PEGA PLATFORM

Pega Platform 8.1

Upgrade Guide

For Tomcat and Oracle

PEGA®

Contents

Overview

Use the Pega documentation to install or upgrade your system.

This guide describes how to upgrade an existing instance of PRPC or Pega Platform to 8.1.

To install a new version of Pega Platform, see the *Pega 8.1 Installation Guide* for your database and application server platform.

Caution: This release introduces new features and functionality that might cause compatibility issues with your existing application. You might need to take additional actions before deploying. For information about new features, see the Pega Community. For information about post-upgrade actions, see Post-upgrade configuration.

In-place or out-of-place upgrades and single or double data migration

Your system requirements determine whether you use the in-place or out-of-place upgrade method, and whether you need to perform a single or double data migration.

In-place and out-of-place upgrades

Depending on the amount of downtime your system can tolerate, you can either perform an in-place or out-of-place upgrade:

- In an in-place upgrade, the existing schemas are upgraded, and no new schemas are created. These upgrades require significant downtime because they directly modify the active schemas. You cannot use Pega Platform during an in-place upgrade. Use in-place upgrades for experimental, development, and most test systems. For more information, see Performing an in-place upgrade.
- To minimize downtime, perform an out-of-place upgrade. Out-of-place upgrades require a temporary upgrade schema that can reside either on the production database or on a second temporary database. The upgrade scripts run on the temporary schema. The migration script moves the rules between the schemas. This minimizes the length of time during which the production system is unavailable. As a best practice, use out-of-place upgrades for split-schema configurations. Out-of-place upgrades can also be reversed.

Double and single migration for out-of-place upgrades

For out-of-place upgrades, the availability of server resources and security features determine whether you perform a single or double migration.

- To minimize consumption of server hardware resources to avoid performance drain on existing systems or when network I/O latency is significant, perform a double migration.
  1. Migrate the contents of the original schemas to a temporary schema on a separate database.
  2. Upgrade the temporary schemas.
  3. Migrate the upgraded schemas back to the original database.

  This is the best practice for database servers that host multiple high-use applications in addition to Pega Platform, for database servers that have restrictive CPU or I/O limitations, or if the upgrade process cannot be executed on the database server or within the same data center.

  For detailed instructions, see Performing an out-of-place upgrade with a double migration.
- If your system includes a firewall or other security measures and the upgrade process does not have database access to both the source system and the target system, perform a double-migration

upgrade with firewall. For detailed instructions, see Performing an out-of-place upgrade with a double migration.

- If your system does not have any of these restrictions but still requires minimal downtime, perform a single-migration upgrade:
  1. Migrate the contents of the original schemas to a temporary schema on the original database.
  2. Upgrade the temporary schema. The temporary schema becomes the new rules schema.

  For more information, see Performing an out-of-place upgrade with a single migration.

Plan your deployment

Pega Platform supports several configuration options that can affect the choices that you make during the deployment. Before beginning, read this section thoroughly.

Plan your architecture and configuration.

- Decide whether to perform an in-place or out-of-place deployment. To minimize downtime, use the out-of-place method. Perform in-place deployments if lengthy downtime is not a concern, for example, for developmental systems. For more information, see In-place or out-of-place upgrades and single or double data migration.
- Choose a configuration type: single-schema or split-schema configuration. Pega recommends a split-schema configuration. See Split-schema and single-schema configurations. For split-schema configurations, choose whether you will maintain separate tablespaces for the data schema and rules schema. This decision depends on your database configuration.
- Choose whether to use the standard product edition or the multitenancy edition. The multitenancy edition has different requirements, different run-time behaviors, and different administrative procedures from the standard edition. Before you select the multitenancy edition, review the *Multitenancy Administration Guide* on the Pega Community.

  Upgrading from one edition to another is not supported. The schema DDLs for the two editions are not compatible. For example, if you install the standard edition and later decide to use the multitenant edition, you must either drop and re-create the database or create a new database.
- Choose whether to use Kerberos functionality. Kerberos is a computer network authentication protocol that allows nodes communicating over a non-secure network to prove their identity to one another in a secure manner. If you enable Kerberos authentication, you must use the command line method to deploy Pega Platform. For more information, see your installation guide.

Consider the following precautions before you continue:

- Always maintain a backup of your system, especially before performing an upgrade.
- Do not change your environment while you are deploying Pega Platform. For example, if you are making changes to your application server or database server, do so before you deploy Pega Platform.
- Conduct a site-specific analysis of Pega Platform and any custom applications to determine the size of your database tablespace.
- The upgrade process requires additional space approximately equal to twice the size of your rules schema. Ensure that your database can allocate enough space to perform this upgrade.

Environment considerations

Consider your application customization, libraries, database, and special site requirements before continuing.

- Custom applications — If you are using any custom applications, confirm whether your custom applications are supported for this version of the Pega Platform. It might be necessary to upgrade your versions of the custom applications to work with the new version of the Pega Platform.
- Customized Pega Platform database — If you made changes to the Pega Platform database schema, incorporate those changes into the database after you upgrade the database schema. In particular, you should merge any changes to triggers, views, or stored procedures that you made in the previous version, and review any custom rules or work tables that you created. The upgrade procedure leaves tables you have added to the schema in place, but you might need to modify the tables to match changes in the Pega schema.

Also verify that schema references in triggers, stored procedures, and views correctly reflect the new schema names.

- Third-party or custom libraries — If you have integrated third-party or custom libraries into your system, make sure you have a copy of them before upgrading. The upgrade might overwrite your deployed libraries.
- Special site requirements — Your site might have special deployment, integration, or security requirements. Schedule an early review of the upgrade procedure with the appropriate system and database administrators.

System requirements

Before you deploy, ensure that your system meets the following minimum requirements.

UI-based tool requirements

If you plan to use the UI-based Installation and Upgrade Assistant, ensure that the system meets these minimum system requirements in addition to all other requirements.

- 1.25 GB minimum available memory
- 10 GB minimum disk space plus at least 8 GB available space in the temporary directory of the root file system. The default temporary directory for the deployment is java.io.tmpdir.
- Java Platform, Standard Edition Development Kit (JDK)

Application server requirements

Install only Pega Platform on the application server. The application server must meet the minimum requirements listed in the *Platform Support Guide* on the Pega Community and in this section.

- Oracle JDBC type 4 driver, such as **`ojdbc7.jar`**. For more information about supported drivers, see the *Platform Support Guide*.
- Supported 64-bit JDK. See the *Platform Support Guide* on the Pega Community for a list of supported versions.
- 1 GB minimum free disk space. You might need additional storage space for debugging and logging.
- Memory requirements: Pega Platform runs in memory (heap) on Java Virtual Machines (JVMs). In general, all activity is distributed over multiple JVMs (nodes) on the application server.
  - Standard suggested system heap size is 4 - 8 GB based on monitoring of memory usage and garbage collection frequency.
  - Larger heaps are advisable if your applications allow a high number of concurrent open tasks per session or cache a large collection of transaction or reference data.
  - Do not deploy Pega Platform in an environment where the heap size exceeds the vendor-specific effectiveness limit.
  - Oracle JDKs use compression to minimize the cost of large heaps. The compression option is labeled CompressedOOPS and is effective up to 32 GB. In current 64-bit JVMs, compression is enabled by default.
  - The host application server memory size must be at least 4 GB larger than the Pega Platform heap size to allow space for the operating system, monitoring tools, operating system network file buffering, and JVM memory size (-XMX option). The minimum host application server memory size is 8 GB:

    4 GB heap + 4 GB for native memory, operating system, and buffering

If the server does not have enough memory allocated to run Pega Platform, the system can hang without an error message. The correct memory settings depend on your server hardware, the number of other applications, and the number of users on the server, and might be larger than these recommendations.

Database server requirements

Your database server must meet the minimum requirements listed in the *Platform Support Guide* on the Pega Community.

Verify that the system also includes:

- Support for Java.
- Support for User-defined functions (UDFs) if you plan to use them.
- A supported version of the JDBC4 driver for your version of the database
- 8 GB minimum RAM
- 10 GB minimum initial tablespace set to auto-extend
- 50 MB logfile size — This default size is sufficient for the initial installation, but will need to be resized to run the application server workload.

Obtain database connection information

Before you configure the data source resources, obtain the correct database connection information from your database administrator.

To determine the database connection URL, obtain the following information from your database administrator:

- Connection method — Service or SID
- Service or SID name
- Host name
- Port number

When you configure the application server, you will enter the connection string, **pega.jdbc.url** as follows. Replace items in *italics* with the values for your system:

Use one of the following formats:

- `jdbc:oracle:thin:@localhost:` *port/service-name*
- `jdbc:oracle:thin:@localhost:` *port*`:`*SID*

Node classification in high availability systems

Optimize performance and provide higher scalability and stability in a cluster by using node classification, which is the process of separating nodes, segregating them by purpose, and predefining their behavior.

Note: Node classification applies to high availability cluster environments only.

By configuring a node with a node type, you dedicate the node to perform particular actions and run only those agents, listeners, job schedulers, and queue processors that are mapped to the node type. For example, if a set of nodes is dedicated to user requests, background processes can be disabled to improve performance.

Every node that is started with the same node type uses the same template and follows the same behavior.

For more information, see *Node classification* on the Pega Community.

Prepare your application server

This section describes how to prepare your application server for the upgrade.

Commit hotfixes

Before you deploy, commit any uncommitted hotfixes on your system. If there are uncommitted hotfixes when you deploy, the hotfixes are automatically committed. For information about committing hotfixes, see the help.

Port configuration

Before you configure your application server, ensure that the following ports are open and available:

- Search (Elasticsearch) — one TCP port in the range 9300-9399 (default 9300). This port is used for internal node-to-node communication only, and should not be externally accessible.
- Cluster communication — leave open the port range 5701-5800. By default, the system begins with port 5701, and then looks for the next port in the sequence (5702, followed by 5703 and so on). To override the default port range, set a different value for the initialization/cluster/ports setting in the **`prconfig.xml`** file.
- Pega Platform can include multiple servers, or nodes, and each node can contain multiple Java Virtual Machines (JVMs). The number of available ports in this range needs to be greater than or equal to the greatest number of JVMs on any one node in the cluster. For example, if there are three JVMs on one node, and seven JVMs on another node, there must be at least seven ports available.

Setting the JVM security parameter for LINUX or UNIX

If you use UNIX or LINUX, set security to urandom.

1. Open the configuration file for your application server:

   `setenv.sh`

2. Enter the following argument to set security to urandom:

   ```
   -Djava.security.egd=file:///dev/urandom
   ```

3. Save the changes.

Setting Stream nodes for Queue Processor rules in high availability systems

Queue Processor rules require at least one stream node. Without a stream node, messages cannot be queued to or retrieved from Kafka. Configure two stream nodes by using node classification to provide a backup in case one stream node fails.

Note: Node classification applies to high availability cluster environments only.

1. Open the configuration file for your application server:

`setenv.sh`

2. Configure at least two nodes as stream nodes by entering the following JVM argument:

```
-DNodeType=Stream
```

3. Save the configuration file.

Preparing your database

Before you begin preparing your database, see the *Platform Support Guide* on the Pega Community to verify that your database is supported. Then have your database administrator make the modifications described in this section.

- If your system includes synonyms to Pega-supplied tables, drop the synonyms before you upgrade. If necessary, reapply the synonyms after the deployment is complete.
- If you are using Oracle 11g, do not use the UCP (Universal Connection Pool) feature in your database. Oracle BUG 8462305 causes a failure when an application tries to call a stored procedure. This error causes Pega Platform to work incorrectly with a database that uses UCP. To determine if UCP is in use, check for the **`ucp.jar`** file in the classpath of the application server.

Backing up your system and database

Upgrading modifies both the data schema and the rules data; use a backup procedure that preserves both schemas. Back up the existing database, your applications, and the system.

1. Verify that all rules are checked in.
2. Use your database utilities to back up the Pega Platform database.
3. If you edited any of the following Pega Platform configuration files in the **`APP- INF\classes`** directory of an EAR deployment or the **`WEB-INF\classes`** directory of a WAR deployment, include these in the backup:
   - **`prbootstrap.properties`**
   - **`prconfig.xml`**
   - logging file: **`prlogging.xml`** or **`prlog4j2.xml`**
   - **`web.xml`**
   - **`pegarules.keyring`** or any other .keyring files

   Note: For upgrades from PRPC 6.1 SP2 or earlier if you added System Settings to your **`prbootstrap.properties`** or **`prconfig.xml`** files, convert them to Dynamic-System-Settings Data- instances. See Upgrading from PRPC 6.1 SP2 and earlier: moving system settings.
4. Back up any third-party or custom JAR files on your system. Redeploying the Pega Platform applications might delete these from your application server.

Verifying that your Oracle database is ready for localization

Oracle supports two types of character semantics, BYTE and CHAR. CHAR supports international character sets.

Before you upgrade, verify that Oracle semantics is set to CHAR to support localization:

1. On the database server, open the file **`SPFILE`** *BNAME***`.ora`** in the database directory.
2. Verify that the settings are as follows:

```
NLS_LENGTH_SEMANTICS=CHAR scope=both;
```

```
NLS_CHARACTERSET=AL32UTF8;
NLS_NCHAR_CHARACTERSET=AL16UTF16;
```

3. If the semantics settings differ, migrate the database character set. For more information, see your Oracle documentation and the Pega Community article Migrating Database Character Sets for Oracle.

Upgrading multitenant systems from Pega 7.1.5 and later

Follow the steps in this section to upgrade a multitenant system from Pega 7.1.5 or later. If you are upgrading from a version prior to Pega 7.1.5, skip this section. If you do not have a multitenant system, skip this section.

The multitenant table architecture requires an additional column, pzTenantID. Several tables are now tenant-qualified; deploying the new version of Pega Platform automatically adds the multitenant column to these tables.

SQL databases do not allow the addition of a non-null column to an existing table unless the table is empty. Therefore, if the tables contain data, upgrading systems on those databases displays an error "Table must be empty to add column" and the deployment fails. For most tables, truncating the data is acceptable; however, the pr_data_admin_product table and the pr_data_tag_relevantrecord table includes important data. Pega provides a script to add the pzTenantID column to the pr_data_admin_product table and the pr_data_tag_relevantrecord table without losing data.

To prepare the tables, follow these steps before you upgrade. The specific steps depend on your starting version of the Pega Platform.

1. Log in to the data schema.
2. For upgrades from Pega 7.1.9 and earlier, add the column to the pr_data_admin_product table without truncating the data:
   a) Navigate to the AdditionalUpgradeScripts directory:
   Pega-image/`ResourceKit/AdditionalUpgradeScripts/MT/719AndEarlier/`
   b) Run the script for your database:
   database`_mt_upgrade_tables.sql`
3. For upgrades from Pega 7.2 and Pega 7.2.1, add the column to the pr_data_tag_relevantrecord table without truncating the data::
   a) Navigate to the AdditionalUpgradeScripts directory:
   Pega-image/`ResourceKit/AdditionalUpgradeScripts/MT/72And721/`
   b) Run the script for your database:
   database`_mt_upgrade_tables_d_a_tag_relevantrecord.sql`

Upgrading from PRPC 6.1 SP2 and earlier: move system settings

If you are upgrading from 6.1 SP2 or earlier, move any custom System Settings from the **`prconfig.xml`** or **`prbootstrap.properties`** configuration files to the Dynamic System Settings (Data-Admin-System-

Settings). Settings in env elements in your current **`prconfig.xml`** or **`prbootstrap.properties`** files continue to work, and this task can be done at any time.

Pega provides a utility to move the settings from the configuration files to Data-Admin-System-Settings instances. See "Upgrading Configuration Settings from Prior Versions to Version 6.2" in the Configuration Settings Reference Guide on the Pega Community for details. Note that the instructions on how to run this utility are the same for Pega 8.1 as they are for version 6.2.

Moving these settings to the database has several advantages.

- Since the settings are stored as Data- instances, they can be read from the database by all nodes running your installation. All nodes will have the same setting values, unlike the values in the **`prconfig.xml`** file, which apply only to the node where that file is located.
- The values of the dynamic system settings can be viewed and modified from Dev Studio. Alternately, values stored in the configuration files must be changed by editing the file, which can require restarting the application nodes.

Database users

This section describes deployment and runtime users and lists all required permissions.

- Deployment user — This user performs actions only during the deployment.
- Oracle users — Because Oracle has a one-to-one relationship between users and schemas, if you have a split-schema configuration, you must have separate users for the rules schema and the data schema.
  - Oracle rules schema owner — Only used to create the schema. The Oracle rules schema owner can be associated with either individual tablespaces or a common tablespace. Pegasystems recommends separate tablespaces for each user in critical SDLC environments.
  - The Oracle data schema owner is the Base runtime user.
- Run-time users — These users perform actions on the Pega Platform after the deployment. In a dual-user configuration, an Admin user is granted full privileges, and a Base user is granted a smaller subset. Pega recommends the dual-user configuration:
  - Base user — The user who runs the Pega Platform. Most run-time operations use the Base user and associated data source.

