracing services.
3. Click **Actions** > **Run**.
4. Complete the form as described in the following table, and then click **Execute**.

Field	Description
Requestor Context	Select a radio button to define the requestor session is to be used in the test:

Field	Description
	• Use current requestor context — Use your current requestor session (including your RuleSet list, privileges, and current Clipboard) • Initialize service requestor context — Create a new requestor session based on the *APP* requestor type and, if the service package requires authentication, another Operator ID instance.
Authentication User ID	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter the Operator ID to be used to test the service
Authentication Password	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter a password for the Operator ID.
Enter Request Data	Select a radio button to define the source of request data values for this test: • Specify individual request values — This option appears only if the EJB method parameters are scalar values, such as strings, numbers, or booleans. • Invoke Initialization activity — A test activity creates values for the EJB method values.

Field	Description
Method Parameter Values	If you selected Specify individual request values for the previous field, enter in the Value field a literal constant value for each EJB method parameter declared on the Parameters tab. Enter a value that corresponds to the Java data type listed.
Activity	If you selected Invoke Initialization activity , enter here the Activity Name key part of an activity that creates EJB method parameters. The system assumes the Applies To class of the activity matches the Primary Page Class value on the Service tab. If the activity applies to a different class, enter the class name, a period, and the activity name.

Unit testing a Service File rule

Use the unit testing feature to verify that the operations of a service file rule Function correctly before you add an external component to your testing process.

To run a unit test, complete the following steps:

1. Save the rule form.
2. Start the Tracer by clicking **Actions** > **Trace**. For more information, see Tracing services.
3. Click **Actions** > **Run**.
4. Complete the form as described in the following table, and then click **Execute**.

Field	Description
Requestor Context	Select a radio button to define the requestor session is to be used in the test:

Field	Description
	• Use current requestor context — Use your current requestor session (including your RuleSet list, privileges, and current Clipboard). • Initialize service requestor context — Create a new requestor session based on the `APP` requestor type and, if the service package requires authentication, another Operator ID instance.
Authentication User ID	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter the Operator ID to be used to test the service.
Authentication Password	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter a password for the Operator ID.
Enter Request Data	Select: • Supply File Content if you will type in or paste in the text of a test file. This option is not available if non-text input is expected — that is, if the Data Mode field on the Method tab is set to a value other than `text only`.

Field	Description
	• Upload a local file if the test file is on your workstation or network.
File Content	Enter the contents of a test file, including delimiters. This text area appears when you choose Supply File Content for the previous field.
File Location	Click **Browse** to upload a test file. This field appears when you choose Upload a local file for the previous field.

- Service File rules

Unit testing a Service HTTP Rule

Use the unit testing feature to verify that the operations of a Service HTTP rule Function correctly before you add the external client to your testing process.

⚠ **IMPORTANT:** Service HTTP Rules are no longer being actively developed. To avoid upgrade issues when these Rules are deprecated, use Service REST Rules instead. For more information about Service REST Rules, see Service REST rules.

Unit testing provides only partial evidence of a correct implementation. For more comprehensive information on testing services, see the Pega Community article *Testing Services and Connectors*.

Before you begin, see How to provide test data when testing service rules.

To run a unit test, complete the following steps:

1. Save the rule form.

2. Start the Tracer by clicking **Actions** > **Trace**. For more information, see Tracing services.
3. Click **Actions** > **Run**.
4. Complete the form as described in the following table, and then click **Execute**.

Field	Description
Requestor Context	Select a radio button to define the requestor session is to be used in the test: • Use current requestor context — Use your current requestor session (including your RuleSet list, privileges, and current Clipboard) • Initialize service requestor context — Create a new requestor session based on the *APP* requestor type and, if the service package requires authentication, another Operator ID instance.
Authentication User ID	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter the Operator ID to be used to test the service.
Authentication Password	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter a password for the Operator ID.
Enter Request Data	Select a radio button to define the source of request data values for this test:

Field	Description
	• Specify individual request values — Select if you want to manually enter the values for the message data in the Message Buffer text box. • Invoke Initialization activity — Select if you want to run an activity that creates the string for the message data.
HTTP Header Values	If you selected Specify individual request values for the previous field, enter in the Value field a literal constant value for each Header Field row on the Request tab.
Message Buffer	If you selected Specify individual request values , enter or paste the test message data in this text box.
Activity	If you selected Invoke Initialization activity, specify the Activity Name key part of an activity that creates the message. The system assumes the Applies To class of the activity matches the Primary Page Class value on the Service tab. If the activity applies to a different class, enter the class name, a period, and the activity name.

Unit testing a Service dotNet Rule

Use the unit testing feature to verify that the operations of a Service dotNet rule Function correctly before you add the external client to your testing process.

⚠ **IMPORTANT:** Service dotNet Rules are no longer being actively developed, and are being considered for deprecation in upcoming releases. Using Service dotNet Rules does not follow Pega development best practices. Use Service SOAP Rules instead. For more information, see Service SOAP rules.

Unit testing provides only partial evidence of a correct implementation. For more comprehensive information on testing services, see *Testing Services and Connectors*, a document on the Integration pages of the Pega Community.

To run a unit test, complete the following steps:

1. Save the rule form.
2. Start the Tracer by clicking **Actions** > **Trace**. For more information, see Tracing services.
3. Click **Actions** > **Run**.
4. Complete the form as described in the following table, and then click **Execute**.

Field	**Description**
Requestor Context	Select a radio button to define the requestor session is to be used in the test: • Use current requestor context — Use your requestor session (including your RuleSet list, privileges, and current Clipboard) • Initialize service requestor context — Create a new requestor session based on the `APP` requestor type and, if the service package requires authentication, another Operator ID instance.

Field	Description
Authentication User ID	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter the Operator ID to be used to test the service
Authentication Password	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter a password for the Operator ID.
Enter Request Data	Select a radio button to define the source of request data values for this test: • Specify individual request values — This option appears only when all elements of the message are simple text values not objects or complex values of type `XML Page` . • Supply SOAP Envelope — You provide the entire SOAP message including the header.
SOAP Header Values	If you selected Specify individual request values for the previous field, enter in the Value field a literal constant value for each Header Field row on the Request tab. Enter a value that matches the XSD type shown.
SOAP Parameter Values	If you selected Specify individual request values for the previous field, enter in the Value field a literal constant value for each Request Parameters row listed on the

Field	Description
	Request tab. Enter a value that corresponds to the XSD data type shown.
SOAP Request Envelope	If you selected Supply Soap Envelope , enter or paste a well-formed and valid XML document in the SOAP Request Envelope text area, starting with the <?xml version="1.0"> declaration. If the service expects requests containing an array element or XML Page elements, a skeleton document is provided as a starting point.

Unit testing a Service SAP Rule

Services start their processing in response to a request from an external application. Before you add the external application to your testing process, use the Simulate SOAP Service Execution feature to verify that the service processes data appropriately. When using this feature, you manually provide some representative data to process. You specify a test page for the rule to use, provide sample data as the input, run the rule, and examine the results to see if they are what you expect.

⚠ **IMPORTANT:** Service SAP Rules are no longer being actively developed, and are being considered for deprecation in upcoming releases. Using Service SAP Rules does not follow Pega development best practices. Use Service SOAP Rules instead. For more information, see Service SOAP rules.

If you have the *AutomatedTesting* privilege (through an access role), you can use the features of Automated Testing such as saved test Cases and unit test suites for testing your Service SAP rule. See About Automated Unit Testing and Working with the Test Cases tab for more information.

Before you begin

Before you begin testing the Service SAP rule, determine how you will provide the sample data for the service rule to process. For help with this step, and for information about additional ways to test your services, see the articles about testing services and connectors in the *Testing Applications* category of the Pega Community.

Run the rule

Complete the following steps:

1. Save the rule.
2. Start the Tracer by clicking **Actions** > **Trace**. For more information, see Tracing services.
3. Click **Actions** > **Run**.
4. Fill out the fields in the form as described in the following table:

Field	Description
Test Cases	If you have the *AutomatedTesting* privilege, **Run Against a Saved Test Case**, **Show Saved Results**, and **Run Test Case** are available if this Rule has saved test Cases. To run the Rule and see how its behavior compares to that in a previously saved test Case, select a choice from the **Run Against a Saved Test Case** drop-down list. After making your selection, click **Run Test Case**. If differences are found between the results of running the current state of the Rule and the saved

Field	Description
	test Case, they are displayed and you have the options of choosing to ignore differences for future test runs, overwriting the saved test Case, and saving the results. See the Playing back saved test cases section in Working with the Test Cases tab. Click **Show Saved Results** to view any previously saved test Case results.
Requestor Context	Select one of the following items to specify which requestor session to use in the test: • Use current requestor context — Runs the Rule in your session, that is, with your Ruleset list, privileges, and current Clipboard. • Initialize service requestor context — Run the Rule in a newly created service requestor session based on the *APP* requestor type and, if the service package requires authentication, another Operator ID instance.
Authentication User ID	If you selected Initialize service requestor context , and the service package for the service requires authentication, enter the Operator ID to use to test the service.
Authentication Password	If you selected Initialize service requestor context , and the service package instance

Field	Description
	for the service requires authentication, enter a password for the Operator ID.
Enter Request Data	Select one of the following to define the source of request data values for this test: • Specify individual request values — This option appears only when all elements of the message are simple text values, not arrays or complex values of type `XML Page`. • Supply SOAP Envelope — You can enter an entire SOAP message including the header.
SOAP Header Values	If you selected Specify individual request values for the previous field, in the Value field, enter a literal constant value for each Header Field row on the Request tab. Enter a value that matches the XSD type shown.
SOAP Parameter Values	If you selected Specify individual request values for the previous field, in the Value field, enter a literal constant value for each Request Parameters row listed on the Request tab. Enter a value that corresponds to the XSD data type shown.
SOAP Request Envelope	If you selected Supply Soap Envelope , enter or paste a well-formed and valid XML document in the SOAP Request Envelope text area, starting with the <?xml version="1.0" ?> declaration.

Field	Description
	If the service expects requests containing an array element or elements, a skeleton document is provided as a starting point.

5. Click **Execute** to run the Rule. The system runs the Rule and displays the results.
6. Click the **Clipboard** icon in the Quick Launch area to see the Clipboard Pages that were generated.
7. Run the rule again using different data, as necessary.
8. Optional. If you have the *AutomatedTesting* privilege, the **Save as Test Case** button is available and you can click it to create a test Case that holds the test data and the results.

- Service SAP rules

Unit testing a Service SAPJCo Rule

Use the unit testing feature to verify that the operations of a Service SAPJCo rule Function correctly before you add the external client to your testing process.

⚠ **IMPORTANT:** Service SAPJCo Rules are no longer being actively developed. To avoid upgrade issues when these Rules are deprecated, use other web-based SAP capabilities.

Unit testing provides only partial evidence of a correct implementation. For more comprehensive information on testing services, see the Pega Community article *Testing Services and Connectors*.

Before you begin, see How to provide test data when testing service rules.

To run a unit test, complete the following steps:

1. Save the Rule form.
2. Start the Tracer by clicking **Actions** > **Trace**. For more information, see Tracing services.
3. Click **Actions** > **Run**.
4. Complete the form as described in the following table, and then click **Execute**.

Field	**Description**
Requestor Context	Select a radio button to define the requestor session is to be used in the test: • Use current requestor context — Use your current requestor session (including your RuleSet list, privileges, and current Clipboard). • Initialize service requestor context — Create a new requestor session based on the *APP* requestor type and, if the service package requires authentication, another Operator ID instance.
Authentication User ID	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter the Operator ID to be used to test the service.
Authentication Password	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter a password for the Operator ID.

Field	Description
Enter Request Data	Select a radio button to define the source of request data values for this test: • Specify individual request values — This option appears only if the Java method parameters are scalar values, such as strings, numbers, or booleans. • Invoke Initialization activity — A test activity creates values for the Java method values.
Method Parameter Values	If you selected Specify individual request values for the previous field, enter in the Value field a literal constant value for each Java method parameter declared on the Parameters tab. Enter a value that corresponds to the Java data type listed.
Activity	If you selected Invoke Initialization activity, enter here the Activity Name key part of an activity that creates Java method parameters. The system assumes the Applies To class of the activity matches the Primary Page Class value on the Service tab. If the activity applies to a different class, enter the class name, a period, and the activity name.

