Pega Platform '24.2

3 September 2025

©2025 Pegasystems Inc

CONTENTS

Reference 12
Methods 12
Branching to another activity 17
Calling another activity 19
Using the Collect instruction 21
Use of Java in an activity step 22
Queue instruction – run an activity asynchronously 28
Connect-Cassandra method 29
Connect-dotNet method (deprecated) 33
Connect-EJB method (deprecated) 35
Connect-File method (deprecated) 37
Connect-FTP method 40
Connect-HBase method 42
Connect-HTTP method 46
About Connect HTTP rules (deprecated) 49
Connect-Java method (deprecated) 51
Connect-JCA method (deprecated) 53
Connect-JMS method 54
Connect-MQ method 56
Connect-REST method 58
Connect-SOAP method 61
Connect-Wait method 64
Commit method 67
Activity-Clear-Status method 70
Activity-End method 71
Activity-List-Add method 72
Setting the status of an activity using the Activity-Set-Status method 74
Apply-DataTransform method 75
Apply-Parse-Delimited method 77

©2025 Pegasystems Inc

Apply-Parse-Structured method. 79
Apply-Parse-XML method. 81
Assert-No-Invocation method. 82
Assign-EstablishContext method. 82
Call-Async-Activity method. 83
Call-Automation method. 84
Call-Function. 85
End-Validate method. 88
Exit-Activity method. 88
Flow-End method. 89
Flow-New method. 89
History-Add method. 90
History-List method. 96
Index-Finish method. 97
Link-Objects method. 97
Using the Load-DataPage method. 99
Log-Message method. 100
Map-Structured method. 103
Obj-Browse method. 106
Obj-Delete method. 118
Obj-Delete-By-Handle method. 121
Obj-Filter method. 126
Obj-List-View method. 128
Obj-Open method. 133
Obj-Open-By-Handle method. 140
Obj-Refresh-and-Lock method. 147
Obj-Save method. 151
Obj-Save-Cancel method. 155
Obj-Sort method. 156
Obj-Validate method. 158
Page-Change-Class method. 159
Page-Clear-Messages method. 163

PEGA ©2025 Pegasystems Inc

Page-Copy method. 164
Page-Merge-Into method. 166
Page-New method. 170
Page-Remove method. 173
Page-Rename method. 174
Page-Set-Messages method. 175
Page-Unlock method. 178
Page-Validate method. 179
Privilege-Check method. 181
Property-Map-DecisionTable method. 182
Property-Map-DecisionTree method. 184
Property-Map-Value method. 186
Property-Map-ValuePair method. 188
Property-Ref method. 190
Property-Remove method. 197
Property-Seek-Value method. 197
Property-Set-Messages method. 199
Property-Set-Stream method. 202
Property-Set method. 204
Property-Set-XML method. 207
Property-Validate method. 209
Property-Set-Corr method. 211
Publish-Notifications. 213
RDB-Delete method. 214
RDB-List method. 216
RDB-Open method. 221
RDB-Save method. 224
Requestor-Stop method. 227
Rollback method. 228
Show-Applet method. 228
Show-Applet-Data method. 229
Show-HTML method. 229

©2025 Pegasystems Inc

Show-Page method. 231
Show-Property method. 232
Show-Stream method. 233
Start-Validate method. 236
StringBuffer-Append method. 236
StringBuffer-Insert method. 237
StringBuffer-Reset method. 238
TaskStatus-Set method. 239
Text-Normalize method. 240
Thread-Clear method. 242
Thread-Stop method. 242
Wait method. 244
Wait-For-Requestor method. 244
Rules. 245
About Connect BPEL rules. 248
Connect BPEL form – Completing the Methods tab. 249
Connect BPEL form - Completing the Deployment tab. 251
About Connect Robot rules. 251
About Calendar data instances. 253
More about Calendar data instances. 254
Creating Calendar data instances. 255
Completing the Schedule tab. 256
Divisions. 258
Creating a division. 259
Organization data instances. 260
Organizations – Completing the Create or Save As form. 261
Organization form – Definition tab. 262
Organization form – Chart tab. 266
More about Organization data instances. 268
Organization landing page. 269
Organization Unit data instances. 272
More about Organization Unit data instances. 272

©2025 Pegasystems Inc

Organization Units – Completing the New or Save As form. 273
Organization Unit form – Accounting tab. 274
Organization Unit form - Chart tab. 276
Summary views. 278
OAuth 1.0 clients. 278
Web Service Security profile. 279
Flow markers. 280
Flow Marker form – Understanding the Inputs tab. 281
Flow Marker form — Understanding the Results tab. 282
About SpellChecker Properties data instances. 282
Spellchecker Properties form – Completing the Dictionaries tab. 283
Spellchecker Properties form - Completing the Properties tab. 285
Word templates. 289
Service Java rules (to be deprecated). 293
Service Java rules - Completing the Create, Save As, or Specialization form. 294
Unit testing a Service Java rule. 296
Service JSR94 rules (to be deprecated). 298
Service JSR94 rules - Completing the Create, Save As, or Specialization form. 299
Unit testing a Service JSR 94 rule. 301
Service MQ rules. 304
Service MQ rules - Completing the Create, Save As, or Specialization form. 305
Unit testing a Service MQ rule. 307
Creating an SMS account. 310
Creating a web service policy data instance. 311
How to call through Java-based services. 312
Registry Java Property Transform data instances - Completing the Save As form 313
About Broadcast data instances. 313
Broadcast form - Completing the Message tab. 315

©2025 Pegasystems Inc

Broadcast data instances - Completing the New or Save As form. 315
Circumstance definitions. 316
Circumstance templates. 317
Function Rule. 318
Creating a Function. 319
Function Alias Rules. 321
Library rules. 322
Creating library rules. 323
Messages. 325
Creating a message. 326
Finding messages in your application. 327
Creating a custom message category. 328
Displaying a warning message in a rule. 330
Standard functions for messages. 331
Updating legacy messages and related rules. 331
Property aliases. 333
Property Qualifier rules. 334
Property Qualifier rules - Completing the New or Save As form. 335
About Static Content Bundle rules. 336
Static Content Bundle rules - Completing the New or Save As form. 337
Static Content Bundle rules - Completing the Definition tab. 338
More about Static Content Bundle rules. 339
Text File rules. 340
Text File rules - Completing the Create, Save As, or Specialization form. . 341
Defining URL patterns for Cases. 343
Function Alias rules — Completing the Create, Save As, or Specialization form.
. 349
About Image Content data instances. 351
Integration Connectors landing page. 354
Integrating your application with external resources. 357
Data mapping in services and connectors. 359
Integration Services landing page. 361

©2025 Pegasystems Inc

Connected apps landing page. 361
About Collection Rules. 362
Collection rules. 364
Completing the Basic/Advanced Collection tab. 365
Completing the Preamble and Stop tab. 367
Completing the Specifications tab. 368
Using conditions in a collection. 370
Setting the context of a collection step. 372
Using response actions in a collection. 373
Rule type. 376
Unit testing a collection. 380
About Connect Cassandra rules. 381
Connect Cassandra rules - Completing the Create, Save As, or Specialization form. 382
Connect Cassandra form - Completing the Service tab. 383
Handling connector exceptions. 385
Connect Cassandra rules - Completing the Mappings tab. 387
About Connect CMIS rules. 388
Creating a Connect CMIS Rule. 389
About EJB connector rules (deprecated). 391
Connect EJB rules - Completing the New or Save As form. 393
Connect EJB form - Completing the Service tab. 394
Connect EJB form - Completing the Create tab. 400
About Connect HBase rules. 402
Connect HBase rules - Completing the Create, Save As, or Specialization form 404
About Connect Java rules. 405
Connect Java rules - Completing the Create, Save As, or Specialization form. 407
About Connect SQL rules. 408
Connect SQL form - Data Mapping. 410

©2025 Pegasystems Inc

Connect SQL rules - Completing the Create, Save As, or Specialization form. 422
Connect SQL form - Completing the Open tab. 423
Connect SQL form - Completing the Delete tab. 426
Connect SQL form - Completing the Save tab. 427
More about Connect SQL rules. 429
Atom Servers. 433
Connect Request Processor data instances. 434
Connect Request Processor – Completing the New or Save As form. 435
About SOAP Server data instances. 435
Web service policy data instances. 437
Map eForm rules. 438
Map eForm rules - Completing the Create, Save As, or Specialization form. 439
Map eForm Rules - Completing the Mapping tab. 439
Map eForm rules - Completing the Pages & Classes tab. 441
About Parse Normalize rules. 441
Parse Normalize rules - Completing the Create, Save As, or Specialization form. 442
Converting from directives to JavaServer Pages tags. 443
Toggling features on and off. 446
Class Management landing page. 447
List rules. 448
Creating list rules. 449
More about List rules. 450
About Edit Validate rules. 451
Edit Validate rules- Completing the New or Save As form. 452
Properties. 453
Creating a property. 455
Properties - Completing the Create, Save As, or Specialization form. 458
Property form - Completing the General tab. 462
Property form - Completing the General tab for Page modes. 473

PEGA ©2025 Pegasystems Inc

Property form: Completing the General tab - Value modes. 473
Property form - Completing the General tab for Java Object modes. 489
Property form: General tab — Configuring validation. 490
Property form - Completing the Advanced tab. 502
More about Properties. 514
Implementing and using the TextEncrypted property type. 519
JavaServer Pages tags. 523
About the static JavaServer Page tag. 527
acDataSource JavaServer Page tag. 530
assert JavaServer Page tag. 533
autoComplete JavaServer Page tag. 535
binaryfile JavaServer Page tag. 538
choose JavaServer Page tag. 539
JSP tag — chart. 540
contentURL JavaServer Page tag. 543
dataGrid JavaServer Page tag. 546
dsField JavaServer Page tag. 546
dsFields JavaServer Page tag. 547
file JavaServer Page tag. 547
forEach JavaServer Page tag. 548
include JavaServer Page tag. 553
listView JavaServer Pages tag. 557
lookup JavaServer Page tag — lookup. 561
onlyonce JavaServer Page tag. 573
p:r JavaServer Page tag. 574
Reference properties indirectly in JSP tags. 580
reference JavaServer Page tag. 584
save JavaServer Page tag. 596
URL JavaServer Page tag. 599
when JavaServer Page tag. 606
withEmbedded JavaServer Page tag. 613
withPage JavaServer Page tag. 614

©2025 Pegasystems Inc

withReference JavaServer Page tag. 615
withTarget JavaServer Page tag. 616
Fields that support the Global Resource Settings syntax. 617
Integration-Connectors category. 625
Wizards. 625
About Edit Input rules. 626
Edit Input rules - Completing the Create, Save As, or Specialization form. 626
Connector and Metadata wizard – Completing the Choose Purpose form. . 627
About the Connector and Metadata wizard. 629
About generating connector rules. 631
Service Wizard: Select Service Purpose. 635
Displaying all Functions in the Expression Builder. 636
About the eForm Rules accelerator. 638
eForm accelerator Help: Completing the Select eForm Properties form. . 639
eForm accelerator Help: Completing the Upload File form. 640
eForm accelerator Help: Completing the Select eForm Fields form. 640
eForm accelerator Help: Completing the Map eForm Fields form. 640
eForm accelerator Help: Completing the Review and Save form. 641
Running other upgrade utilities. 641

PEGA ©2025 Pegasystems Inc

Reference

This book contains reference topics for Pega Platform.

- Methods
- Rules
- Wizards

Methods

This book contains reference topics for methods in Pega Platform.

- Branching to another activity
- Calling another activity
- Using the Collect instruction
- Use of Java in an activity step
- Queue instruction – run an activity asynchronously
- Connect-Cassandra method
- Connect-dotNet method (deprecated)
- Connect-EJB method (deprecated)
- Connect-File method (deprecated)
- Connect-FTP method
- Connect-HBase method
- Connect-HTTP method

©2025 Pegasystems Inc

- Connect-Java method (deprecated)
- Connect-JCA method (deprecated)
- Connect-JMS method
- Connect-MQ method
- Connect-REST method
- Connect-SOAP method
- Connect-Wait method
- Commit method
- Activity-Clear-Status method
- Activity-End method
- Activity-List-Add method
- Setting the status of an activity using the Activity-Set-Status method
- Apply-DataTransform method
- Apply-Parse-Delimited method
- Apply-Parse-Structured method
- Apply-Parse-XML method
- Assert-No-Invocation method
- Assign-EstablishContext method
- Call-Async-Activity method
- Call-Automation method

PEGA ©2025 Pegasystems Inc

- Call-Function
- End-Validate method
- Exit-Activity method
- Flow-End method
- Flow-New method
- History-Add method
- History-List method
- Index-Finish method
- Link-Objects method
- Using the Load-DataPage method
- Log-Message method
- Map-Structured method
- Obj-Browse method
- Obj-Delete method
- Obj-Delete-By-Handle method
- Obj-Filter method
- Obj-List-View method
- Obj-Open method
- Obj-Open-By-Handle method
- Obj-Refresh-and-Lock method

©2025 Pegasystems Inc

- Obj-Save method
- Obj-Save-Cancel method
- Obj-Sort method
- Obj-Validate method
- Page-Change-Class method
- Page-Clear-Messages method
- Page-Copy method
- Page-Merge-Into method
- Page-New method
- Page-Remove method
- Page-Rename method
- Page-Set-Messages method
- Page-Unlock method
- Page-Validate method
- Privilege-Check method
- Property-Map-DecisionTable method
- Property-Map-DecisionTree method
- Property-Map-Value method
- Property-Map-ValuePair method
- Property-Ref method

©2025 Pegasystems Inc

- Property-Remove method
- Property-Seek-Value method
- Property-Set-Messages method
- Property-Set-Stream method
- Property-Set method
- Property-Set-XML method
- Property-Validate method
- Property-Set-Corr method
- Publish-Notifications
- RDB-Delete method
- RDB-List method
- RDB-Open method
- RDB-Save method
- Requestor-Stop method
- Rollback method
- Show-Applet method
- Show-Applet-Data method
- Show-HTML method
- Show-Page method
- Show-Property method

PEGA ©2025 Pegasystems Inc

- **Show-Stream method**
- **Start-Validate method**
- **StringBuffer-Append method**
- **StringBuffer-Insert method**
- **StringBuffer-Reset method**
- **TaskStatus-Set method**
- **Text-Normalize method**
- **Thread-Clear method**
- **Thread-Stop method**
- **Wait method**
- **Wait-For-Requestor method**

Branching to another activity

Use the Branch instruction to cause the current activity to find another specified activity and branch to it without a return. When the system executes a Branch step, control transfers to another activity found through rule resolution. Execution of the original activity pauses.

When the branched activity ends, processing of the current activity ends also; no steps after the Branch step are executed.

You can pass parameters to the target activity, or you can share the current activity parameter page with the target activity.

Do not include any steps after a Branch instruction, unless the branch occurs only part of the time (because of a transition) or the later steps can be reached through a label and Jump.