    The Base user is the Oracle data schema user.

  Pega recommends that you use the dual-user configuration with separate Admin and Base users; however, you can create a single Base user with both sets of privileges. If there is no separate Admin user, the Pega Platform uses the Base user for all run-time operations.

  Note: If you have only a Base user, the system cannot perform automatic schema-change operations.

General user permissions

The following table describes the general permissions for each user and the purpose of each permission.

Pega recommends the dual-user configuration. For a single-user configuration, the Base user also requires the permissions listed for the Admin user.

Permission	Deployment User	Base User	Admin User
Insert/select/update/delete on data and rules tables	The deployment process saves instances to data and rules tables.	User has basic read and write access to data and rules tables.	

Permission	Deployment User	Base User	Admin User
Select data and rule schema metadata	The deployment process reads metadata about tables and other objects to determine the most efficient way to access and store data.	PegaRULES reads metadata about tables and other objects to determine the most efficient way to access and store data.	
Execute stored procedures in data and rules schemas	The deployment process uses stored procedures for system operations.	PegaRULES uses stored procedures for system operations.	
Create/update/drop tables, indexes, and constraints in data and rules schema	The deployment process installs the tables, indexes, and constraints used by PegaRULES.		Various system management tools allow you to create and modify tables and indexes. For data schemas, various facilities in decisioning create short-lived tables to assist with strategy analysis.
Create/update/drop views in data and rules schemas	The deployment process installs the views used by PegaRULES.		Various facilities in decisioning create short-lived views to assist with strategy analysis. Import also requires this when importing views.
Create/update/drop stored procedures, triggers, and functions	The deployment process installs stored procedures, triggers, and functions used by PegaRULES.		
Enable and disable triggers on rule tables	The installation and upgrade processes disable triggers in order to save large amounts of records more quickly.		
Truncate rule and data tables	Various tables must be truncated during a PegaRULES upgrade.		
Grant object-level privileges on rules schema to data user	When the install and upgrade processes create tables and other objects in the rules schema, they must grant the Base user access to these objects.		When system management utilities create tables and other objects in the rules schema, they must grant the Base user access to these objects.

Note: If you plan to manually install the user-defined functions (UDFs) from Pega, the database user who will install the UDFs cannot have the sysadmin role. Having the sysadmin role changes the default schema name and causes installation problems. For more information about UDFs, see Installing user-defined functions.

Oracle user permissions

Use either an SQL command or the Oracle Enterprise Manager to create users with the privileges and roles listed in this section. Because Oracle maintains a one-to-one relationship between schemas and database users, creating users also creates the schemas.

All Oracle database users require unlimited tablespace. For information about creating the users with unlimited tablespace privileges, see Creating Oracle users from an SQL statement. For information about using the Oracle Enterprise Manager to create users and assign privileges and roles, see Creating Oracle users by using the Enterprise Manager.

Deployment user privileges and roles

The Oracle rules schema owner requires only unlimited tablespace and is only used for deployment.

The Deployment user requires the following privileges and roles for all configurations:

- UNLIMITED TABLESPACE
- CREATE SESSION
- CREATE ANY TABLE
- ALTER ANY TABLE
- INSERT ANY TABLE WITH ADMIN OPTION
- SELECT ANY TABLE
- UPDATE ANY TABLE
- DELETE ANY TABLE
- CREATE ANY INDEX
- CREATE ANY PROCEDURE
- EXECUTE ANY PROCEDURE
- CREATE ANY VIEW
- CREATE ANY TYPE
- CREATE ANY TRIGGER
- ALTER ANY TRIGGER
- GRANT ANY OBJECT PRIVILEGE
- DROP ANY PROCEDURE
- DROP ANY TRIGGER
- DROP ANY TABLE
- DROP ANY VIEW
- DROP ANY INDEX
- ANALYZE ANY
- ANALYZE ANY DICTIONARY
- SELECT_CATALOG_ROLE (This is a role, not a privilege.)

  Note: For custom applications, you must grant the SELECT_CATALOG_ROLE to the Deployment or Admin user. Some custom applications use triggers, so the user will need the SELECT_CATALOG_ROLE to drop triggers that read from the update cache and rule view tables. The deployment automatically drops custom triggers. Manually re-create custom triggers after you deploy Pega Platform.

- SESSIONS_PER_USER: When the upgrade begins to import rules, it opens multiple parallel connections. Consider setting SESSIONS_PER_USER to UNLIMITED to avoid an error similar to the following:

  ```
  Exceeded simultaneous SESSIONS_PER_USER limit
  ```

Run-time users — privileges and roles

The run-time users require different permissions depending on whether you have a dual-user configuration.

Note: The run-time users of the rules and data schemas can share the same tablespace. If you create separate tablespaces for the rules schema and the data schema users, base the size of the tablespace on the estimated number of work objects in the application.

Dual-user configuration — Admin and Base users

In a dual-user configuration, grant these privileges and roles:

- Admin user
  - UNLIMITED TABLESPACE
  - CREATE SESSION
  - CREATE ANY TABLE
  - ALTER ANY TABLE
  - INSERT ANY TABLE WITH ADMIN OPTION
  - SELECT ANY TABLE
  - UPDATE ANY TABLE
  - DELETE ANY TABLE
  - CREATE ANY INDEX
  - CREATE ANY PROCEDURE
  - EXECUTE ANY PROCEDURE
  - CREATE ANY VIEW
  - CREATE ANY TYPE
  - CREATE ANY TRIGGER
  - ALTER ANY TRIGGER
  - GRANT ANY OBJECT PRIVILEGE
  - DROP ANY PROCEDURE
  - DROP ANY TRIGGER
  - DROP ANY TABLE
  - DROP ANY VIEW
  - DROP ANY INDEX
  - ANALYZE ANY
  - ANALYZE ANY DICTIONARY
  - SELECT_CATALOG_ROLE (This is a role, not a privilege.)
- Base user—The Base user is the Oracle data schema owner.
  - Basic read and write access to data and rules tables including rules resolution.
  - UNLIMITED TABLESPACE
  - CREATE SESSION

Single-user configuration— Base user only

The Base user is the Oracle data schema owner.

Note: Pega recommends that you create an Admin user separate from the Base user; if you opt for a single Base user, the system cannot perform automatic schema-change operations.

- Basic read and write access to data and rules tables including rules resolution.
- UNLIMITED TABLESPACE
- CREATE SESSION

Creating Oracle users from an SQL statement

Use SQL statements to create users. For information about using the Oracle Enterprise Manager to create users and assign privileges and roles, see your Oracle documentation.

1. On the database server, run the following SQL statement to create users and grant the users unlimited access to the default USERS tablespace.

   ```
   ALTER USER <user> DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USERS;
   ```
2. Use the Oracle tools to assign the appropriate roles and privileges to this user.
3. Repeat steps 1 and 2 for the remaining users:
   - Oracle schema users:
     - For single schemas, create one Oracle schema user
     - For split-schemas, create separate Oracle rules and data schema users.
   - Deployment user
   - Base user
   - Admin user (for dual-user configurations)

Creating Oracle users by using the Enterprise Manager

Follow these steps to create a user:

1. Log in to the Enterprise Manager using the URL provided by the Database Configuration Assistant. The URL is usually in the form: `https://`*host*`:5501/em`
2. Enter the user name and password and click **Login**.
   - User name = sys
   - Password = *password*
3. Select **Security > Users**.
4. Select **Actions > Create User**. Accept the other defaults.
5. On the User Account step, enter the name and password for the user you are creating.
6. Click the right arrow.
   a) If you created a dedicated tablespace, choose that tablespace from the menu.
   b) Accept the other defaults.
7. Click the right arrow.
8. Select the privileges for this user and click **OK**.
9. Repeat these steps to configure the remaining users.

Performing an out-of-place upgrade with a double migration

Perform an out-of-place upgrade to minimize downtime. Use a double migration to avoid performance drain on existing database servers. The rules schema upgrade occurs out-of-place, and the data schema upgrade occurs in place.

Note: To further minimize downtime, use the High Availability features. For more information about High Availability, see the Pega Platform High Availability Administration Guide.

This upgrade involves four schemas:

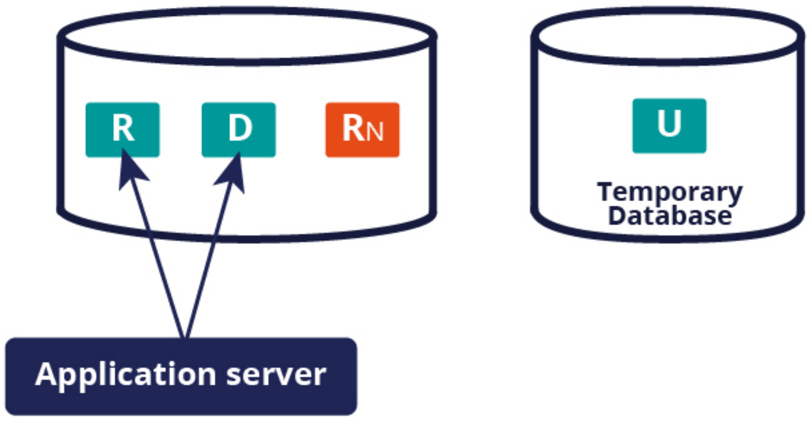
- Data schema - your current data schema. This will be your data schema after the upgrade as well.
- Rules schema - your current rules schema. This schema will be replaced after the upgrade.
- Temporary upgrade schema - a temporary schema on a separate database used for staging the upgrade. This will include the rules and data tables during the upgrade.
- New rules schema- the new rules schema. This will become the rules schema after the upgrade.

The generic process for upgrading a split-schema configuration can be summarized in the following steps:

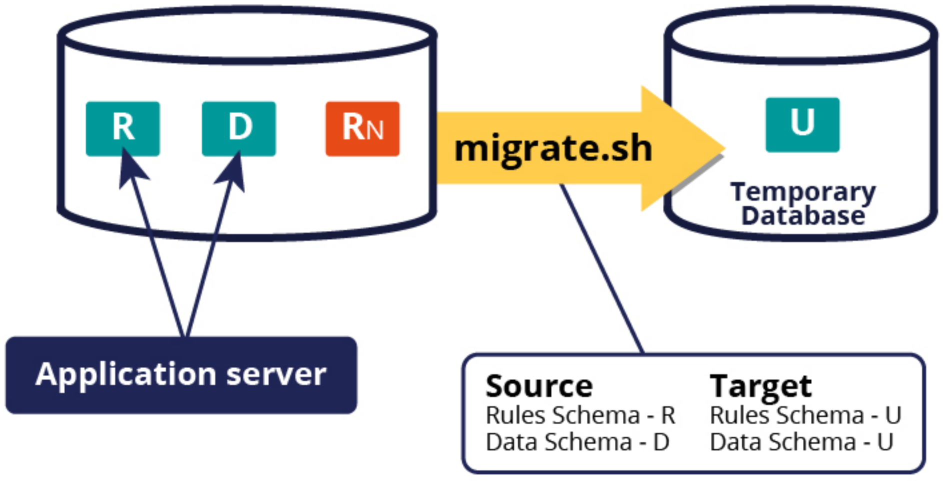
1. For upgrades to high-availability systems on Pega 7.x, disable rule creation on the rules schema. See For high availability systems: Disabling rule creation on the rules schema.

   Note: During the upgrade, you can still use the original system, but any rules created after the migration will be lost when you switch to the upgraded rules schema. The data schema retains all new data and work.

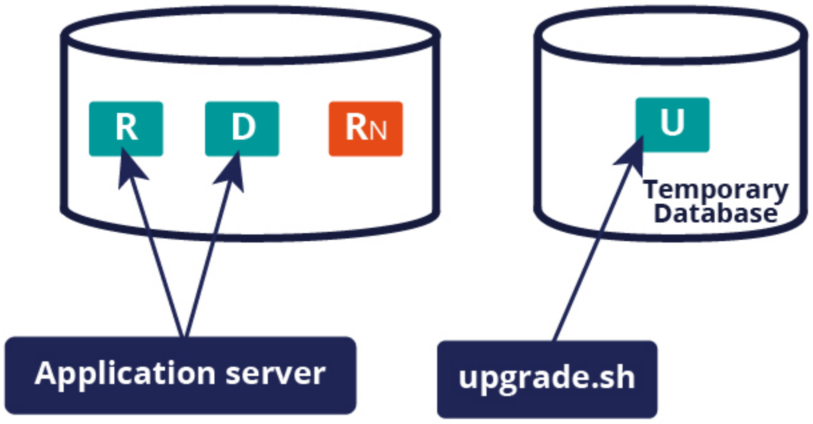
2. Create two blank schemas and a temporary database. See Create two new physical schemas on two databases.



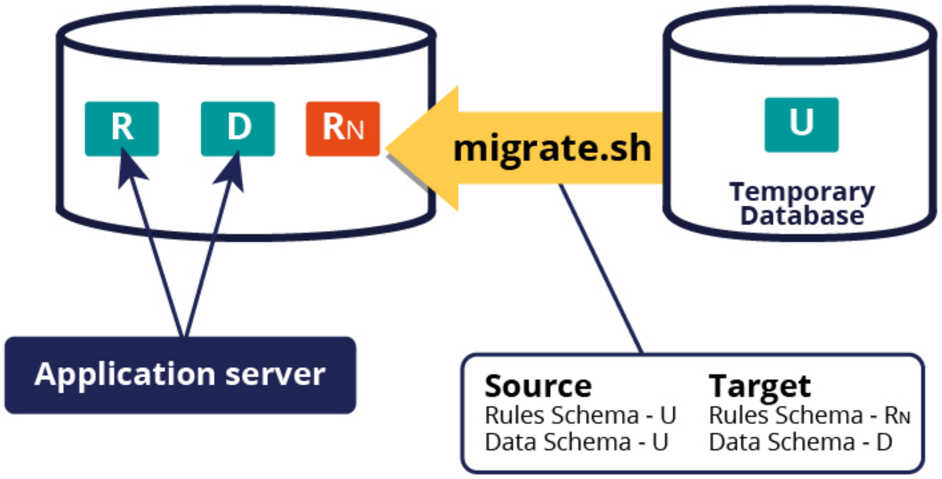
3. Migrate the current rules schema to the temporary upgrade schema. See Migrating the existing rules schema.



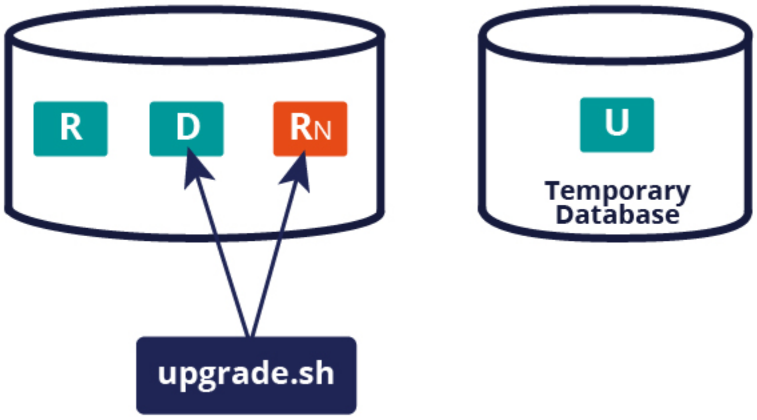
4. Upgrade the temporary upgrade schema. See Upgrade methods for the migrated rules schema.



5. Migrate the upgraded temporary upgrade schema to the new rules schema. See Migrating to the new rules schema.



6. Shut down the existing system.
7. Use the upgrade script to upgrade the data schema and reference the new rules schema. See [Upgrading the data schema](#).



For high availability systems: Disabling rule creation on the rules schema

If you are using the recommended High Availability option, you can disable rule creation on the rules schema to speed the deployment. If you are not using the High Availability option, skip this section.

Before you deploy, commit all uncommitted hotfixes. After you begin the deployment, ensure that no changes to the rules, including hotfixes, are applied until after the deployment is complete.

1. Log in as a user with the PegaRULES:HighAvailabilityAdministrator role.

2. Navigate to **System > High Availability > HA Cluster Settings**.
3. Under **Cluster Upgrade**, select **Cluster Upgrading - Disable saving of rules**.
4. Click **Submit**.
5. Log out.

Create two new physical schemas on two databases

To use two databases for an out-of-place upgrade, create a new database and two new schemas:

- Create the temporary database of the same type and version as your current database.
- Create the temporary upgrade schema in the temporary database.
- Create the new rules schema in your original database.

Migrating the existing rules schema

Use the migrate script to migrate the rules tables and other required database objects from the existing schema to the new rules schema.

You will also use the migrate script later to generate and apply rules objects, such as functions and stored procedures, after the upgrade, but the property settings will be different. The Deployment user performs the migrations.