Understanding Pega Platform 7.2.2 and later behavior when switching between Pega unit testing and Automated Unit Testing features

Beginning with Pega 7.2.2, you can use Pega unit testing to create test Cases to validate the quality of your application by comparing the expected test output with results that are returned by running Rules. In previous versions, you used Automated Unit Testing (AUT) to create test Cases.

If you have the *AutomatedTesting* privilege, you can use Automated Unit Testing (AUT) and switch between Pega unit testing and AUT, for example, if you want to view test Cases that you created in AUT. The following list describes the application behavior when you use Pega unit testing and AUT:

- When you unit test activities that are supported by both Pega unit testing and AUT, the **Run Rule** dialog box displays updated options for creating unit tests for Pega unit testing. However, you cannot create unit test Cases for AUT by using this dialog box.
- When you use Pega unit testing, you can create, run, and view the results of Pega unit testing on the Test Cases tab for the supported rule types.
- You can view, run, and view the results of Pega unit test Cases by clicking Dev Studio > Automated Testing > Test Cases. You can also switch to the AUT landing page by clicking Switch to old version.
- When you switch to the AUT landing page, you can create, run, and view the results of unit test Cases for AUT on the Test Cases tab for activities, Data Transforms, and data tables, which are supported by both Pega unit testing and AUT. You can create unit test Cases only by clicking the Record test Case button and using the older **Run Rule** dialog box.
- In the Automated Unit Testing landing page, you can restore the Automated Rule Testing landing page by clicking Switch to new version. When you click the Test Cases tab in an activity, Decision table, or Decision tree, the tab displays options for creating Pega unit test Cases.

- If you use the Automated Unit Testing landing page, and then log out of the system, Dev Studio displays the **Dev Studio** > **Application** > **Automated Unit Testing** > menu option instead of the **Dev Studio** > **Application** > **Automated Testing** > option. To return to the Automated Unit Testing landing page, click **Switch to new version** on the Automated Unit Testing landing page.

Working with the deprecated AUT tool

In older versions of Pega Platform, automated unit tests were created using the Automated Unit Testing (AUT) tool, which has since been replaced by PegaUnit testing. If you have automated unit tests that were created using AUT and they haven't been changed to PegaUnit test Cases, then you can switch back to AUT to manage those tests.

NOTE: AUT has been deprecated and is not supported in the current version of Pega Platform. Switch to and use AUT only if you have existing automated unit tests created with AUT. See PegaUnit testing more information.

Note the following behavior:

- To use AUT, your operator ID must have the *AutomatedTesting* privilege through an access role.
- Switch from PegaUnit testing to AUT by clicking **Configure** > **Automated** > **Testing** > **Test Cases** and clicking **Switch to old version** on the Automated Unit Testing landing page.
- Click the **Test cases** tab of the Automated Unit Testing landing page to display options for creating unit tests for activities, decision tables, and decision trees.
- If you are using the Automated Unit Testing landing page and then log out of the system, you can click **Configure** > **Application** > **Automated Unit** > **Testing**, and then click **Switch to new version** to restore the Automated Testing landing page.

Viewing, playing back, and rerecording test Cases

1. Click the **Automated Unit Tests** tab.
2. Select **Unit Test Cases** in the **Show** field.
    - To play back a test Case, click its name in the **Name** column.
    - To rerecord a test Case, right-click the test Case name and click **Re-record**.

> **NOTE:** If the underlying test Case Rule belongs to a Ruleset that uses the check-out feature, you must have the test Case rule checked out to you before re-recording the test Case.

Opening Rules in test Cases and unit test suites

1. Click the **Automated Unit Tests** tab.
2. Right-click a test Case or suite and click **Open** to open its Rule.

Withdrawing test Cases and unit test suites

1. Click the **Automated Unit Tests** tab.
2. Right-click a test Case or suite and click **Withdraw**.

Withdrawn test Cases and suites are not displayed on the Automated Unit Tests tab.

Unit test suite run results

You can view the results of your recent unit test suite runs in either the **Dashboard** tab or **Reports** tab. The **Dashboard** tab displays the ten most recent runs. The **Reports** tab displays earlier results and, for a given unit test suite, shows results from the last fifty (50) runs of that unit test suite.

If you ran a unit test against a saved test Case for a Decision table, Decision tree, activity, or Service SOAP rule and selected the **All Cases** option in the **Run Rule** form, those results also appear in the **Dashboard** tab.

For activity test Cases, if the activity test Case has an approval list, differences are reported only for pages and properties on the list. If the test Case has an approval list and the only differences are for pages and properties not on the list, those differences are not reported. If differences are found for items on the approval list, you can remove the item from the approval list for that test Case.

Creating and scheduling unit test suites

To create a unit test suite:

1. Click the Schedule tab.
2. Click **Create Suite**.
3. In the New Rule form, enter the requested information for creating a unit test suite.

To run a unit test suite or to schedule a run:

1. Click the **Schedule tab**.
2. Click the **Calendar** icon in the **Schedule** column for the unit test suite you want to run.
3. In the **Pattern** section in the **Schedule Unit Test Suite** window, specify how to run this unit test suite. When the run is complete, the system displays the results in the **Dashboard** tab. When you select **To run immediately**, the system runs the test suite in the foreground; for all other options, the system runs the test in the background.
4. For scheduled runs, you can specify additional options.
    a. Select to run the unit test suite by using a different operator ID. In the **Advanced Settings** section, enter the Operator ID in the **Override Default and Run Suite As** field. The system runs the unit test suite by using the Rulesets and access rights associated with that operator. If the operator ID form has multiple Access Groups, the default Access Group is used.
    b. Send the completion email to multiple email addresses. Use the **Send Completion Email to** field to specify the email addresses.

If you do not want any emails sent, clear the **Send Completion Email** field.

5. Click **OK**.

By default, the Pega-AutoTest agents run scheduled unit test suites run every five minutes. When the suite is finished, the agent activity sends an email with the results. By default, this email is sent to the operator who requested the unit test suite run and to any email addresses listed in the **Send Completion Email** array. If no email addresses appear in that field, no email message is sent.

- **Creating test Cases with AUT**
- **Creating unit test suites with AUT**

- Contents form in AUT test suite
- Managing unit tests and test suites

Creating test Cases with AUT

You can automate testing of Rules by creating test Cases for automated unit testing. Automated unit testing validates application data by comparing expected output to the actual output that is returned by running Rules.

To create test Cases, you must have a Ruleset that can store test Cases. For more information, see Creating a test Ruleset to store test Cases.

> **NOTE:** AUT is deprecated. Use PegaUnit testing instead to create automated test Rules.

Automated unit testing information is available on the Testing Applications landing page on Pega Community and in the automated unit testing topics in the help.

- **AUT test Cases**

- Creating Rules
- Copying a Rule or data instance
- Creating a Rule specialized by circumstance

AUT test Cases

Create test Cases for automated unit testing to validate application data by comparing expected output to the actual output that is returned by running Rules.

You can create automated unit testing test Cases for the following Rule Types:

- Activity
- Decision table
- Decision tree
- Flow
- Service SOAP

NOTE: AUT is deprecated. Use PegaUnit testing instead to create automated test Rules.

Automated unit testing information is available on the Testing Applications landing page on Pega Community and in the automated unit testing topics in the help.

- Creating test Cases with AUT

Creating unit test suites with AUT

Unit Test Suites identify a collection of Test Cases and their Rulesets, and a user (Operator ID) whose credentials are used to run the Unit Test Suite. Unit Test Suites are used to automatically run groups of test Cases together and make unit testing more efficient.

The **Unit Test Suite** rule form consists of the Contents tab.

NOTE: AUT is deprecated. Use PegaUnit testing instead to create automated test Rules.

Automated unit testing information is available on the **Testing Applications** landing page on Pega Community and in the automated unit testing topics in the help.

You must have the *AutomatedTesting* privilege to work with unit test suite Rules.

You can create a unit test suite that includes all the test Cases for a specific Rule Type, or you can select individual Rules and specify the sequence in which to run them.

To run a unit test suite, use the **Schedule** gadget on the Automated Unit Testing landing page. For more information, see Working with the deprecated AUT tool. From that gadget, you can choose to run the unit test suite immediately or schedule the run for a future time.

For unit test suites that are scheduled to run at future times, an agent activity in the *Pega-AutoTest* agents rule checks for unit test suite requests every five minutes and runs those that are due. When the agent activity finishes running a unit test suite, it sends an email message with the results. By default, this completion email message is sent to the person who scheduled the unit test suite run, and to any additional email addresses specified at the time the run is scheduled. If no email addresses are specified at the time the run was scheduled, no email message is sent.

Access

For more information, see Working with the deprecated AUT tool to work with the Unit Test Suites that are available to you. You can:

- Use the **Automated Unit Tests** gadget to see the Unit Test Suites and the test Cases in each.
- Use the Create Suite button on the **Schedule** gadget to create unit test suites.
- Use the calendar button on the **Schedule** gadget to run unit test suites immediately and to schedule unit test suite runs.

Category

Unit Test Suites are instances of the *Rule-AutoTest-Suite* class. They belong to the SysAdmin category.

- **Create or Save as form in AUT test suite**
- **Contents form in AUT test suite**

Create or Save as form in AUT test suite

Unit Test Suites – Completing the Create or Save As form

To create a unit test suite rule, use the **Create Suite** button on the *Schedule* gadget of the Automated Unit Testing landing page. To open the Automated Unit Testing landing page, select Dev Studio > *Application* > *Automated Unit Testing*.

You must have the *AutomatedTesting* privilege to be able to create unit test suites. For information about how to enable this privilege, see About Automated Unit Testing.

A unit test suite rule has a single key part, the unit test suite name:

Field	Description
Name	Enter a short, descriptive name for the unit test suite.

Create a separate RuleSet to hold test Cases and unit test suites, rather than using a RuleSet that will be moved to a production system. For more information, consult the articles in the *Testing Applications* category of Pega Community.

For general information about the Create and Save As forms, see:

- Creating Rules.
- Copying a Rule or data instance .

> Rule resolution

As with most Rules, when you search for a Unit Test Suite, the system shows you only those Rules that belong to a RuleSet and version that you have access to.

Unit Test Suite Rules cannot be qualified by circumstance or time.

Contents form in AUT test suite

Use the **Contents** tab to define the unit test suite. Specify a user (Operator ID) that the *Pega-AutoTest* agents are to use by default when running the suite, and select the test Cases to include.

The Operator ID specified here is the default one used to run the unit test suite. When defining the unit test suite's run schedule using the *Schedule* gadget of the Automated Unit Testing landing page, you have the option to specify a different Operator ID and override the one specified here.

You can specify Test Cases in both the **Rule Types To Include** section and the **Query Test Cases To Include** section of this form. If you specify Test Cases in both sections, when the unit test suite runs, those test Cases defined in the **Rule Types To Include** section will run before the test Cases in the **Query Test Cases To Include** section.

1. In the **RuleSets for Test Cases** field, select the RuleSet that holds the test Cases you want to include in this test suite.
   If the test Cases are in more than one RuleSet, click the **Add** icon to add rows to specify the additional RuleSets.
2. In the **User ID for Agent Processing** field, select the Operator ID for the *Pega-AutoTest* agents to use by default when they run this test suite.
   This ID must provide access to the RuleSet that this test suite belongs to, as well as access to the RuleSets listed in the RuleSets field.
3. **Optional:** To specify that the work items created during the test Case execution are to be deleted afterwards, select the **Remove Test Work Objects?** check box.

The fields in the **Application Test Cases To Include** section provide options to specify the test Cases by application name and version.

4. In the Application Name field, select the name of the application that has the test Cases you want to include in the unit test suite.
5. In the Application Version field, select the version of the application that has the test Cases you want to include in the unit test suite.

The fields in the **Rule Types To Include** section provide options to select the test Cases by Rule Type. You can specify that all the test Cases for a particular Rule Type are included in this unit test suite, or you can constrain the list with a When Condition rule.

6. In the Rule Type field, select those Rule Types for which you want to include their test Cases in this unit test suite:
   - Activities
   - Decision Tables
   - Decision Trees
   - Flows
   - Service SOAP service records
7. In the When Filter field, do one of the following:
   - Leave blank to include all the test Cases that were created for Rules of the type specified in the Rule Type field.
   - Select the appropriate when Condition Rule to constrain the list.

     The test Cases that meet the Conditions in the when Condition Rule are included in the unit test suite

The fields in the **Query Test Cases To Include** section provide options to select specific Test Cases to include in this unit test suite. List the test Cases in the order in which you want them to be run.

8. In the Test Case Name field, enter a search string for the test Case you want to find.
9. To list test Cases that match the query string in the Test Case Name field, click Query.