PEGA ©2025 Pegasystems Inc

Parameters

The Branch instruction requires the name of the target activity branched to, and parameters for that activity.

Parameter	Description
activity	In the **Method** field, type the word `Branch`, a space, and the activity name to branch to. SmartPrompt is available. If the second activity is not in the same class as this activity, and not in the same class as the page identified in the **Step Page** field, identify the **Applies To** key part of the second activity also, using the format: `Branch Data-Admin-Operator-ID.Analyze`
Pass current parameter page?	Select if you want the current activity's parameter page to be passed to (shared with) the branch activity, which can alter its contents in any way. Clear this check box to use the array that appears below the check box to pass parameters to the branch activity, on its own parameter page. If you select this box, make sure that input parameter values needed by the branch activity are present on the parameter page before the Branch instruction.

©2025 Pegasystems Inc

Parameter	Description
Value	Complete values for parameters of the branch activity, if you did not check the **Pass current parameter page?** box. Enter literal or property reference values for each parameter that you want to have a value. The system creates a parameter page for the branch activity that is unrelated to the calling activity's parameter page.

Results

At runtime, the system uses rule resolution to locate the activity identified in the branch. It transfers control to the activity and passes the parameters supplied. The step page of this step becomes the primary page of the target activity. Processing of the original activity pauses.

When the branched-to activity ends, processing of the original activity also ends. If the original activity was started by another activity using a CALL instruction, processing resumes with that activity.

Checking the method status

This method updates the *pxMethodStatus* property only if it fails. See How to test method results using a transition.

Calling another activity

Use the Call instruction to cause the current activity to find another specified activity and execute it. When that activity completes, control returns to the calling activity.

The calling activity can pass parameter values to the called activity in two ways.

PEGA ©2025 Pegasystems Inc

- Only parameter values entered are passed if the **Pass current parameter page** check box is not selected in the **Steps** tab.
- The parameter page is shared between the two activities if the **Pass current parameter page** check box is selected in the **Steps** tab. This approach is known as call-by-reference.

You can also pass a reference to a clipboard page as a parameter. The called activity can then use this Page Name parameter as if it were the clipboard page. See Passing Page Name parameters.

Notes on Out parameters

The Call instruction in an activity examines the **In/Out** value of each parameter in the called activity:

- If the parameter is declared as `Out` and the entire parameter page of the current activity is shared with the calling activity, then the value of the Out parameter in the call may be of the form param.name. If the called activity updates the output parameter, the resulting value is available on the parameter page of the calling activity.

- **NOTE:** If a parameter is declared as `Out` and the value of the parameter in the Call panel is itself a property reference, then traditional call-by-reference semantics applies. That is, if the called activity updates the parameter value, the value of the property itself is updated.

In a chain of calling activities (activity Alpha calls Beta which calls Gamma), a property passed by reference from Alpha to Beta can also be passed by reference to Gamma, and changes to the `Out` parameter made within Gamma can be passed all the way back to Alpha. This corresponds to the behavior of call-by-reference in many programming languages.

©2025 Pegasystems Inc

Results

The system uses rule resolution to locate the activity. It then constructs or shares a parameter page and transfers control to the activity.

Accessing iteration values in a called activity

If an activity step involving iteration contains a Call instruction, special parameters are added to the parameter page and can be used within the called activity to access iteration details. See Activity Form — Completing the Steps tab — Entering loops.

Asynchronous execution

As an alternative to the Call instruction, you can use the Queue instruction to execute an activity asynchronously in parallel, in a child requestor.

- Queue instruction – run an activity asynchronously

Using the Collect instruction

Use the Collect instruction in an activity step to call a collection rule. Collections define an ordered sequence of rules to execute.

1. Insert a step on the **Steps** tab of the activity where you want to call the collection.
2. In the **Method** field of the step, type the `Collect` keyword, followed by one space and the name of the collection.
   If you do not know the collection name, press the down arrow key to view a list of available collections in your application.
3. Enter a clipboard page name in the **Step Page** field of the step. This page sets the context, or primary page, for the collection when the step is executed.
   Leave this field blank to use the primary page of the activity instead.
4. Expand the step and enter values in the **Method Parameters** section.

©2025 Pegasystems Inc

Result:

During the processing of a Collect instruction, the system uses rule resolution to locate and execute the collection specified in the **Method** field.

If any step calls the **Stop** method, the collection ceases execution and the Collect instruction ends. In this case, control is returned to the activity and the next step on the Steps tab is processed.

The *pxMethodStatus* property is only updated when a Collect instruction fails. Use the Tracer tool to get more details on how your Collect instruction is processed.

Related tasks

- Creating an activity

Use of Java in an activity step

Most activities can be defined by using only methods and instructions. Some activities require inline Java.

WARNING: Pega does NOT support directly writing Java code against libraries (JAR files) that are distributed with Pega Platform™, even if such code might work. Pega Platform updates and patch releases do NOT guarantee backwards compatibility for any direct calls against third-party APIs. If your business requirements cannot be fulfilled by using official APIs and extension points, contact Pega by going to the Pega Support Center.

©2025 Pegasystems Inc

Reviewing generated Java

Activities run as server-side Java, so you can include custom Java statements within your activities. However, this requires experience with designing, coding, and testing Java.

When you save an activity, the system validates Java code from information in the form and in other Rule instances. This generated Java is read-only. To review the generated Java code, select **Actions** > **View Java**.

The window provides a preview of the Java that implements this Rule instance. The Java code presented is similar to but not identical to the Java that executes at run time, which includes inline Java code from other Rule instances and reflects Rules in the requestor's Ruleset list.

The first time a requestor session selects this activity for execution, the Rules assembly occurs again, generating and compiling Java that is specific to the Ruleset list and other context of that requestor. The next execution of the same execution by that requestor, or other requestors with identical access, does not require assembly or compilation.

The Java instruction

To run your custom Java within the generated Java for an activity, add inline Java into an activity step. These are known as Java steps. Your Java code must be Java 7-compliant.

Inline Java steps can introduce complex processing into your application. Use this facility only when necessary. Following the SmartBuild guardrails, avoid Java steps when other approaches can achieve the same computational result. Activities containing a Java step are flagged with a warning message when you save the Activity form, and when you run the Guardrails tool.

1. Type `Java` in the **Method** field of the step.
2. Click the **Expand** icon to expand the **Method** field. A text box is displayed, in which you can enter normal Java code and the Pega Platform -specific Java.

©2025 Pegasystems Inc

Your Java code can use two sources of the Pega Platform -specific Java:

- Functions defined through the *Rule-Utility-Function* Rule Type. For more information, see Function Rule.
- Java methods defined in the Pega Platform PublicAPI.

In addition, your Java can call methods and Functions in external libraries, if all these conditions are true:

- The JAR file for the library is in the proper directory of the application server.
- The name is included in the class path.
- The Java classes are loaded.

See the Pega Community article *About the Process Commander class paths*. Consider Connect JAVA Rules or Connect EJB Rules as alternatives to Java steps.

Because the Java you write in an activity step is inline Java, you cannot import your Java libraries or add Java `import` statements. However, you can refer to Java classes using a fully qualified class name.

For example, Pega Platform cannot accept the following code:

```
Vector x = new Vector();
```

However, you can specify:

```
java.util.Vector x = new java.util.Vector();
```

Use the `String.equals()` Java method, not the == or != operators, to compare the value of a property to a literal string or to the value of another property.

Calling PublicAPI Java methods

For a complete list of available PublicAPI Java methods, consult the Public Javadoc API documentation. By convention, the `tools` variable identifies the PublicAPI class.

PEGA ©2025 Pegasystems Inc

For example, the Java methods *getProperty()* and *setProperty()* get and set property values on a page. Similarly, the Java methods *getParameterValue()* and *setParameterValue()* get and set parameter values:

- `getProperty(string prop_ref)`
- `setProperty(string dict_ref, string value)`
- `getParameterValue(string param_name)`
- `setParameterValue(string param_name, string value)`

In many cases, your Java code needs to reference variables defined in the Java generated for a step. For example, the generated Java often uses the variable `myStepPage` to refer to the current step page. The step page must be on the clipboard before the Java step starts.

Your Java activity step can access Pega Platform facilities, using the classes and interfaces in the `com.pega.pegarules.pub.*` packages.

For example, the following PublicAPI Java methods are part of the *ClipboardProperty* interface to the `com.pega.pegarules.pub.clipboard` package:

- `public ClipboardProperty getProperty(java.lang.String propertyreference)`
- `public boolean ClipboardProperty setValue(ClipboardProperty avalue)`

Calling Function Rules

In addition to the normal Java libraries, your Java step can call Functions in the *Rule-Utility-Function* class. This class contains Functions (*Rule-Utility-Function* Rule Type), organized into libraries (*Rule-Utility-Library* Rule Type).

Each Function Rule belongs to a Ruleset. To call a Function Rule from a Java step, refer to each library by its Ruleset and library name. For example, a standard library named

PEGA ©2025 Pegasystems Inc

String is defined in the *Pega-RULES* Ruleset. To reference this library in your Java, use this syntax:

```
pega_rules_string.
```

After the dot, enter the name of the *String* library Function using exact case, and any required parameters:

```
pega_rules_string.equalsIgnoreCase ("ABC123DEF", pyWorkPage.Account);
```

Reserved variable names

Certain variables have assigned purposes and are reserved. See Activity Rules — Reserved names for parameters and local variables.

Java EE enterprise support

In many cases, a Java EE application cannot directly access the file system on the application server. To allow customers to prepare for this capability, the PublicAPI contains Java classes that mimic many features of java.io facilities. Together, these are known as the Virtual File Interface.

The *com.pega.pegarules.pub.util* Package contains these classes:

- *PRFile* class — Mimics a subset of methods in `java.io.File`
- *PRInputStream* — Emulates `java.io.InputStream`
- *PROutputStream* class — Emulates `java.io.OutputStream`
- *PRReader* class — Emulates `java.io.Reader`
- *PRWriter* class — Emulates `java.io.Writer`

Consult the PublicAPI Javadoc documentation for details on these Java classes.

In Web tier deployments of Pega Platform, these facilities operate the same as the corresponding java.io facilities.

PEGA ©2025 Pegasystems Inc

In e-tier deployments, operation of file access depends on the following dynamic system setting (DSS):

Dynamic system setting

`prconfig/storage/class/default`

The `prconfig/storage/class/default` setting provides the configuration for the storage mechanisms, as well as the mapping between the resource types and those mechanisms.

The following values are valid for this DSS:

Value	Description
`filesystem`	The regular, physical file system (Default)
`resadaptor`	A Java EE resource adapter
`embedded`	(Reserved)
`datasource`	A database accessed through a JDBC Datasource

> ⚠ **CAUTION:** The *prconfig* setting can have broad impact on the operation of your system. Update with caution.

Checking for security issues

As a best practice for activities that contain hand-crafted Java, run the Rule Security Analyzer to check for code that might expose your application to security vulnerabilities.

PEGA ©2025 Pegasystems Inc

Checking status

Technically, the text `Java` in the **Method** column does not reference a Pega Platform method. But like methods, the Java instruction updates the *pxMethodStatus* property. See Testing method results using a transition.

- Function Rule
- Library rules
- Repeating steps in an activity

Queue instruction – run an activity asynchronously

The Queue instruction uses a Java thread from a limited thread pool. If using the queue instruction, set the agent/threadpoolsize value in the `prconfig.xml` file or dynamic system settings appropriately based on concurrency needs.

Use the Queue instruction:

- To start asynchronous execution of a second activity. The Queue instruction provides an alternative to the Call instruction, in situations where there is no need to wait for completion of the second activity, or where processing in parallel is feasible and beneficial.
- Only when appropriate, where the queued activity can operate in a "fire-and-forget" style.

Do not use the Queue instruction:

- If you have disabled batch requestor processing by setting the *agent/threadpoolsize* value to 0 in the `prconfig.xml` file or dynamic system settings.
- If parallel execution of the activity may introduce locking issues, deadlocks, race conditions or other concurrency risks into your application. Use the Call instruction instead.

PEGA ©2025 Pegasystems Inc

- To allow a background agent, operating as a separate process, to execute an activity asynchronously from the foreground processing, especially when locking is involved. Use the Queue-for-Agent instruction instead.

Results

At runtime, the system uses rule resolution to locate and start the activity.

The activity does not receive a step page or a primary page.

Execution of the queued activity occurs asynchronously, in a batch requestor. The execution progress of the two activities is not predictable and may vary from execution to execution.

Checking the method status

This method updates the *pxMethodStatus* property only if it fails. See How to test method results using a transition.

- Calling another activity

Connect-Cassandra method

Use the Connect-Cassandra method to start a request/response interaction between the Pega Platform and the external data store identified by a Connect Cassandra instance.

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the Step Page specified in the activity step that calls this method is set to the appropriate page name.

This method has one required and two optional parameters:

©2025 Pegasystems Inc

Parameter	Description
ServiceName	Required. Enter the name of the Connect Cassandra instance you want to invoke.
ExecutionMode	Select to indicate the timing of execution: • `Run` — Execute the connector synchronously (default). The next step of the activity containing the Connect-Cassandra method does not execute until the connector response is received. • `Run in Parallel` — This connector runs in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to access results from the connector. By default, an invoked connector retains control of the requestor session until it returns a response. Select this parameter if you want the connector to return control of the session directly after completing the request without waiting for a response. If a subsequent connector in the flow has a shape to accomplish, that connector can send its request before the first connector returns a response. • `Queue` — Add this connector request to a queue, defined by a Connect Request Processor data instance, for background processing by the Pega-

©2025 Pegasystems Inc

Parameter	Description
	IntSvcs *ProcessConnectQueue* agent. This mode works only if the Request Processor field on the Service tab of the service instance identifies a *Data-Admin-RequestProcessor-Connect* data instance. When the agent executes a queued connector request, the execution runs with the authorization profile of the original requestor.
EndPointURL	Optional. Specify the URL of an external system to which you want to connect. A value specified for this parameter overrides the value set in the Connection settings of the Service tab of the Connect Cassandra rule.
Operation	Select the type of HTTP request: • Put • Get

Results

The Connect-Cassandra method works as follows:

1. Using rule resolution, locates the Connect Cassandra instance identified by the ServiceName parameter and the class of the step page or primary page.
2. Uses request information to assemble the request message.
3. If the step page does not exist, creates a new page, of the class identified in the Pages & Classes tab.

©2025 Pegasystems Inc

4. Checks whether simulation is in force for this instance. If true, results depend on the Execution mode value:

   `Run`
   : Executes the simulation activity synchronously

   `Run In Parallel`
   : Executes the simulation activity in a separate requestor

   `Queue`
   : Queues the simulation activity

5. If simulation is not in force, calls the service, or queues the request, depending on the Execution mode value:

   `Run`
   : Calls the service and waits for a response.

   `Run in parallel`
   : Executes in a separate requestor. (This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)

   `Queue`
   : Creates a queue item based on the class and other characteristics identified in the Connect Request Processor, for later background processing by the Pega-IntSvcs agent.