Note: Pega strongly recommends that you use the migration script. The use of vendor tools to manage this step is not recommended. If you do not use the migrate script to migrate the schema, there are additional manual steps required that are not documented here.

This process depends on whether the system has access to both the original and temporary databases. For more information, see:

- Migrating the rules schema when you have access to both databases
- Migrating the rules schema when you have access to one database

Note: To minimize the time required, run the migration scripts from the same data center as the database server.

Migrating the rules schema when you have access to both databases

If you have access to both the temporary and original databases, run the migrate script once to migrate the rules schema.

1. Use a text editor to edit the `migrateSystem.properties` file in the scripts directory:
   Pega-image`\scripts\migrateSystem.properties`
2. Configure the source properties. For more information, see Migrate script properties.

   Note: If you are starting with a single-schema system, the pega.source.rules.schema and pega.source.data.schema names are the same.

```
# Connection Information
pega.source.jdbc.driver.jar=full path/DRIVER.jar
pega.source.jdbc.driver.class=database driver class
pega.source.database.type=database vendor type
pega.source.jdbc.url=URL of database
pega.source.jdbc.username=Deployment user name
pega.source.jdbc.password=password
pega.source.rules.schema=original rules schema name
pega.source.data.schema=original data schema name
```

3. Configure the target properties. See Properties file parameters for more information. Leave the target data schema name blank:

   ```
   pega.target.jdbc.driver.jar=full path/DRIVER.JAR
   pega.target.jdbc.driver.class=database driver class
   pega.target.database.type=database vendor type
   pega.target.jdbc.url=database URL
   pega.target.jdbc.username=Deployment user name
   pega.target.jdbc.password=password
   pega.target.rules.schema=temporary upgrade schema
   pega.target.data.schema=
   ```

   Note: If pega.target.data.schema is blank, the rules schema is used by default.

4. Configure the bulkmover directory:

   `pega.bulkmover.directory=`*full path to output directory*

5. Configure a temporary directory:

   `pega.migrate.temp.directory=`*full path to temporary directory*

6. Configure the operations to be performed by the utility as shown below:

   ```
   pega.move.admin.table=true
   pega.clone.generate.xml=true
   pega.clone.create.ddl=true
   pega.clone.apply.ddl=true
   pega.bulkmover.unload.db=true
   pega.bulkmover.load.db=true
   ```

7. Disable the operations as shown below:

   ```
   pega.rules.objects.generate=false
                 pega.rule.objects.apply=false
   ```

8. Save the properties file.
9. Open a command prompt, and navigate to the scripts directory.
10. Type **`migrate.bat`** or **`./migrate.sh`** to run the script.

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

Migrating the rules schema when you have access to one database

If you can only access one database at a time (for example, if there is a firewall between the two servers), run the migration script twice: first on a system that can access the original source database, and then where it can access the temporary target database.

Make sure that the system that accesses the temporary database has access to the bulkmover directory and the DDL generated from the source database.

1. On a system that can access the original database, export rules from the original database.

   a) Use a text editor to edit the **`migrateSystem.properties`** file in the scripts directory:
   Pega-image`\scripts\migrateSystem.properties`

b) Configure the source properties. For more information, see Migrate script properties.

> **Note:** If you are starting with a single-schema system, the pega.source.rules.schema and pega.source.data.schema names are the same.

```
# Connection Information
                  pega.source.jdbc.driver.jar=full path/DRIVER.jar
                  pega.source.jdbc.driver.class=database driver class
                  pega.source.database.type=database vendor type
                  pega.source.jdbc.url=URL of database
                  pega.source.jdbc.username=Deployment user name
                  pega.source.jdbc.password=password
                  pega.source.rules.schema=original rules schema name
                  pega.source.data.schema=original data schema name
```

c) Configure the target properties. See Properties file parameters for more information. Leave the target data schema name blank:

```

                    pega.target.jdbc.driver.jar=full path/DRIVER.JAR
                    pega.target.jdbc.driver.class=database driver class
                    pega.target.database.type=database vendor type
                    pega.target.jdbc.url=database URL
                    pega.target.jdbc.username=Deployment user name
                    pega.target.jdbc.password=password
                    pega.target.rules.schema=temporary upgrade schema
                    pega.target.data.schema=
```

> **Note:** If pega.target.data.schema is blank, the rules schema is used by default.

d) Configure the bulkmover directory:

```
pega.bulkmover.directory=full path to output directory
```

e) Configure a temporary directory:

```
pega.migrate.temp.directory=full path to temporary directory
```

f) Configure the operations to be performed by the utility as shown below:

```

                  pega.move.admin.table=true
                  pega.clone.generate.xml=true
                  pega.clone.create.ddl=true
                  pega.clone.apply.ddl=false
                  pega.bulkmover.unload.db=true
                  pega.bulkmover.load.db=false
                  pega.rules.objects.generate=false
                  pega.rule.objects.apply=false
```

g) Save the properties file.

h) Open a command prompt, and navigate to the scripts directory.

i) Type **`migrate.bat`** or **`./migrate.sh`** to export the rules.

2. On a system that can access the temporary database, import the rules to the temporary database.

a) Copy the **`migrateSystem.properties`** file you created in step 1 to a system that can access the temporary database.

b) Verify the target, bulkmover, and temporary directory properties.

c) Configure the operations to be performed by the utility as shown below:

```
pega.move.admin.table=true
                    pega.clone.generate.xml=false
                    pega.clone.create.ddl=false
                    pega.clone.apply.ddl=true
                    pega.bulkmover.unload.db=false
                    pega.bulkmover.load.db=true
                    pega.rules.objects.generate=false
                    pega.rule.objects.apply=false
```

d) Save the properties file.

e) Open a command prompt, and navigate to the scripts directory.

f) Run **`migrate.bat`** or **`./migrate.sh`** to import the rules.

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

Upgrade methods for the migrated rules schema

Use one of these methods to upgrade the migrated rules schema:

- Installation and Upgrade Assistant
- Command line script: **`upgrade.bat`** or **`upgrade.sh`**

Note: To minimize the time required to upgrade, run the upgrade from the same data center as the database server.

Upgrading the rules schema by using the Installation and Upgrade Assistant

For a UI-based upgrade experience, use the Installation and Upgrade Assistant.

Because of the large volume of data, run the IUA on the same network as the database server. If this is not possible, run the tool on a system with fast, direct access to the database server. The Deployment user performs these steps.

The upgrade can last for several hours and the time can vary widely based on network proximity to the database server.

1. Double-click the **`PRPC_Setup.jar`** file to start the IUA.

   Note: If JAR files are not associated with Java commands on your system, start the IUA from the command line. Navigate to the directory containing the **`PRPC_Setup.jar`** file, and type `java -jar PRPC_Setup.jar`.

   The IUA loads and the Pega icon is displayed in your task bar.
2. Click **Next** to display the license agreement.
3. Review the license agreement and click **Accept**.
4. **Optional:** If you are resuming after a previous failed upgrade and the **Resume Options** screen is displayed, select either **Resume** or **Start Over**.

- If you select **Resume**, the system uses the previously entered database configuration information to resume the upgrade from the last successful process. Continue these instructions at step 8.
- If you select **Start Over**, continue at step 5 to reenter the configuration information.

5. On the **Installer Mode** screen, choose **Upgrade** and click **Next**.
6. Choose your database type and click **Next**.
7. Configure the database connection. The JDBC drivers allow the Pega Platform application to communicate with the database. Specify the new rules schema name for both the **Rules Schema Name** and **Data Schema Name** fields. For more information, see Appendix A — Properties files.

   Note: Some of the fields on the **Database Connection** screen are pre-populated based on the type of database you selected. If you edit these or any other fields on this screen, and then later decide to change the database type, the IUA might not populate the fields correctly. If this occurs, enter the correct field values as documented below, or exit and rerun the IUA to select the intended database type. If you are resuming after a failed upgrade, some pre-populated values might be disabled.

   - **JDBC Driver Class Name** — Verify that the pre-populated values are correct.
   - **JDBC Driver JAR Files** — Click **Select Jar** to browse to the appropriate driver files for your database type and version.
   - **Database JDBC URL** — Verify that the pre-populated value is accurate.
   - **Database Username and Password** — Enter the Deployment user name and password.
   - **Rules Schema Name** — Enter the new rules schema name.
   - **Data Schema Name** — For in-place upgrades, enter the existing data schema name.
   - **Customer Schema Name** — Optional: For in-place upgrades, enter the existing customer data schema name.
8. Click **Test Connection**. If the connection is not successful, review your connection information, correct any errors, and retest. When the connection is successful, click **Next** to choose how to apply the data schema.
9. Specify whether you will have your database administrator manually apply the DDL changes to the schema. These changes include the user-defined functions (UDF) supplied by Pega. By default, the IUA generates and applies the schema changes to your database.
   - To generate and apply the DDL outside the IUA, select **Bypass Automatic DDL Application** and continue the deployment. After you complete the deployment, manually generate and apply the DDL and UDF. For more information, see Optional: Generating and applying DDL and Optional: Installing user-defined functions.
   - To have the IUA automatically apply the DDL changes and the UDF, clear **Bypass Automatic DDL Application**.
10. Click **Next**.
11. Select the upgrade options and click **Next**:
    - Optional: Select **Update applications schema**. The Update Applications Schema utility updates all auto-generated tables with the schema changes in the latest base tables. You can also run the update applications schema utility later from the `prpcUtils.bat` or `prpcUtils.sh` script, or from Dev Studio. For information about using the Update Applications Schema utility, see the online help.
    - Optional: Select **Run rulebase cleanup** to permanently remove old rules. In most cases, removing older rules improves the general performance of the system. Running the cleanup script permanently removes rules older than the upgraded version.

- Optional: Select **Update existing applications** to modify your existing applications to support the upgraded version of the Pega Platform. The specific actions depend on your current version of PRPC. If you do not automatically update the applications as part of the IUA, follow the instructions in Updating existing applications to update the applications as part of the post-upgrade process.
- Optional: Select **Rebuild database indexes** to have the IUA to rebuild the database indexes after the rulebase loads. The IUA rebuilds the database indexes to ensure good performance in the upgraded system. The amount of time this process adds to the upgrade procedure depends on the size of your database.

12. Click **Start** to begin loading the rulebase. During the upgrade, the log window might appear inactive when the IUA is processing larger files.
13. Click **Back** to return to the previous screen, and then click **Exit** to close the IUA.

Pega Platform writes command-line output to a file in the **Pega-image\scripts\logs** directory.

Upgrading the rules schema from the command line

To use the command line, configure the **setupDatabase.properties** file and run either **upgrade.bat** or **upgrade.sh**. The Deployment user runs these scripts.

1. If you have not done so already, edit the **setupDatabase.properties** file.
   a) Open the **setupDatabase.properties** file in the scripts directory of your distribution image: *Directories.distributionDirectory***scripts\setupDatabase.properties**
   b) Configure the connection properties. Use the temporary upgrade schema name for the rules schema and data schema names. If your system includes a separate customer data schema, use the temporary upgrade schema name for the customer data schema too. See Properties file parameters for more information.
   c) Optional: If you are repeating a failed upgrade, configure the resume property:
      - To resume the upgrade from the last successful step, set `automatic.resume=true`.
      - To restart the upgrade from the beginning, set `automatic.resume=false`.
   d) Save and close the file.
2. Open a command prompt and navigate to the scripts directory.
3. Run either **upgrade.bat** or **upgrade.sh**.

The rulebase upgrade can take several hours, depending on the proximity of the database to the system running the script and available resources.

Pega Platform writes command-line output to a file in the **Pega-image\scripts\logs** directory.

Migrating to the new rules schema

Migrate to the new rules schema. Use the **migrate.bat** or **migrate.sh** script again for this process, but with different properties in the **migrateSystem.properties** file. The Deployment user runs these scripts.

- If the system can access both databases at the same time, follow the instructions in Migrating to the new rules schema when the system has access to both databases.
- If there is a firewall or other security restriction and the system can only access one database at a time, follow the instructions in Migrating to the new rules schema when the system has access to one database at a time (firewall).

Note: Pega strongly recommends that you use the migration script. The use of vendor tools to manage this step is not recommended. If you do not use the migrate script to migrate the schema, there are additional manual steps required that are not documented here.

This process depends on whether the system has access to both the original and temporary databases at the same time.

Note: To minimize the time required , run the migration scripts from the same data center as the database server.

Migrating to the new rules schema when the system has access to both databases

If your system has access to the temporary and original databases, run the migrate script once to migrate to the new rules schema.

1. Use a text editor to edit the `migrateSystem.properties` file in the scripts directory:

   Pega-image`\scripts\migrateSystem.properties`

2. Configure the source properties. For more information, see Migrate script properties.

```
# Connection Information
pega.source.jdbc.driver.jar=full path/DRIVER.jar
pega.source.jdbc.driver.class=database driver class
pega.source.database.type=database vendor type
pega.source.jdbc.url=database URL
pega.source.jdbc.username=Deployment user name
pega.source.jdbc.password=password
pega.source.rules.schema=temporary upgrade schema
pega.source.data.schema=temporary upgrade schema
```

3. Configure the target properties. See Properties file parameters for more information:

```
pega.target.jdbc.driver.jar=full path/DRIVER.JAR
pega.target.jdbc.driver.class=database driver class
pega.target.database.type=database vendor type
pega.target.jdbc.url=database URL
pega.target.jdbc.username=Deployment user name
pega.target.jdbc.password=password
pega.target.rules.schema=new rules schema
pega.target.data.schema=original data schema name
```

   Note: If `pega.target.data.schema` is blank, the rules schema is used by default.

4. Configure the bulkmover directory:

   `pega.bulkmover.directory=`*full path to output directory*

5. Configure a temporary directory:

   `pega.migrate.temp.directory=`*full path to temporary directory*

6. If the system has access to both the original and temporary databases, configure the operations to be performed by the utility as shown below:

```
pega.move.admin.table=false
pega.clone.generate.xml=true
pega.clone.create.ddl=true
pega.clone.apply.ddl=true
pega.bulkmover.unload.db=true
pega.bulkmover.load.db=true
pega.rules.objects.generate=true
pega.rules.objects.apply=true
```

7. Save the properties file.
8. Open a command prompt, and navigate to the scripts directory.
9. Type **`migrate.bat`** or **`./migrate.sh`** to run the script.

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

Migrating to the new rules schema when the system has access to one database at a time (firewall)

If the system can only access one database at a time (for example, if there is a firewall between the two servers), run the migration script twice: first on a system that can access the original source database, and then on a system that can access the temporary target database.

Make sure that the system that accesses the temporary database has access to the bulkmover directory and the DDL generated from the source database.

1. On a system that can access the original database, export rules from the original database.
   a) Use a text editor to edit the **`migrateSystem.properties`** file in the scripts directory:
   Pega-image\ `scripts\migrateSystem.properties`
   b) Configure the source properties. For more information, see Migrate script properties.

```
# Connection Information
pega.source.jdbc.driver.jar=/path-to-the-database-JAR-file/DRIVER.jar
pega.source.jdbc.driver.class=database driver class
pega.source.database.type=database vendor type
pega.source.jdbc.url=URL of database
pega.source.jdbc.username=Deployment user name
pega.source.jdbc.password=password
pega.source.rules.schema=temporary upgrade schema
pega.source.data.schema=temporary upgrade schema
```

   c) Configure the bulkmover directory:
   `pega.bulkmover.directory=`*full path to output directory*
   d) Configure a temporary directory:
   `pega.migrate.temp.directory=`*full path to temporary directory*
   e) Configure the operations to be performed by the utility as shown below:

```
pega.move.admin.table=false
pega.clone.generate.xml=true
pega.clone.create.ddl=false
pega.clone.apply.ddl=false
pega.bulkmover.unload.db=true
pega.bulkmover.load.db=false
pega.rules.objects.generate=false
pega.rules.objects.apply=false
```

   f) Save the properties file.
   g) Open a command prompt, and navigate to the scripts directory.

h) Type **`migrate.bat`** or **`./migrate.sh`** to export the rules.

2. On a system that can access the temporary database, import the rules to the temporary database.

   a) Copy the **`migrateSystem.properties`** file you created in step 1 to a system that can access the temporary database.

   b) Verify the target, bulkmover, and temporary directory properties. Set the target rules schema to the original rules schema:

   `pega.target.rules.schema=`*new rules schema name*

   c) Configure the operations to be performed by the utility as shown below:

   ```
   pega.admin.table=false
   pega.clone.generate.xml=false
   pega.clone.create.ddl=true
   pega.clone.apply.ddl=true
   pega.bulkmover.unload.db=false
   pega.bulkmover.load.db=true
   pega.rules.objects.generate=true
   pega.rules.objects.apply=true
   ```

   d) Save the properties file.

   e) Open a command prompt, and navigate to the scripts directory.

   f) Type **`migrate.bat`** or **`./migrate.sh`** to import the rules.