The list is not limited by RuleSet. If test Cases exist that match the search string, the **List Test Case** window appears. Select the test Cases you want to include and then click OK. The test Cases are added to the list in this section of the form.

10. In the Test Case Key field, enter the three-part key of a Test Case rule.

    The key consists of the following parts:

    - Class Name
    - Instance Name (Ins Name)
    - Purpose

    When you use the Query button to find and add a test Case, the system automatically fills in this field.

11. In the Description field, enter the short description of the Test Case.

    When you use the Query button to find and add a test Case, the system automatically fills in this field.

12. In the RuleSet field, enter the RuleSet of the test Case.

    When you use the Query button to find and add a test Case, the system automatically fills in this field.

    Verify that this RuleSet is included in the RuleSets for Test Cases list at the top of this form. If the RuleSet for the test Case is not in that list, add it now. Otherwise, the Test Case does not run when the unit test suite runs.

Running scenario tests against a user interface

Run scenario tests against a user interface to verify that the end-to-end scenarios are functioning correctly. The UI-based scenario testing tool allows you to focus on creating functional and useful tests, rather than writing complex code.

You can test either a specific Case Type or an entire Portal by clicking Scenario Testing in the run-time toolbar to open the test recorder. When you use the test recorder and hover over a testable element, an orange highlight indicates that the element can be tested. Interactions are recorded in a visual series of steps and the execution of a test step can include a delay.

Provide data to your test Cases with a predefined Data Page. This Data Page can provide unique values for each execution of the test Case. You can populate the Data Page by using any source, including activities or Data Transforms.

Tests are saved in a test Ruleset. After they are saved, tests are available on the Scenario Testing landing page. From the landing page you can run a test or view the results of a previous test run.

> **NOTE:** Scenario testing using the Automation Toolkit is not supported with Constellation UI – The recommendation is to use the Selenium starter kit for testing your Pega application.

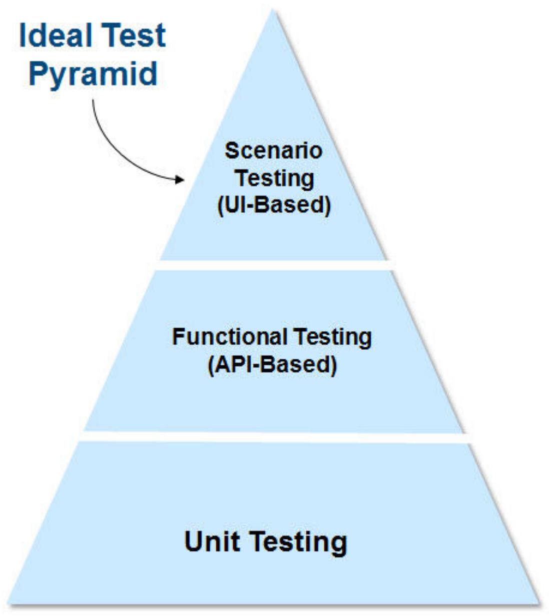
Testing applications in the DevOps pipeline

Having an effective Automation test suite for your application in your continuous delivery DevOps pipeline ensures that the features and changes that you deliver to your customers are of high-quality and do not introduce regressions.

At a high level, the recommended test Automation strategy for testing your Pega applications is as follows:

- Create your Automation test suite based on industry best practices for test automation
- Build up your Automation test suite by using Pega Platform capabilities and industry test solutions
- Run the right set of tests at different Stages of your delivery pipeline
- Test early and test often

Industry best practices for test Automation can be graphically shown as a test pyramid. Test types at the bottom of the pyramid are the least expensive to run, easiest to maintain, take the least amount of time to run, and should represent the greatest number of tests in the test suite. Test types at the top of the pyramid are the most expensive to run, hardest to maintain, take the most time to run, and should represent the least number of tests in the test suite. The higher up the pyramid you go, the higher the overall cost and the lower the benefits.



Ideal test pyramid

- **Scenario testing on Pega Platform**
- **Scenario testing as an add-on component**

Scenario testing on Pega Platform

Run scenario tests against a user interface to verify that the end-to-end scenarios are functioning correctly. You can record a set of interactions for a Case Type or Portal in your scenario tests. You can also group related scenario tests into test suites to run multiple scenario test Cases in a specified order.

In versions of Pega Platform earlier than 8.5, scenario testing is provided as a native Pega Platform and is released as an independent component.

NOTE: Scenario testing using the Automation Toolkit is not supported with Constellation UI – The recommendation is to use the Selenium starter kit for testing your Pega application.

Application: Scenario testing landing page

The scenario testing landing page provides a graphical test creation tool that you can use to increase test coverage without writing complex code.

On the Scenario Testing landing page, you can view and run scenario test Cases. By viewing reports, you can also identify Case Types and Portals that did not pass scenario testing.

On the Scenario Testing landing page, you can perform the following tasks:

- View test execution and coverage information for Case Type tests and Portal tests.
- Open a test Case Rule where you can add assertions to your test.
- View the results of the most recent test run.
- Select and run individual test Cases, or group tests into test suites that you can use to run multiple tests in a specified order.
- Select and run role-specific tests that apply to access roles that your operator includes.
- Download a list of tests, their type, the name of a Portal or Case that is tested, and the time and the result of the last run.
- View the history of your test runs so that you can quickly analyze the history of all your test runs.

- View which scenario tests were automatically rerun if they failed.

- Creating scenario tests

Creating scenario tests

Verify and improve the quality of your application by recording a set of interactions for a Case type or Portal in scenario tests. As a result, you ensure that your application runs correctly and that the end-user experience that you deliver is error-free and works as expected.

Before you begin:

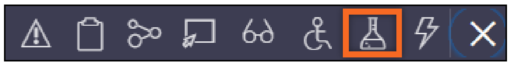
- Create a test Ruleset in which to store the scenario test. For more information, see Creating a test Ruleset to store test Cases.
- Add the *pxScenarioTestAutomation* privilege to your access role. For more information, see Granting privileges to an Access Role.
- Set the *pzPegaSUT* dynamic system setting to `true`. For more information, see Configuring dynamic system settings.

1. Launch the Portal in which you want to do the test.

   For example:

   In the header of Dev Studio, click **Launch Portal**, and then select the Portal in which you want to create a scenario test. Scenario testing works best with Portals that you build on UI Kit.

2. In the footer of the Portal, click **Toggle runtime toolbar**, and then click **Toggle Automation Recorder**, as shown in the following figure:

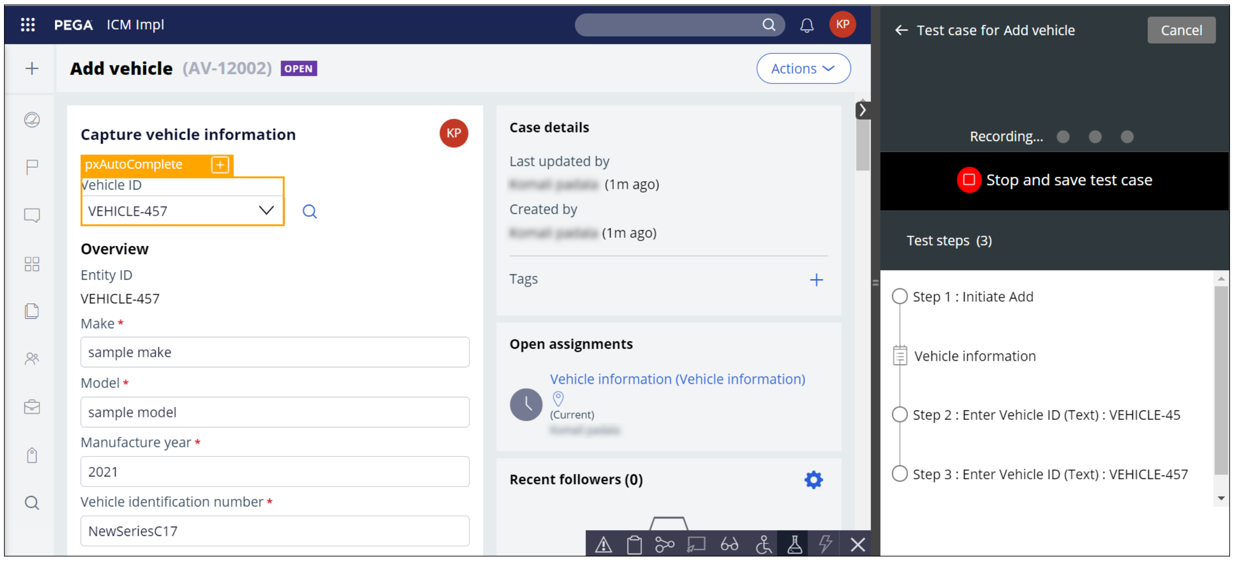


Toggle Automation Recorder icon

3. In the **Scenario tests** pane, click Create test Case, and then select the test type:
   - To record a test for a Portal, select Portal.
   - To record a test for a Case, select Case Type, and then select the type of Case for which you want to record the test.

   > **NOTE:** When you select the Case Type, the system creates a new Case of that type.

4. Record the Steps for the test by clicking the user interface elements.
   When you hover over a testable element, an orange highlight box appears. When you click an element, you record an implicit assertion and add the interaction to the list of test Steps, as shown in the following figure:



Scenario testing

5. **Optional:** To provide specific UI attributes, add an explicit assertion to the test:
   a. Hover over an element.
   b. Click the Mark for assertion icon on the orange highlight box.
   c. In the **Expected results** section, click Add assertion.
   d. Define the assertion by completing the Name, Comparator, and Value fields.

    e. Click Save Step.
6. To add latency to a web browser and actions on a web page to prevent tests from failing during complex processing or slow UI rendering, delay the processing of a Step:
    a. Click the Mark for assertion icon on the orange highlight box.
    b. In the Wait field, enter the number of milliseconds by which to delay the processing of the Step.
    c. Click Save Step.

> **TIP:** You can add delays to all Steps in all scenario tests in your application at once. For more information, see Delaying scenario test execution.

> **NOTE:** Delaying the processing of a single Step takes precedence over delaying all of the Steps at the application level.

7. When you finish adding Steps, in the **Test Case** pane, click Stop and save Test Case.
8. On the **New Test Case** form, save the test:
    a. In the Name field, enter a name for the test.
    b. **Optional:** To provide additional information about the test, in the Description field, enter extra information.
    c. In the **Context** section, select a branch or Ruleset in which you want to save the test.
    d. In the Apply to field, enter the name of a class that is relevant to the test.
    e. Click Save.

> **Result:**
>
> The test Case appears on the Scenario testing landing page.

What to do next:

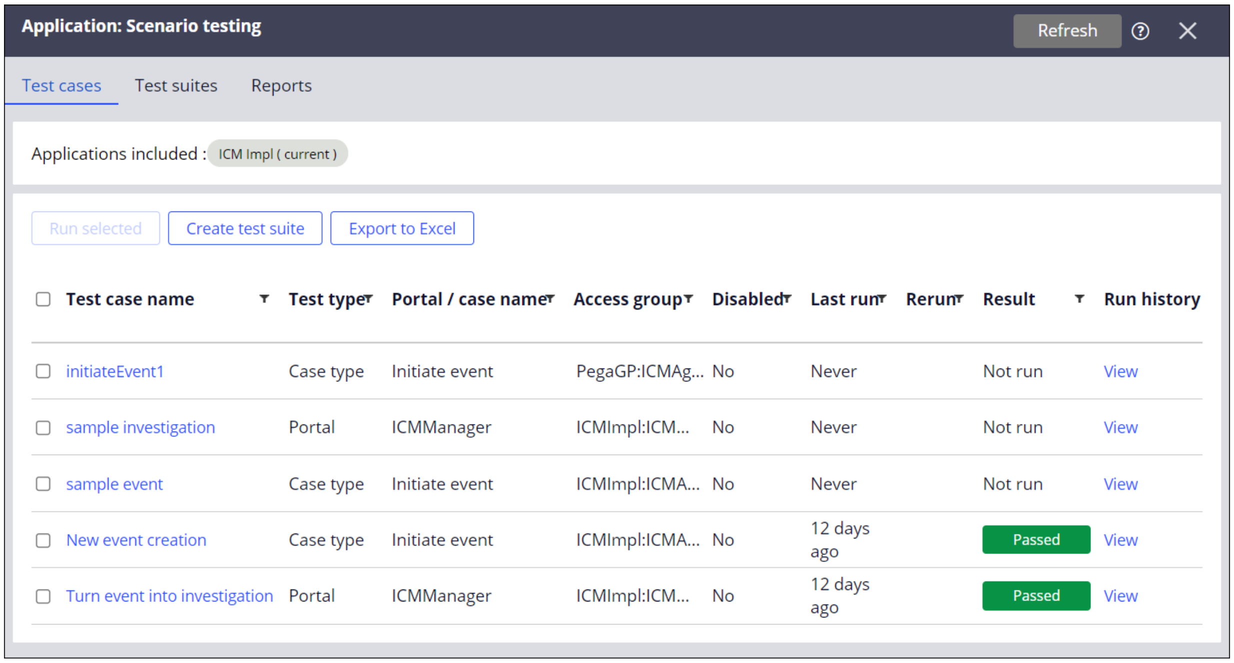
To record interactions between your application and a user of a specific role, for example a manager, create a role-specific scenario test. For more information, see Creating role-specific scenario tests.