6. Processes the response message when it is received. (No response is available for queued connector requests.)

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> ⚠ **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-Cassandra method with the RunInParallel check box selected fails in a way that

©2025 Pegasystems Inc

> causes the Connect-Wait method to be skipped, the child requestor can fail. If the limit of parallel requestors is then eventually reached, the parent requestor will fail as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

- Connect-Wait method

Connect-dotNet method (deprecated)

Use the Connect-dotNet method to invoke a Web service identified in a Connect dotNet rule (an instance of the *Rule-Connect-dotNet* rule type).

> **NOTE:** This method is deprecated and is no longer supported.

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the Step Page column that calls this method contains the non-blank name of such a page.

This method has two parameters:

Parameter	Description
ServiceName	Enter the Service Name (second key part) of a Connect dotNet rule (*Rule-Connect-dotNet* rule type). To find the Connect dotNet rule, rule resolution uses the class of the step page as the first key part of the Connect dotNet rule.
RunInParallel	Select if this connector is intended to run in parallel as a child requestor of the current

©2025 Pegasystems Inc

Parameter	Description
	requestor. If selected, include the Connect-Wait method later in the current or a follow-on activity to access results from the connector. Do not select this box if the Connect dotNet rule supports compensating actions.
EndPointURL	Optional. Enter an endpoint URL value to override the value in the Connect dotNet rule.

Results

The Connect-dotNet method works as follows:

1. Using rule resolution, the method locates the Connect dotNet rule identified by the parameter.
2. Uses request information to assemble a SOAP request for the Web service.
3. If the step page does not exist, creates a new page, of the class identified in the Pages & Classes tab.
4. Checks whether simulation is in force for this rule. If so, the simulation stub activity is called.
5. (If simulation is not in force) sends the SOAP message (or calls the simulation activity), in a temporary separate requestor session if the RunInParallel box is selected. (This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)
6. Processes the response message when it is received.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

PEGA ©2025 Pegasystems Inc

Design the activity to detect and handle exceptions. If the Connect-dotNet method with the RunInParallel check box selected fails in a way that causes the Connect-Wait method to be skipped, the child requestor can be left hanging. If the limit of parallel requestors is then eventually reached, the parent requestor will fail as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

- Creating connector simulators

Connect-EJB method (deprecated)

Use the Connect-EJB method to invoke a connector to an external Enterprise JavaBean.

> **NOTE:** This method is deprecated and is no longer supported.

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the Step Page column that calls this method contains the non-blank name of such a page.

This method has two parameters:

Parameter	Description
ServiceName	Enter the Service Name (second key part) of a Connect EJB rule (*Rule-Connect-EJB* rule type).
RunInParallel	Select if this connector is intended to run in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait

©2025 Pegasystems Inc

Parameter	Description
	method later in the current or a follow-on activity to access results from the connector.

Results

The Connect-EJB method works as follows:

1. Using rule resolution, the system locates the Connect EJB rule identified by the **ServiceName** parameter and the class of the step page or primary page.
2. The system uses request information to call one of the `EJB Create()` methods or one of the Java class business methods.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.
4. It checks whether simulation is in force for this rule. If so, calls the simulation stub activity.
5. If simulation is not in force, calls the EJB method(s), in a separate requestor if the **RunInParallel** box is selected. (This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)
6. Processes the response when it is received.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> ⚠ **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-EJB method with the **RunInParallel** check box selected fails in a way that causes the Connect-Wait method to be skipped, the child requestor can be left hanging. If the limit of parallel requestors is then eventually reached, the parent requestor will fail as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or

PEGA ©2025 Pegasystems Inc

> dynamic system settings using the `services/maxRequestorChildren` setting.

- About EJB connector rules (deprecated)
- Creating connector simulators
- Connect-Wait method

Connect-File method (deprecated)

Use the Connect-File method to invoke a Connect File rule (*Rule-File-Connect* rule type). The rule can create, add contents to, or close an output file.

> **NOTE:** This method is deprecated and is no longer supported.

Parameters

This method has three parameters:

Parameter	Description
ConnectorName	Enter the **Connector Name** (second key part) of a Connect File rule, an instance of the *Rule-Connect-File* rule type. The system uses the class of the step page as the first key part of the rule.
Operation	Select to identify the source of output to be added to the file. • `Write from HTML Stream Rule` — An HTML rule. • `Write from XML Stream Rule` — An XML Stream rule.

©2025 Pegasystems Inc

Parameter	Description
	• `Write from eForm` — A Map eForm rule • `Work Attachment` — A work item attachment • `Write from Clipboard Property` — The value of a single property. • `Write from Map Structured rule` — The output data structure produced by a map structured rule. • `Close File` — No further output; close the file. (This is meaningful and necessary only when the **Keep File Open** check box is selected on a Connect File rule.
Parameter	Identify the key of the **Operation** parameter. Identify the property, key of the rule, or key of the work item attachment designated in the **Operation** field. • For `Clipboard Property`, enter a property reference, literal text, parameter, local variable, or complex function. Literal text must be surrounded by double quotes. • For `HTML Stream rules`, `XML Stream rules`, or `Write from Map Structured` rule, enter the second key part of the rule. The system uses the class of the step page (or the class

©2025 Pegasystems Inc

Parameter	Description
	of the activity) with rule resolution to find the rule. (For XML Stream rules, the third key part must be `MapFrom`.) • For `eform rules`, enter a valid *Code-Pega-eForm* page. Ensure that the *pyEform* property of this page contains the PDF content, in the form of a byte array. • For `Work Attachment`, enter the name of a valid page of class *Data-WorkAttach-File*. • Leave blank if the **Operation** parameter is `Close File`.

Results

The Connect-File method works as follows:

1. Using rule resolution, locates the Connect File rule.
2. Uses information in the **Service** tab of the rule to identify the path and file name of the output file, and processing instructions.
3. Locates the rule or other object identified by the **Operation** and **Parameter**.
4. Opens the file if not already open, and appends the output.
5. Generates the output using stream processing, map eform processing or map structured processing, if specified. If the map structured rule specifies a character encoding other than `Default`, output characters are converted to the specified character set using normal Java methods.
6. Closes the file, unless the **Keep File open option** on the Connect File rule is selected.
7. Updates the value on the clipboard of the status and status message properties identified in the rule.

©2025 Pegasystems Inc

If the **Operation** is `Close File,` steps 5 and 6 do not occur and the output file is closed.

Operation of this method is synchronous. Further processing in the activity does not resume until the file processing completes.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Example

For an example using this method, see the Pega Community article *How to write a file to the local file system using a file connector*.

- Map eForm rules
- Map Structured rules

Connect-FTP method

Use this method to copy a file from one file system to another file system using File Transfer Protocol.

Parameters

The Connect-FTP method has four parameters:

Parameter	Description
FTPServer	Select the name of an FTP server data instance to use for this file transfer.
RemotePath	Specify the relative path to the destination directory specified on the FTP service. Use forward slashes regardless of whether the FTP Service is running on Windows or Linux.

©2025 Pegasystems Inc

Parameter	Description

	For example, if the FTP directory on the FTP service is set to c:\ftp, specify the relative path as a blank path between quotes:
	
	`""`
	
	If the c:\ftp directory on the FTP service has a documents directory under it, c:\ftp\documents, specify the relative path as:
	
	`"documents"`
	
	You can enter a property name, property reference or expression that at runtime evaluates to the remote file path.
	
	For Pega Cloud clients specifying a relative path in the Pega Cloud File storage folder, enter `file://pegacloudfilestorage:/attachments`, as an example.

LocalFile	Identify the source file path and file name. Use syntax appropriate to the operating system that hosts the Pega Platform server. If the path contains backslash characters, escape each backslash character by doubling it.

©2025 Pegasystems Inc

Parameter	Description
	You can enter a property name, property reference or expression that at runtime evaluates to the local file path and name.
TransferMode	Select `ASCII` for character-mode data transfers, or `Binary` for binary data. The default is ASCII.

Results

The system uses information on the **Environment** tab of the FTP server data instance to log into the FTP server. Upon successful log-in, the system transfers the file identified in the *LocalFile* parameter to the filename and directory identified in the *RemoteFile* parameter, and then ends the FTP connection.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- FTP Server data instances
- Secure file transfer in Pega

Connect-HBase method

Use the Connect-HBase method to start an request/response interaction between the Pega Platform and the external data store identified by a Connect HBase instance (*Rule-Connect-HBase* rule type).

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the **Step Page** specified in the activity step that calls this method is set to the appropriate page name.

©2025 Pegasystems Inc

This method has one required and two optional parameters:

Parameter	Description
ServiceName	Required. Enter the name of the Connect HBase instance you want to invoke.
Execution Mode	Select to indicate the timing of execution: • `Run` — Execute the connector synchronously (default). The next step of the activity containing the Connect-HBase method does not execute until the connector response is received. • `Run in Parallel` — This connector runs in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to access results from the connector. By default, an invoked connector retains control of the requestor session until it returns a response. Select this parameter if you want the connector to return control of the session directly after completing the request without waiting for a response. If a subsequent connector in the flow has a shape to accomplish, that connector can send its request before the first connector returns a response. • `Queue` — Add this connector request to a queue, defined by a Connect Request Processor data instance, for

©2025 Pegasystems Inc

Parameter	Description
	background processing by the Pega-IntSvcs *ProcessConnectQueue* agent. This mode works only if the **Request Processor** field on the **Service** tab of the service instance identifies a *Data-Admin-RequestProcessor-Connect* data instance. When the agent executes a queued connector request, the execution runs with the authorization profile of the original requestor.
EndPointURL	Optional. Specify the URL of an external system to which you want to connect. A value specified for this parameter overrides the value set in the **Connection** settings of the **Service** tab of the Connect HBase rule.
Operation	Select the type of HTTP request: • Put • Get

Results

The Connect-HBase method works as follows:

1. Using rule resolution, locates the Connect HBase instance identified by the **ServiceName** parameter and the class of the step page or primary page.
2. Uses request information to assemble the request message.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.

©2025 Pegasystems Inc

4. Checks whether simulation is in force for this instance. If true, results depend on the **Execution mode** value:
   - `Run` — Executes the simulation activity synchronously
   - `Run In Parallel` — Executes the simulation activity in a separate requestor
   - `Queue` — Queues the simulation activity
5. If simulation is not in force, calls service, or queues the request, depending on the Execution mode value:
   - `Run` — Calls the service and waits for a response.
   - `Run in parallel` — Executes in a separate requestor. (This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)
   - `Queue` — Creates a queue item based on the class and other characteristics identified in the Connect Request Processor, for later background processing by the Pega-IntSvcs agent.
6. Processes the response message when it is received. (No response is available for queued connector requests.)

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> ⚠ **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-HBase method with the RunInParallel check box selected fails in a way that causes the Connect-Wait method to be skipped, the child requestor can fail. If the limit of parallel requestors is then eventually reached, the parent requestor will fail as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

- About Connect HBase rules

PEGA ©2025 Pegasystems Inc

- Connect-Wait method

Connect-HTTP method

Use the Connect-HTTP method to start an HTTP request/response interaction between the Pega Platform and the external system identified by a Connect HTTP rule (*Rule-Connect-HTTP* rule type).

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the **Step Page** specified in the activity step that calls this method is set to the appropriate page name.

This method has one required and two optional parameters:

Parameter	Description
ServiceName	Required. Enter the **Service Name** — second key part — of the Connect HTTP rule you want to invoke. The system uses the class of the step page as the first key part.
Execution Mode	Select to indicate the timing of execution • `Run` — Execute the connector synchronously; the next step of the activity containing the Connect-HTTP method does not execute until the connector response is received. (This is the default.) • `Run in Parallel` — This connector is to run in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to

©2025 Pegasystems Inc

Parameter	Description
	access results from the connector. By default, an invoked connector retains control of the requestor session until it returns a response. Select this parameter if you want the connector to return control of the session directly after completing the request without waiting for a response. If a subsequent connector in the flow has a shape to accomplish, that connector can send its request before the first connector returns a response. • `Queue` — Add this connector request to a queue, defined by a Connect Request Processor data instance, for background processing by the Pega-IntSvcs *ProcessConnectQueue* agent. This mode works only if the **Request Processor** field on the **Service** tab of the HTTP Service rule identifies a *Data-Admin-RequestProcessor-Connect* data instance. When the agent executes a queued connector request, the execution is performed with the authorization profile of the original requestor.
EndPointURL	Optional. Specify the URL of an external system to which you want to connect. A value specified for this parameter overrides

©2025 Pegasystems Inc

Parameter	Description
	the value set in the **End Point URL** field on the **Service** tab of the Connect HTTP rule.

Results

The Connect-HTTP method works as follows:

1. Using rule resolution, locates the Connect HTTP rule identified by the **ServiceName** parameter and the class of the step page or primary page.
2. Uses request information to assemble a HTTP request message for the Web service.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.
4. Checks whether simulation is in force for this rule. If true, results depend on the **Execution mode** value:
   - `Run` — Executes the simulation activity synchronously
   - `Run In Parallel` — Executes the simulation activity in a separate requestor
   - `Queue` — Queues the simulation activity
5. If simulation is not in force, calls the Web service, or queues the request, depending on the Execution mode value:
   - `Run` — Calls the Web service and waits for a response.
   - `Run In Parallel` — Executes in a separate requestor. (This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)
   - `Queue` — Creates a queue item based on the class and other characteristics identified in the Connect Request Processor, for later background processing by the Pega-IntSvcs agent.
6. Processes the HTTP response message when it is received. (No response is available for queued connector requests.)

©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property.

> **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-HTTP method with the **RunInParallel** check box selected fails in a way that causes the Connect-Wait method to be skipped, the child requestor can fail. If the limit of parallel requestors is then eventually reached, the parent requestor will fail as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

- **About Connect HTTP rules (deprecated)**

- About Connect HTTP rules (deprecated)
- Connect-Wait method

About Connect HTTP rules (deprecated)

The HTTP integration interface supports interactions between your Pega Platform applications and other systems through HTTP.

> **IMPORTANT:** Connect HTTP rules are deprecated. After an upgrade to Pega Platform 8.6 and later, you cannot create new Connect HTTP rules. However, existing Connect HTTP rules will continue to function. Use Connect REST rules instead. For more information, see Creating a REST integration by using the Create REST integration wizard.

The following tabs are available on this form:

- Service
- Request

©2025 Pegasystems Inc

- Response

Use Connect HTTP rules when you want your application to send XML or string data (text) as messages to an external system without the need to comply with messaging standards or protocols like SOAP.

Connect HTTP rules provide a low-overhead, stateless alternative to SOAP.

Access

To list the Connect HTTP rules in your application, use the Application Explorer or click **Configure** > **Integration** > **Connector Definitions & Simulations**.

To list all Connect HTTP rules that are available to you, use the Records Explorer.

Development

To create Connect HTTP rules directly in your application, click Create > **Integration-Connectors** > **Connect HTTP**.

The HTTP connector supports sending and receiving attachments with the HTTP response message. For more information, see Pega Community article *How to configure a SOAP or dotNET connector to send or receive SOAP with attachments*.