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

Optional: importing applications and other rule changes for highly available systems

If you plan to import applications or other rule changes in a high available system, doing so during an out-of-place upgrade rather than after the upgrade improves performance.

1. Open the **`prpcUtils.properties`** file in the Pega_HOME\scripts\utils directory.
2. Configure the following property:

   `import.oop.upgrade=true`
3. Save and close the file.
4. Import the application or other rule change. For more information, see the Dev Studio help for import tools.

Upgrading the data schema

The Deployment user runs a script to upgrade the data schema.

1. If you have not already done so, configure the connection properties. Use your current data schema name for **data.schema.name**. Use the new rules schema name for **rules.schema.name**. If you have an optional customer data schema separate from the Pega data schema, enter the **customerdata.schema.name.** For more information, see Editing the setupDatabase.properties file.

   ```
   # Connection Information
   pega.jdbc.driver.jar=/path-to-the-database-JAR-file/DRIVER.jar
   ```

```
pega.jdbc.driver.class=database driver class
pega.database.type=database vendor type
pega.jdbc.url=URL of the database
pega.jdbc.username=Deployment user name
pega.jdbc.password=password
rules.schema.name=new rules schema
data.schema.name=current data schema
customerdata.schema.name=optional-customer-data-schema
```

2. Shut down the application server and ensure that no other processes are using the data schema.
3. Open a command prompt, and navigate to the scripts directory.
4. Run **`upgrade.bat`** or **`./upgrade.sh`** for Linux, passing in the --dataOnly argument and true parameter, for example:

   ```
   upgrade.bat --dataOnly true
   ```

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

Performing an out-of-place upgrade with a single migration

Perform an out-of-place upgrade to minimize downtime. Use a single migration if limited server resources or security features are not an issue. The rules schema upgrade occurs out-of-place, and the data schema upgrade occurs in place.

Note: To further minimize downtime, use the High Availability features. For more information about High Availability, see the Pega Platform High Availability Administration Guide.

This upgrade involves three schemas:

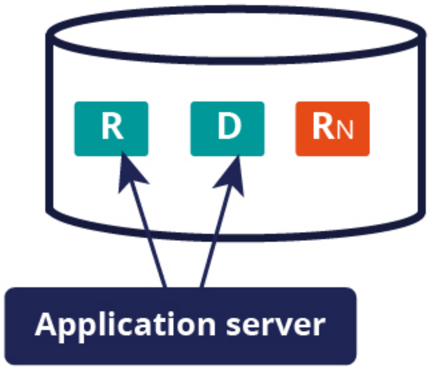
- Data schema - your current data schema. This will be your data schema after the upgrade as well.
- Rules schema - your current rules schema. This schema will be replaced after the upgrade.
- Temporary upgrade schema - a temporary schema on the same database used for staging the upgrade. This will include the rules and data tables during the upgrade and will become the new rules schema.

The generic process for upgrading a split-schema configuration can be summarized in the following steps:

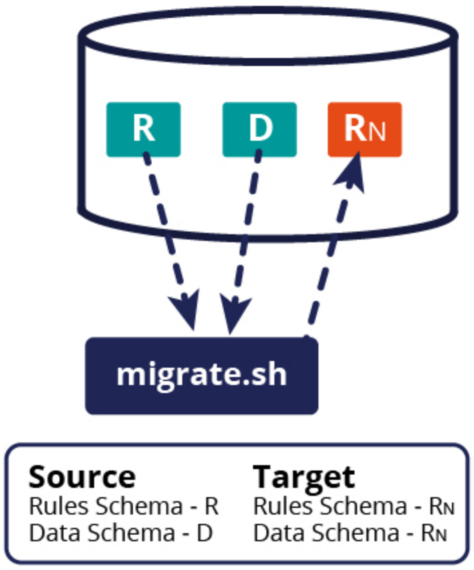
1. For upgrades to high-availability systems on Pega 7.x, disable rule creation on the rules schema. See For high availability systems: Disabling rule creation on the rules schema.

   Note: During the upgrade, you can still use the original system, but any rules created after the migration will be lost when you switch to the upgraded rules schema. The data schema retains all new data and work.

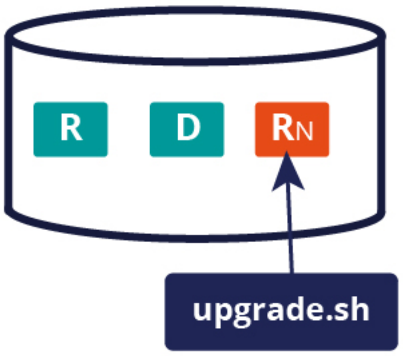
2. Create a new blank rules schema in your existing database. See Create a new rules schema.



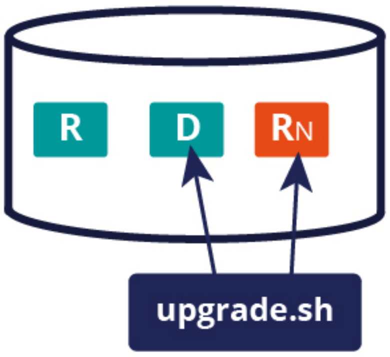
3. Migrate only the rules from the current rules schema to the new rules schema. See Migrating the existing rules schema.



4. Upgrade the new rules schema. See Upgrade methods for the migrated rules schema.



5. Shut down the existing system.
6. Upgrade the data schema. See Upgrading the data schema.



For high availability systems: Disabling rule creation on the rules schema

If you are using the recommended High Availability option, you can disable rule creation on the rules schema to speed the deployment. If you are not using the High Availability option, skip this section.

Before you deploy, commit all uncommitted hotfixes. After you begin the deployment, ensure that no changes to the rules, including hotfixes, are applied until after the deployment is complete.

1. Log in as a user with the PegaRULES:HighAvailabilityAdministrator role.
2. Navigate to **System > High Availability > HA Cluster Settings**.
3. Under **Cluster Upgrade**, select **Cluster Upgrading - Disable saving of rules**.
4. Click **Submit**.
5. Log out.

Create a new rules schema

Create a blank rules schema.

platform="db-Oracle">Create the schema names in all uppercase letters. Oracle requires schema names to be in all uppercase letters, or the deployment may not work correctly.

Migrating the rules schema with one database

Use the migrate script to migrate the rules tables and other required database objects from the existing rules schema to the new rules schema. The Deployment user performs the migrations.

Note: Pega strongly recommends that you use the migration script. The use of vendor tools to manage this step is not recommended. If you do not use the migrate script to migrate the schema, there are additional manual steps required that are not documented here.

To use the migrate script, complete the following steps:

Note: To minimize the time required, run the migration scripts from the same data center as the database server.

1. Use a text editor to edit the `migrateSystem.properties` file in the scripts directory of your distribution image:

   Pega-image`\scripts\migrateSystem.properties`

2. Configure the source properties. For more information, see Migrate script properties.

```
# Connection Information
pega.source.jdbc.driver.jar=full path/DRIVER.jar
pega.source.jdbc.driver.class=database driver class
pega.source.database.type=database vendor type
pega.source.jdbc.url=URL of database
pega.source.jdbc.username=Deployment user name
pega.source.jdbc.password=password
pega.source.rules.schema=original rules schema name
pega.source.data.schema=original data schema name
```

3. Configure the target properties. Leave the target data schema name blank:

```
pega.target.jdbc.driver.jar=full path/DRIVER.JAR
pega.target.jdbc.driver.class=database driver class
pega.target.database.type=database vendor type
pega.target.jdbc.url=database URL
pega.target.jdbc.username=Deployment user name
pega.target.jdbc.password=password
pega.target.rules.schema=new rules schema
```

   Note: If `pega.target.data.schema` is blank, the rules schema is used by default.

4. Configure the bulkmover directory.
   `pega.bulkmover.directory=`*full path to output directory*
5. Configure a temporary directory:
   `pega.migrate.temp.directory=`*full path to temporary directory*
6. Configure the operations to be performed by the utility as shown below:

```
pega.move.admin.table=true
pega.clone.generate.xml=true
pega.clone.create.ddl=true
pega.clone.apply.ddl=true
pega.bulkmover.unload.db=true
pega.bulkmover.load.db=true
pega.rules.objects.generate=false
pega.rule.objects.apply=false
```

7. Save the properties file.
8. Open a command prompt, and navigate to the scripts directory.
9. Type **`migrate.bat`** or **`./migrate.sh`** to run the script.

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

Upgrade methods for the migrated rules schema

Use one of these methods to upgrade the migrated rules schema:

- Installation and Upgrade Assistant
- Command line script: **`upgrade.bat`** or **`upgrade.sh`**

Note: To minimize the time required to upgrade, run the upgrade from the same data center as the database server.

Upgrading the rules schema by using the Installation and Upgrade Assistant

For a UI-based upgrade experience, use the Installation and Upgrade Assistant.

Because of the large volume of data, run the IUA on the same network as the database server. If this is not possible, run the tool on a system with fast, direct access to the database server. The Deployment user performs these steps.

The upgrade can last for several hours and the time can vary widely based on network proximity to the database server.

1. Double-click the **`PRPC_Setup.jar`** file to start the IUA.

   Note: If JAR files are not associated with Java commands on your system, start the IUA from the command line. Navigate to the directory containing the **`PRPC_Setup.jar`** file, and type `java -jar PRPC_Setup.jar`.

   The IUA loads and the Pega icon is displayed in your task bar.
2. Click **Next** to display the license agreement.
3. Review the license agreement and click **Accept**.
4. **Optional:** If you are resuming after a previous failed upgrade and the **Resume Options** screen is displayed, select either **Resume** or **Start Over**.
   - If you select **Resume**, the system uses the previously entered database configuration information to resume the upgrade from the last successful process. Continue these instructions at step 8.
   - If you select **Start Over**, continue at step 5 to reenter the configuration information.
5. On the **Installer Mode** screen, choose **Upgrade** and click **Next**.
6. Choose your database type and click **Next**.
7. Configure the database connection. The JDBC drivers allow the Pega Platform application to communicate with the database. Specify the new rules schema name for both the **Rules Schema Name** and **Data Schema Name** fields. For more information, see Appendix A — Properties files.

   Note: Some of the fields on the **Database Connection** screen are pre-populated based on the type of database you selected. If you edit these or any other fields on this screen, and then later decide to change the database type, the IUA might not populate the fields correctly. If this occurs, enter the correct field values as documented below, or exit and rerun the IUA to select the intended database type. If you are resuming after a failed upgrade, some pre-populated values might be disabled.
   - **JDBC Driver Class Name** — Verify that the pre-populated values are correct.
   - **JDBC Driver JAR Files** — Click **Select Jar** to browse to the appropriate driver files for your database type and version.
   - **Database JDBC URL** — Verify that the pre-populated value is accurate.
   - **Database Username and Password** — Enter the Deployment user name and password.
   - **Rules Schema Name** — Enter the new rules schema name.
   - **Data Schema Name** — For in-place upgrades, enter the existing data schema name.
   - **Customer Schema Name** — Optional: For in-place upgrades, enter the existing customer data schema name.
8. Click **Test Connection**. If the connection is not successful, review your connection information, correct any errors, and retest. When the connection is successful, click **Next** to choose how to apply the data schema.
9. Specify whether you will have your database administrator manually apply the DDL changes to the schema. These changes include the user-defined functions (UDF) supplied by Pega. By default, the IUA generates and applies the schema changes to your database.
   - To generate and apply the DDL outside the IUA, select **Bypass Automatic DDL Application** and continue the deployment. After you complete the deployment, manually generate and apply the DDL and UDF. For more information, see Optional: Generating and applying DDL and Optional: Installing user-defined functions.

- To have the IUA automatically apply the DDL changes and the UDF, clear **Bypass Automatic DDL Application**.

10. Click **Next**.
11. Select the upgrade options and click **Next**:
    - Optional: Select **Update applications schema**. The Update Applications Schema utility updates all auto-generated tables with the schema changes in the latest base tables. You can also run the update applications schema utility later from the **`prpcUtils.bat`** or **`prpcUtils.sh`** script, or from Dev Studio. For information about using the Update Applications Schema utility, see the online help.
    - Optional: Select **Run rulebase cleanup** to permanently remove old rules. In most cases, removing older rules improves the general performance of the system. Running the cleanup script permanently removes rules older than the upgraded version.
    - Optional: Select **Update existing applications** to modify your existing applications to support the upgraded version of the Pega Platform. The specific actions depend on your current version of PRPC. If you do not automatically update the applications as part of the IUA, follow the instructions in Updating existing applications to update the applications as part of the post-upgrade process.
    - Optional: Select **Rebuild database indexes** to have the IUA to rebuild the database indexes after the rulebase loads. The IUA rebuilds the database indexes to ensure good performance in the upgraded system. The amount of time this process adds to the upgrade procedure depends on the size of your database.
12. Click **Start** to begin loading the rulebase. During the upgrade, the log window might appear inactive when the IUA is processing larger files.
13. Click **Back** to return to the previous screen, and then click **Exit** to close the IUA.

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

Upgrading the rules schema from the command line

To use the command line, configure the **`setupDatabase.properties`** file and run either **`upgrade.bat`** or **`upgrade.sh`**. The Deployment user runs these scripts.

1. If you have not done so already, edit the **`setupDatabase.properties`** file.
   a) Open the **`setupDatabase.properties`** file in the scripts directory of your distribution image: *Directories.distributionDirectory*`\scripts\setupDatabase.properties`
   b) Configure the connection properties. Use the temporary upgrade schema name for the rules schema and data schema names. If your system includes a separate customer data schema, use the temporary upgrade schema name for the customer data schema too. See Properties file parameters for more information.
   c) Optional: If you are repeating a failed upgrade, configure the resume property:
      - To resume the upgrade from the last successful step, set `automatic.resume=true`.
      - To restart the upgrade from the beginning, set `automatic.resume=false`.
   d) Save and close the file.
2. Open a command prompt and navigate to the scripts directory.
3. Run either **`upgrade.bat`** or **`upgrade.sh`**.

The rulebase upgrade can take several hours, depending on the proximity of the database to the system running the script and available resources.

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

Optional: importing applications and other rule changes for highly available systems

If you plan to import applications or other rule changes in a high available system, doing so during an out-of-place upgrade rather than after the upgrade improves performance.

1. Open the **`prpcUtils.properties`** file in the Pega_HOME\scripts\utils directory.
2. Configure the following property:
   ```
   import.oop.upgrade=true
   ```
3. Save and close the file.
4. Import the application or other rule change. For more information, see the Dev Studio help for import tools.

Upgrading the data schema

The Deployment user runs a script to upgrade the data schema.

1. If you have not already done so, configure the connection properties. Use your current data schema name for **data.schema.name**. Use the new rules schema name for **rules.schema.name**. If you have an optional customer data schema separate from the Pega data schema, enter the **customerdata.schema.name.** For more information, see Editing the setupDatabase.properties file.
   ```
   # Connection Information
   pega.jdbc.driver.jar=/path-to-the-database-JAR-file/DRIVER.jar
   pega.jdbc.driver.class=database driver class
   pega.database.type=database vendor type
   pega.jdbc.url=URL of the database
   pega.jdbc.username=Deployment user name
   pega.jdbc.password=password
   rules.schema.name=new rules schema
   data.schema.name=current data schema
   customerdata.schema.name=optional-customer-data-schema
   ```
2. Shut down the application server and ensure that no other processes are using the data schema.
3. Open a command prompt, and navigate to the scripts directory.
4. Run **`upgrade.bat`** or **`./upgrade.sh`** for Linux, passing in the --dataOnly argument and true parameter, for example:
   ```
   upgrade.bat --dataOnly true
   ```

Pega Platform writes command-line output to a file in the **Pega-image`\scripts\logs`** directory.

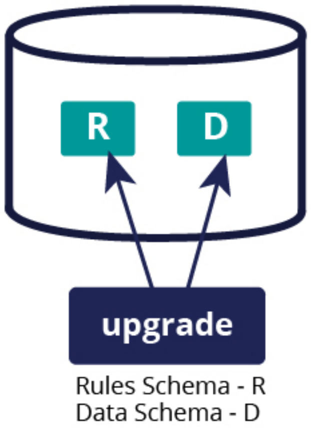
Performing an in-place upgrade

Perform an in-place upgrade on systems that do not require minimal downtime.

For more information about in-place and out-of-place upgrades, see In-place or out-of-place upgrades and single or double data migration

The generic process for performing an in-place upgrade can be summarized in the following steps:

1. Shut down the system.
2. Upgrade the rules schema. See Upgrade methods for the rules schema.



3. Restart the system.

Upgrade methods for the rules schema

Use one of these methods to upgrade the rules schema in place:

- Installation and Upgrade Assistant
- Command line script: **`upgrade.bat`** or **`upgrade.sh`**

Upgrading the rules schema by using the Installation and Upgrade Assistant

For a UI-based upgrade experience, use the Installation and Upgrade Assistant.