- Application: Scenario testing landing page

Opening a scenario test Case

To conveniently test the UI of different Portals and Cases in your application, you can view a list of the scenario test Cases for your application and select the test that you want to open.

1. In the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Automated Testing** > **Scenario Testing** > **Test Cases**.
2. In the **Test Case name** column, click the test Case that you want to open.
   The following figure shows a sample list of scenario Cases for different access groups. You can use an Access Group as a filter to open and run test Cases.



Test case name	Test type	Portal / case name	Access group	Disabled	Last run	Rerun	Result	Run history
initiateEvent1	Case type	Initiate event	PegaGP:ICMAg...	No	Never		Not run	View
sample investigation	Portal	ICMManager	ICMImpl:ICM...	No	Never		Not run	View
sample event	Case type	Initiate event	ICMImpl:ICMA...	No	Never		Not run	View
New event creation	Case type	Initiate event	ICMImpl:ICMA...	No	12 days ago		Passed	View
Turn event into investigation	Portal	ICMManager	ICMImpl:ICM...	No	12 days ago		Passed	View

Scenario Cases

What to do next:

You can run scenario tests that are specific to an access group that your operator includes. After opening a scenario test, you can also define if the test is role-specific, and then provide a relevant Access Group. For more information, see Creating role-specific scenario tests.

Editing scenario tests

Ensure that the test covers your current Portal or Case type scenario by updating an existing scenario test when the user interface or process flow changes. You can save time and effort by keeping existing tests functional instead of creating new ones.

Before you begin:

Create a scenario test Case for a Portal or Case Type. For more information, see Creating scenario tests

1. Launch the Portal in which you want to do the test
2. Do one of the following to open the Automation Recorder:
   - In App Studio, on the lower-left side of the screen, click the Test icon.
   - In Dev Studio, on the lower-right side of the screen, toggle the run-time toolbar, and then click the Toggle Automation Recorder icon.
3. In the **Scenario tests** pane, click the name of the test that you want to edit, and then click Edit.
4. Update the test sequence by clicking the More icon next to a Step, and then selecting an action:
   - To remove a Step from the test Case, click Remove Step.
   - To add a Step to the test Case, click Add Steps.

The test runs from the start and stops at the selected Step so that you can add Steps to the specific part of the test sequence.

- To record the test Case again from a specific Step, click Re-record from here.

  All Steps after the selected Step are removed. The test runs from the start and stops at the selected Step so that you can add Steps to the end of the test sequence.

5. **Optional:** To modify a Step, click the Edit icon next to a Step, modify the assertions and other properties of the Step, and then click Save Step.
6. Click Save.

What to do next:

Your scenario test Case now matches the current user interface and process flow. Run the test to check the quality of your current Portal or Case Type scenario.

Grouping scenario tests into suites

Group related scenario tests into test suites to run multiple scenario test Cases in a specified order. You can then run the scenario test suites as part of purpose-specific tests, such as smoke tests, regression tests, or outcome-based tests. Additionally, you can disable or quarantine individual scenario tests for an application so that they are not executed when the test suite runs.

- **Creating scenario test suites**
- **Running scenario test suites**
- **Viewing scenario test suite results**

- Creating scenario tests
- Application: Scenario testing landing page

Running scenario test suites

Run scenario test suites to check application functionality. You can check the run history, add or remove test Cases from the suite, or reorder the test Cases before running the suite.

1. In the header of Dev Studio, click Configure > Application > Quality > Automated Testing > Scenario Testing > Test Suites.
2. **Optional:** View or modify the test Cases included in a test suite.
   a. Click the name of the suite.
   b. View summary information about previous test results in the header.
      - To view more information about the latest test results, click View details.
      - To view information about earlier results, click View previous runs.
   c. Modify the test Cases in the suite from the Scenario test Cases section.
      - To remove test Cases from the suite, click Remove, and then click Save.
      - To include additional test Cases in the suite, click Add, select the test Case, click Add, and then click Save.
   d. To change the order in which the test Cases will run, drag a Case to a different position in the sequence and then click Save.
   e. To prevent individual test Cases from running as part of the suite, select the Case, click Disable, click Save, and then close the test Case.
   f. Close the test suite, return to the **Application: Scenario** testing page, and then click Actions > Refresh.
3. **Optional:** To view details about previous test results, click View in the Run history column.
4. Select the check box for each test suite that you want to run and then click Run selected. The test suites run and the Result column is updated with the result, which you can click to open test results.

- Grouping scenario tests into suites
- Running scenario test suites

- Viewing scenario test suite results
- Creating scenario tests
- Application: Scenario testing landing page

Viewing scenario test suite results

After you run a scenario test suite, you can view the test results. For example, you can view the expected and actual output for assertions that did not pass.

1. In the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Automated Testing** > **Scenario Testing** > **Test Suites**.
2. To view results of the most recent run, click the result in the **Result** column. For information about why a test failed, click **Failed** in the **Result** column.
3. To view historical details about a specific test suite, in the **Run history** column, click **View**.

- Grouping scenario tests into suites
- Running scenario test suites
- Creating scenario tests
- Application: Scenario testing landing page

Viewing scenario test results

After you run a scenario test, you can view the test results. For example, you can view the expected and actual output for assertions that did not pass.

1. In the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Automated Testing** > **Scenario Testing** > **Test Cases**.
2. To view results of the most recent run, click the result in the **Result** column.
3. To view historical details about a specific test, in the **Run history** column, click **View**.

- Creating scenario tests
- Opening a scenario test Case

- Application: Scenario testing landing page

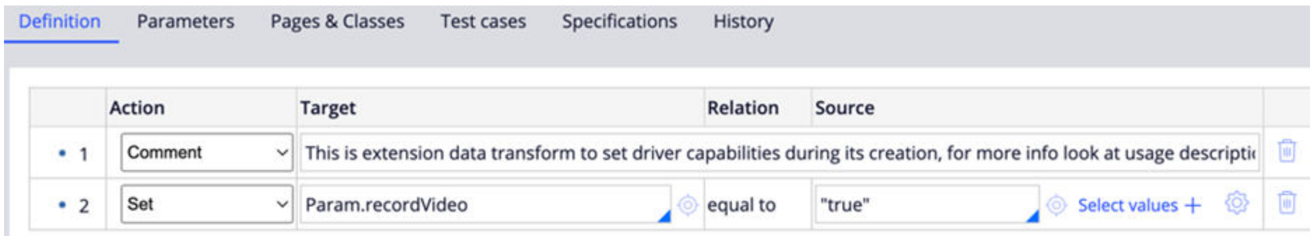
Other features

- **Extension parameters to run scenario tests**
- **Delaying scenario test execution**
- **Automatically rerunning failed scenario tests**
- **Grouping scenario tests into suites**

Extension parameters to run scenario tests

Create a Data Transform to perform testing actions, such as recording a video, or any other configurations that you want to use as the default action. Use the Data Transform when you run automated tests with supported providers such as Cross Browser Testing (CBT), Browser Stack, Sauce-Labs, and Standalone Selenium Grid.

Use the Data Transform *SetScenarioTestExecParams* in the current application Ruleset in the system under test (SUT) to set the required capabilities and values. For example, you can record a video of a test execution while running the automated tests using CBT. The following figure shows the configuration to set the record video action:

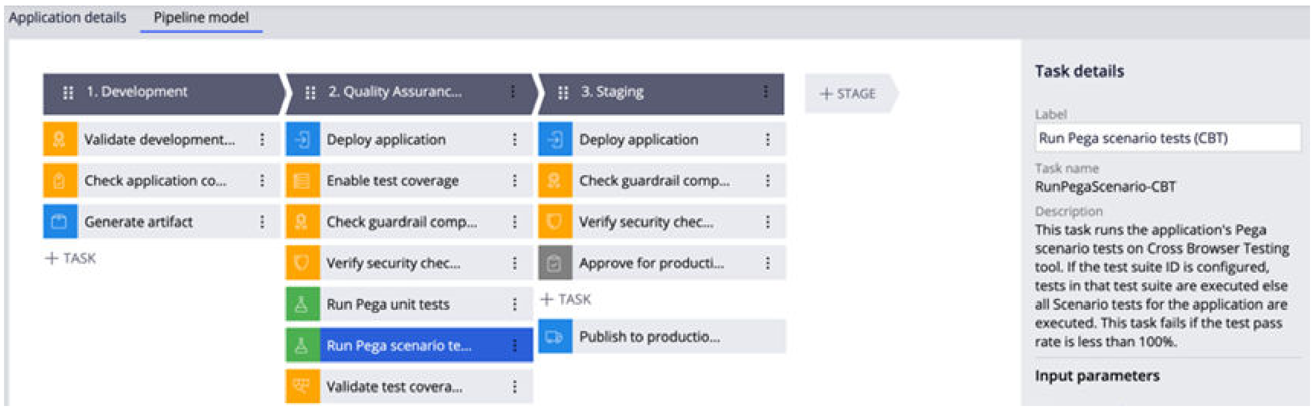


	Action	Target	Relation	Source
1	Comment	This is extension data transform to set driver capabilities during its creation, for more info look at usage descripti		
2	Set	Param.recordVideo	equal to	"true"

Record video action

You can integrate the API into the continuous integration and delivery (CI/CD) process either in a Jenkins job or by configuring this task in Deployment Manager.

For example, the following figure shows a `Run Pega scenario tests (CBT)` task in the QA Stage of Deployment Manager. In this example, when you run the task, the system records a video in CBT.



Task details

Label
Run Pega scenario tests (CBT)

Task name
RunPegaScenario-CBT

Description
This task runs the application's Pega scenario tests on Cross Browser Testing tool. If the test suite ID is configured, tests in that test suite are executed else all Scenario tests for the application are executed. This task fails if the test pass rate is less than 100%.

Input parameters

Deployment Manager scenario test in the QA Stage

Delaying scenario test execution

You can delay the execution of a test to add latency to a web browser and actions on a web page. This prevents tests from failing when a dynamic web page does not load all page elements at once, but the test finds the page elements that render immediately.

Use this procedure to delay the execution of all the steps in all the scenario tests in an application. To delay the execution of a specific step, configure the time of delay on each step when you create or edit a scenario test.

NOTE: Delaying the execution of a single step takes precedence over delaying all the steps at the application level.

1. In the header of Dev Studio, click Configure > Application > Quality > Settings.
2. On the **Application: Quality Settings** page, in the **Scenario test Case execution** section, select the Configure delay for scenario test execution check box.
3. In the Milliseconds field, enter the number of milliseconds by which to delay the execution of all the steps in all your scenario tests.

4. Click Save.

Related concepts

- Changing application quality metrics settings

Related tasks

- Creating scenario tests

Automatically rerunning failed scenario tests

You can automatically rerun scenario test Cases after they fail. Test Cases can sometimes fail if there are temporary stability issues on your environment, or if there are first-time First Use Assembly (FUA) issues that could cause the application UI to be rendered slowly.

To automatically rerun scenario tests that fail, do the following steps:

1. In the header of Dev Studio, click Configure > Application > Quality > Settings.
2. On the **Application: Quality Settings** page, in the **Scenario test Case execution** Section, select the Re-run tests automatically on failure check box.
3. In the Number of runs list, select the maximum number of times to rerun failed scenario test Cases.
4. Click Save.

Result:

The Application: Scenario testing landing page indicates whether a scenario test has been rerun.

Related concepts

- Changing application quality metrics settings

Grouping scenario tests into suites

Group related scenario tests into test suites to run multiple scenario test Cases in a specified order. You can then run the scenario test suites as part of purpose-specific tests, such as smoke tests, regression tests, or outcome-based tests. Additionally, you can disable or quarantine individual scenario tests for an application so that they are not executed when the test suite runs.