Where referenced

Connect HTTP rules can be referenced in an Integrator shape of a flow rule, or on the **Definition** tab of a Data Page rule, as the source of data for the contents of the data page.

Category

A Connect HTTP rule is an instance of the *Rule-Connect-HTTP* rule type. Connect HTTP rules belong to the Integration-Connectors category.

- Integration-Connectors category

PEGA ©2025 Pegasystems Inc

- Viewing Rule history

Connect-Java method (deprecated)

Use the Connect-Java method to call a public method of an external Java class, using a *Rule-Connect-Java* rule.

> **NOTE:** This method is deprecated and is no longer supported.

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the **Step Page** column that calls this method contains the non-blank name of such a page.

This method has two parameters:

Parameter	Description
ServiceName	Enter the **Service Name** (second key part) of a Connect Java rule, an instance of the *Rule-Connect-Java* rule type. The system uses the class of the step page as the first key part.
RunInParallel	Select if this connector is intended to run in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to access results from the connector.

Results

The Connect-Java method works as follows:

©2025 Pegasystems Inc

1. Using rule resolution, locates the Connect Java rule identified by the **ServiceName** parameter and the class of the step page or primary page.
2. Uses request information to call one of the Java class constructors or one of the Java class business methods method.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.
4. Checks whether simulation is in force for this rule. If so, calls the simulation stub activity.
5. If simulation is not in force, calls the Java method. It creates a separate requestor if the **RunInParallel** box is selected. This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.
6. Processes the response when it is received.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> ⚠ **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-Java method with the RunInParallel check box selected fails in a way that causes the Connect-Wait method to be skipped, the child requestor can be left hanging. If the limit of parallel requestors is then eventually reached, the parent requestor will fail as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

Restrictions

This method cannot be used in a step that involves iteration.

- About Connect Java rules

©2025 Pegasystems Inc

- Connect-Wait method

Connect-JCA method (deprecated)

Use the Connect-JCA method to communicate through a JCA resource adapter to an external system.

> **NOTE:** This method is deprecated and is no longer supported.

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the **Step Page** column that calls this method contains the non-blank name of such a page.

This method has two parameters:

Parameter	Description
ServiceName	Enter the **Service Name** (second key part) of a Connect JCA rule (*Rule-Connect-JCA* rule type). The system uses the class of the step page as the first key part.
RunInParallel	Select if this connector is intended to run in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to access results from the connector.

Results

The Connect-JCA method works as follows:

©2025 Pegasystems Inc

1. Using rule resolution, locates the Connect JCA rule identified by the **ServiceName** parameter and the class of the step page or primary page.
2. Uses request information to assemble a request.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.
4. Checks whether simulation is in force for this rule. If so, calls the simulation stub activity.
5. If simulation is not in force, sends the request to a resource adapter identified in the Connect JCA rule, using a separate requestor if the **RunInParallel** box is selected. (This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)
6. Processes the response when it is received.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> ⚠ **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-JCA method with the **RunInParallel** check box selected fails in a way that causes the Connect-Wait method to be skipped, the child requestor can be left hanging.If the limit of parallel requestors is then eventually reached, the parent requestor will fail as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

- Creating connector simulators
- Connect-Wait method

Connect-JMS method

Use the Connect-JMS method to send a JMS message to an external system using the Java Message Service application programmer interface.

©2025 Pegasystems Inc

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the **Step Page** column that calls this method contains the non-blank name of such a page.

This method has two parameters:

Parameter	Description
ServiceName	Enter the **Service Name** (second key part) of a Connect JMS rule (*Rule-Connect-JMS* rule type). The system uses the class of the step page as the first key part.
RunInParallel	Select if this connector is intended to run in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to access results from the connector.

Results

The Connect-JMS method works as follows:

1. Using rule resolution, locates the Connect JMS rule identified by the **ServiceName** parameter and the class of the step page or primary page.
2. Uses request information to assemble a request JMS message.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.
4. Checks whether simulation is in force for this rule. If so, calls the simulation stub activity.
5. If simulation is not in force, sends a JMS message, in a separate requestor if the **RunInParallel** box is selected. (This temporary requestor has the same access

©2025 Pegasystems Inc

group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)

6. Processes the response when it is received.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-JMS method with the RunInParallel check box selected fails in a way that causes the Connect-Wait method to be skipped, the child requestor can be left hanging. If the limit of parallel requestors is then eventually reached, the parent requestor fails as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

- Connect JMS rules
- Creating connector simulators
- Connect-Wait method

Connect-MQ method

Use the Connect-MQ method to invoke a connector to an external system that uses IBM WebSphere MQ messaging services.

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the **Step Page** column that calls this method contains the non-blank name of such a page.

This method has two parameters:

©2025 Pegasystems Inc

Parameter	Description
ServiceName	Enter the **Service Name** (second key part) of a Connect MQ rule (*Rule-Connect-MQ* rule type). The system uses the class of the step page as the first key part of the rule.
RunInParallel	Select if this connector is intended to run in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to access results from the connector.

Results

The Connect-MQ method works as follows:

1. Using rule resolution, locates the Connect MQ rule identified by the **ServiceName** parameter and the class of the step page or primary page.
2. Uses request information in the Connect MQ rule to assemble an MQ request message.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.
4. Checks whether simulation is in force for this rule. If so, calls the simulation stub activity.
5. If simulation is not in force, sends the MQ message, in a separate thread if the **RunInParallel** box is selected. (This temporary requestor has the same access group and clipboard as the original.) If the limit of parallel requestors is then eventually reached, the parent requestor will fail as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.
6. Processes the MQ response message if any when it is received.

©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Connect MQ rules
- Creating connector simulators
- Connect-Wait method

Connect-REST method

Use the Connect-REST method to start an HTTP request/response interaction between the Pega Platform and the external system identified by a Connect REST rule (*Rule-Connect-REST* rule type).

Parameters

This method accepts three required and one optional parameter.

Parameter	Description
ServiceName	Service name of the Connect REST rule to run.
EndPointURL	Optional. Specify the URL of an external system to which you want to connect. A value specified for this parameter overrides the value set in the **End Point URL** field on the **Service** tab of the Connect REST rule.
MethodName	HTTP method to execute. Select GET, POST, PUT or DELETE.
ExecutionMode	Select to indicate the timing of execution • `Run` — Execute the connector synchronously; the next step of the

©2025 Pegasystems Inc

Parameter	Description
	activity containing the Connect-REST method does not execute until the connector response is received. (This is the default.) • `Run in Parallel` — This connector is to run in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to access results from the connector. By default, an invoked connector retains control of the requestor session until it returns a response. Select this parameter if you want the connector to return control of the session directly after completing the request without waiting for a response. If a subsequent connector in the flow has a shape to accomplish, that connector can send its request before the first connector returns a response. Parallel execution of the REST connector with an authorization code is not supported. • `Queue` — Add this connector request to a queue, defined by a Connect Request Processor data instance, for background processing by the Pega-IntSvcs *ProcessConnectQueue* agent. This mode works only if the **Request Processor** field on the **Service** tab of the Connect REST rule identifies a *Data-*

©2025 Pegasystems Inc

Parameter	Description
	Admin-RequestProcessor-Connect data instance. When the agent executes a queued connector request, the execution is performed with the authorization profile of the original requestor.

Results

The Connect-REST method works as follows:

1. Using rule resolution, locates the Connect REST rule identified by the **ServiceName** parameter and the class of the step page or primary page.
2. Uses the HTTP method identified by the **MethodName** parameter to assemble a HTTP request message.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.
4. Checks whether simulation is in force for this rule. If true, results depend on the Execution mode value:
   - `Run` — Executes the simulation activity synchronously
   - `Run In Parallel` — Executes the simulation activity in a separate requestor
   - `Queue` — Queues the simulation activity
5. If simulation is not in force, calls the Web service, or queues the request, depending on the Execution mode value:
   - `Run` — Calls the Web service and waits for a response.
   - `Run In Parallel` — Executes in a separate requestor. (This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)

©2025 Pegasystems Inc

- `Queue` — Creates a queue item based on the class and other characteristics identified in the Connect Request Processor, for later background processing by the Pega-IntSvcs agent.

6. Processes the response message when it is received. (No response is available for queued connector requests.)

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-REST method with the **Execution Mode** of `Run In Parallel` fails in a way that causes the Connect-Wait method to be skipped, the child requestor can be left hanging. If the limit of parallel requestors is then eventually reached, the parent requestor fails as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

- Connect REST rules
- Connect-Wait method

Connect-SOAP method

Use the Connect-SOAP method to invoke a Web service identified in a Connect SOAP or Connect SAP rule (*Rule-Connect-SOAP* rule type).

Operation of this method can be affected by dynamic system settings. See More about Connect SOAP rules.

©2025 Pegasystems Inc

Parameters

This method operates on a named clipboard page that is the source and destination of data mappings. Make sure the **Step Page** column that calls this method contains the non-blank name of such a page.

This method has four parameters:

Parameter	Description
ServiceName	Enter the **Service Name** (second key part) of a Connect SOAP rule, an instance of the *Rule-Connect-SOAP* rule type. The system uses the class of the step page as the first key part of the rule.
Execution Mode	Select to indicate the timing of execution • `Run` — Execute the connector synchronously; the next step of the activity containing the Connect-SOAP method does not execute until the connector response is received. • `Run in Parallel` — This connector is to run in parallel as a child requestor of the current requestor. If selected, use the Connect-Wait method later in the current or a follow-on activity to access results from the connector. Do not select this value if the Connect SOAP rule supports compensating actions. • `Queue` — Add this connector request to a queue, defined by a Connect Request Processor data instance, for

©2025 Pegasystems Inc

Parameter	Description
	background processing by the Pega-IntSvcs *ProcessConnectQueue* agent. This mode works only if the **Request Processor** field on the **Service** tab of the SOAP Service rule identifies a *Data-Admin-RequestProcessor-Connect* data instance. When the agent executes a queued connector request, the execution is performed with the authorization profile of the original requestor.
EndPointURL	Optional. Enter an endpoint URL value to override the value in the Connect SOAP rule.

Results

The Connect-SOAP method works as follows:

1. Using rule resolution, locates the Connect SOAP rule identified by the **ServiceName** parameter and the class of the step page or primary page.
2. Uses request information to assemble a SOAP request message for the Web service.
3. If the step page does not exist, creates a new page, of the class identified in the **Pages & Classes** tab.
4. Checks whether simulation is in force for this rule. If true, results depend on the **Execution mode** value:
   - `Run` — Executes the simulation activity synchronously
   - `Run In Parallel` — Executes the simulation activity in a separate requestor
   - `Queue` — Queues the simulation activity

©2025 Pegasystems Inc

5. If simulation is not in force, calls the Web service, or queues the request, depending on the **Execution mode** value:
   - `Run` — Calls the Web service and waits for a response.
   - `Run In Parallel` — Executes in a separate requestor. (This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.)
   - `Queue` — Creates a queue item based on the class and other characteristics identified in the Connect Request Processor, for later background processing by the Pega-IntSvcs agent.
   - Processes the SOAP response message when it is received. (No response is available for queued connector requests.)

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> ⚠ **CAUTION:** Design the activity to detect and handle exceptions. If the Connect-SOAP method with the **Execution Mode** of `Run In Parallel` fails in a way that causes the Connect-Wait method to be skipped, the child requestor can be left hanging. If the limit of parallel requestors is then eventually reached, the parent requestor fails as well. By default, the limit is 10 requestors in parallel; you can establish a higher or lower value in the `prconfig.xml` file or dynamic system settings using the `services/maxRequestorChildren` setting.

- Creating a Connect SOAP rule
- Creating connector simulators
- Connect-Wait method

Connect-Wait method

Use the Connect-Wait method

PEGA ©2025 Pegasystems Inc

- to join the current requestor session with a parallel connector requestor session created previously. Any of the four RDB- methods or eight Connect- methods can operate in parallel.
- with the Load-DataPage method. Load-DataPage uses the PoolID property to group requestors loading data as part of the same activity. Connect-Wait lets processing wait for the stated timeout interval, or until all requestors with the same PoolID have finished loading data.

Parameters

This method accepts two parameters:

Parameter	Description
WaitSeconds	Enter a positive number of seconds as a timeout interval, or enter -1 to cause an indefinite wait.
PoolID	PoolID is an arbitrary identifier used in the activity methods Load-DataPage and Call-Async-Activity to help manage asynchronous data loading. Provide a value when using Connect-Wait in an activity of type Asynchronous. PoolID can be any valid String, a property reference, or a parameter. PoolID exists in the context of the activity using it, so you can have identical PoolID values in different activities without causing a problem.

©2025 Pegasystems Inc

Results

1. WaitSeconds has a value and PoolID has a value: Wait for child requestors to finish (if any) and then wait for data pages which belong to PoolID to finish loading. The entire operation has a timeout of the specified WaitSeconds.
2. WaitSeconds has no value and PoolID has a value: Wait for child requestors to finish (if any) and then wait for data pages which belong to PoolID to finish loading. The entire operation has a timeout of indefinite.
3. WaitSeconds has a value and PoolID has no value: Wait for child requestors to finish (if any). The entire operation has a timeout of specified WaitSeconds.
4. WaitSeconds has no value and PoolID has no value: Wait for child requestors to finish (if any). The entire operation has a timeout of indefinite.

When awakened, if the method status is `Good`, the activity can examine the results pages of the connector methods. If the method status is `Fail`, this may indicate that one or more of the parallel child requestors has not completed.

The Connect-Wait method sets the method status to `Good` or `Fail` even if you simulate the connector rule through a Connector Simulation data instance.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- About Connect SQL rules
- Creating connector simulators
- Connect-dotNet method (deprecated)
- Connect-EJB method (deprecated)
- Connect-HTTP method
- Connect-Java method (deprecated)
- Connect-JMS method
- Connect-MQ method
- Connect-SOAP method

PEGA ©2025 Pegasystems Inc

- RDB-Open method
- RDB-Save method
- RDB-Delete method
- RDB-List method

Commit method

Use this method to commit all uncommitted database changes. This method writes all the instances specified by one or more earlier Obj-Save methods to the PegaRULES database (for internal classes) and to external databases (for external classes).

Basics

The Commit method operates on Thread pages that can operate across multiple instances. After an object instance is saved into the database, it is persistent.

Use the Rollback method or the Obj-Save-Cancel method to cancel a previous Obj-Save before commit.

> ⚠ **CAUTION:** Do not use the Commit method in an activity that has an Activity Type of `Utility, Route, Assign, Notify`, or `Integrator`. Flow processing performs commits automatically when the flow ends and when an assignment is created or completed. To avoid interference with this approach, do not use Commit in the flow-related activities in your application. (When you save an activity that uses Commit, if the Applies To key part of the activity is derived from the Work- base class, a warning message appears).