Because of the large volume of data, run the IUA on the same network as the database server. If this is not possible, run the tool on a system with fast, direct access to the database server. The Deployment user performs these steps.

The upgrade can last for several hours and the time can vary widely based on network proximity to the database server.

1. Double-click the **`PRPC_Setup.jar`** file to start the IUA.

   Note: If JAR files are not associated with Java commands on your system, start the IUA from the command line. Navigate to the directory containing the **`PRPC_Setup.jar`** file, and type `java -jar PRPC_Setup.jar`.

   The IUA loads and the Pega icon is displayed in your task bar.
2. Click **Next** to display the license agreement.

3. Review the license agreement and click **Accept**.
4. **Optional:** If you are resuming after a previous failed upgrade and the **Resume Options** screen is displayed, select either **Resume** or **Start Over**.
   - If you select **Resume**, the system uses the previously entered database configuration information to resume the upgrade from the last successful process. Continue these instructions at step 8.
   - If you select **Start Over**, continue at step 5 to reenter the configuration information.
5. On the **Installer Mode** screen, choose **Upgrade** and click **Next**.
6. Choose your database type and click **Next**.
7. Configure the database connection. The JDBC drivers allow the Pega Platform application to communicate with the database. Specify the new rules schema name for both the **Rules Schema Name** and **Data Schema Name** fields. For more information, see Appendix A — Properties files.

   Note: Some of the fields on the **Database Connection** screen are pre-populated based on the type of database you selected. If you edit these or any other fields on this screen, and then later decide to change the database type, the IUA might not populate the fields correctly. If this occurs, enter the correct field values as documented below, or exit and rerun the IUA to select the intended database type. If you are resuming after a failed upgrade, some pre-populated values might be disabled.

   - **JDBC Driver Class Name** — Verify that the pre-populated values are correct.
   - **JDBC Driver JAR Files** — Click **Select Jar** to browse to the appropriate driver files for your database type and version.
   - **Database JDBC URL** — Verify that the pre-populated value is accurate.
   - **Database Username and Password** — Enter the Deployment user name and password.
   - **Rules Schema Name** — Enter the new rules schema name.
   - **Data Schema Name** — For in-place upgrades, enter the existing data schema name.
   - **Customer Schema Name** — Optional: For in-place upgrades, enter the existing customer data schema name.
8. Click **Test Connection**. If the connection is not successful, review your connection information, correct any errors, and retest. When the connection is successful, click **Next** to choose how to apply the data schema.
9. Specify whether you will have your database administrator manually apply the DDL changes to the schema. These changes include the user-defined functions (UDF) supplied by Pega. By default, the IUA generates and applies the schema changes to your database.
   - To generate and apply the DDL outside the IUA, select **Bypass Automatic DDL Application** and continue the deployment. After you complete the deployment, manually generate and apply the DDL and UDF. For more information, see Optional: Generating and applying DDL and Optional: Installing user-defined functions.
   - To have the IUA automatically apply the DDL changes and the UDF, clear **Bypass Automatic DDL Application**.
10. Click **Next**.
11. Select the upgrade options and click **Next**:
    - Optional: Select **Update applications schema**. The Update Applications Schema utility updates all auto-generated tables with the schema changes in the latest base tables. You can also run the update applications schema utility later from the `prpcUtils.bat` or `prpcUtils.sh` script, or from Dev Studio. For information about using the Update Applications Schema utility, see the online help.

- Optional: Select **Run rulebase cleanup** to permanently remove old rules. In most cases, removing older rules improves the general performance of the system. Running the cleanup script permanently removes rules older than the upgraded version.
- Optional: Select **Update existing applications** to modify your existing applications to support the upgraded version of the Pega Platform. The specific actions depend on your current version of PRPC. If you do not automatically update the applications as part of the IUA, follow the instructions in Updating existing applications to update the applications as part of the post-upgrade process.
- Optional: Select **Rebuild database indexes** to have the IUA to rebuild the database indexes after the rulebase loads. The IUA rebuilds the database indexes to ensure good performance in the upgraded system. The amount of time this process adds to the upgrade procedure depends on the size of your database.

12. Click **Start** to begin loading the rulebase. During the upgrade, the log window might appear inactive when the IUA is processing larger files.
13. Click **Back** to return to the previous screen, and then click **Exit** to close the IUA.

Pega Platform writes command-line output to a file in the **Pega-image\scripts\logs** directory.

Upgrading the rules schema in place from the command line

To use the command line, configure the **setupDatabase.properties** file and run either **upgrade.bat** or **upgrade.sh**. The Deployment user runs these scripts.

1. If you have not done so already, edit the **setupDatabase.properties** file.
   a) Open the **setupDatabase.properties** file in the scripts directory of your distribution image: *Directories.distributionDirectory***\scripts\setupDatabase.properties**
   b) Configure the connection properties. Use the temporary upgrade schema name for both the rules schema and data schema names. See Properties file parameters for more information.
   c) Optional: If you have a separate customer data schema, set the target schema name:
   `pega.target.customerdata.schema=`*current-customer-data-schema*
   d) Optional: If you are repeating a failed upgrade, configure the resume property:
   - To resume the upgrade from the last successful step, set `automatic.resume=true`.
   - To restart the upgrade from the beginning, set `automatic.resume=false`.

   e) Save and close the file.
2. Open a command prompt and navigate to the scripts directory.
3. Run either **upgrade.bat** or **upgrade.sh**.

The rulebase upgrade can take several hours, depending on the proximity of the database to the system running the script.

Pega Platform writes command-line output to a file in the **Pega-image\scripts\logs** directory.

Post-upgrade configuration

This section describes additional tasks to perform after you finish upgrading the system.

Upgrading from PRPC 6.1 SP2 and earlier: updating ruleset columns

When upgrading from any release prior to PRPC 6.2, you must run additional scripts to update the ruleset columns.

Depending upon the database platform and the size of the database the script might require significant time and resources to execute.

1. Navigate to **ResourceKit\AdditionalUpgradeScripts** and locate the scripts for your database:
   - *database*`_rulesetversion_columns_data.sql`
   - *database*`_rulesetversion_columns_rules.sql`

   For example, `oracle_rulesetversion_columns_data.sql`
2. Run the scripts:
   - For split schema environments:
     - Run *database*`_rulesetversion_columns_data.sql` against the data schema.
     - Run *database*`_rulesetversion_columns_rules.sql` against the rules schema.
   - For single schema environments, run both scripts against the single schema.

For Docker, multiple VMs, or multiple NICs: Setting the public address

If the cluster is set up in Docker, uses separate virtual machines (VMs), or multiple network interfaces (NICs), set the public address in the `prconfig.xml` file for each Pega Platform node.

1. Open the `prconfig.xml` configuration file in the prweb/WEB-INF/classes subdirectory of the application server directory. For more information, see the Pega Community article *How to change prconfig.xml file settings*.
2. Modify the `prconfig.xml` file. Add the following setting to set the public address:

   `<env name=" identification/cluster/public/address" value="` *IP address* `" />`

   For example, if the IP address of the computer on which you run the Pega Platform node is 10.254.34.210, add the following setting:

   ```
   <env name=" identification/cluster/public/address" value="10.254.34.210" />
   ```

   The new setting controls the address that is used by the Pega Platform node.
3. Save and close the `prconfig.xml` file.
4. Repeat steps 1 to 3 for the remaining nodes.

Reconfiguring the application server

To use the upgraded rules schema, you must reconfigure the application server. The process is different for each application server.

Apache Tomcat: Defining default schemas

If you performed an out-of-place upgrade on Apache Tomcat, redefine your default schema names. Define the default schemas in the **`context.xml`** file.

Follow these steps to define the default schemas:

1. Open **`context.xml`**.
2. In the <context> element, insert the following lines to define the default schemas. Replace *RULES* with your rules schema name and replace *DATA* with your data schema name. For single-schema systems, use the rules schema name for both *RULES* and *DATA*.

```
<Environment name="prconfig/database/databases/PegaRULES/defaultSchema"
value="RULES" type="java.lang.String" />
<Environment name="prconfig/database/databases/PegaDATA/defaultSchema"
 value="DATA"
type="java.lang.String" />
```

3. **Optional:** If your customer data schema is different than your PegaDATA schema, insert the following entry to specify the customer data schema name. Replace *customer-data-schema* with your customer data schema name.

```
<Environment name="prconfig/database/databases/CustomerData/defaultSchema"
value="customer-data-schema" type="java.lang.String"/>
```

4. Save the file.

Redeploying the Pega Platform WAR or EAR file

Remove the existing **`prweb.war`** from your application server and deploy the new file.

Do not start the redeployed application while the rulebase deployment is running. By default, your application server might start the application automatically when they are deployed. If you deploy and start the application before creating the database, the application generates an error and fail to start. This error is not harmful, and you can restart the application successfully when the database is available.

Note: When the server restarts after the application deploys, the first node you bring up becomes the default search node.

Apache Tomcat: Redeploying Pega Platform

1. Make sure that **`prweb.war`** or the EAR file for your application server is not running.
2. Remove each of the current versions of the applications.
   a) In the **Tomcat Web Application Manager** page, find the row for each application and select **Undeploy**.
   b) In the **WEBAPPS** directory, delete any remaining folders or WAR files.
3. Copy the **`prweb.war`** file from the **Pega-image`\archives\`** directory to the *Tomcat_home***`\webapps\`** directory.

4. Restart the application server.
5. Shut down the server and delete the **`prweb.war`** file from the *Tomcat_home*`\webapps\` directory to prevent Tomcat from redeploying the application each time the server restarts.
6. Verify that any third-party or custom JAR files that you installed with the applications are still in place on the application server. Restore any JAR files that were deleted when the Pega Platform redeployed.

For upgrades from Pega 7.x: Enabling rule creation on the production system

If you are upgrading from PRPC 5.x or PRPC 6.x, or if you did not disable rule creation, skip this section.

For upgrades from Pega 7.x, enable rule creation on the production system:

1. Log in as a user with the PegaRULES:HighAvailabilityAdministrator role.
2. Navigate to **System > High Availability> HA Cluster Settings**.
3. Under **Cluster Upgrade**, clear **Cluster Upgrading - Disable saving of rules**.
4. Click **Submit**.
5. Log out.

Upgrades from 7.2.2 and earlier: Port Apache logging file customizations to the new logging file

Starting with Pega 7.3, the Pega Platform uses the Apache Log4j 2 logging service. Prior to Pega 7.3, the Pega Platform used Apache Log4j 1. Because of the change to the logging service, the name of the logging configuration file has changed from **`prlogging.xml`** to **`prlog4j2.xml`** and the format has changed considerably. If you customized your **`prlogging.xml`** file, port the customizations to the new **`prlog4j2.xml`** file. If you do not edit the new **`prlog4j2.xml`** file, the Pega Platform uses the default **`prlog4j2.xml`** file and disregards your customized **`prlogging.xml`** file. For upgrades to systems that were using the default logging configuration, no changes are needed.

Pega Platform supports all appenders and layouts supported by Apache Log4j 2. The following commonly-customized appenders define the logging file locations, names, and archiving policy:

- **RollingRandomAccessFile** – Configures the **`PegaRULES.log`** file that includes all logs except alerts
- **RollingRandomAccessFileAlert** – Configures the **`PegaRULES-ALERT.log`** file that includes all performance alerts
- **RollingRandomAccessFileAlertSecurity** – Configures the date and time-stamped **`PegaRULES-ALERTSECURITY`** log file that includes all security alerts

For more information about customizing these appenders, see the Apache Log4j 2 documentation.

Restarting Pega Platform

Restarting the Pega Platform

1. Stop and restart the application server.
2. Ensure that the Pega Platform **`prweb.war`** or **`prpc_ *.ear`** file has started.
3. Access the Pega Platform through a browser. Enter the URL for the Pega Platform application:

`http://` *server:portnumber/context_root*

For example: `http://prpc-server:8080/prweb`

4. Log in as administrator@pega.com.

The **What's New** section of the page includes a welcome message and links to application development tools.

Locking and rolling ruleset versions

If your system includes rule set names using ruleset prerequisite validation instead of application-based validation, lock your existing ruleset and roll them into new versions before continuing development. This ensures that future development is based on the upgraded rulesets and that your applications consistently reference the latest features.

Note: The upgrade process automatically upgrades any prerequisite for Pega-ProCom to the highest version of the ruleset in the system.

1. In the header of Dev Studio, click **Configure** > **Application** > **Structure** > **Referencing Applications**.
2. For each application, click **Lock and Roll** to display the application page.
3. Select **+** next to **Prerequisites** to see the ruleset version prerequisites for the application ruleset. Click the name of the ruleset version to open and modify its rule form.
4. Select the **Lock** box and select **Update my application to include the new ruleset versions**.
5. Enter the **Password** for this ruleset version and select the **Roll** box.
6. In the **Roll to Version** field, enter a new ruleset version or accept the default version which increments the current version by one.
7. In the **NEW** section of the **Prerequisites** verify the list of ruleset prerequisites. In particular, verify that the lowest ruleset points to Pega Platform 8.1.
8. Click **Run** to apply the changes to your ruleset.
9. Repeat these steps for each application in your system.

Upgrading from Pega 7.1.7 through 7.2.1: Rebuilding search indexes

Elasticsearch has been updated to version 5.6.9. If you are upgrading from Pega 7.2.2 or higher, no action is required. You can continue to use your existing search indexes.

If you are upgrading from Pega 7.1.7 through 7.2.1, you must discard your existing indexes and build new indexes in a new empty index directory. Use the Search landing page or the full text indexer to create the search indexes. For more information, see the help.

Optional: Upgrading from Pega 7.1.6 and earlier: Configuring the default search nodes and storage directory

If you are upgrading from Pega 7.1.6 or earlier, you can manually build the Elasticsearch indexes and configure the search index host node settings to configure the default initial search node and index storage directory.

Starting in Pega 7.1.7, the underlying platform for full-text search transitioned from Lucene to Elasticsearch. Elasticsearch provides a more robust, fault-tolerant search capability and does not require manual configuration of switchover activities. Existing search customizations through Pega Platform APIs are intact and used exactly the same way with Elasticsearch; only the search query generation changes from Lucene to Elasticsearch.

Indexing starts when you start the application server. The first node that starts after the deployment becomes the default initial search node. You can configure a different search node and can also configure multiple search nodes.

The default index directory is PegaSearchIndex in your temporary directory. The contents of this directory might be deleted. As a best practice, store your indexes in a more permanent location accessible to all search nodes.

Follow these steps to build the indexes, configure search nodes, and update the index directory:

1. Check your directory sizes. Ensure that the directories for all Elasticsearch host nodes have sufficient free space to hold the Elasticsearch indexes.
   - Ensure that the host node directories have sufficient free space to hold the Elasticsearch indexes. Elasticsearch indexes are approximately three times the size of the Lucene indexes.
   - Ensure that the directory for the initial host node has sufficient space to initially hold both the Lucene index and the Elasticsearch index.
2. Use the prpcUtils tool to reindex the rules:

   a) On the node that you want to index, open the **`prpcUtils.properties`** file in the Pega_HOME\scripts\utils directory.

   b) Configure the connection properties. For more information about parameter values, see Properties file parameters.

   ```
   # Connection Information
   pega.jdbc.driver.jar=/path-to-the-database-JAR-file/DRIVER.jar
   pega.jdbc.driver.class=database driver class
   pega.database.type=database vendor type
   pega.jdbc.url=URL of the database
   pega.jdbc.username=Deployment username
   pega.jdbc.password=password
   ```

   c) Optional. Configure a directory to store the new indexes; by default, indexes are created in the directory specified in the `user.temp.dir` property.

   `indexing.indexdirectory=`*full-path*`/index/`

   d) Configure the indexing type parameter in the **SETTINGS FOR FULL TEXT INDEXER TOOL** section; leave all other indexing parameters commented out:

   `indexing.indextype=`*Rule*

   e) Save and close the **`prpcUtils.properties`** file.

f) Run the **`prpcUtils.bat`** or **`prpcUtils.sh`** script to reindex the rules. For example:

```
prpcUtils.bat indexing
```

3. Repeat step 2 to reindex the data files. Set the index type to Data:

   ```
   indexing.indextype=Data
   ```

4. Repeat step 2 to reindex the work files. Set the index type to Work:

   ```
   indexing.indextype=Work
   ```

5. Use Dev Studio to delete the existing index nodes:

   a) In the header of Dev Studio, click **Configure** > **System** > **Settings** > **Search**.

   b) Expand **Search Index Host Node Setting**.

   c) Click the **X** to the right of each node to delete all existing nodes.

   d) Click **Submit**.