- Creating scenario tests
- Application: Scenario testing landing page

Scenario testing as an add-on component

As of Pega Platform version 8.5 and newer, the scenario testing feature is being decoupled from the native Pega Platform and is released as an independent component. See Release notes for release-specific enhancements and updates to the functionality.

- **Installing the add-on component**
- **Release notes**

Installing the add-on component

For on-premises environments, you can download the add-on component from the Pega Marketplace, here.

To install the scenario testing add-on on-premises, do the following steps:

1. On each system, browse to the Pega Marketplace page, and then download the **PegaTestAutomationKit_1.5.zip** file for your version of the add-on component.
2. Use the Import wizard to import the **PegaTestAutomationKit_1.5.zip** into the appropriate systems.
3. After the import, add major version (1) of the **PegaTestAutomationKit** as a built-on application in the target application app stack.

Release notes

Release notes contain important information that you should know before you install a release of the scenario testing component. These notes describe the changes that are included in each component release. Before you install the component, familiarize yourself with the changes, new features, and resolved issues that are listed in respective release below.

- **Pega Test Automation Kit 1.6.2**
- **Pega Test Automation Kit 1.6**
- **Pega Test Automation Kit 1.5**
- **Pega Test Automation Kit 1.4**
- **Pega Test Automation Kit 1.3**
- **Pega Test Automation Kit 1.2**
- **Pega Test Automation Kit 1.1**

Pega Test Automation Kit 1.6.2

The Pega Test Automation Kit 1.6.2 release includes the following new features and improvements.

Enhancements

The following enhancement is now available:

Configure extension parameters to run scenario tests

Default properties at the providers end while executing scenario tests. For example, you can use this to record a video of run with BrowserStack, or provide additional launch information like the region of deployment. You can achieve this by extending the **SetScenarioTestExecParams** Data Transform on the system where you run scenario tests. For more information, see Extension parameters to run scenario tests.

Improvements

The following issues were improved in this release:

- Multi-select control Actions are now captured by scenario test.
- Addressed BAC issue for users on version 8.7 and later who were accessing user Portals by default.

Pega Test Automation Kit 1.6

The Pega Test Automation Kit 1.6 release includes several new features.

Enhancements

The following new features are now available:

Custom steps to identify non-recordable elements

The new release of Pega Test Automation Kit provides you with a convenient way to record any elements that scenario tests do not identify. With this enhancement, you can record any user interface element by providing an Xpath of that element. For example, if the test does not recognize a non out-of-the-box element, you can use the element's Xpath to include that element in tests.

For more information about Xpath, visit the W3schools webpage.

For more information about custom steps, see Adding custom steps to scenario tests.

Reuse of data captured from the UI

In your current session, you can now copy values from the user interface of certain tests, and reuse those values in another step of the same test, or in another test. You specify a unique property name to store the captured data, and then refer to the property name in other steps or tests. In this way, you ensure a smooth and automated testing experience.

The following video shows the steps to reuse data from the UI:

For more information, see Capturing values from the user interface.

Adding custom steps to scenario tests

Provide users with an error-free experience by ensuring that scenario tests recognize all elements of your application's interface. With comprehensive scenario tests, you enhance the quality of the application that you deliver.If tests do not identify controls that are not out-of-the-box, you can add a custom step to the scenario to include the missing elements in the tests.

1. In Dev Studio, create a scenario test.

   For more information, see Creating scenario tests.

2. Add custom steps to the scenario:
   a. In the **Test steps** Section on the right, next to a selected step, click More > Add custom step.
   b. In the **Custom Step Modal** dialog box, provide details of the custom step.

      For example:

      In the Xpath of UI element field, provide the Xpath of the element to make it recognizable by the test.For more information about Xpath, visit the W3schools webpage.

   c. Click Submit.
3. In the **Test Case** pane, click Stop and save test Case.
4. On the **New test Case** form, provide details of the test, and then click Save.

Capturing values from the user interface

Enhance the process of scenario testing by reusing properties in the same test, or in another test. You can copy values that users enter in the interface, and reuse those values to ensure a smooth testing experience.

For example, you can capture the names of members of a household, and reuse that data in another testing scenario. You can also capture values while recording a new test Case.

NOTE: The reuse is available for the ongoing session only.

1. Open a scenario test Case in which you want to capture a certain value.
   For more information, see Opening a scenario test Case.
2. Capture the value of a selected Step:
   a. In Dev Studio, in the test Case View, in the **Test Case Steps** Section, click the Step whose value you want to capture.
   b. In the **Actions** Section, select the **Capture value of Step to use at another Step/test Case** checkbox, and then provide a unique property name to store the value of that Step.
3. Reuse the captured value:
   a. In the **Test Case Steps** Section, click the Step in which you want to reuse the captured value.
   b. In the **Actions** Section, in the **Value type** list, select **Captured value**.
   c. In the **Value** field, enter the property name that stores the value that you want to reuse.
4. In the upper-right corner of the rule, click **Check out**, and then click **Save**.

Pega Test Automation Kit 1.5

The Pega Test Automation Kit 1.5 release includes the following new features and resolved issues.

Enhancements

The following new features are available in this release:

- **Extended test coverage reports**
  - From Pega Platform version 8.7 onwards, test coverage reports include metrics for the application Rules that the system invokes when running queue processors and file processors. With these new metrics, you can capture more accurate test coverage statistics for your applications.
- **Role-specific scenario tests**
  - Scenario tests now can be specific to roles in your application, for more comprehensive scenario testing. You can create separate scenario tests that are specific to different roles, such as an HR worker or a manager. You can create and open scenario tests specific to Access Groups that your operator includes. For more information, see Creating role-specific scenario tests.

  Review the video.

Resolved issues

The following issues were resolved in this release:

- **Lack of support for an email ID field**
  - Pega Test Automation Kit now supports testing an email ID field when you create, edit, and run scenario tests in your application.
- **Lack of support for recording attachments during scenario tests**
  - For more comprehensive and reliable testing, scenario testing now supports an Attach content control that you can include in your application.

- **Incorrect refreshing of test Case Status and Step status data**
  - To avoid inaccurate statuses, now when you run a scenario test, test Case Status and Step status refreshes correctly, and displays a correct color that marks the final result.
- **Insufficient data when a scenario test fails**
  - When a scenario test fails due to an issue with Javascript in your application, now the error message includes relevant information so that you can identify and fix the issue.
  - When a scenario test fails due to a UI element that does not load before the test ends, now the error message includes the element ID so that you can easily identify the missing element.

Creating role-specific scenario tests

For more comprehensive and improved testing, you can create role-specific scenario tests that record interactions between your application and a user with a specific role. Then, you can conveniently run scenario tests for all access roles that your operator includes. For example, if you are an administrator with access to access roles of a manager and an HR worker in an HR application, you can create and then run tests that focus on interactions assigned to an administrator, a manager, and an HR worker.

Before you begin:

- Ensure that you have the Access Group for which you want to create a scenario test. For more information, see Defining user contact information and application access.
- Create a test Ruleset in which to store the scenario test. For more information, see Creating a test Ruleset to store test Cases.
- Add the *pxScenarioTestAutomation* privilege to your access role. For more information, see Granting privileges to an Access Role.

- Set the *pzPegaSUT* dynamic system setting to `true`. For more information, see Configuring dynamic system settings.

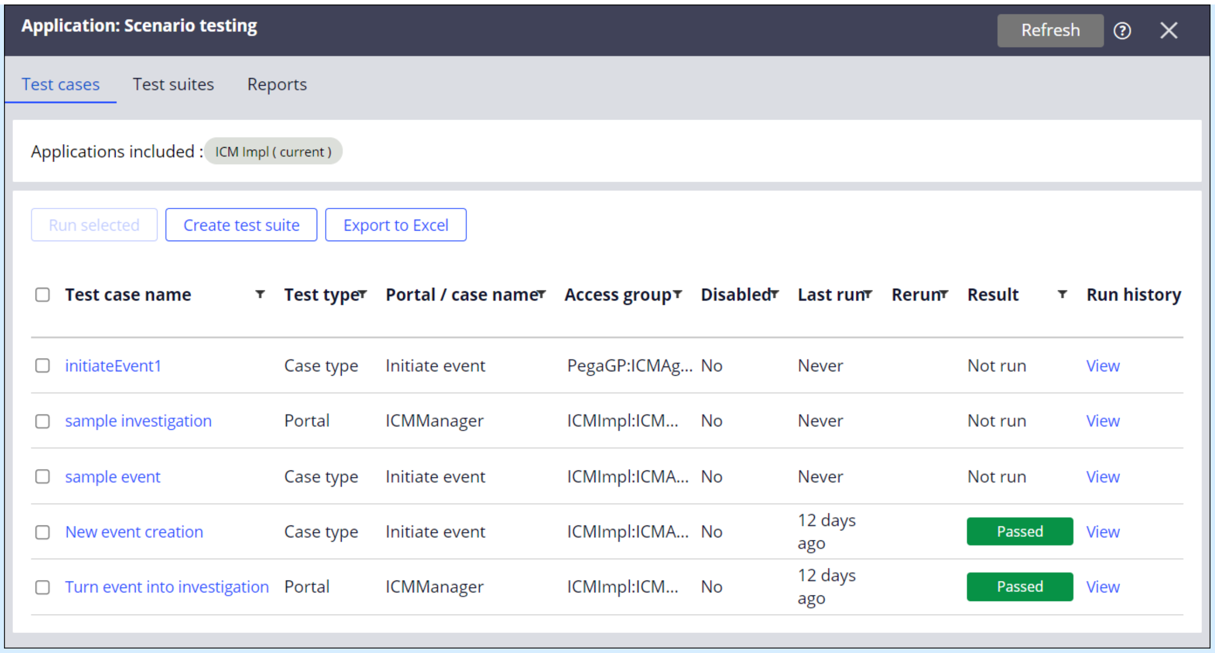
1. Start creating a scenario test.
   For more information, see Creating scenario tests.
2. After you finish recording the scenario test, on the **New Test Case** form, select the Role specific test checkbox.
3. In the AccessGroupInfo field, provide an Access Group that applies to the test.

   For example:

   To indicate that the scenario test applies to the UI that an HR worker views, select MyApp:HRWorker.
4. Click Save.

Result:

The Test Cases tab on the **Application: Scenario testing** landing page includes a field with an Access Group related to the role specific to the test, as shown in the following figure. For greater convenience, you can use an Access Group as a filter to open and run role-specific scenario tests.


Application: Scenario testing

Applications included : ICM Impl (current)

Test case name	Test type	Portal / case name	Access group	Disabled	Last run	Rerun	Result	Run history
initiateEvent1	Case type	Initiate event	PegaGP:ICMAg...	No	Never		Not run	View
sample investigation	Portal	ICMManager	ICMImpl:ICM...	No	Never		Not run	View
sample event	Case type	Initiate event	ICMImpl:ICMA...	No	Never		Not run	View
New event creation	Case type	Initiate event	ICMImpl:ICMA...	No	12 days ago		Passed	View
Turn event into investigation	Portal	ICMManager	ICMImpl:ICM...	No	12 days ago		Passed	View

Test Cases

You can ran multiple test Cases with different Access Groups in a batch. For every role, the context switches and after running all selected tests, the context switches back to an initial Access Group. If you want to run a role-specific test for an access group that your operator does not include, an exception occurs. To handle the exception, add the relevant Access Group to your operator.

What to do next:

Explore scenario tests available in your application. After opening a scenario test, you can also define if the test is role-specific. For more information, see Opening a scenario test Case.

Pega Test Automation Kit 1.4

The Pega Test Automation Kit 1.4 release includes the following new features and resolved issues.

Enhancements

The following new features are available in this release:

- **Support for reusable scenario tests**
    - You can now reuse scenario tests and embed them within other scenario tests. By reusing scenario tests, you do not need to record the same scenario test multiple times. For more information, see Embedding and reusing scenario tests.

Embedding and reusing scenario tests

You can record a scenario test and save it as a reusable component to embed in one or more scenario tests. Reusing scenario tests reduces the effort to create the same scenario test multiple times.

When you run a scenario test with embedded scenario tests, the system runs the embedded scenario tests in the sequence in which the tests are listed. Then, the system runs any other setup steps.
Note the following information about reusing scenario tests:

- The scenario tests that you want to embed has to belong to the same user Portal.
- There is no limit on the number of reusable scenario tests that you can embed in a scenario test.