Most database updates require that the requestor hold a lock on the object. Typically a requestor acquires this lock when the requestor first opens the object with the Obj-Open, Obj-Open-By-Handle, or Obj-Refresh-and-Lock method. Note these two exceptions:

©2025 Pegasystems Inc

- A lock is not required for the first-ever commit operation of a newly created object.
- A lock is not required for objects of classes that don't support locking.

Parameters

The Commit method has no parameters.

Results

The system commits all deferred write operations (including deletes) for this requestor in the queue to the database. If any of the writes fail, they all fail.

Commit operations can trigger the execution of Declare Trigger rules. The system updates indexes as required by executing Declare Index rules.

All locks held by the requestor Thread (except those that were acquired with the **ReleaseOnCommit** box not selected in the Obj-Open method) are released.

Any clipboard pages which hold objects retrieved through linked property processing are deleted, as potentially stale.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

When a deferred — that is, non-immediate — Obj-Save or Obj-Delete method fails, the next Commit method will also fail unless one of the following is performed successfully before the Commit method is attempted again:

- Another deferred Obj-Delete (or Obj-Save) method is performed on the instance
- An Obj-Save-Cancel is performed
- A Rollback method is performed

PEGA ©2025 Pegasystems Inc

> ⚠ **CAUTION:** If during the database save, any of the four properties listed below contain a value that is wider than the corresponding database column, the Commit method fails. The system associates the message *Database-ColumnTruncated* with the property in error. (This message begins "Property Zzzzz has been truncated...")

- *pxObjClass*
- *pxInsName*
- *pzInsKey*
- *pyClassName*

If any property other than these four properties contains a value that is longer than the width of the database column, the system associates a warning message *Database-BadTableMapping-KeyPropertyTooBig* with the property and returns a warning status. The database is updated and data for the column is truncated to fit. The system changes the last character of the truncated value to a plus sign (+) character, as a more enduring indication of this problem in case the warning message is missed or ignored.

Java EE two-phase Commit support

In a Java EE application server setting, the JTA transaction aspects of this method depend on how the Pega Platform system is deployed. If your system is deployed appropriately in the Enterprise tier of a Java EE application server, a Commit operation can be processed as a full Java Transaction API (JTA) transaction. This is useful when objects are saved to more than one database,

For example a single Commit operation may update both the PegaRULES database and an external database (made visible through the External Database Table wizard).

Troubleshooting deferred write operations

As a debugging aid, the standard HTML rule *@baseclass.DeferredOps* interrogates system internals and presents an ordered list of deferred (uncommitted) database operations for the current Thread. In a step immediately before a Commit method,

©2025 Pegasystems Inc

your activity can display this HTML rule using the Show-Page method, to see what will be committed.

Design consideration and notes

- When you include the Commit method in an activity, design the processing so that the Commit method occurs only after errors are intercepted, checked, and corrected. If a Commit method fails, it is unlikely that processing can continue in any reasonable way, so it is important to include a transition in the Commit step that checks the status.
- Even when processing can continue after Commit failure, design your activity (after debugging and fixing the cause of the failure) to collect any required user input that corrects the problems, and re-perform any Obj-Save methods that preceded the Commit call.

Performance statistics

Three statistics in the Full Details display of the Performance tool report on Commit operations from your requestor session

- Elapsed time writing to the database (property *Code-Pega-PAL.pxCommitElapsed*)
- Database Commit Count (property *Code-Pega-PAL.pxCommitCount*)
- Database Rows Committed (property *Code-Pega-PAL.pxCommitRowCount*)

Additional resources

For more information about JTA operation, see *Distributed Transactions and PegaRULES Process Commander*, a file on Pega Community.

- Configuring Declare Index Rules
- Declare Trigger Rules
- Obj-Save method
- Obj-Save-Cancel method
- Rollback method

PEGA ©2025 Pegasystems Inc

Activity-Clear-Status method

Use this method to clear the activity status value, setting it to `Good` or another status determined by the operation of the parameter.

This method can be useful following an Obj-Open method or Obj-Open-by-Handle method in situations where failure is considered likely.

Parameters

This method uses a single parameter.

Parameter	Description
BackoutWorstOnlyStatus	Select to clear the most severe status values (for example `Error`) and retain a less severe status value (for example `Warn` or `Good`).

Results

If the parameter is not selected, the *pyMethodStatus* property is forced to the value `Good`. If the parameter is selected, the most severe status is dropped, and *pxMethodStatus* is set to the next most severe status (`Fail` or `Good`).

Checking the method status

This method may update the *pxMethodStatus* property. See How to test method results using a transition.

- Setting the status of an activity using the Activity-Set-Status method

©2025 Pegasystems Inc

Activity-End method

This method appears in activities that support the Pega Platform infrastructure, but is otherwise rarely needed. In ordinary circumstances, use the Exit-Activity method, not Activity-End, to return to a calling activity from an internal step of the current activity.

Use the Activity-End method to cause the system to:

1. End the current activity and all calling activities (by throwing an exception that is caught).
2. Transmit any HTML code previously produced to an interactive user.

For example, you can use the Activity-End method to end an activity conditionally in mid-processing.

Processing continues with the next activity on the internal Activity List. If the list is empty, the system presents Status Message information to the user. To hide the Status Message form from the user, provide the user with an HTML form.

This method exits all activities in the current call stack. If one activity calls another activity that executes the Activity-End method, both activities end; the system does not return to the calling activity. Similarly, if Alpha calls Beta, which calls Gamma, which calls Delta, which performs the Activity-End method, all four activities are ended.

Parameters

This method has no parameters.

Results

All activities on the current execution stack are ended. You cannot check for status messages.

- Exit-Activity method

©2025 Pegasystems Inc

Activity-List-Add method

Three activity lists are maintained for each requestor, stored in Java constructs. The lists are:

- Current list — highest priority
- Activity list — intermediate priority
- Workflow list — lowest priority

Use this method to add an activity to one of the three lists, as an alternative to a Call. If the activity requires parameters, enter literal parameters in the method's parameter fields.

This method does not check the validity of the activity you add. Make sure that the activity exists and that users have the necessary security to execute it.

Parameters

The Activity-List-Add method has two parameters plus an optional array of parameter values for an activity.

Parameter	Description
Activity	Use SmartPrompt to select an activity.
Onlist	Optional. Generally, leave this blank. If blank, the system adds the activity to the end of the Activity list. You can enter `Current` or `Workflow`, within double quotes, to instruct the system to add the activity to the end of the Current List or the Workflow List, rather than to the Activity List.

©2025 Pegasystems Inc

Parameter	Description
ActivityPrmName	Optional. Enter a literal as a parameter name for the activity.
ActivityPrmValue	Optional. Enter the value for the parameter.

Results

This method adds the activity to the bottom of the selected list.

When activities are in the Current list, they are executed in order before returning control to the user.

When the Current list becomes empty, activities in the Activity list move to the Current list. When the Activity list is empty, activities in the Workflow list move to the Activity list.

> **NOTE:** You cannot examine these lists with the Tracer or Clipboard tool.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Setting the status of an activity using the Activity-Set-Status method

Use the Activity-Set-Status method to set the execution outcome of an activity. This method sets the status (*pxMethodStatus* property) of the activity to the severity that you specify.

> **NOTE:** Many other methods also set the *pxMethodStatus* property value. If such a method runs in a step after the Activity-Set-Status method, the value that you set in this method could be overwritten.

©2025 Pegasystems Inc

1. On the **Steps** tab of the activity, in the **Method** field, press the Down arrow key and then select **Activity-Set-Status**.
2. Expand the **Method** list.
3. In the **Severity** field, enter one of the following return statuses:
   - `Good`
   - `Info`
   - `Warn`
   - `Fail`
4. In the **Message** field, specify a message that you want to associate with this status by doing one of the following:
   - Enter a literal constant.
   - Press the Down arrow key and select a message rule that contains the message.
   - Click the **Open** icon to create a new message rule.
5. Save the rule form.

- Activity-Clear-Status method

Apply-DataTransform method

Use the Apply-DataTransform method to invoke a data transform for updating an existing clipboard page or pages. (In contrast, use the Page-New method to create a new page and invoke a data transform in a single step.)

The calling activity can share its parameter page with the data transform (by selecting the **Pass the current parameter page** check box).

Parameters

This method accepts two parameters:

Parameter	Description
DataTransform	Specify the data transform to be applied.

©2025 Pegasystems Inc

Parameter	Description
	To locate the data transform with rule resolution at runtime, the system uses the class of the activity's step page (specified in the row on the Activity rule form with the Apply-DataTransform method) as the initial **Applies To** key part, and searches the class hierarchy for a data transform with the name as specified in this **DataTransform** field. If the **Step Page** field is blank for this line on the Activity rule form, the system uses the **Applies To** key part of the activity as the initial **Applies To** key part to locate the data transform. Optionally, you can identify the **Applies To** key part of the data transform explicitly, using the format `class.data_transform_rule`; for example: Apply-DataTransform Data-Admin-Operator-ID.pyDefault The explicit class format causes rule resolution to bypass all class hierarchy searches — the data transform must be present in the specific class identified before the dot. (Other aspects of rule resolution operate normally.) Because use of explicit class names can introduce inflexibility, complicate debugging, and reduce

©2025 Pegasystems Inc

Parameter	Description
	opportunities for rule reuse, avoid this format whenever possible.
Pass the current parameter page	Select to have the values from the activity's parameter page be available to (shared with) the data transform, and vice versa.

Results

The system finds the data transform rule to invoke using the **Applies To** class of the step page (or the **Applies To** class of the activity, if a step page is not specified on the line for the Apply-DataTransform method in the activity) and the value in the **DataTransform** field.

The system invokes the actions in the data transform, evaluating rows of the rule in sequence. If the **Pass the current parameter page** is checked, values in that page can be altered by the data transform. If the data transform has the **Call superclass data transform** check box selected, data transforms from higher classes are applied first.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Data Transforms
- Page-New method

Apply-Parse-Delimited method

Use this method to locate and execute a Parse Delimited Rule (*Rule-Parse-Delimited* Rule type). Parse Delimited Rules parse character-delimited data into multiple property values on the clipboard.

©2025 Pegasystems Inc

Parameters

This method has three required parameters:

Parameter	Description
NameSpace	Enter the second key part of a Parse Delimited Rule.
RecordType	Enter the third key part of a Parse Delimited Rule.
SourceProperty	Identify a property that contains as its value a text string to parse, as input to this parse Rule: • If the property is on the step page, precede the property name with a period. • If this property is on a different page, include the page name in the reference; ensure that this page appears on the **Pages & Classes** tab of the Activity form.

Results

Parse Delimited Rules are identified by a three key parts: the **Applies To** class, a **Namespace** value, and a **Record Type**.

To execute this method, the system:

1. First forms a key from the class of the step page combined with the **Namespace** and **RecordType** parameters to this method. If the **Step Page** field for this step on

©2025 Pegasystems Inc

the Activity form is blank, it uses the class that is the **Applies To** key part of the activity.
2. Next, Rule resolution retrieves a Parse Delimited Rule.
3. Finally, the system executes the Parse Delimited Rule in the current context. The value of the property referenced in the **SourceProperty** parameter is parsed into other property values, as defined by the Parse Delimited Rule.

Checking the method status

This method updates the *pxMethodStatus* property. For more information, see How to test method results using a transition.

Apply-Parse-Structured method

Use this method to locate and execute a Parse Structured rule (*Rule-Parse-Structured* rule type). Parse Structured rules parse structured text string into property values added to the clipboard. The source of the input data depends on which rule type calls this method:

- In an activity (*Rule-Obj-Activity* rule type), the system takes the source data from the property specified in the **SourceProperty** parameter.
- In a Parse Structured rule (*Rule-Parse-Structured* rule type), you can chain two or more rules together by leaving the **SourceProperty** parameter blank. Processing by the second (or later) Parse Structured rule continues from the current position within the input data.

Parameters

This method has two parameters:

- *RecordType* - Enter a literal value that is the second key part of a Parse Structured rule.

- *SourceProperty* - Optional; required for activities. Identify a property reference that contains as its value the input to the Structured Parse rule.

©2025 Pegasystems Inc

If the property is on the step page, precede the property name with a period. If this property is on a different page, include the page name in the reference and ensure that this page appears on the **Pages & Classes** tab of the Activity form or Parse Structured form.

Results

Parse Structured rules have two key parts: an **Applies To** class and a **Record Type**.

First, the system assembles a key for the desired Parse Structured rule. It uses the class of the step page as the **Applies To** class of the key, and the value of the **RecordType** parameter as the second key part. (If the **Step Page** field is blank, it uses the class that is the **Applies To** key part of the activity or Parse Structured rule.)

Next, the system uses rule resolution to locate the Parse Structured rule.

Finally, it applies the Parse Structured rule in the current context:

- If the **SourceProperty** parameter is not blank, the value of the property is used as input data.
- If this method is called from Parse Structured rule and the **SourceProperty** parameter is blank, parsing of the input file continues using the rule identified through the first parameter and the step page.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Using parent activity's primary page

Usually when an activity calls another activity, the primary page and default step page of the second activity would be the step page of first activity. *Rule-Parse-Structured* rules can be considered special activities, but with slightly different behavior:

PEGA ©2025 Pegasystems Inc

If an activity calls *Rule-Parse-Structured* type of rules using "Apply-Parse-Structured" method, the primary page of the Rule-Parse-Structured rule would be null and will eventually refer to parent's primary page - that is, the primary page of the activity, and the default step page of Rule-Parse-Structured rule would be that of the step page mentioned in the "Apply-Parse-Structured" step method.

- Parse Structured rules

Apply-Parse-XML method

Use this method in activities to parse an XML document contained in the value of a property and add them to the step page. This method locates and executes a Parse XML rule (*Rule-Parse-XML* rule type) and parses the XML document text into values for multiple other properties.

Parameters

The system uses the class of the step page as the **Applies To** key part of the desired Parse XML rule. (It determines this class from the **Pages & Classes** information on the Activity form.)

This method has three required parameters:

Parameter	Description
NameSpace	Enter a value that is the second key part of a Parse XML rule.
ElementName	Enter a value that is the third key part of a Parse XML rule. **NOTE:** If left blank, the parsed results will not be formatted correctly.

©2025 Pegasystems Inc

Parameter	Description
SourceProperty	Identify a property reference that contains a well formed XML document as its value: • If the property is on the step page, precede the property name with a period. • If this property is on a different page, include the page name in the reference and ensure that this page appears on the **Pages & Classes** tab of the Activity form.

Results

The system determines the class of the page identified in the **Step Page** column using the **Pages & Classes** information for the activity (If the **Step Page** field is blank, it uses the class that is the **Applies To** key part of the activity, the class of the primary page). The system assembles the key of a Parse XML rule from that class, the **NameSpace** parameter, and the **ElementName** parameter.

Finally, it executes the Parse XML rule in the current context. This parses the input XML document (the value of the property referenced in the **SourceProperty** parameter) into other property values and adds these to the step page.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Assert-No-Invocation method

Reserved. Do not use this method.

©2025 Pegasystems Inc

Assign-EstablishContext method

Do not use this deprecated method.