6. Use Dev Studio to add the host nodes. The system replicates the indexes on the new nodes.

   Note:

   - Configure a minimum of two Elasticsearch host nodes. Pegasystems recommends that you configure a minimum of three nodes for maximum fault tolerance. You might need more than three nodes depending on the size of your cluster.
   - You can specify that a node is either always an index host node or that it never becomes an index host node even if it is the first node that is started after installation using the `–Dindex.directory` JVM setting. To specify that a node is always an index host node specify the directory name. To specify that a node is never an index host node, leave this setting blank. For more information about configuring index host nodes, see Managing Elasticsearch index host nodes outside of the Search landing page on the Pega Community.

   a) In the header of Dev Studio, click **Configure** > **System** > **Settings** > **Search**.

   b) Expand **Search Index Host Node Setting**.

   c) Enter the information for the primary host node. The first node you enter is the node on which Elasticsearch indexes will be built.

   - Enter the Search index host node ID on which you built the indexes.

     For example:

     ```
     /dsk01/tomcat7/system7/work/Catalina/localhost/prweb/PegaSearchIndex
     ```

   - In the Search index file directory, enter the directory in which prpcUtils saved the indexes.

   d) **Optional:** Add any needed additional host nodes.

   e) Verify the **Search Index Host Node ID** and the **Search Index File Directory**.

   f) Expand **Automated Search Alerts**, and enable **Automatically Monitor Files**.

   g) Click **Submit** to save the settings.

7. To enable communication between Elasticsearch host nodes in the cluster, open a TCP port in the range 9300-9399 on each node. (By default, Elasticsearch uses port 9300.) These ports are used for internal node-to-node communication only, and should not be externally accessible.

Do not stop or bring down the default node until the search indexes build completely. The Search Landing Page displays the status. After the search indexes are completely built, you can change the default settings.

Final Rules Conflict Report

The Final Rules Conflict Report lists rules in your system that reference Pega rules that have been made Final in this release. The rules listed vary depending on the specific release from which you are upgrading.

Rules that are marked Final can no longer be overridden. If you have custom rules in your applications that override default rules and your custom rules are final, your existing rules will continue to execute correctly. However, you will not be able to modify them after you upgrade to the new RuleSet.

Note: If you modify and try to save a custom rule that overrides a Final rule, validation fails and you receive an error message. To resolve the conflict, you must delete application rules that override Final system rules, and replace the functionality in your application with other rules. If you are unsure how to respond to a Final rule, see the Support area on the Pega Community.

To run the report, select **Dev Studio** > **System** > **Release** > **Upgrade** > **Final Conflicts**.

For upgrades from Pega 7.2.2 and earlier: Adopting APIs and rules for Pega Survey

If you are upgrading from Pega 7.2.2 or earlier, upgrade your application to use the latest survey features. Skip this section if you are upgrading from a version that is later than Pega 7.2.2, or if your application does not use Pega Survey.

You cannot revert surveys to their original implementation after you adopt the rules and APIs that are provided in the `Pega-Survey` ruleset.

For each application that uses survey capabilities, repeat the following steps:

1. Remove the reference to the legacy `PegaSurvey` ruleset.
    a) In the header of Dev Studio, click **[Your application name]** > **Definition** to open the Application form.
    b) In the **Application rulesets** section, delete the entry for the `PegaSurvey` ruleset.
    c) Click **Save**.
2. Upgrade your overrides and custom rules that rely on standard rules that have been renamed to find rules with invalid references.
    a) In the header of Dev Studio, click **Configure** > **Application** > **Tools** > **Validation**.
    b) In the **Select Application** list, select the name of your application.
    c) Click **Run Validation**.
    d) Review the list of rules with invalid references, and resolve each invalid reference by performing one of the following tasks:
        - Redirect the invalid reference to a valid rule in the `Pega-Survey` ruleset.

          Because only prefixes were added to the names of standard rules, you can inspect the ruleset for a rule name that is similar to your invalid reference.
        - Recreate your override by copying the renamed version of the rule in the `Pega-Survey` ruleset.

          Ensure that all references to your original override are redirected to your new override, before you delete the original override.

    e) Manually review and upgrade your application for references, such as Java steps in an activity, that are not detected by the validation tool.

3. Upgrade the rules that support the surveys in your application.
    a) Click **Dev Studio** > **System** > **Release** > **Upgrade** > **Validate** to access tools for validation.
    b) Click **Revalidate and Save**.
    c) In the **Update Rule Forms** dialog box, enter values in the fields to perform validation on the following classes:
        - `Rule-PegaQ-Question`
        - `Rule-PegaQ-QuestionCollection`
        - `Rule-PegaQ-QuestionGroup`
        - `Rule-PegaQ-Questionnaire`

    For more information about the options that you can choose while running the Revalidate and Save tool, see the Pega Platform help.
4. Find the surveys in your application that run on an embedded page instead of the context, or primary page, of the parent flow.
    a) In the Application Explorer, expand **Survey > Survey** to display a list of surveys in your application.
    b) Click a survey name to open the Survey form.
    c) Click **Actions** > **View references**to find the flow that calls your survey.
    d) Click the **Open** icon next to the flow name.
    e) On the flow diagram, inspect the configuration of the Subprocess shape that calls your survey.
    f) If the **Define flow** field is set to **On embedded page**, note the value in the **Page property** field.
    g) Repeat steps b through f for each survey in your application.
5. Customize the upgrade utility so that it finds and edits the correct pages for in-flight surveys. If you do not have any surveys that run on embedded pages, you can skip this step.
    a) Find the `Work-.pyUpgradeSurveyProperties` data transform by searching for it or by using the Application Explorer.
    b) Save a copy of the rule to an unlocked ruleset version in your application.
    c) On the **Definition** tab of the Data Transform form, use the **Update Page** action to set the current page to the embedded page that you noted from step 4.
    d) Enclose the **Update Page** action with a **When** action if only some surveys run on the embedded page.
    e) Repeat steps c and d for each embedded page that you noted from step 4.
    f) Click **Save**.
6. Run the upgrade utility for in-flight surveys.
    a) Click **Dev Studio** > **System** > **Release** > **Upgrade** > **Upgrade Tools**.
    b) Click **Update Survey Work Objects**.
    c) In the **Upgrade survey work objects** dialog box, select the check box next to each class that defines a survey.
    d) Click **Run utility**.
7. Correct references to deprecated APIs. For more information about deprecated APIs and the APIs that supersede them, see the release notes for Pega Platform 7.3.

Scheduling column population jobs

Upgrading the Pega Platform exposes the pxApplication column on all Work object tables. Because the expose process may temporarily degrade performance, it is a best practice to schedule this process to run during off-peak hours.

You can specify a start time for the job and a timeout length. When the job reaches the end of the timeout, it stops and restarts the next day at the same point. The job continues to recur until all tables are upgraded. The number of times the job runs depends on the amount of data in your work tables.

Users must have the SysAdm4 role to schedule column population jobs.

1. Configure the start time for the column population:
   a. In the Explorer panel, click **Records > SysAdm > Agent Schedule**.
   b. Filter the list. In the **Key Contains** box at the top of the screen, enter `Pega-ImportExport` and click **Run**.
   c. Click any instance of **Pega-ImportExport**.

      Note: You can edit any instance of Pega-ImportExport. Check the **NODE NAME** field for information about the node associated with each instance.
   d. Click **Advanced** next to the pxAutomaticColumnPopulation activity line.
   e. Enter the start time and schedule for the job and click **OK**.
   f. In the `Column Pattern` area, select **Recurring**.
   g. Select **Enabled**.
   h. Click **Save**.
2. Optional: Configure the timeout to set the maximum length of time the column population job will run.
   a. In the Explorer panel, click **Records > SysAdm > Dynamic System Settings**.
   b. In the **Owning Ruleset** field, click the filter icon, enter `ImportExport`, and click **Apply**.
   c. Double-click **AutomaticColumnOptimization/Timeout**.
   d. Enter the timeout value in minutes.

      Note: The default setting is 120 minutes. Change the timeout to reflect the duration of your off-peak schedule. For example, if your lowest usage occurs from 9 P.M. EST until 4 A.M. EST, start this job at 9 P.M. and have it run for 420 minutes.
   e. Click **Save**.

Upgrading from Pega 7.2.2 or earlier: Upgrading access role names to enable notifications

When upgrading from any release prior to Pega 7.2.2, you must upgrade all the user access role names of an application with specific classes so that users can receive notifications.

Edit the user access role names for these classes:

- `Data-Notification-Parameters`
- `Pega-Notification`
- `Data-Notification-Recipient`
- `Data-Preference-Operator`

To save time, clone any access role name that contains the preceding classes and assign it to application users instead of updating the access role names manually. For more information on how to clone an access role name, see the help.

To edit access role names:

1. In the Records Explorer, click **Security > Access Role Name**.
2. Open the access role name that needs to be edited.
3. Click the **Plus** icon to open the **Add Access Role Object** dialog box.
4. In the **Class** field, enter the class name that you want to add to the access role name.
5. Under Access Control, enter `5` in all the fields to provide access to this access role name.
6. Click **Save**.
7. Perform steps 3 through 6 for each of the remaining classes.

Upgrades from 7.2.2 and earlier: Enabling access to environmental information

Prior to Pega 7.3, all roles included access to environmental information for the current node. This information can include version numbers of third-party platforms and JVM information. This access appears as a flaw in some security audits. With Pega 7.3, the new `@baseclass.pxViewSystemInfo` privilege controls access to environmental information. Only the PegaRULES:SysAdm4 role has this privilege by default.

After upgrading from Pega 7.2.2 or earlier, add the `@baseclass.pxViewSystemInfo` privilege to all system administrator roles that need access to environmental information.

1. In the header of Dev Studio, click **Configure** > **Org & Security** > **Tools** > **Security** > **Role Names**.
2. In the pop-up window that displays roles, click the role that you want to edit.
3. In the Dev Studio click the `@baseclass` class in the **Access Class** column.
4. In the **Privileges** section, click the **Plus** icon and select the `pxViewSystemInfo` privilege in the **Name** column.
5. Enter `5` for the production level in the **Level** column. Production level 5 provides the highest security.
6. Click **Submit**.
7. Repeat steps 1 - 6 for each role that requires modification.

Optional: Leveraging the current UI Kit rules

The UI Kit ruleset contains rules and skins that you can use for building or customizing user interfaces for your applications. Employing the UI Kit ruleset provides you with the latest standard user interface elements, including templates and icons. Add the latest version of the UIKit application as a built-on application to take advantage of the latest features and styles.

Enabling operators

Pega Platform deployment security requires an administrator to enable new operators shipped with Pega Platform and requires password changes after the first login.

The administrator and new operators shipped with Pega Platform must change their passwords when they first log in:

- Batch@pega.com
- DatabaseAdmin@pega.com
- ExternalInviteUser
- IntSampleUser
- PRPC_SOAPOper
- PortalUser@pega.com
- UVUser@pega.com
- External

For more information about changing the administrator password, see Logging in and changing the administrator password.

1. In the header of Dev Studio, click **Configure** > **Org & Security** > **Authentication** > **Operator Access**.
2. In the **Disabled operators** list, click the link for the Pega-provided operator that you want to enable. The following standard operators are installed but disabled by default. When these standard operators first log on, they are required to change their passwords. Enable only those operators you plan to use:
   - Batch@pega.com
   - DatabaseAdmin@pega.com
   - ExternalInviteUser
   - IntSampleUser
   - PRPC_SOAPOper
   - PortalUser@pega.com
   - UVUser@pega.com
   - External
3. On the **Edit Operator ID** page, on the Security tab, select **Force password change on next login** and clear **Disable Operator**.
4. Select **Update password**.
5. Enter a password that conforms to your site standards and click **Submit**.
6. Click **Save** and close the operator page.
7. Repeat steps 2 through 6 for the remaining operators.

Running upgrade utilities

The Pega Platform includes several upgrade utilities to help you to upgrade your application to use new features. Run all of the upgrade utilities, even though some utilities might not return results for your application:

1. Log in as the administrative user.
2. In the header of Dev Studio, click **Configure** > **System** > **Release** > **Upgrade** > **Upgrade Tools**.
3. Expand **General Utilities**.
4. Click each utility and then click **Run utility**.

Cleaning up unused tables

Pegasystems recommends that you drop unused rules tables in the data schema after deploying a split-schema. If you have only one database, also drop unused data tables in the rules schema.

1. Verify that you have the correct privileges to view and edit the **Optimize Schema** landing page. Set these parameters to true:
   - ViewAndOptimizeSchema
   - Dynamic System Setting (DSS) databases/AutoDBSchemaChanges
2. In the header of Dev Studio, click **Configure** > **System** > **Database** > **Optimize Schema**.
3. Select the PegaDATA database.
4. Click **view the unused tables** to display a list of Pega Platform tables without class mappings. Either select the ones you want to delete and click **Proceed with Changes** to have Pega Platform drop the tables, or drop them manually in your database.
5. Repeat steps 3 and 4 for the PegaRULES database.

Upgrading your custom applications

If you did not opt to upgrade your existing applications automatically, Run the Update Existing Application utility to ensure that your existing applications take advantage of new functionality in Pega Platform. Run the utility first on your development system and test the changes. Then, run the utility again on the production system. The specific actions required for your application depend on your current version.

The utility lists the actions that will be performed, the number of records that will be modified, and an estimate of how long each action will take.

1. In the header of Dev Studio, click **Configure** > **System** > **Release** > **Upgrade** > **Upgrade Existing Applications**.
2. If any actions are listed, click **Run** to start the utility.
3. Test the application. If the test results are acceptable, repeat these steps on your production system.

Upgrading your application schema

If you did not opt to upgrade your application schema automatically, run the Upgrade Application Schema utility.

1. In the header of Dev Studio, click **Configure** > **System** > **Release** > **Upgrade** > **Upgrade Applications Schema**.
2. If any actions are listed, click **Run** to start the utility.
3. Test the application. If the test results are acceptable, repeat these steps on your production system.

Review log files

The upgrade creates a series of log files in the **Pega-image** \scripts\logs directory. After you upgrade, even if the upgrade is successful, review the log file.

In particular, review the Prebuild Conclusion for messages about conclusions that cannot be built. These messages do not indicate a problem with the upgrade but rather identify issues with the Pega Platform application that you must correct.

This is a sample of the Prebuild Conclusions section:

```
#Prebuild Conclusions:
java] May 1, 2015 1:00:21 PM
 com.pega.pegarules.internal.bootstrap.PRBootstrapDataSource
[java] 19830421: Loading bootstrap properties from file:///e:\temp/
PegaInstallTemp-24-November-2014-11.07.15/prbootstrap.properties
[java] May 1, 2015 1:00:21 PM
 com.pega.pegarules.internal.bootstrap.SettingReaderJNDI
java] 19830421: Could not find java:comp/env/prbootstrap/ in the local JNDI
 context, skipping prconfig setting lookup
```

Look for any warning or error messages. One common issue is that a conclusion cannot be built because a class is invalid, for example:

```
[java] 2015-05-01 13:13:52,911 [  STANDARD] [ ] (ionary.ClassInfoConclusionImpl)
 WARNING  -
[java]  Unable to initialize a ClassInfoConclusion for PEGACARD-CPM-WORK-
GENERALCUSTOMERCASE, classDef=null, classRule=null
```

The warning or error message includes information about the invalid class. See the Pega Community for information. If you cannot resolve the issue, see the Support section of the Pega Community.

Test your applications

The post-upgrade procedures remove the known compatibility issues between Pega Platform and earlier versions. However, depending on your development methods, you might discover additional modifications that need to be made in your existing applications when they are upgraded to Pega Platform. Perform full testing of your application functionality after the upgrade.

Enabling server-side screen captures for application documents

Regardless of which application server platform you use, you must set up a Tomcat server to support taking and storing screen captures on a server rather than on a client. By taking and storing screen captures on a server, you avoid client-side limitations, such as browser incompatibilities or client software requirements.

As a best practice, virtually install Tomcat and deploy the **`prScreenShot.war`** file on the same server that is running Pega Platform. Otherwise, use a standalone Linux or Windows server. If you use a Linux server, you must include the following components:

- fontconfig
- freetype
- libfreetype.so.6
- libfontconfig.so.1
- libstdc++.so.6

You can include screen captures in an application document that is generated by the Document Application tool. Screen captures provide stakeholders with a realistic picture of an application's user interface. Install a PhantomJS REST server to include screen captures in an application document.

1. Download the following WAR file: **`Pega_DistributionImage\Additional_Products\PhantomJS\prScreenShot.war`**
2. Deploy the WAR file on a Tomcat server.
3. Edit the **`tomcat-users.xml`** file to add the following role and user. This file is located at \apache-tomcat-XX\conf\ tomcat-users.xml.