1. Record and save the scenario test that you want to configure as a reusable test. For more information about recording scenario tests, see Creating scenario tests.
2. In Dev Studio, open the test case. For more information, see Opening a scenario test Case.
3. On the **Definition** tab, at the top of the **Scenario test Case** page, select the **Reusable** check box.

> **NOTE:** If a scenario test is configured to use embedded scenario tests, you cannot mark that scenario test as reusable.

4. Click **Save**.
5. Open the scenario test into which you want to embed the reusable scenario test.
6. Click the **Setup & Cleanup** tab.
7. Expand **Embed and execute other test Cases on start**.
8. Click **Add scenario test Case**.
9. In the **Of class** field, press the Down arrow key, and then select the class to which the scenario test Case belongs.
10. In the **Name of test Case** field, press the Down arrow key, and then select the test Case that you want to reuse.
11. Click **Save**.

> **What to do next:**
>
> Review the video.

Pega Test Automation Kit 1.3

The Pega Test Automation Kit 1.3 release includes the following new features and resolved issues.

Enhancements

The following new features are available in this release:

- **Alert on new version of Pega Test Automation Kit.**
  - The scenario testing landing page now provides an announcement when a new Pega Test Automation Kit is released.

- **Attachment content control support in scenario testing.**
  - Scenario tests now support the attach content control during recording. This control allows users to select documents or images and attach them to their application by either browsing in a local directory for a file or dragging and dropping files on to the attachment window. For more information, see Attachment control support in scenario testing.
- **More operators for string data type in scenario testing.**
  - All string attributes (value, placeholder, label, tooltip, and button caption) now include new operators for comparing values in the validations.

Resolved issues

The following issues were resolved in this release:

- **Not enough information when scenario test execution fails from a pipeline.**
  - Pega Test Automation Kit now returns appropriate failure or exception messages if a scenario test does not execute successfully when invoked from a CI pipeline.

Attachment control support in scenario testing

Scenario tests now support the attach content control, `pxAttachContent`, during recording. This allows you to attach images or files directly into your Case and record it for testing and playback.

1. Launch App Studio or Dev Studio, based on where you want to perform the test.
2. Do one of the following to open the Automation Recorder:
   - In App Studio, on the lower-left side of the screen, click the Test icon.
   - In Dev Studio, on the lower-right side of the screen, toggle the run-time toolbar, and then click the Toggle Automation Recorder icon.
3. In the **Scenario tests** pane, click Create test Case, and then select the test type:
   - To record a test for a Portal, select Portal.

- To record a test for a Case, select **Case type**, and then select the type of Case for which you want to record the test.

> **NOTE:** When you select the Case type, a new Case of that type is created.

4. Record the Steps for the test by clicking the UI elements.

> **Result:**
>
> When you hover over a testable element, an orange highlight box appears. When you click an element, you record an implicit assertion and add the interaction to the list of test Steps.

5. Click on **Upload file** to open the attachment dialogue page.
   a. Navigate to the file you want to upload and select it.
   b. Enter a name for the attachment in the **Enter name** field.
   c. Enter a description for the attachment in the **Enter description field.**
   d. Click **Publish.**
6. Continue recording as many Steps as necessary. When you finish adding Steps, in the **Test Case** pane, click **Stop and save Test Case.**

> **Result:**
>
> The Test Case appears on the **Scenario testing landing page**.

Pega Test Automation Kit 1.2

The Pega Test Automation Kit 1.2 release includes the following new features and resolved issues.

Enhancements

The following new features are available in this release:

- **Anypicker and Multi-select support in scenario testing.**
    - Scenario tests now support the Anypicker and Multi-select controls in inspectable text fields during recording. Anypicker is used to enhance navigation in your scenario test by grouping drop-down list items into expandable categories, and Multi-select allows you to record more than one value for a single field on a form.
- **Setting up and cleaning the context for a scenario test.**
    - A new Setup & Cleanup tab has been added, which you can use to create or fetch test data using one of the setup options available from the drop down menu. You can apply a Data Transform, execute an activity, or load an object that your scenario test Case can create or fetch dynamically.

      A cleanup feature has been added to automatically remove all referenced Data Pages, data objects, and user pages at the end of each test run. See Setting up and cleaning the context for a scenario test for more information.

      Scenario tests now apply dynamic data to text fields for real-time application of information in a UI recording.

- **Standalone selenium grid setup supports 3.x and 4.x versions of Selenium.**
    - Execution of scenario tests from a CI tool using standalone selenium grid now supports Selenium versions 3.x and 4.x.

Anypicker and Multi-select support in scenario testing

Scenario tests now support multiple new controls in text fields to include both Anypicker and Multi-select. Anypicker is used to enhance navigation in your scenario test by grouping drop-down list items into expandable categories, and Multi-select allows you to record more than one value for a single field on a form.

1. Launch App Studio or Dev Studio, based on where you want to perform the test.

2. Do one of the following to open the Automation Recorder:
    - In App Studio, on the lower-left side of the screen, click the Test icon.
    - In Dev Studio, on the lower-right side of the screen, toggle the run-time toolbar, and then click the Toggle Automation Recorder icon.
3. In the **Scenario tests** pane, click Create test Case, and then select the test type:
    - To record a test for a Portal, select Portal.
    - To record a test for a Case, select Case type, and then select the type of Case for which you want to record the test.

    > **NOTE:** When you select the case type, a new Case of that type is created.

4. Record the Steps for the test by clicking the UI elements.

    > **Result:**
    >
    > When you hover over a testable element, an orange highlight box appears. When you click an element, you record an implicit assertion and add the interaction to the list of test Steps.

5. Click on an inspectable text field, denoted by an orange highlight.
    - For Anypicker, use the dropdown menu to select the required input.
    - For Multi-Select, add as many input values as necessary.
6. Continue recording as many Steps as necessary. When you finish adding Steps, in the **Test Case** pane, click Stop and save Test Case.

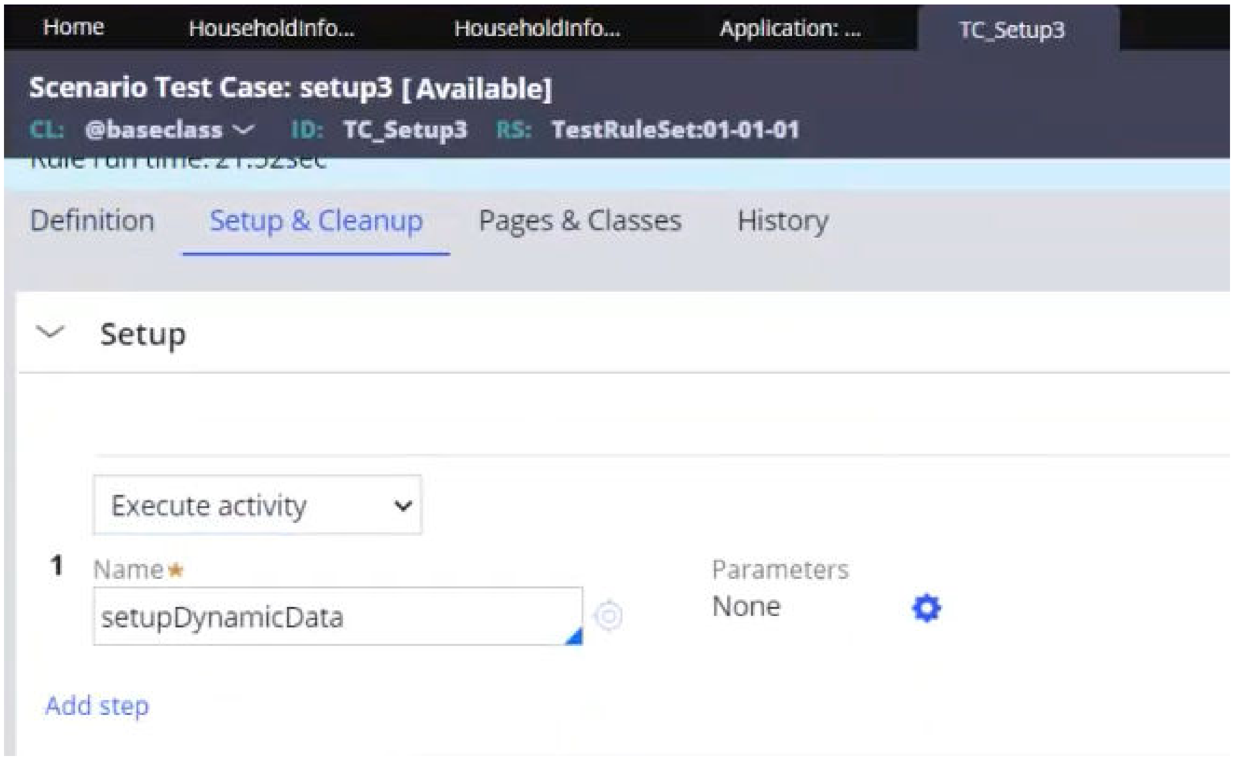
    > **Result:**
    >
    > The test Case appears on the **Scenario testing landing page**.

Setting up and cleaning the context for a scenario test

Scenario tests can now apply dynamic data to text fields for real-time application of information in a UI recording.

After an initial scenario test Case has been recorded, any static text field can now be re-characterized as a dynamic field to apply referenced data using the editable scenario test Case dashboard. A new **Setup & Cleanup** tab has been added to the Scenario Test Case dashboard to set up dynamic data. To perform this action:

1. Record a scenario test using static data. See Creating scenario tests for more information.
2. In the navigation page of Dev Studio click Configure > Application > Quality > Automated Testing > Scenario Testing > Test Cases.
3. Select the test Case for which you want to edit.
4. Navigate to the **Setup & Cleanup** tab of the test Case.
5. In the Setup section of this tab, you can create or fetch test data using one of the setup options available from the drop down menu. You can apply a data transform, execute an activity, or load an object. Scenario tests can fetch and use any of this setup information as dynamic input data.
6. Input a name for the activity.



Scenario Test Case: setup3 [Available]

CL: @baseclass ID: TC_Setup3 RS: TestRuleSet:01-01-01

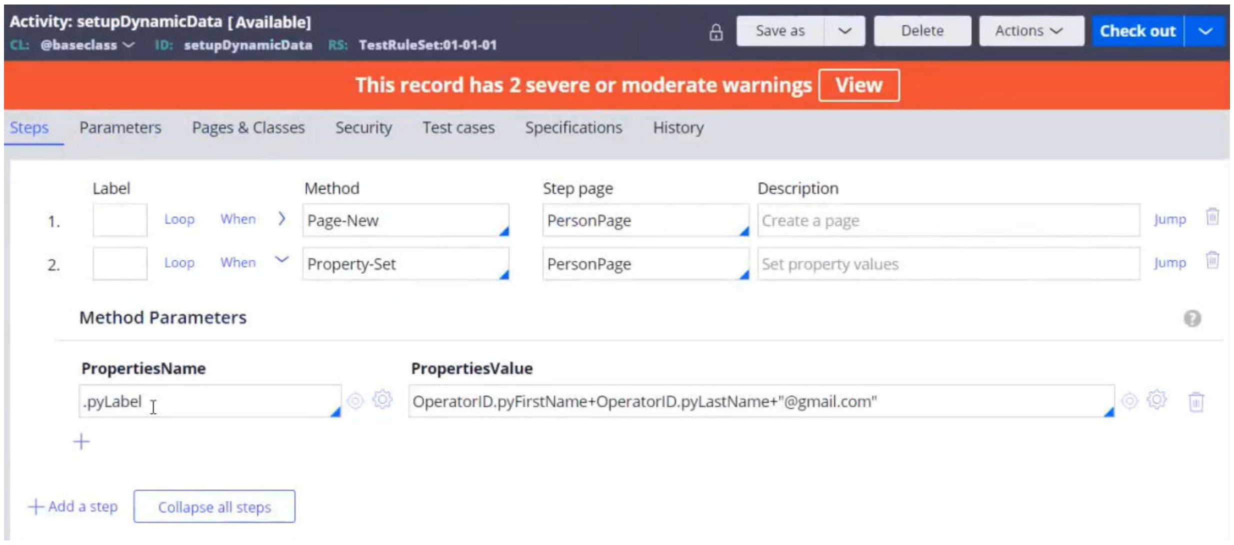
Setup

Execute activity

1 Name* setupDynamicData

Parameters None

7. On the setupDynamicData Page, create the dynamic value and associate it to a property for use in your scenario test Case. For instance, on the **PersonPage** in the example below, the *.pyLabel* property consists of the *OperatorID.pyFirstName+OperatorID.pyLastName"@gmail.com"* dynamic data structure.