Call-Async-Activity method

Use the Call-Async-Activity method to load data pages asynchronously (into background threads) to allow other processing to continue.

> **NOTE:** To run an activity asynchronously, set **Usage** on the activity's Security tab to "Asynchronous".

Only the following step methods are available:

- Load-DataPage
- Connect-Wait
- Call-Async-Activity (step instruction to allow the user to load an asynchronous activity)

Parameters

Parameter	Description
Activity	Name of the activity to execute asynchronously. Select an activity whose Usage on the Security tab is set to "Asynchronous".
pool ID	This is a token used to group multiple data pages that are loaded in the background and used by the Connect-Wait method to wait for the data pages with the same pool ID to finish loading.

©2025 Pegasystems Inc

Results

The system uses rule resolution to locate the correct instances of the data pages.

> **NOTE:** This method supports asynchronous loading of data pages.

Checking the method status

This method updates the *pxMethodStatus* property only if it fails. See How to test method results using a transition.

Call-Automation method

Call automations from an activity without using a Java step.

Parameters

This method has the following parameters:

Parameter	Description
Resource	Select the type of automation that you want to call.
Automation	Select the automation that you want to call. The options listed are based on the resource that you selected. Click the Gear icon to view the automation record.
Inputs	Lists all of the inputs for the selected automation. Map the fields that you want to use. Hover over the gear icon to display information about the input. Click the Gear icon to create a new property.

©2025 Pegasystems Inc

Parameter	Description
Outputs	Lists all of the outputs for the selected automation. Map the fields that you want to use. Hover over the gear icon to display information about the output. Click the Gear icon to create a new property.
Errors	Displays a read-only list of the errors that can occur when running this automation.

Results

The system invokes the automation using the invokeAutomation API.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Automations
- Calling an Automation from an activity

Call-Function

Use the Call-Function method to call a Rule-Utility-Function that returns a value for a property or parameter. Unlike calling functions by using the Property-set method, functions that are called by using the Call-Function method can also return a value of void.

This Call-Function method uses an array of parameters. Each row of the array contains an expression (the source) and the property or parameter (the destination) in which the return value is stored.

PEGA ©2025 Pegasystems Inc

The system evaluates each expression that is identified in the **Function** column of the parameter array and assigns the result to the property or property part that you identified in the corresponding **Results** column.

Parameters

- **Function**: Expression that computes the return value. You can:
  - Enter an expression in the Expression Builder.
  - Click the **Down arrow** icon and select a function alias. Your list of referenced rules is updated when the underlying function has parameters mapped to rule types.
- **Return Value**: Optional. Name of the property or parameter in which to store the return value.

> **NOTE:** Enter one or more target property references. If you precede the property with a period but no page name, the system expects this property to be on the step page. If the Step Page field for this step is blank, the system uses the primary page of the activity.

For example:

```
.myProperty
Primary.myProperty
MyPage.myProperty
```

For Value List, Value Group, Page List, and Page Group properties, enter a literal, property reference, or symbolic index value (subscripts):

```
myList(15)
myChild(.Subscript)
myGroup("Massachusetts")
```

©2025 Pegasystems Inc

```
myGroup(<APPEND>)
.myGroup(<LAST>)
.myList(><INSERT>2)
.myList(<PREPEND>)
```

Do use this method to update a property that is the target property of a Declare Expression rule. You cannot save the Activity form if such a conflict is detected.

Results

If the step page does not exist, the system creates a page of the class that is identified in the Pages & Classes tab.

Any property messages previously associated with the property are cleared.

If the type of the expression does not match the type of the destination, the value is converted following the same scheme as within expressions.

Each property value is validated against the property rule.

Rows are processed in sequence, so an expression that is set in one row (as a destination) might be a result in a later row.

NOTE:

- The standard function PropertyExists() can check whether a property is already on the clipboard.
- You cannot use this method to update a property that is also the target property of a Declare Expression rule.
- Do not attempt to return the value of a property that has a name starting with pz; changes to such properties are reserved.
- Use the standard activity Work-.UpdateStatus — not this method — to return the value of a work item status (pyStatusWork property).

©2025 Pegasystems Inc

Checking the method status

This method updates the pxMethodStatus property.

- Expressions
- Constants in Expressions

End-Validate method

This method is used only for custom rule types, to end the processing that develops rule references. Use this method as the last step in an activity of Activity Type `Validate` when the class of the activity is a *Rule-* class and the Start-Validate method was used as the first step.

Parameters

None.

Results

The system performs processing to develop a list of other rules that reference this rule.

- Start-Validate method

Exit-Activity method

The Exit-Activity method ends the current activity and returns control to the calling activity.

In rare cases, write your activity to call the Activity-End method (which ends the current activity and all calling activities too rather than Exit-Activity method).

Parameters

This method uses no parameters and returns no results.

©2025 Pegasystems Inc

Results

If no calling activity exists, the system starts the next activity in the activity list.

- Activity-End method

Flow-End method

Use this method to immediately end a flow execution, before execution reaches a Flow-End shape.

This method is needed only in unusual situations. Design flows so that flow processing ends at a Flow-End shape.

Parameters

This method has no parameters.

Results

The flow rule is no longer associated with the primary page of the activity.

This method does not perform any database updates. As appropriate, design your activity to clean up assignments or other objects created by the incomplete flow execution.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Flow-New method

This instruction spins off a new flow for a case.

It provides the same functionality as the SpinOff shape. If you need to specify a flow using a parameter or property reference, use the activity *StartFlow*.

PEGA ©2025 Pegasystems Inc

Parameters

Type `Flow-New` in the **Method** field, followed by one space and a flow name (the second key part of a *Rule-Obj-Flow* record) as a literal constant. If the flow takes parameters, you can display them by expanding the step.

Results

A flow is spun off for the case.

Checking the method status

This instruction updates the processing status so that you can use the standard when conditions *StepStatusFail* or *StepStatusGood* on the transition. See How to test method results using a transition.

History-Add method

History classes are created automatically in many cases:

- Work objects — Instances of concrete classes derived from History-Work- class are created when a work item advanced through a flow, typically through the *Work-.AddHistory* activity. Others are added automatically. History are visible on the work history report, typically accessed from the History tab on the user form. See the Pega Community article *How to control history instances written to the audit trail.*
- Rules — Instances of the concrete *History-Rule* class are created when you create a rule (with Save or Save As after New) when you save a rule into a ruleset not requiring check-out, and when you check in a rule into a ruleset that requires check-out. These are visible from the **History** tab of the rule form. The history instance for a checked-in rule contains the entire previous contents of the rule before update.
- Data — Instances of concrete classes derived from the *History-Data-* class are created when a form (such as the Operator ID form) is saved or when a Data Table row is saved.

©2025 Pegasystems Inc

> **NOTE:** Data tables are now superseded by local data storage.

In other situations where history is needed, your application can include an activity that calls the History-Add method. Use this method to cause the system to record an audit trail of the processing of a work item by users (or other requestors). Each use of the method appends a new instance to similar instances belonging to a concrete class derived from the *History-* base class. Typically, these record important events and changes in the course of the object.

History details are "append only." The system saves the new instance to the class *History-ClassGroup* of the underlying work item. (For rules, the system saves the new instance to the fixed class *History-Rule*.)

History processing is automatic for work items, rules, and most data objects.

Work item history

Use these built-in facilities to add to the history of a work item:

- When updating a flow rule enter text into the **Audit Note** field in the shape properties panel. The **Audit Note** field is available for several flow shapes, including connector flow actions (arrows), Decision shapes, Subprocess shapes, and End shapes. During flow execution, the Pega Platform records the date, time, current operator, and text as the shape is processed. Your **Audit Note** text can include reference JSP tags, to record the value of specific properties.
- If your application needs to record history instances at points in the flow beyond those in existing, add into the flow a Utility shape that references the standard activity *Work-.AddHistory*.
- Your application activities can call the *Work-.AddHistory* activity as often as needed to extend the information kept as history.

Work item history instances are added to the concrete class named History-Zzzzz, where Zzzzz is the name of the class corresponding to a class group (work pool).

©2025 Pegasystems Inc

A temporary work item has no history. If an activity calls the History-Add method for a temporary work item, the method returns a `Good` status but nothing else occurs.

Disabling or limiting work item history

Optionally, your application can write no work item history records, or only a subset of the default history records, based on the values of an application setting rule and a decision tree.

History for rules and data instances

When you save a rule or data form, the system creates a history instance. In addition, the system creates a history instance to record rule check-in.

You can modify the text that is added for such events.

Other cases

Your application can record history for an object in additional situations. In such situations, use the History-Add method. History instances promote accountability and save supporting detail. They are not used for database rollback or transaction back out.

Parameters

This method has five parameters:

Parameter	Description
ForOperatorID	Enter an Operator ID value (literal or property reference) to be recorded in history. Typically, enter *pxRequestor.pyUserIdentifier* to record the current user.
HistoryMemo	Optional. Enter a property or literal string that becomes memo text. Leave this blank if

©2025 Pegasystems Inc

Parameter	Description
	the memo text will be provided by a field value rule.
RuleAction	Optional. Enter one or two words characterizing the action that you want to record in history, such as "Save" or "Copy". To capture instance history for Rule-, Data-, and Link- instances, set RuleAction to one of the following values: • Save • Delete • Version Restore
Category	Optional. Identify a localization category to be used when the system displays this history memo and the output will be localized based on field value rules in a language-specific ruleset. The default is *pyHistoryMemo*, corresponding to the property *Work-.pyHistoryMemo*. which applies to work item history Leave blank if no category is appropriate, if you prefer alternate text to that produced by the standard field value rules, or when this method adds to the history of an object that is not a work item.
MessageKey	Optional. The meaning of this field depends on the value in the previous field.

©2025 Pegasystems Inc

Parameter	Description
	• `pyHistoryMemo` — If you choose `pyHistoryMemo` in the **HistoryMemo** field, select the third key part of a field value rule which has `Work-` as the first key part and *pyHistoryMemo* as the second key part. As a foundation, your system includes a few dozen standard field values supporting common work item processing. For example, the standard field value rule with key *Work-.pyHistoryMemo.FileAttached* produces on the history display (in English) the text: `A file has been attached {1}.` where {1} is replaced by the file name. This approach supports localization of the work item history display. • Another field value — If you selected an application-specific value in the **HistoryMemo** field, enter the third key part of a field value rule for which `pyHistoryMemo` is the second key part. The first key part is a class related to the current primary page. • blank — If you left the **HistoryMemo** field blank, enter the key of a *Rule-Message* rule. If the text of that message rule contains parameters, follow the

©2025 Pegasystems Inc

Parameter	Description
	key with " \t" and the value to be substituted for parameter {1}, then "\t" and the value to be substituted for parameter {2}, and so on.

Results

The system creates and saves a history instance in the concrete class derived from the *History-* base class that is associated with the class of the object.

For rules and work items, history is maintained automatically. For example, when you save a rule form, an instance is added to the *History-Rule* class. When you update a SimpleTask work item in the PegaSample sample application, an instance is added to the *History-PegaSample-SimpleTask* class. Your application can add to history at other points in the progress of a work item.

The new history instance is saved but not committed. Make sure that your activity or a later activity always performs a Commit method to make this new data instance a permanent part of the database. (This occurs in flow executions automatically when the flow ends, or when the next assignment is created.)

Each history instance contains a date, type, user ID, and memo. If you supply the values in the **HistoryMemo** parameter, the history instance can contain additional properties, such as the "before" values of certain properties. For example, you can create a history instance that indicates:

```
Credit limit increased to $12,000 from $10,500.
```

Similarly, if dates, parties, addresses, or other important facts are modified, the text can record the previous value and the new value. Of course, your processing must retain the old value in a property, parameter, or local variable at least temporarily.

©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property.

See How to test method results using a transition.

History-List method

Use the History-List method to obtain a list of History- instances for a class.

This method is similar to the Obj-List method, which creates embedded pages corresponding to a list of instances of a class.

Parameters

This method has five parameters:

Parameter	Description
ToPage	Identify the name of the destination clipboard page to contain the results of this method. This can be an existing or new page.
ClearToPage	Select if the system is to clear the current contents of the page identified in the **ToPage** parameter before adding history detail.
IncludeLinks	Select this box to cause HTML <A > anchors from the history instances to be included on the destination page.
MaxReturn	Optional. Enter an integer value that is the maximum number of history instances to search and return.
Descending	Select this box to cause the embedded pages to correspond to the history instances with oldest first, corresponding to index value (1).

©2025 Pegasystems Inc

Results

This method retrieves history instances up to the count identified in the **MaxReturn** parameter and creates an embedded page for each instance, sorted in accord with the **Descending** check box.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- History-Add method

Index-Finish method

Do not use this deprecated method. Use Declare Index rules to define indexing.

Link-Objects method

Use this method to create an instance of a concrete class derived from the Link- base class. A link functions as a bidirectional cross reference to associate two objects together.

For example, instances of the standard *Link-Attachment* class associate attachments of various types with work item history instances. The properties in this class identify the work item ID, the attachment's class and key, and the date and time of the history instance. (Attachments themselves belong to a different concrete class, one derived from the *Data-WorkAttach-* class.)

Parameters

This method has several parameters. The last two form an array of property names (not arbitrary property references) and property values.

©2025 Pegasystems Inc

Parameter	Description
LinkToPage	Enter the name of an existing clipboard page that is to be linked to the step page.
LinkClass	Enter the concrete class for the linked objects, a class derived from the *Link-* base class. To create a work item attachment, enter `Link-Attachment`.
LinkDataTransform	Optional. Enter the name of a data transform that applies to the LinkClass class, or a parent class of that class.
LinkMemo	Optional. Enter text that becomes the *pyMemo* value of the link object.
LinkPageProperty	Optional. Enter in this array a property or properties to be set by this method.
LinkPageValue	Optional. Enter a value for each property, a literal, a property reference, or a linked property reference.

Results

1. The system creates a page of the class specified in the LinkClass page.
2. If the LinkData Transform field is not blank, the system finds and applies the data transform to the new page.
3. It adds the memo text, and applies the other properties identified in the array.
4. It adds the properties and values (if any) specified in the array. It adds property values identifying both the object in the LinkToPage page and the object in the step page.
5. The system saves the Link- page (as with the Obj-Save method) but does not commit the page.

©2025 Pegasystems Inc

For instances of the *Link-Attachment* class, the system forms the key from the class name *pxLinkedRefFrom* property value (for example PEGASAMPLE W-153) and the date and time of the save operation, ensuring uniqueness.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Link tables

Using the Load-DataPage method

Use the Load-DataPage method to load a data page in a background thread.

Enter the data page name as a parameter to the step page, then expand the tab to see the parameters of the data page, along with the pool ID. Provide values for the required parameters, and optionally for the non-required parameters.

The values of the parameters of the Data Page could be: literal values, parameter references or property references.

Parameters

Parameter	Description
data page	Enter the name of the data page to load.
pool ID	This is a token used to group multiple data pages that are loaded in the background and used by the Connect-Wait method to wait for the data pages with the same pool ID to finish loading.
data page parameters	The parameters of the named data page appear, with indications showing which are required and which are optional.