   `<role rolename="pegascreencapture" /> <user username="`*restUser*`" password="`*rules*`" roles="pegascreencapture" />`
4. Start the Tomcat server. The service is hosted at http://*IPaddress*:*port*/prScreenShot/rest/capture, where *IPaddress* is the address of the system where Tomcat is hosted, and *port* is the port on which the service is deployed.
5. Log in to your Pega Platform application and make the following changes:

   a) Edit the Data-Admin-System-Setting instance *Pega-AppDefinition* - *CaptureScreenshotsResourcePath* with the URL of the service, for example, `http://10.224.232.91:8080/prScreenShot/rest/capture`.

   b) Add the user that you created in step 3 to the Data-Admin-Security-Authentication profile instance CaptureScreenshotsAuthProfile.

What to do next: Continue at Configuring PhantomJS REST server security for including screen captures in an application document.

Configuring PhantomJS REST server security for including screen captures in an application document

To ensure a secure installation of Pega Platform, enable the PhantomJS REST server to take and store server-side screen captures. In application documents generated by the Document Application tool, screen captures provide stakeholders with a realistic picture of the application's user interface.

1. Obtain the SSL certificate from the Pega Platform administrator.
2. Add the SSL certificate to the list of trusted certificates:

   a) Double-click the certificate.

   b) Click **Install certificate** to start the **Certificate Import** wizard.

   c) Click **Next**, and select **Place all certificates in the following store**.
   d) Click **Browse**, select **Trusted Root certificate**, and click **OK**.
   e) Click **Next**, and then click **Finish** to complete the wizard.
3. Add the certificate to the truststore of the JVM on which the REST server is installed:
   a) Open a command prompt.
   b) Change the root directory to the security folder in the Java installation folder. For example, `C:\Program Files (x86)\Java\jre7\lib\security`.
   c) Run the following command:
   `keytool -keystore cacerts -importcert -alias` *certificate alias* `-file` *certificate name*
   d) When prompted, enter the password for the cacerts keystore. The default password is `changeit`.

Adding special privileges to access the Requester Management landing page

To access the Requester Management landing page in your application, you need to add privileges to the `@baseclass` and `Pega-Landing` access classes in your access roles.

Add the following privileges for the type of access that is needed:

- `pzSystemOperationsObserver` – Required to access the Requester Management landing page and to view performance and trace entry details.
- `pzSystemOperationsAdministrator` – Required to access the Requester Management landing page and perform most actions on requestors. To trace requestors and view the clipboard you also need to have the `pzDebugRemoteRequestor` privilege.

To add the privileges, complete the following steps:

1. Click the **Operator** menu in the Dev Studio header and select **Operator**.
2. In the **Application Access** section, expand an access group and click the role that you need to modify.
3. Click the `@baseclass` class in the **Access Class** column.
4. In the **Privileges** section, click the **Plus** icon and select the appropriate privilege in the **Name** column.
5. Enter `5` for the production level in the **Level** column. Production level 5 provides the highest security.
6. Click **Submit**.
7. Click the `Pega-Landing` class in the **Access Class** column and repeat steps 4 through 6.

   Note: If the `Pega-Landing` class is not in the table, add it by clicking the **Plus** icon at the end of the table and entering `Pega-Landing` in the **Class** field.
8. Save the access role form.

Upgrading from Pega 7.2.2: customizing the agent schedules for the standard Pega Platform agents

If you developed agent schedules on Pega 7.2.2 with the Node Classification feature, manually edit all customized schedules for standard Pega 7.2.2 agents. You can update the agent schedules after starting a node with a node type, when the agent schedule is re-created.

If you did not develop agent schedules with the Node Classification feature of Pega 7.2.2, skip this section.

1. In Dev Studio, open the Agent type to customize.
2. In the agent schedule form, modify any settings that need to be updated. For more information, see the help for the agent schedule data instances.
3. Click **Save**.

Updating the service email for Pulse email replies

If you have a service email that you created to configure replies to Pulse email notifications, you must add the Message–ID, In–Reply–To, and References fields to the message header to ensure that these replies are posted in Pulse. Adding the fields allows the system to interact with email clients by using email headers when users reply to Pulse emails.

1. Search for the service email that you created to configure replies to Pulse email notifications:
   a) Open the Records Explorer.
   b) Expand the **Integration-Services** category and click **Service Email**.
   c) Click the service email that you created for Pulse email replies.
2. In the **Message header** section on the **Request** tab of the service email, add the following fields:

Field name	Description	Map to	Map to key
Message–ID	Message–ID	Clipboard	`.pyInboundEmail.pyMessageID`
In–Reply–To	In-Reply–To	Clipboard	`.pyInboundEmail.pyInReplyTo`
References	References	Clipboard	`.pyInboundEmail.pyReferences`

3. Save the service email.
4. Restart the email listener that is configured with the service email.

Appendices

The appendices include information about optional processes and troubleshooting.

Migrate script properties

The migrate script, **`migrate.bat`** or **`migrate.sh`** migrates the rules objects from the existing schema to a new rules schema, and generates and applies rules schema DDL objects after upgrading the new rules schema. Edit the **`migrateSystem.properties`** file to configure the migrate script. The Deployment user performs the migrations.

Note: Pega strongly recommends that you use the **`migrate.bat`** or **`migrate.sh`** script to perform these steps. The use of vendor tools is not recommended.

The migrate script is designed to automate many aspects of the data movement process, including:

- Export of appropriate tables and data objects from the source system schema;
- Generation and application of DDL to the target system schema;
- Import of appropriate tables and data objects to the target system schema.

Common properties

The following common properties must be configured in the **`migrateSystem.properties`** file to log on to the databases used for each schema. If you are using one database for each schema, these properties will be the same for each step. However, if you are using different databases for the rules schema and the temporary upgrade schema, these properties will be different, depending on which database the schema is hosted on.

The table below lists the common properties, their descriptions, and valid values. Source properties apply to the system being migrated from, and target properties apply to the system being migrated to. Set the properties by adding the appropriate value after the equals sign in the properties file. Set the properties by adding the appropriate value after the equals sign in the properties file.

Property	Description
pega.source.jdbc.driver.jar pega.target.jdbc.driver.jar	The path to the JDBC JAR file. For databases that use multiple JDBC driver files, specify semicolon-separated values.
pega.source.jdbc.driver.class pega.target.jdbc.driver.class	Valid values are: • `oracle.jdbc.OracleDriver`
pega.source.database.type pega.target.database.type	The database type: oracledate
pega.source.jdbc.url pega.target.jdbc.url	The database connection URL. For more information, see Obtaining database connection information.
pega.source.jdbc.username pega.target.jdbc.username	Deployment user name.
pega.source.jdbc.password pega.target.jdbc.password	Deployment user password.

Custom properties

The following properties are used during migration to configure custom settings.

Property	Description
pega.source.jdbc.custom.connection.properties pega.target.jdbc.custom.connection.properties	An optional semi-colon-delimited list of custom connection properties.
pega.source.data.schema pega.target.data.schema pega.source.rules.schema pega.target.rules.schema	Used to correctly schema-qualify tables in stored procedures, views and triggers. These properties are not required if migrating before performing an upgrade.
pega.target.bypass.udf	Set this property to bypass UDF generation on the system.

Migration directory properties

Set the directories for migration objects.

Property	Description
pega.bulkmover.directory	The full path to the directory where output from the bulk mover will be stored. This directory will be cleared when pega.bulkmover.load.db is set to true. This property must be set if either pega.bulkmover.unload.db or pega.bulkmover.load.db is set to true.
pega.migrate.temp.directory	The full path to the temporary directory that is created by the migrate system utilities.

Operational properties

Use the following properties to migrate Rules objects. Set to true or false.

Property	Description
pega.move.admin.table	Migrate the admin tables required for an upgrade with the rules tables.
pega.clone.generate.xml	Generate an XML document containing the definitions of tables in the source system. It will be found in the schema directory of the distribution image.
pega.clone.create.ddl	Create DDL from the generated xml document. This DDL can be used to create copies of rule tables found on the source system.
pega.clone.apply.ddl	Apply the generated clone DDL to the target system.
pega.bulkmover.unload.db	Unload the data from the rules tables on the source system into the pega.bulkmover.directory.
pega.bulkmover.load.db	Load the data onto the target system from the pega.bulkmover.directory.

Rules schema object properties

This table describes operations to run when migrating upgraded rules:

Property	Description
pega.rules.objects.generate	Generate the rules schema objects (views, triggers, procedures, and functions). The objects will be created in the pega.target.rules.schema but will contain references to the pega.target.data.schema where appropriate.
pega.target.bypass.udf	Set this property to bypass UDF generation on the system.
pega.rules.objects.apply	Apply the rules schema objects (views, triggers, procedures, and functions) to pega.target.rules.schema.

Editing the setupDatabase.properties file

Edit the **`setupDatabase.properties`** file to configure deployment scripts.

Skip this section if your deployment meets all the following criteria:

- You will use the Installation and Upgrade Assistant.
- You will allow the Installation and Upgrade Assistant to automatically apply the schema changes and do not need to create a DDL file.
- You will not enable Kerberos authentication.

If your deployment does not meet all these criteria, follow the steps in this section to edit the **`setupDatabase.properties`** file. The **`setupDatabase.properties`** file controls scripts which perform the following tasks:

- Upgrade Pega Platform and enable Kerberos authentication. Use the **`upgrade.bat`** or **`upgrade.sh`** script.
- Generate a DDL file of schema changes. Use the **`generateddl.bat`** or **`generateddl.sh`** script. You can use the generateddl script regardless of whether you use the IUA or the command-line script.
- Generate user-defined functions. Use the **`generateudf.bat`** or **`generateudf.sh`** script.

1. Open the **`setupDatabase.properties`** file in the scripts directory of your distribution image: *Directories.distributionDirectory*<b>`\scripts\setupDatabase.properties`</b>
2. Specify the properties for your system. For each property, add the appropriate value after the equal sign. See Database connection properties and script arguments.
3. Optional: If you are repeating a failed upgrade, configure the resume property:
    - To resume from the last successful step, set `automatic.resume=true`.
    - To restart from the beginning, set `automatic.resume=false`.
4. Save and close the file.

Database connection properties and script arguments

The database connection properties in the **`setupDatabase.properties`** file specify the settings needed to connect to the database. The script arguments specify the same settings when you use command-line scripts. Command-line settings override property file settings.

Script argument	Property	Description
--driverJAR	pega.jdbc.driver.jar	Path and file name of the JDBC driver.
--driverClass	pega.jdbc.driver.class	Class of the JDBC driver

Script argument	Property	Description
--dbType	pega.database.type	Database vendor type. Enter the correct database vendor: • Oracle: `oracledate`
--dbURL	pega.jdbc.url	The database JDBC URL. For more information, see Obtaining database connection information.
--dbUser	pega.jdbc.username	User name of the Deployment user.
--dbPassword	pega.jdbc.password	Password of the Deployment user. For encrypted passwords, leave this blank.
--adminPassword	pega.admin.password	For new installations only.
--rulesSchema	rules.schema.name	In a single schema environment, sets rules schema and data schema. In a split-schema configuration, sets the rules schema only.
--dataSchema	data.schema.name	For split-schema configurations only, sets the data schema name.
--customerDataSchema	customerdata.schema.name	An optional customer data schema separate from the default Pega data schema.
	user.temp.dir	Optional: The location of the temp directory. Set this location to any accessible location. For example, C:\TEMP.
--mtSystem	multitenant.system	Specifies whether this a multitenant system.

Additional upgrade properties

The properties in the **`setupDatabase.properties`** file help you to customize the upgrade.

The following table describes additional properties in the **`setupDatabase.properties`** file that you might need to edit only for upgrades.

Property	Description
bypass.pega.schema	To bypass both creating the upgrade schema and automatically generating the user-defined functions, set **bypass.pega.schema** to `true`. This implies that the upgrade DDL is already applied. To create the upgrade schema and automatically generate the UDFs, leave this property blank or set it to `false`.
bypass.udf.generation	If you set **bypass.pega.schema** to `false` to create the upgrade schema, but still want to bypass automatically generating the user-defined functions, set **bypass.udf.generation** to `true`.
rebuild.indexes	Rebuilds database rules indexes after the rules load to improve database access performance. If **rebuild.indexes**=`false`, you can rebuild the indexes later by running the stored procedure SPPR_REBUILD_INDEXES. The amount of time this process adds to the upgrade depends on the size of your database.

Property	Description
update.existing.applications	Set to true to run the Update Existing Applications utility. Run the Update Existing Application utility to ensure that your existing applications take advantage of new functionality in Pega Platform. Run the utility first on your development system and test the changes. Then, run the utility again on the production system. The specific actions required for your application depend on your current version. You can also run this utility later from the Dev Studio. The default setting is false.
update.applications.schema	Specifies whether to run the Update Applications Schema utility to upgrade the auto-generated rule, data, work, and work history tables with the schema changes in the latest base tables as part of the upgrade. You can also run this utility later from the **prpcUtils.bat** or **prpcUtils.sh** script, or from Dev Studio. The default setting is false.
run.ruleset.cleanup	Removes older rules. In most cases, removing older rules improves the general performance of the system. Running the cleanup script permanently removes rules older than the upgraded version.
reversal.schema.file.name	Schema file name to be used for reversal.
automatic.resume	If the upgrade fails, specifies whether the system restarts the upgrade from the step where the failure occurred. The default value is true.

Optional: Generating and applying DDL

If you opted not to have the Installation and Upgrade Assistant automatically apply the DDL, generate and apply the DDL manually.

Manually generating and applying DDL changes must be done in each step of the deployment. Some steps use the **generateddl** script. Other steps use the **migrate** script. These scripts write platform-specific DDL to a file. You can then view and edit the file or directly apply it by using database management tools. For detailed instructions, communicate with your database administrator.

The process for generating and applying DDL differs depending on whether you are performing an out-of-place upgrade or an in-place upgrade.

Generating and applying DDL in an out-of-place upgrade

Use the **migrate** and **generateddl** scripts to generate and apply DDL changes as part of an out-of-place upgrade.

Manually generating and applying DDL changes must be done in each step of the upgrade. Some steps use the **generateddl** script. Other steps use the script. These scripts write platform-specific DDL to a file. You can then view and edit the file or directly apply it by using database management tools. For detailed instructions, communicate with your database administrator.

This example shows an out-of-place upgrade with double-migration.

1. Optional: If you are upgrading out-of-place, clone the DDL:

   a) Clone the DDL. For details about running the migrate script, see Migrate script properties.

      1. Edit the **migrateSystem.properties** file to set the source schema names:

```
pega.source.rules.schema=original rules schema name
pega.source.jdbc.url=URL of database
pega.source.jdbc.username=Deployment user name
pega.source.jdbc.password=password
pega.source.data.schema=original data schema name
```

2. Edit the **migrateSystem.properties** file to set the target schema names. The settings depend on whether you have one database or two databases:

   - One database:

     ```
     pega.target.rules.schema=new rules schema name
     pega.target.data.schema=new rules schema name
     ```

   - Two databases:

     ```
     pega.target.rules.schema=upgrade schema name
     pega.target.data.schema=upgrade schema name
     ```

3. Edit the **migrateSystem.properties** file to create the DDL:

   ```
   pega.move.admin.table=true
   pega.clone.generate.xml=true
   pega.clone.create.ddl=true
   pega.clone.apply.ddl=false
   pega.bulkmover.unload.db=false
   pega.bulkmover.load.db=false
   pega.rules.objects.generate=false
   pega.rules.objects.apply=false
   ```

4. Run the **migrate.sh** or **migrate.bat** script to create the DDL.

b) Have the database administrator apply the DDL.

c) Populate the tables. For details about running the migrate script, see Migrating the existing rules schema.

1. Leave the source and target schema properties as in step 1a.
2. Edit the **migrateSystem.properties** file to populate the table:

   ```
   pega.move.admin.table=true
   pega.clone.generate.xml=false
   pega.clone.create.ddl=false
   pega.clone.apply.ddl=false
   pega.bulkmover.unload.db=true
   pega.bulkmover.load.db=true
   pega.rules.objects.generate=false
   pega.rules.objects.apply=false
   ```

3. Run the **migrate.sh** or **migrate.bat** script to populate the table.

2. Upgrade the rules schema and apply the DDL for the rule schema changes:

a) Create the DDL of changes to the rules schema.

1. Edit the **setupDatabase.properties** file to set the rules and data schema names:

   - One database:

     ```
     rules.schema.name=new rules schema name
     data.schema.name=new rules schema name
     ```

- Two databases:

  ```
  pega.target.rules.schema=upgrade schema name
  pega.target.data.schema=upgrade schema name
  ```

2. Optional: If your customer data schema is different than your Pega data schema, insert the following entry to specify the customer data schema name. Replace *customer-data-schema* with your customer data schema name.

   `pega.customerdata.schema=`*customer-data-schema*

3. Run the **`generateddl.bat`** or **`generateddl.sh`** script.

b) Have the database administrator apply the DDL.

c) Use the command line to upgrade the rules schema.