Activity: setupDynamicData [Available]

CL: @baseclass ID: setupDynamicData RS: TestRuleSet:01-01-01

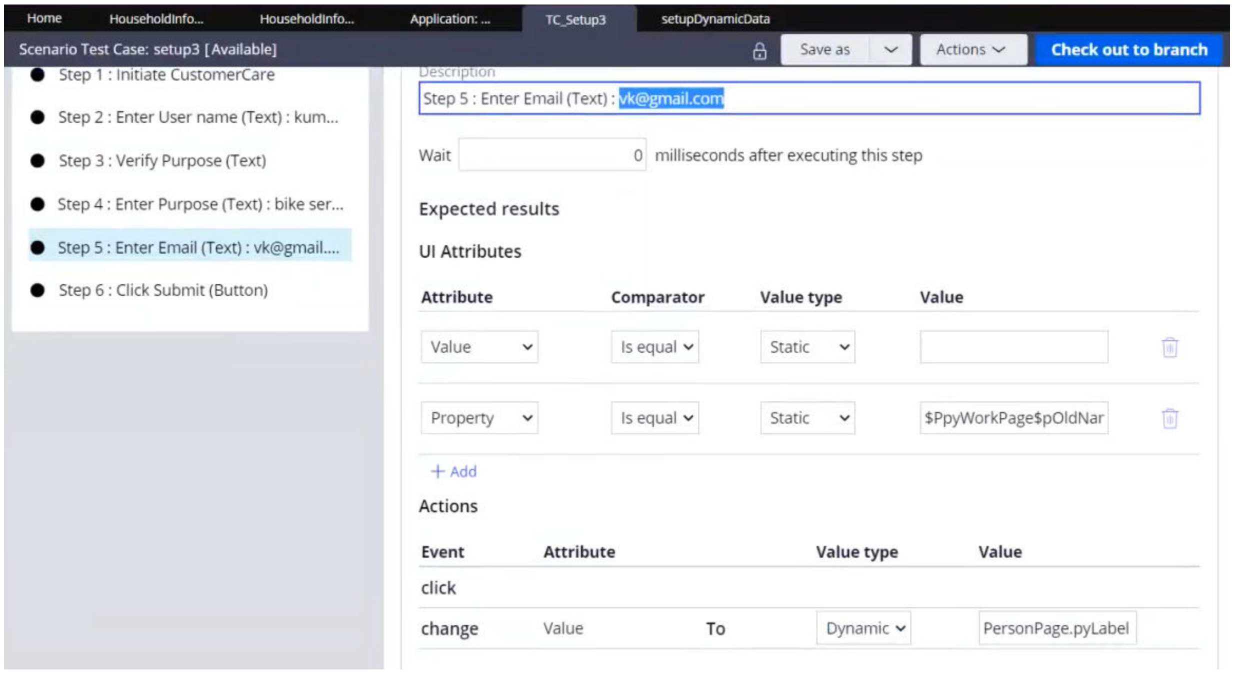
This record has 2 severe or moderate warnings

	Label	Method	Step page	Description
1.		Page-New	PersonPage	Create a page
2.		Property-Set	PersonPage	Set property values

Method Parameters

PropertiesName	PropertiesValue
.pyLabel	OperatorID.pyFirstName+OperatorID.pyLastName+"@gmail.com"

8. To utilize this dynamic field in your test Case, navigate back to your scenario test Case.
9. From the **Test Case Steps** section, select the Step that contains the text field.
10. After selecting the test case Step, navigate to the **Actions** field, and change the value type to **Dynamic**.
11. In the Value field, enter the page.property value created as part of your activity in Step 8. In this case, it is *PersonPage.pyLabel*.


Home
HouseholdInfo...
HouseholdInfo...
Application: ...
TC_Setup3
setupDynamicData
Scenario Test Case: setup3 [Available]
Save as
Actions
Check out to branch
Step 1 : Initiate CustomerCare
Step 2 : Enter User name (Text) : kum...
Step 3 : Verify Purpose (Text)
Step 4 : Enter Purpose (Text) : bike ser...
Step 5 : Enter Email (Text) : vk@gmail....
Step 6 : Click Submit (Button)
+ Add

Description: Step 5 : Enter Email (Text) : vk@gmail.com

Wait 0 milliseconds after executing this step

Expected results

UI Attributes

Attribute	Comparator	Value type	Value
Value	Is equal	Static	
Property	Is equal	Static	$PpyWorkPage$pOldNar

Actions

Event	Attribute		Value type	Value
click				
change	Value	To	Dynamic	PersonPage.pyLabel

12. Click **Save**.
https://players.brightcove.net/1519050010001/obdhcLv77W_default/index.html?videoId=6261768015001

Result:

To automatically remove all referenced Data Pages, data objects, and user pages at the end of each test run, select the checkbox for **Clean up the test data at the end of run**.

- Setting up your test environment

- Cleaning up your test environment

Pega Test Automation Kit 1.1

The Pega Test Automation Kit 1.1 release includes the following new features and resolved issues.

New features

The following new features are available in this release:

- Run scenario tests from rule form action menu in Dev Studio
  - You can now view and run scenario tests directly from Dev Studio on the rule form action menu, eliminating the need to launch the Portal for every instance.
- **Attachment support in scenario testing**
  - Enhance your end-to-end UI testing of run-time scenarios by simulating the user experience of interacting with attachments. When recording scenario tests, you can add and edit files attached to a test Case and view the interaction during playback. For more information, see Attachment support in scenario testing.
- **Scenario testing compliance metrics are available in the Application Quality dashboard.**
  - The metrics tile of the Application Quality dashboard now includes scenario testing compliance metrics and execution rate, and provides a centralized location for all testing-specific application data.

Resolved issues

The following issues were resolved in this release:

- **Unable to view failed scenario tests information directly from the Portal.**
  - If a scenario test fails, you can now click on the **Failed** dialogue in the testing window to view and debug the test results. This dialogue provides the same

test result summary information as in Dev Studio, without navigating out of the Portal.

- **Tracer tool not supported for scenario tests.**
    - All scenario testing functionality now supports the tracer tool for easier debugging.

Attachment support in scenario testing

Enhance your end-to-end UI testing of run-time scenarios by simulating the user experience of interacting with attachments. When recording scenario tests, you can add and edit files attached to a test Case and view the interaction during playback.

To interact with attachments during a scenario test recording, perform the following steps:

NOTE: You can only add attachments in a scenario test if the attachment field is configured with the pzMultiFilePath control.

1. Launch App Studio or Dev Studio, based on where you want to perform the test.
2. Do one of the following to open the Automation Recorder:
    - In App Studio, on the lower-left side of the screen, click the Test icon.
    - In Dev Studio, on the lower-right side of the screen, toggle the run-time toolbar, and then click the Toggle Automation Recorder icon.
3. In the **Scenario tests** pane, click Create Test Case, and then select the test type:
    - To record a test for a Portal, select Portal.
    - To record a test for a Case, select Case type, and then select the type of Case for which you want to record the test.

    NOTE: When you select the Case type, a new Case of that type is created.

4. Record the Steps for the test by clicking the UI elements.

> **Result:**
>
> When you hover over a testable element, an orange highlight box appears. When you click an element, you record an implicit assertion and add the interaction to the list of test Steps.

5. Click **Attachments** to attach a file to the work object. You can also attach files from **Recent attachments**, or anywhere that files can be attached from the UI.
   a. To attach a local file, select **File from device**.
   b. To attach a link, select **URL**.
6. Provide the required information and select **Submit** to attach the file to the work object. Attached files appear on the **Files & documents** pane on the right side.
7. Continue recording as many Steps as necessary. When you finish adding Steps, in the **Test Case** pane, click **Stop and save Test Case**.
8. You can edit attached files by changing the **Attachment key** in the test Case Step. To edit the attachment:
   a. Navigate to the test Case that you want to edit and select **Edit**. See Opening a scenario test Case for more information.
   b. Identify the Step that includes the attachment and select the pencil icon to edit that attachment.
   c. Edit the **Attachment key** to configure the test Case attachment as required.
   d. Click **Save Step**.
   e. Click **Save** to exit the editor and update the test Case.

   https://players.brightcove.net/1519050010001/obdhcLv77W_default/index.html?videoId=6261765810001

> **Result:**
>
> The test Case appears on the **Scenario testing landing page**.

Analyzing application quality metrics

Quickly identify areas within your application that need improvement by viewing metrics related to your application's health on the Application Quality dashboard.

- **Viewing application quality metrics**
- **Changing application quality metrics settings**
- **Estimating test coverage**
- **Viewing test coverage reports**

Viewing application quality metrics

Quickly identify areas within your application that need improvement by viewing metrics related to your application's health on the Application Quality dashboard.

For example, view your application's compliance score and see the number and severity of guardrail violations that were found in your application. You can then improve your application's compliance score and overall quality by investigating and resolving the violations.

To open the Application Quality dashboard, from the Dev Studio header, click Configure > Application > Quality > Dashboard.

You can view the following metrics:

- Rule, Case, and application – View the number of executable Rules (functional Rules that are supported by test coverage) and the number of Case types in the selected applications. To view metrics for a different combination of applications, select a different list on the **Application: Quality Settings** page.
- Guardrail compliance – View the compliance score and the number of guardrail violations for the included applications, as well as a graph of changes to the compliance score over time. To see more details about the application's guardrail compliance, click View details.

- Test coverage – View the percentage and number of Rules that are covered by tests, and the last generation date of the application-level coverage report for the selected applications, as well as a graph of changes to application-level coverage over time. To see test coverage reports or to generate a new coverage report, click View details.

> **NOTE:** If the *EnableBuiltOnAppSelectionForQuality* switch is turned on, then coverage sessions metrics are also displayed on the Application Quality Dashboard for the built-on applications selected in Application: Quality Settings.

- Unit testing – View the percentage and number of Pega unit test Cases that passed for the selected applications, over the period selected on the **Application Quality Settings** landing page. The graph illustrates the changes to the test pass rate over time. To see reports about test compliance and test execution, click View details.
- Case types – View guardrail score, severe guardrail warnings, test coverage, unit test pass rate, and scenario test pass rate for each Case type in the applications. To view additional details about a Case type, click View details.
- Data types – View guardrail score, severe guardrail warnings, test coverage, and unit test pass rate for each data type in the applications. To view additional details about a data type, click View details.
- Other Rules – View guardrail score, test coverage, test pass rate, the number of warnings, a list of Rules with warnings, the number and list of uncovered Rules, and the number and list of failed test Cases for Rules that are used in the selected applications but that are not a part of any Case type.

- **Application quality metrics**

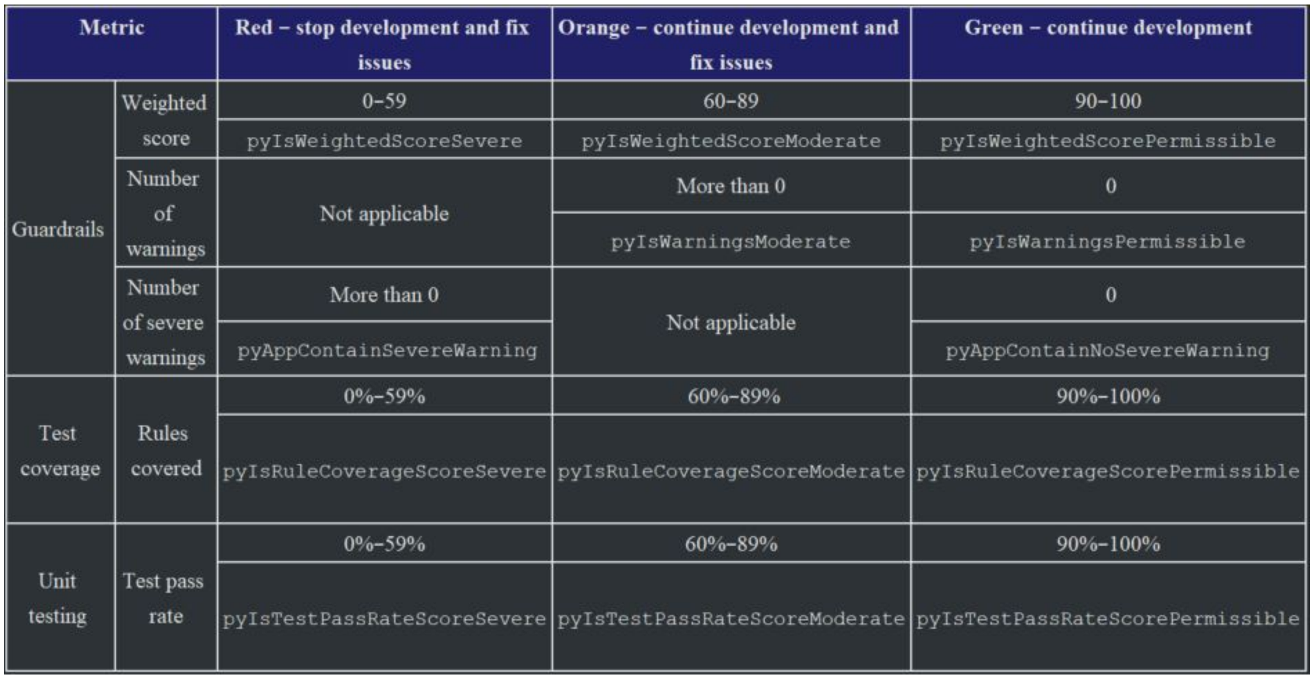
- Changing application quality metrics settings
- Application quality metrics

- Estimating test coverage

Application quality metrics

The Application Quality dashboard displays metrics for guardrails, test coverage, and unit testing that you can use to assess the overall health of your application and identify areas that require improvement. You can change the default ranges for the color codes by modifying the corresponding when Rules in the *Data-Application-Quality* class.