©2025 Pegasystems Inc

Results

The system uses rule resolution to locate the correct instance of the data page.

> **NOTE:**
>
> This method enables asynchronous loading of data pages.

Checking the method status

This method updates the *pxMethodStatus* property only if it fails. See How to test method results using a transition.

Log-Message method

Use the Log-Message method to append a message to the Pega log when the step is reached. Depending on method parameters and the current logging level, a line is added to the Pega log on the current node.

Additions to the Pega log consume system resources. Select the **LoggingLevel** parameter carefully. For best performance, remove or convert to comments the steps containing this method when they are no longer needed.

Parameters

The Log-Message method accepts four parameters:

Parameter	Description
Message	Enter the text of the message between double quotes, or an expression that results in a text value. For example:

©2025 Pegasystems Inc

Parameter	Description
	"Test45: Work item "+.pyID+" status= "+ .pyStatusWork The predefined Java variable myStepPage identifies the step page of the method. For example, to include the entire contents of the step page in the log as an XML document, set the **Message** parameter to: "The step page:" + @getStepPageHandle(tools) + "contains the following properties: \n + @getXMLOfPage(myStepPage) You can include a linked property reference in the expression.
LoggingLevel	Optional. Enter one of the following values: • `Debug` — Record as DEBUG severity, produces the largest number of messages; least severe. • `Info` — Record as INFO severity. • `Warn` — Record as WARN severity. • `Error` — Record as ERROR severity; most severe. • `InfoForced` — Always record as INFO event.

©2025 Pegasystems Inc

Parameter	Description
	Choose a type; the default value is `INFO`. The initial `prlog4j2.xml` file installed with Pega Platform shows only `Error` severity. Messages appear when their **LoggingLevel** parameter equals or is more severe than the internal logging level setting. If you choose `Debug`, `Info`, or `Warn`, use the Set Logging Levels tool if needed to cause these messages to appear as you debug the activity. If you choose `InfoForced`, a message is added to the Pega log each time the method executes, independent of settings in the `prlog4j2.xml` file or settings through the Set Logging Levels tool. To reduce clutter in the Pega Log and improve performance, be sure to remove such `InfoForced` steps, or convert them to comments, in production systems.
GenerateStackTrace	Select to cause execution of this method to create a Java stack trace in the Pega log, in addition to other results. If the **SendToTracer** check box is selected, the stack trace also appears in the Tracer tool output.
SendToTracer	Select to cause a Tracer event when this method executes, in addition to other results.

©2025 Pegasystems Inc

Results

If the **LoggingLevel** parameter is set to `InfoForced`, a message is added to the Pega log on the current node.

If the **LoggingLevel** parameter is set to `Debug`, `Info`, `Warn`, or `Error`, results of this method depend on a comparison of the **LoggingLevel** parameter with the `prlog4j2.xml` setting on the current node for the *Rule-Obj-Activity* class. If the value of the parameter is equal to or greater than the severity value of the setting, the message is added to the log and other processing occurs as described above. You can temporarily override the setting in the `prlog4j2.xml` file using the Set Logging Level tool. Severity levels range from `DEBUG` (lowest level, most messages) to `INFO`, `WARN`, `ERROR`, and `FATAL` (highest level, fewest messages).

If requested, a Java stack trace is also added to the Pega log. If requested, a Tracer event is generated. These events appear in Tracer output if the Log-Message check box is selected on the Tracer Settings panel.

When debugging is complete, you can delete the steps that use this method or change them to comment steps. This is needed for steps where LoggingLevel is set to `InfoForced`. For others, if the `prlog4j2.xml` setting for the *Rule-Obj-Activity* class is higher than the **LoggingLevel** parameter, executing this method does not update the Pega log and does not affect performance.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> **NOTE:** This method is a wrapper for the Java LogHelper class for logging diagnostic data. When practical, use this method rather than a Java step to add to the Pega log.

©2025 Pegasystems Inc

Map-Structured method

Use the Map-Structured method to invoke a map structured rule (*Rule-Map-Structured* rule type) to assemble a text property value from property values on a clipboard page, or disassemble a single text property value into other property values.

Parameters

This method has three parameters:

Parameter	Description
Direction	Select the direction of processing: • `Map From Clipboard` — Assemble one structured value from multiple clipboard properties. • `Map To Clipboard` — Disassemble a single structured string into multiple clipboard properties.
Map	Identify the second key part (Purpose) of a map structured rule. At runtime, rule resolution uses the class of the step page (or the class of the primary page of the activity) as the Applies To key part to retrieve this rule.
Property	Enter a scalar property reference that is the source of the input data structure (when the **Direction** is `Map From`) or the destination data structure (when the **Direction** is `Map To`).

©2025 Pegasystems Inc

Parameter	Description
	Enter only properties that have a character-oriented type, such as `Text`, `Date`, `DateTime`, `Identifier`, `Time of Day`, or `Decimal`. Do not use `Double`, `TrueFalse`, `Integer`, or `Password` properties. If the **Direction** is `Map From`, you can enter a linked property reference here.

Results

The Map-Structured method works as follows:

1. Locates the map structured rule using rule resolution.
2. Uses information in the **Mapping** tab of the rule to parse the value of the property identified in the **Property** parameter into other clipboard values, or to assemble the value of that property from other clipboard values, depending on the **Direction** parameter.
3. If the direction is `Map From Clipboard`, the value of properties listed on the **Mapping** tab are copied to the character offset position, truncated (if necessary) to the length specified in the **Length** field, padded and justified according to information on the **Mapping** tab.
4. If the direction is `Map To Clipboard`, the mapping is executed in the reverse direction. The value of the property in the **Property** parameter is disassembled into the multiple properties listed on the **Mapping** tab. No padding or justification occurs. If the mapping defines a character length longer than the **Max Length** size of the property, the value is truncated to the **Max Length** size.

Property validation does not occur.

PEGA ©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Map Structured rules

Obj-Browse method

Use the Obj-Browse method to search instances of one class and copy the entire instances, or specified properties, to the clipboard as an array of embedded pages.

Only properties exposed as columns can be used as selection criteria. However, values of properties that are not exposed as columns, including embedded properties, can be returned.

> **NOTE:** Obj-Browse does not work with encrypted properties.

Use the Obj-Browse method optionally followed by the Obj-Filter method to create a list of search results. The Obj-Browse method selects instances based on values of properties exposed as a column. The Obj-Filter method qualifies and filters list results based on all property values, exposed or not. Together these two methods provide greater control over which properties are returned, and better performance, than the Obj-List method.

Parameters

The Obj-Browse method has five base parameters and an optional array of selection criteria.

©2025 Pegasystems Inc

Parameter	Description
PageName	Enter the name of the destination page to contain search results. The system uses *Code-Pega-List* as the class of this page.
ObjClass	Identify a class to search. You can search one concrete class, or all classes in a class group. Within the PegaRULES database, the class can correspond to a database table or to a database view.
MaxRecords	Optional. Enter the maximum number of instances you want returned in the list at run time. If left blank, the default value is 10,000. If the value is 0, up to 50 instances are returned. As a best practice, specify a **MaxRecords** value to reduce the demand for database activity memory. If necessary, your activity can test the property *pxMore* on the results page to see whether additional rows were not returned because the **MaxRecords** limit was reached.
GetRowKey	Select to include the primary key in the set of property values returned. Clear to exclude the primary key. (Selected by default.)
RowKey	This field is required if **GetRowKey**, above, is selected. Otherwise, leave blank unless the class in the **ObjClass** parameter corresponds to an external table.

©2025 Pegasystems Inc

Parameter	Description
	Enter the name of a property in the external table, or an expression involving one or more property names, that specifies the unique key of rows of the external table.
Lightweight List	Select to have the embedded pages for the selected instances use lightweight lists for better performance. Leave unselected if the activity performs operations unsupported by lightweight lists, like forward chaining declarative processing, messaging, or value vetting. Refer to the Pega Community article *About lightweight lists* for more information. As a best practice, select this check box. This can reduce memory requirements during processing. Having the embedded pages use lightweight lists does not affect your ability to remove an embedded page completely (with the Obj-Filter or Page-Remove method). Two dynamic system settings control the use of lightweight lists by the Obj-Browse method. There is a setting each for the Pega Platform and application rules: • clipboard/lightWeightList/enableForCorePRPCRules - this is set to true by default.

©2025 Pegasystems Inc

Parameter	Description
	• clipboard/lightWeightList/enableForCustomerRules - this is set to false by default. When true, these dynamic system settings override the **Use Lightweight List** check box setting on the Activity rule form.
Logic	Optional. Enter a boolean statement that defines how the rows of the parameter array are to be combined. Identify rows by the Label field. You can use the operators AND and OR in this statement, and parentheses for grouping. For example, A AND (B OR C OR (E AND F)). If you leave this blank, the system assumes the AND operation applies to all rows of the array.
	Optional. Complete one or more rows of this array to define the selection criteria and the properties to be returned. If you leave this array blank, all rows (and all columns including the *pzPVStream* column) are selected, up to the MaxRecords limit. For an external class, complete at least one row of the array.
Label	Enter a unique letter or identifier for this row, referenced in the **Logic** field.

©2025 Pegasystems Inc

Parameter	Description
Select	Select to return the value of this property on each embedded page. In the resulting SQL statement at run time, this property is listed in the SELECT list (as well as the WHERE condition).
Field	Enter a property reference. You can specify a `Single Value` property, a list or group property, or individual elements of a list or group property. You can return any property — exposed as a column or not — in the results by setting the **Condition** field to `Value Only`. However, only `Single Value` properties exposed as database columns can be used as selection criteria. Click the **Open** icon to open the property. Do not use the **Gear** icon. To restrict the search to specific values of a property, enter a property reference that is exposed as a database column. You can review the database schema to determine which properties are exposed for a class. Consult your database administrator for information on the current database schema, or use the Modify Schema facility. See Property optimization.

©2025 Pegasystems Inc

Parameter	Description
Condition	Select a comparison, such as `Is Equal` or `Is Less Than`. • Select `Value Only` if the search results are to include the property value, but the property value is not to be used as a selection criterion. • Select `Is Null` to require that the field a have null value. • Select `Is Not Null` to require that the field have a value. If the property is not exposed as a database column, select `Value Only`.
Value	Enter a constant value, a property reference or other expression for the comparison. Leave blank if the **Condition** is `Value Only`, `Is Null`, `Is Not Null`, `Is True` or `Is False`. Click the **Open** icon to open a property. Click the **Open expression builder** icon to start the Expression Builder. • If you enter an expression involving function rules, Java, and properties on various clipboard pages, then at run time the expression is evaluated only once. • If you enter a property (that is exposed as a column) that applies to the

©2025 Pegasystems Inc

Parameter	Description
	ObjClass class, the Value value is evaluated for each row. For example, if the **Field** field contains a `DateTime` property ExpireDateTime, the **Condition** is `IS GREATER THAN`, and the **Value** field contains a function computing the current date and time, the function is evaluated only once, not once per row. **NOTE:** The second value in the comparison refers to a property on the current clipboard page, or a hard-coded value, not to a value in the database record.
Sort	Select `No Sorting`, `Ascending`, or `Descending` to control the sorting of results by this field. (Sorting occurs through database software, on the database server.) If the **Condition** is `Value Only` or `Is Null`, select `No Sorting`.

Results

At run time, the system converts the parameters to an SQL query statement, creating a SELECT WHERE and ORDER BY query based on the parameters, and sends the query to the PegaRULES database.

©2025 Pegasystems Inc

This method operates on the database response to create a results page containing one embedded page for each instance retrieved.

If the **Field** parameters identify only `Single Value` properties exposed as columns, each embedded page in the results contains only those properties. However, if one or more of the Field parameters identifies a property that is not an exposed column, the entire *pzPVStream* column, also known as the Storage Stream or Blob column, is also returned.

The Obj-Browse method returns the data either to a previously created step page — which is cleared and reused — or to a new page of that this method creates of class *Code-Pega-List*. The system writes search results in the specified step page, along with information that controlled the search.

> **NOTE:** Execution of this method does not trigger declarative rule processing that depends on change tracking. Even when a property retrieved by the Obj-Browse method is involved in a Declare Expression, Declare Constraint or other declarative rule, retrieval does not cause the declarative rule to re-evaluate.

Properties of Code-Pega-List

This table identifies some of the main properties of the *Code-Pega-List* class, which is output from various methods. Properties marked with a yellow diamond can hold input parameters for the search. Other properties specified as read-only contain outputs from the search.

©2025 Pegasystems Inc

Property	Purpose
pyCacheFile	For list view reports that save results as an XML file in the lookup list cache, identifies the file name.
pyCondition	A `Page List.` Identifies the parameters that define search comparisons, for the Obj-List method: • *pyConditionFieldName* — A property reference resulting in a single text value • *pyConditionOperation* — A comparison operator, such as "<" • *pyConditionValue1* — An expression for the comparison value
pyDetailsAction	For list view and summary view rules, identifies the value of the `pyAction` parameter to the *ShowView* activity, such as `PREPARE` or `SHOW`.
pyDetailsWindow	For list view and summary view rules, identifies the name of the window that is to hold the resulting display.
pxElapsedTime	Elapsed time in seconds to process the RDB-List operation. Read-only.

©2025 Pegasystems Inc

Property	Purpose
pxMore	Indicates whether there are more instances not returned. Values are true or false. Read-only.
pyMaxRecords	Optional input to limit the number of rows returned.
pxNextKey	Indicates the key of the first row (instance) not returned, if *pxMore* is true. Read-only.
pyObjClass	Identifies the class containing the instances that are to be searched. This property is different from *pxObjClass*, the class of this page, which always has the value *Code-Pega-List*.
pyPageSize	For list view or summary view rules that support report paging, the maximum number of results to presented on a page.
pyQueryTimeStamp	A `DateTime` value recording the start of the operation. Read-only.
pxResults	A Page List containing pages with individual search results. each of the class defined in the *pxObjClass* property. (Not used for Obj-

©2025 Pegasystems Inc

Property	Purpose
	List methods that specify the lightweight results format.) Read-only.
pxResultCount	Number of rows of data returned by the search. Read-only.
pyReturnLightweightResults	For the Obj-List method, indicates that the results are in a special format rather than in *pxResults* pages. Read-only.
pxSelectStatement	For the Obj-List method, the SQL statement sent to the database before substitution. Read-only.
pxSQLStatementPost	For the RDB-List method, SQL statements as sent to the SQL database. Appears only when requested through the `{SQLPage}` syntax. Read-only.
pxSQLStatementPre	For the RDB-List method, SQL statements before substitution. Appears only when requested through the `{SQLPage}` syntax. Read-only.
pxTimeElapsed	For the Obj-List method, elapsed seconds taken for the method. Read-only.