1. Edit the **`setupDatabase.properties`** file to bypass the schema upgrade because the DDL is already applied: `bypass.pega.schema=true`
2. Leave the rules and data schema names as in step 2a.
3. Run the **`upgrade.bat`** or **`upgrade.sh`** script.

3. Migrate the changes to the new rules schema; create rules schema objects, and create links between the new rules schema and the data schema.

a) Clone the DDL.

1. Edit the **`migrateSystem.properties`** file to set the source and target schema properties:

   ```
   pega.source.rules.schema=upgrade schema name
   pega.source.data.schema=upgrade schema name
   pega.target.rules.schema=new rules schema
   pega.target.data.schema=original data schema
   ```

2. Edit the **`migrateSystem.properties`** file to create the DDL:

   ```
   pega.move.admin.table=false
   pega.clone.generate.xml=false
   pega.clone.create.ddl=true
   pega.clone.apply.ddl=false
   pega.bulkmover.unload.db=false
   pega.bulkmover.load.db=false
   pega.rules.objects.generate=false
   pega.rules.objects.apply=false
   ```

3. Run the **`migrate.sh`** or **`migrate.bat`** script to create the DDL.

b) Give the DDL to the database administrator to apply.

c) Populate the tables.

1. Leave the source and target schema properties as in step 3a.
2. Edit the **`migrateSystem.properties`** file to populate the table:

   ```
   pega.move.admin.table=false
   pega.clone.generate.xml=false
   pega.clone.create.ddl=false
   pega.clone.apply.ddl=false
   pega.bulkmover.unload.db=true
   pega.bulkmover.load.db=true
   pega.rules.objects.generate=true
   ```

```
pega.rules.objects.apply=false
```

3. Run the **`migrate.sh`** or **`migrate.bat`** script to populate the table.

d) Give the DDL to the database administrator to apply the rules objects.

4. Upgrade the data schema and apply the DDL for the data schema changes:

   a) Create the DDL of changes to the rules schema.

   1. Edit the **`setupDatabase.properties`** to set the rules and data schema names:

      ```
      rules.schema.name=new rules schema
      data.schema.name=original data schema
      ```

   2. Optional: If your customer data schema is different than your Pega data schema, insert the following entry to specify the customer data schema name. Replace *customer-data-schema* with your customer data schema name.

      `pega.customerdata.schema=`*customer-data-schema*

   3. Run the **`generateddl.bat`** or **`generateddl.sh`** script with the --upgradeDataOnly argument and true parameter, for example: `generateddl.bat --upgradeDataOnly true`

   b) Have the database administrator apply the DDL to the data schema.

   c) Use the command line to upgrade the data schema. Follow the instructions in Upgrading the data schema.

   1. Edit the **`setupDatabase.properties`** file to bypass the schema upgrade because the DDL is already applied: `bypass.pega.schema=true`

   2. Run the **`upgrade.bat`** or **`upgrade.sh`** script with the --dataOnly argument and true parameter, for example: `upgrade.bat --dataOnly true`

Generating and applying DDL in an in-place upgrade

Use the **`migrate`** and **`generateddl`** scripts to generate and apply DDL changes as part of an in-place upgrade.

Generating the DDL file

Follow these steps to generate a DDL file for your database administrator to apply manually.

1. Edit the **`setupDatabase.properties`** file.

   a) Configure the connection properties. For more information about parameter values, see Properties file parameters. The customer data schema is optional.

   ```
   # Connection Information
   pega.jdbc.driver.jar=\path-to-the-database-JAR-file\DRIVER.jar
   pega.jdbc.driver.class=database driver class
   pega.database.type=database vendor type
   pega.jdbc.url=URL of the database
   pega.jdbc.username=Deployment username
   pega.jdbc.password=password
   rules.schema.name=rules-schema-name
   data.schema.name=data-schema-name
   customerdata.schema.name=optional-customer-data-schema
   ```

   b) Save and close the file.

2. At a command prompt, navigate to the **Pega-image** \scripts directory.
3. Run **`generateddl.bat`** or **`generateddl.sh`** and pass in the required --action argument:

   ```
   #generateddl.bat --action upgrade
   ```

If you do not specify an output directory, the script writes the output to the default directory: **Pega-image`\schema\generated\`**

Note: The output directory is deleted and re-created each time the generateddl script runs. To save a copy of the DDL, rename the directory before you run the script.

Applying the DDL file

Before you continue, have your database administrator follow these general steps to apply the schema changes; these schema changes can include changes to user-defined functions:

1. Review the DDL file in the output directory and make any necessary changes. The default directory is:

   Pega-image`\schema\generated\database\oracledate>`

2. Apply the DDL file.

   a) Register the DDL file with the database. Register the .jar file with the database.

   b) Apply the **CREATE FUNCTION** DDL.

The output directory is deleted and re-created each time the generateddl script runs. To save a copy of the DDL, rename the directory before you rerun the script.

Editing the setupDatabase.properties file to bypass DDL generation

After your database administrator applies the changes to your database, configure the **`setupDatabase.properties`** file to bypass applying a schema that already exists. Reapplying an existing schema would cause the deployment to fail.

1. Open the **`setupDatabase.properties`** file in the scripts directory of your distribution image: *Directories.distributionDirectory***`\scripts\setupDatabase.properties`**
2. Set the property `bypass.pega.schema=true`.
3. Save and close the file.

Installing user-defined functions

The user-defined functions (UDFs) enable the Pega Platform to read data directly from the BLOB without creating and exposing columns. Skip this section if you installed the UDFs when you deployed Pega Platform.

There are several ways you might have bypassed generating and installing the UDFs when you deployed:

- Setting either `bypass.pega.schema=true` or `bypass.udf.generation=true` in the **`setupDatabase.properties`** file
- Setting `pega.target.bypass.udf=true` in the **`migrateSystem.properties`** file
- Selecting **Bypass Automatic DDL Application** from the Installation and Upgrade Assistant

Before you install the UDFs, verify that you have the appropriate user permissions.

For more information about user permissions, see your Pega Platform installation guide.

1. Edit the **`setupDatabase.properties`** file.

a) Configure the connection properties. For more information about parameter values, see Properties file parameters.

```
# Connection Informationpega.jdbc.driver.jar=\path-to-the-database-JAR-file\DRIVER.jar
pega.jdbc.driver.class=database driver class
pega.database.type=database vendor type
pega.jdbc.url=URL of the database
pega.jdbc.username=Deployment user name
pega.jdbc.password=password
rules.schema.name= rules-schema-name
data.schema.name=data-schema-name
```

b) Save and close the file.

2. On the rules schema, run the following commands to remove any partially installed UDFs:

```
DROP FUNCTION rules-schema-name.pr_read_from_stream;
DROP FUNCTION rules-schema-name.pr_read_decimal_from_stream;
DROP FUNCTION rules-schema-name.pr_read_int_from_stream;
```

3. Optional: If you have a split-schema, on the data schema run the following commands:

```
DROP FUNCTION data-schema-name.pr_read_from_stream;
DROP FUNCTION data-schema-name.pr_read_decimal_from_stream;
DROP FUNCTION data-schema-name.pr_read_int_from_stream;
```

4. From the **Pega-image** \scripts directory, run the **`generateudf.bat`** or **`generateudf.sh`** script with the `--action install` argument.

```
generateudf.bat --action install --dbType oracledate
```

Switching to Hazelcast embedded mode from Apache Ignite client-server mode

If you were using Apache Ignite in client-server mode, you must switch to Hazelcast in embedded mode because Apache Ignite is no longer supported. You can switch back to embedded mode from client-server mode during a rolling restart.

Perform a rolling restart to keep your system always available during the upgrade. You remove nodes from the load balancer, shut them down, upgrade, and start them again one by one. You do not add them back to the load balancer until you have upgraded half of the nodes.

To perform a rolling restart, complete the following steps.

1. Prepare the database.
   a) Disable rule saving. For more information, see Disabling rule creation on the rules schema.
   b) Migrate the PegaRULES schema to a temporary schema. For more information, see Migrating the existing rules schema.
   c) Upgrade the new rules schema, for example, a framework or application upgrade. For more information, see Upgrading the migrated rules schema.
   d) Copy the new rule schema to the production database. For more information, see Migrating to the new rules schema.
2. Upgrade half of the nodes one by one.

a) Configure the load balancer to disable a node.

- Disabling the node does not allow new connections, but it allows existing users and services to complete work.
- Quiescing a Pega Platform node that has not been disabled in the load balancer results in error conditions for users of that Pega Platform node, because new users cannot log in. The Pega Platform must be disabled in the load balancer so that new users are redirected to another active Pega Platform node.

b) Quiesce the Pega Platform node, by using the Autonomic Event Services (AES) or high availability landing pages. For more information, see the help for Cluster management and quiescing.

c) Ensure that all requestors are passivated and the system run state is set to "Quiesce Complete", by using the HA Cluster Management landing page.

d) Shut down the node.

e) Upgrade the data source to connect to the upgraded schema (to reflect changes made in step 1). For more information, see Upgrading the data schema.

f) To switch back to embedded mode from client-server mode, modify the **prconfig.xml** file and remove the following settings that were added during the switch to client-server mode.

```
<env name="cluster/clientserver/clientmode" value="true " />
<env name="identification/cluster/protocol" value="ignite " />
```

g) Start the node.

3. Perform any needed post-upgrade activities and tests. For more information, see Post-upgrade configuration.
4. After you upgrade half of the nodes, disable the remaining non-upgraded nodes in the load balancer.
5. Add all of the upgraded nodes that you upgraded in step 3, back to the load balancer to start taking traffic.
6. Upgrade the remaining half of the nodes (the non-upgraded nodes) one by one.

   a) Perform steps 2 b through 3 g.

   b) Add the node back to the load balancer to start taking traffic.
7. To switch back to embedded mode from client-server mode, shut down all stand-alone Apache Ignite servers after all of the nodes are upgraded and no longer use the stand-alone Apache Ignite server cluster.

Reverse an out-of-place upgrade

You can revert to a previous release after performing an out-of-place upgrade.

Reversing an upgrade requires the original (pre-update) rules schema, which is available only if you perform an out-of-place upgrade.

Limitations

You can reverse upgrades in many situations, but there are limitations.

Reversing an upgrade is not supported for:

- In-place upgrades
- Single-schema systems

- Multitenancy systems
- Multi-hop upgrades: For example,
  - You can:
    1. Upgrade from Pega 7.2 to 8.1.
    2. Reverse the upgrade to return Pega 7.2.
  - You cannot:
    1. Upgrade from Pega 7.2 to 7.4 (first hop).
    2. Upgrade again from Pega 7.4 to 8.1 (second hop).
    3. Reverse the upgrade back to Pega 7.2.

Upgrade reversal details

To reverse an upgrade, run the **`reverse.bat`** or **`reverse.sh`** script on the pre-upgrade rules schema to revert the changes to the data schema.

Upgrading the data schema creates a **`Reverse_`** *timestamp* **`.xml`** file. The **`reverse.bat`** or **`reverse.sh`** script uses the **`Reverse_`** *timestamp* **`.xml`** file to re-create the data schema. If you upgrade your system more than once, there might be multiple versions of the **`Reverse_`** *timestamp* **`.xml`** file. The **`setupDatabase.properties`** file specifies the correct **`Reverse_`** *timestamp* **`.xml`** file.

The reverse process restores database functions, procedures, triggers, and views in the data schema. New or altered columns, constraints, indexes, and tables are left intact. The process creates the **`CLI-Reverse-Log-`** *timestamp* **`.log`** file in the scripts/logs directory.

If new Data- instances (including Work- instances) are created by using data objects modified by the upgrade process, the new instances might not function properly after reversal.

The table below describes the reverse behavior for some database objects that are either added or modified (before or after an upgrade), or added or modified by the product during the upgrade process.

Object Type	**How created**	**Added or Modified**	**Status after Reversal**
Tables, Columns, Indexes	--	--	Ignored
Procedures, Triggers, Views	Pega Platform shipped	Added	Dropped
--	Pega Platform shipped	Modified	Restored
Functions	Pega Platform shipped	Added	Dropped
--	Pega Platform shipped	Modified	Restored
--	Manually	Added	Ignored
--	Manually	Modified	Ignored
Data instances	Pega Platform shipped	Added	Deleted
--	Pega Platform shipped	Modified	Restored
--	Manually	Added	Ignored
--	Manually	Modified	Ignored

Reversing an upgrade

Use a script to reverse an upgrade.

1. Check the scripts/log directory for the presence of multiple **Reverse_** *timestamp***.xml** files. Note the file name with the latest time stamp.
2. If you have not done so already, edit the **setupDatabase.properties** file to configure the reversal.
   a) Open the **setupDatabase.properties** file in the scripts directory of your distribution image: *Directories.distributionDirectory***\scripts\setupDatabase.properties**

   b) Configure the connection properties. For more information about the connection properties, see Appendix A — Properties files.

   ```
   # Connection Information
   pega.jdbc.driver.jar=/path-to-the-database-JAR-file/DRIVER.jar
   pega.jdbc.driver.class=database driver class
   pega.database.type=database vendor type
   pega.jdbc.url=URL of the database
   pega.jdbc.username=Deployment user name
   pega.jdbc.password=password
   rules.schema.name=new-rules-schema-name
   data.schema.name=data-schema-name
   ```

   c) If there are multiple versions of the **Reverse_***timestamp***.xml** file, specify the file name from step 1 in the `reversal.schema.file.name` property, for example:

   `reversal.schema.file.name=Reverse_2016-06-18-23:59:59.xml`

   d) Save and close the file.
3. In the scripts directory of the upgrade distribution image, run the **generateDDL.bat** or **generateDDL.sh** script to generate the reverse create and drop statements.
4. In the scripts directory of the upgrade distribution image, run the **reverse.bat** or **reverse.sh** script. If there are any errors, review the information in the **scripts\logs\CLI-Reverse-Log-***timestamp***.log** file.
5. Reconfigure the application server:
   a) Restart the application server.

   b) Verify that the **prweb** application is configured to use the correct rules and data schema names.

   c) Deploy the **prweb** application.
6. To verify that the reversal was successful, check the version of Pega Platform on the log-in screen.

Troubleshoot upgrade errors

Use the information in this section to troubleshoot upgrade errors.

Error logs are displayed in the Installation and Upgrade Assistant window and are also stored in the **Pega-image\scripts\logs** directory.

Upgrades from PRPC 5.4 and earlier: System-Work-Indexer not found in dictionary

If you are upgrading from PRPC 5.4 or earlier, an indexing error can cause the upgrade to fail with a class not defined message.

```
Class not defined in dictionary: System-Work-Indexer aClassName
```

To fix the problem, first follow the instructions in Upgrading from PRPC 5.4 and earlier: setting indexing, and then repeat the upgrade.

Resuming or restarting after a failed deployment

If the deployment fails, you can opt to either resume or start over:

- **Resume** — The system uses the previously-entered configuration information to resume a failed deployment from the last successful step. This is the default behavior.
- **Start Over** — The system discards all previously-entered configuration information, drops the database schema, and starts the deployment from the beginning.

1. Review the failure message for information about the source of the error. Use the information in the error message to correct the error before you continue.
2. Optional. If you used the IUA, the select either **Resume** or **Start Over** when the system displays the **Resume Options** screen.
3. Optional. If you used the command-line script, set the automatic.resume property in the **`setupDatabase.properties`** file:
   - To resume the deployment from the last successful step, set `automatic.resume=true`.
   - To start over, set `automatic.resume=false`.
4. Repeat the deployment. Use the same procedure that you used for the initial deployment.

Recovering from a faulty split-schema migration

If the rules schema objects were not applied successfully, run the migration recovery scripts to remove duplicate rules.

The migration recovery scripts remove duplicate rules created as a result of a faulty split schema migration, where indexes and primary keys were not created on rules tables. To check for this issue, see if your rules tables, such as pr4_base and pr4_rule, are missing primary key indexes.

1. Take down any application servers that use the failed schema.
2. Backup your database.
3. In **`ResourceKit\AdditionalUpgradeScripts\MigrationRecoveryScripts\`***database***`_cleanDups.sql`**, replace all instances of @RULES_SCHEMA with the name of the schema that contains the pr4_base table.
4. Use your vendor tools to run the *database***`_cleanDups.sql`** script on the database.
5. In *database***`_fix_vw_table.sql`**, replace all instances of @RULES_SCHEMA with the name of the schema that contains the pr4_base table.
6. Use your vendor tools to run the *database***`_fix_vw_table.sql`** script on the database.
7. Use the **`generateddl.bat`** or **`generateddl.sh`** script to generate and apply the DDL. See Optional: Generating and applying DDL.
8. Use your vendor tools to rebuild the indexes for the tables in your rules schema.

PEGA0055 alert — clocks not synchronized between nodes

The Pega Platform validates time synchronization to ensure proper operations and displays a PEGA0055 alert if clocks are not synchronized between nodes.

For information about how to reference a common time standard, see the documentation for your operating system.