The following table describes the relationship between colors, default ranges, and when Rules. For each metric in the Red, Orange, and Green columns, the top row indicates the default range for each color and the bottom row indicates the corresponding when rule.



Metric		Red – stop development and fix issues	Orange – continue development and fix issues	Green – continue development
Guardrails	Weighted score	0–59	60–89	90–100
		pyIsWeightedScoreSevere	pyIsWeightedScoreModerate	pyIsWeightedScorePermissible
	Number of warnings	Not applicable	More than 0	0
			pyIsWarningsModerate	pyIsWarningsPermissible
	Number of severe warnings	More than 0	Not applicable	0
		pyAppContainSevereWarning		pyAppContainNoSevereWarning
Test coverage	Rules covered	0%–59%	60%–89%	90%–100%
		pyIsRuleCoverageScoreSevere	pyIsRuleCoverageScoreModerate	pyIsRuleCoverageScorePermissible
Unit testing	Test pass rate	0%–59%	60%–89%	90%–100%
		pyIsTestPassRateScoreSevere	pyIsTestPassRateScoreModerate	pyIsTestPassRateScorePermissible

- Viewing application quality metrics

Changing application quality metrics settings

The Application Quality settings provides configurable options related to quality metrics. You can change the default settings for metrics displayed to meet your business needs.

> **NOTE:** To change settings to enable the *EnableBuiltOnAppSelectionForQuality* toggle that allow you to select which built-on applications are included, your operator ID must have the *SysAdm4* privilege.

On the Application Quality settings landing page, you can modify the following settings for application quality:

- **Application(s) included** – If you want the test coverage report to include only Rules from the current application, select **Current application only**. If you want the test coverage report to also include Rules from built-on applications, select **Include built-on applications**. By default, only the current application is selected. If you enable the *EnableBuiltOnAppSelectionForQuality* toggle, you can select which built-on application will be included.

  > **NOTE:** If a master user starts an application-level coverage session for an application, then that user's configuration of this setting is in effect for all users that execute test coverage for the duration of the session.

- **Ignore test Rulesets when calculating Guardrail score** – When you enable this setting, the guardrail score is calculated without taking test Rulesets into account. This is the default behavior. When you disable this setting, test Rulesets are taken into account when calculating the guardrail score.
- **Quality trends** – Use this setting to change the date range of the trend graphs on the Application Quality, Application: Test coverage and Application: Unit testing landing pages. The default value is **Last 2 weeks**.

- **Test Case execution** – Use this setting to change the number of days, from the time that they are executed that tests are treated as executed by the Application Quality dashboard and coverage reports. By default, a test executed later than the last seven days is excluded from the Application Quality dashboard and in reports.
- **Scenario test Case execution** – Use this setting to add a delay (in milliseconds) to the execution of all the Steps in all the scenario tests in an application. For more information, see Delaying scenario test execution.
- **Re-run tests automatically on failure** – Use this setting to automatically rerun scenario tests if they fail, up to a maximum of two times. For more information, see Automatically rerunning failed scenario tests.

- Improving your compliance score
- Viewing application quality metrics
- Estimating test coverage
- Application quality metrics

Estimating test coverage

View historical test coverage metrics and generate reports containing the number of executable Rules and their test coverage. Use the data to analyze changes in test coverage, and to verify which Rules require testing.

On the Test Coverage landing page, view a chart displaying test coverage metrics and generate specific user-level, application-level, and merged coverage reports. User-level reports contain the results of a single test coverage session that a user performs, while application-level reports contain results from multiple test coverage sessions that many users run. Merged reports contain results from multiple most recent application-level reports.

The following Rule Types are included in test coverage reports.

• Activity • Case type	• Declare Expression • Declare trigger	• Paragraph • Queue processors*

• Collection • Correspondence • Data Page • Data Transform • Decision data • Decision table • Decision tree	• File listeners* • Flow • Flow Action • Harness • HTML • HTML fragment • Map value • Navigation	• Report definition • Scorecard • Section • Strategy • Validate • When • XML Stream

NOTE: To include file listeners and queue processors in your test coverage reports, you must have the PegaTestAutomationKit application in your application stack.

NOTE: Ensure you have the appropriate access to generate reports by associating the *pzStartOrStopMasterAppRuleCoverage* privilege with your account. For more information, see Granting privileges to an Access Role.

- **Generating a user-level test coverage report**
- **Generating an application-level test coverage report**
- **Participating in an application-level test coverage session**
- **Generating a merged coverage report**

Generating a user-level test coverage report

Generate a user-level test coverage report to identify which executable Rules in your currently included applications are covered and not covered by tests. The results of this type of report are not visible on the Application Quality Dashboard.

1. In the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Test Coverage**.
2. Click **User level**.

> **NOTE:** If the `Application level coverage is in progress` message is displayed, you cannot start a user-level coverage session.

3. Click **Start new session**.
4. Enter the title of the coverage report, and then click **OK**.
5. To provide data for the report, run all of the tests that are available for your included applications, for example, Pega unit automated tests and manual tests.
6. Click **Stop coverage**, and then click **Yes**.

> **NOTE:** If you close the tab or log out without clicking **Stop**, the report is not generated.

7. Review the results of the coverage session. In the **Coverage history** section, click **Show Report**.
8. **Optional:** To see whether coverage reports were generated by other users, click **Refresh**.
9. **Optional:** To see a list of application-level coverage reports, click **Application level**.

- Participating in an application-level test coverage session

Generating an application-level test coverage report

Generate an application-level coverage report that contains coverage results from multiple users. Use this report to identify which executable Rules in your currently included applications are covered and not covered by tests. The results of this type of report are visible on the Application Quality Dashboard.

1. In the header of Dev Studio, click Configure > Application > Quality > Test Coverage.
2. Click Application level.
3. Click Start new session.

> **NOTE:** To start application-level coverage, your operator ID must have the *pzStartOrStopMasterAppRuleCoverage* privilege.

4. Enter the title of the coverage report, and then click OK.
5. **Optional:** To provide data for the report, run all of the tests that are available for your currently included applications, for example, Pega unit automated tests and manual tests.
6. Inform all relevant users that they can log in to the application and start running tests.
7. Wait until all users have completed their tests and have logged off.

> **NOTE:** If you stop an application coverage session before a user has logged off, the coverage data of this user is not included in the report.

8. Click Stop coverage, and then click Yes.
9. Review the results of coverage session. In the Coverage history section click Show Report.
10. **Optional:** To see whether coverage reports were generated by other users, click Refresh.
11. **Optional:** To see a list of user-level coverage reports, click User level.

Participating in an application-level test coverage session

When an application-level coverage session is running, you can perform tests of the application to contribute to an application-level test coverage report that identifies the executable Rules in your application that are covered and not covered by tests.

Before you begin:

Ensure that application-level coverage is in progress before you log in. If application coverage is started after you log in, you cannot contribute to it unless you log off and log in again. Only users with the *pzStartOrStopMasterAppRuleCoverage* privilege can initiate application-level coverage.

1. Check if application-level coverage is in progress.
   a. In the header of Dev Studio, click **Configure** > **Application** > **Quality** > **Test Coverage**.
   b. Verify that you see the `Application level coverage is in progress` message.
      If you do not see the message, application-level coverage is not active, however, you can still start a user-level test coverage session.
2. To provide data for the report, execute all the tests that are available for the included applications, for example, Pega unit automated tests and manual tests.

   NOTE: During the coverage session your local configuration for included applications is overridden by the configuration of the user that started the application-level coverage session.

3. Click your profile icon and then click **Log off**.

> **NOTE:** If you do not log off before the rule coverage session is stopped, you will not contribute to the report. If you log off and then log in again while the coverage session is still active, your test coverage sessions are saved as a new session that will be included in the application coverage report.

- Generating an application-level test coverage report

Generating a merged coverage report

Generate application-level coverage reports for every application in your system and in your application stack, and then merge the most recent reports to a single report, to gain a consolidated overview of test coverage for all your top-level or built-on applications.

Insight from a merged application report helps you avoid creating duplicate tests for Rules that are used across multiple applications.

> **NOTE:** Because a merged report is an instance of an application-level report, when a merged report is the most recent one for an application, it is included in the next merged report.

> **Before you begin:**
>
> Ensure that your operator ID has the *pzStartOrStopMasterAppRuleCoverage* privilege. Generate at least one application-level coverage report for another application in your system or for a built-on application in your current application. For more information, see Generating an application-level test coverage report.

1. Switch to the main application that you want to use as the baseline for the merged report. See Switching between applications in Dev Studio.

2. In the header of Dev Studio, click Configure > Application > Quality > Test Coverage.
3. Click the Application level tab.
4. In the **Coverage history** section, click Merge reports.
5. Enter the title of the merged report, and then click Next.
6. In the list of the most recent reports, select the reports that you want to include in the merged report, and then click Create.
7. Close the **Merge confirmation** window.

> **Result:**
>
> Dev Studio automatically adds the `MRG_` prefix to every merged report to differentiate them from standard application-level coverage reports and to facilitate finding them.

8. Open the merged report. In the Coverage history section find the merged report that you created and click Show report.
9. **Optional:** To open a report that is included in the merged report, in the Merged reports section, click the report name.

Viewing test coverage reports

View a report that contains the results of test coverage sessions to determine which Rules in your application are not covered with tests. You can improve the quality of your application by creating tests for all uncovered Rules that are indicated in the reports.

1. In the header of Dev Studio, click Configure > Application > Quality > Test Coverage.
2. Choose the type of report that you want to view:
    - To view application-level test coverages, click the Application level tab.
    - To view user-level test coverages, click the User level tab.
3. In the **Coverage history** section, hover over the row with the relevant test coverage session, and then click Show Report.

4. **Optional:** Choose the data you want to include in the report:
   - To include only the Rules that were updated after a specific date, in the Rules updated after field, click the calendar icon, select a date and time, and then click Apply.
   - To include all the Rules that are covered with tests, click Covered.
   - To include all the Rules that are not covered with tests, click Uncovered.
   - To filter the Rules, in the column header that you want to filter, click the filter icon, enter the filter criteria, and then click Apply.
   - To open a single report that is included in a merged report, in the **Merged reports** section, click the report name.
   - To open a rule that is included in the report, click the rule name.

Setting up for test automation

Before you create Pega unit test Cases and test suites, you must configure a test Ruleset in which to store the tests.

- **Creating a test Ruleset to store test Cases**

Creating a test Ruleset to store test Cases

Before you can create unit test Cases or scenario tests, you must configure a test Ruleset in which to store the tests.

1. In the Dev Studio header, open your application rule form by clicking your application and selecting Definition.
2. In the Application Rulesets section, click Add Ruleset.
3. Complete one of the following Actions:
   - Enter a Ruleset name and version of an existing Ruleset.
   - Click the Open icon and create a new Ruleset.

> **NOTE:** The test Ruleset must always be the last Ruleset in your application stack.

4. Open the test Ruleset and click the **Category** tab.
5. Select the **Use this Ruleset to store test Cases** check box.
6. Save the test and application Ruleset forms.

> **Result:**
>
> When you save test Cases for Rules, they are saved in this Ruleset.

- PegaUnit testing