©2025 Pegasystems Inc

Property	Purpose
pxTotalResultCount	For the RDB-List method, the total number of results returned by the query. This is set only if *pyReturnTotalResultCount* is true. (If *pyReturnTotalResultCount* is false, then this is zero.) Read-only.
pyUseAlternateDb	Set to true if an alternate database, not the primary PegaRULES database, was the source of data for a list view or summary view report.

Performance

This method can return thousands of database rows and create large clipboard pages. Use care to retrieve only the rows (instances) and columns (properties) you truly need. Similarly, specify sorting only when needed.

> **NOTE:** Execution of this method may cause an alert to be added to the Alert log, of type `PEGA004` - `PEGA007`, `PEGA0025`, `PEGA0027`, or `PEGA0028`. Review the Alert log to understand the frequency and sources of such alerts, and for best performance, alter your application or database to eliminate those alerts that occur frequently. For example, if execution of an Obj-Browse method accesses properties within the Storage Stream column, a `PEGA0025` alert occurs. The alert indicates that exposing one or more columns in the PegaRULES database table that holds instances of the class is likely to improve performance.

©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Pega Community Article

See the Pega Community article *Comparing the Obj-List-View, Obj-Browse and Obj-List methods*.

- Obj-Filter method
- RDB-List method

Obj-Delete method

Use the Obj-Delete method to delete a database instance corresponding to a clipboard page and optionally to delete the clipboard page too. You can cause the deletion to occur immediately, or (more commonly) defer deletion until a later execution of a Commit method.

This method can operate on objects of both internal classes (corresponding to rows in a table in the PegaRULES database) and external classes (corresponding to rows in an external relational database).

The Obj-Delete method uses the class of the page to obtain the appropriate *Rule-Obj-Class* instance. It uses the table name, key fields, and other aspects of the class rule to mark the instance for deletion. Unless you check the Immediate check box on the activity form, the database deletion does not occur until the system next processes the Commit method.

You can reverse or cancel a previously executed Obj-Delete method by using the Obj-Save-Cancel method (like the method Obj-Save can be reversed), unless the **Immediate** box is selected or the Commit method has already occurred.

©2025 Pegasystems Inc

If you use the Obj-Delete method in an activity with Activity Type of `Trigger`, do not check the **Immediate** check box.

Do not use Obj-Delete method on instances in a class derived from the Index- base class. Use only Declare Index rules to delete these instances.

Use the Obj-Delete-By-Handle method, not the Obj-Delete method, when the object is not open on your clipboard and the object handle (*pzInsKey* value) is known. The Obj-Delete-By-Handle method combines the Obj-Open-By-Handle method and the Obj-Delete method in a single step, making your activity shorter.

Parameters

This method has two parameters:

Parameter	Description
Remove	If selected, the system deletes the page identified in the **Step Page** column of this step from your clipboard and marks the corresponding persistent instance for deletion.
Immediate	If selected, the system deletes the database instance immediately (commits this deletion). To delete the database instance immediately from the database, select this box. If this box is cleared, the database instance is deleted by the next Commit operation.

©2025 Pegasystems Inc

Results

The system finds the clipboard page identified in the **Step Page** column of the current step and checks whether it the current requestor holds a lock on the corresponding object.

If the object is not locked (and the class rule allows locking), the method fails and no clipboard or database changes occur.

If the page is locked and the **Remove** parameter is selected, the system deletes the page. It marks the persistent object in the database for deletion, or deletes it immediately, depending on the **Immediate** parameter.

The system automatically performs index processing for an immediate delete, or later when a Commit method follows the Obj-Delete method.

Fail and Warn conditions

These situations cause the method status to be `Fail` or `Warn`.

You cannot delete an instance without locking it first (usually with the Obj-Open method). You cannot delete an instance while it is locked by another requestor.

Troubleshooting deferred delete operations

As a debugging aid, the standard HTML rule *@baseclass.DeferredOps* interrogates system internals and presents an ordered list of non-immediate (uncommitted) database operations, including Obj-Delete operations. Your activity can display this HTML rule using the Show-Page method.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

©2025 Pegasystems Inc

When a deferred — that is, non-immediate — Obj-Delete method fails, the next Commit method will also fail unless one of the following is performed successfully before the Commit method is attempted:

- Another deferred Obj-Delete (or Obj-Save) method is performed on the instance
- An Obj-Save-Cancel is performed
- A Rollback method is performed

- Commit method
- Obj-Delete-By-Handle method
- Obj-Save-Cancel method
- Rollback method

Obj-Delete-By-Handle method

Use the Obj-Delete-by-Handle method to delete an instance identified by its handle. This method executes the operations of the Obj-Open-By-Handle method followed (if successful) the Obj-Delete method.

Unlike the Obj-Delete method, the Obj-Delete-By-Handle method does not require that the instance be open on the clipboard. Using Obj-Delete-By-Handle, you can delete it without first opening the object.

Parameters

This method has five parameters:

Parameter	Description
InstanceHandle	Enter the handle that identifies the instance to be opened as a literal or a property reference. This corresponds to the value of the *pzInsKey* property for an internal class, and a simulated handle for an external class.

©2025 Pegasystems Inc

Parameter	Description
	Because the handle is lengthy and not easily displayed, it is often easier to identify it by a property reference, rather than attempt to enter the handle by typing.
Lock	If the class allows locking, a lock will be required to delete the object. Select this option to obtain the lock when executing the Obj-Delete-By-Handle method. If the instance is already locked at the time Obj-Delete-By-Handle is called, you can leave this option cleared. If selected, but the class does not allow locking, the Obj-Delete-By-Handle method deletes the instance but produces a warning: `Database-LockFailure-NotSupported`. As a best practice, select this box if the object belongs to a class that allows locking.
Release on Commit	Select this box to cause a lock held by your requestor to be released when your changes are completed through the Commit method. In most cases, select this box if you selected the **Lock** box. In the unusual case that you want to retain the lock after Commit (perhaps because additional changes are expected), your activity can later use the Page-Unlock method to release the lock.

©2025 Pegasystems Inc

Parameter	Description
LockInfoPage	Optional. As a debugging aid, identify the name of a page to be created by this method to hold the results of the locking attempt when it fails. The page, of class *System-Locks*, contains the following properties: • *pxOwnerID* — requestor who holds the lock. • *pxExpireDateTime* — date and time the lock will go soft. • *pxUpdateOperator* — *pyUserIdentifier* value for the requestor who holds the lock. • *pxLockHandle* — text string that defines the lock. If the Obj-Delete-By-Handle fails to acquire a lock, this page is created and can be viewed with the Clipboard tool or the Tracer. By convention, this page is named `LockInfo` in standard rules. When a lock failure occurs during a flow execution, the flow can display the standard harness *Work-.LockInfo* (or a similar harness incorporating the standard section *System-Locks.LockInfo*) to report the situation. If the lock not acquired is held by the same user in a separate requestor session, the user can click the **Release Lock** button to relinquish it.

©2025 Pegasystems Inc

Parameter	Description
Immediate	To delete the database instance immediately from the database, select this box. If this box is cleared, the database instance is deleted by the next Commit operation.

Results

If the **Lock** check box is selected, the system attempts to acquire a lock on the object. This attempt does not require the object to be opened. If **Lock** is selected, the class supports locking, but the lock cannot be acquired, the method fails.

Next, if **Immediate** is selected, the method deletes the instance from the clipboard and from the PegaRULES database. If **Immediate** is not selected, the deletion operation is queued until the next Commit occurs.

Fail and Warn conditions

These situations cause the method status to be `Fail` or Warn:

- **Lock** is selected, the class rule enables locking, but a lock cannot be acquired because another requestor session holds the lock (`Fail - Database LockFailure - LockHeldByAnother` .)
- The **InstanceHandle** parameter identifies an instance that does not exist (Fail).
- **Lock** is selected, but the class rule does not allow locking (Warn). The object is deleted.

Use the Obj-Delete-By-Handle method only if you can determine the unique handle that permanently identifies which instance to delete. The handle of an instance is a unique key, in an internal format, assembled by the system that identifies an instance in the PegaRULES database. It may differ from the visible key generally used to identify instances. (Every instance has a unique handle, but more than one instance may share

©2025 Pegasystems Inc

the same visible key.) For rule instances, the handle includes system-generated identifying values (including the create date and time) that make it unique.

This method does not access or depend on any information in the **Step Page** field of the step. To avoid a warning message, leave the **Step Page** field blank.

You can reverse or cancel a previously executed Obj-Delete-By-Handle method by using the Obj-Save-Cancel method, unless the **Immediate** box is selected or the Commit method has already occurred.

If you use the Obj-Delete-By-Handle method in an activity with Activity Type of `Trigger`, do not check the **Immediate** check box.

> ⚠ **CAUTION:** Do not use Obj-Delete-By-Handle method on instances in a class derived from the *Index-* base class. Use only Declare Index rules to delete these instances.

This method cannot be used in a step that involves iteration.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

When a deferred — that is, non-immediate — Obj-Delete-By-Handle method fails, the next Commit method will also fail unless one of the following is performed successfully before the Commit method is attempted:

- Another deferred Obj-Delete-By-Handle (or Obj-Save) method is performed on the instance
- An Obj-Save-Cancel is performed
- A Rollback method is performed

- Commit method

©2025 Pegasystems Inc

- Obj-Delete method
- Obj-Open-By-Handle method

Obj-Filter method

Use the Obj-Filter method to filter the embedded pages of a results page (such as a *Code-Pega-List* page produced by the RDB-List, Obj-List-View or Obj-Browse methods) based on one or more when condition rules.

For example, if each embedded page on results page contains a `Page List` property Child and each Child page contains a `Single Value` property FirstName, you can use the Obj-Filter method to remove all embedded pages where no value of FirstName is "Robert".

> **NOTE:** For best performance, use an Obj-Browse method to search and retrieve instances of a class where the selection criteria involve only `Single Value` properties exposed as columns in the PegaRULES database. Use the Obj-Filter method to refine search results by deleting pages based on when condition tests that involve non-exposed property values, including aggregate properties.

Parameters

This method has two required parameters and an array parameter.

Parameter	Description
ListPage	Name of the results page, of class *Code-Pega-List*.
ResultClass	Select the **Applies To** key part of one or more when condition rules (*Rule-Obj-When*) to be used to test each result. Must match or be an

©2025 Pegasystems Inc

Parameter	Description
	ancestor class of the class of the embedded pages in the ListPage page.
When	Complete at least one row of this array, specifying a when condition rule. Enter the second key part (**When Name**) of a when condition rule. Typically, the comparisons and expressions in the when condition rule involve one or more aggregate properties or unexposed properties. **NOTE:** Because the embedded pages may contain only a portion — not all — of the properties of the listed instances, make sure the when condition rule you identify here involves only the properties that are expected to be present. If your when condition rule mentions a property not present on the embedded page, the system uses the null value in the when rule evaluation, not the true value that is saved in the PegaRULES database.

Results

At run time, the system uses rule resolution to find the when condition rules.

©2025 Pegasystems Inc

Use the Obj-Filter method to filter the embedded pages of a results page (such as a *Code-Pega-List* page produced by the RDB-List, Obj-List-View or Obj-Browse methods). When one or more when rules evaluate to false, the page is removed from the results page.

The embedded pages for which all when condition rules evaluated to true are not affected.

The contents of surviving embedded pages are not affected.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Obj-Browse method
- Obj-List-View method

Obj-List-View method

Use the Obj-List-View method to execute the retrieval and sorting operations, but not the formatting and display processing, of a list view rule.

Obj-List-View processing can return a smaller set of columns and a more precisely selected set of rows than the similar Obj-List method, and so can require less memory and less network traffic than an Obj-List method.

Parameters

Specify all three key parts of a list view rule.

Parameter	Description
ObjClass	Select the **Applies To** key part of a list view rule. This may identify an internal class or class group, or an external class.

©2025 Pegasystems Inc

Parameter	Description
ListView	Select the **Purpose** key part of a list view rule.
Owner	Select the **Owner** key part of a list view rule.

Results

The system uses rule resolution to find the list view rule and executes it, but does not produce any HTML output display. The selection criteria of the list view rule and the sorting requirements (recorded on the **Content** tab) are converted to an SQL query that is sent to the PegaRULES database.

The database results are organized into a results page of class *Code-Pega-List*. The name of this page is determined by the **Content Page Name** field on the **Content** tab of the List View form, (The **Step Page** field on the step is ignored.)

Each embedded pages contains the properties listed on the **Content** tab. If any of these properties is not exposed as a column in the database, the entire Storage Stream (blob) column is accessed. If the **Maximum Value** field on the **Content** tab is not blank, the method returns at most the number of embedded pages specified in the **Maximum Value** field.

Most information in the **Organize**, **Display Field**, and **Format** tabs is not used. However, if report paging is enabled on the **Organize** tab, only rows for a single page are returned.

Properties of Code-Pega-List

This table identifies some of the main properties of the *Code-Pega-List* class, which is output from various methods. Properties marked with a yellow diamond can hold input parameters for the search. Other properties specified as read-only contain outputs from the search.

©2025 Pegasystems Inc

Property	Purpose
pyCacheFile	For list view reports that save results as an XML file in the lookup list cache, identifies the file name.
pyCondition	A `Page List.` Identifies the parameters that define search comparisons, for the Obj-List method: • *pyConditionFieldName* — A property reference resulting in a single text value • *pyConditionOperation* — A comparison operator, such as "<" • *pyConditionValue1* — An expression for the comparison value
pyDetailsAction	For list view and summary view rules, identifies the value of the `pyAction` parameter to the *ShowView* activity, such as `PREPARE` or `SHOW`.
pyDetailsWindow	For list view and summary view rules, identifies the name of the window that is to hold the resulting display.
pxElapsedTime	Elapsed time in seconds to process the RDB-List operation. Read-only.
pxMore	Indicates whether there are more instances not returned. Values are true or false. Read-only.
pyMaxRecords	Optional input to limit the number of rows returned.

©2025 Pegasystems Inc

Property	Purpose
pxNextKey	Indicates the key of the first row (instance) not returned, if *pxMore* is true. Read-only.
pyObjClass	Identifies the class containing the instances that are to be searched. This property is different from *pxObjClass*, the class of this page, which always has the value *Code-Pega-List*.
pyPageSize	For list view or summary view rules that support report paging, the maximum number of results to presented on a page.
pyQueryTimeStamp	A `DateTime` value recording the start of the operation. Read-only.
pxResults	A Page List containing pages with individual search results. each of the class defined in the *pxObjClass* property. (Not used for Obj-List methods that specify the lightweight results format.) Read-only.
pxResultCount	Number of rows of data returned by the search. Read-only.
pyReturnLightweightResults	For the Obj-List method, indicates that the results are in a special format rather than in *pxResults* pages. Read-only.
pxSelectStatement	For the Obj-List method, the SQL statement sent to the database before substitution. Read-only.
pxSQLStatementPost	For the RDB-List method, SQL statements as sent to the SQL database. Appears only when requested through the `{SQLPage}` syntax. Read-only.

©2025 Pegasystems Inc

Property	Purpose
pxSQLStatementPre	For the RDB-List method, SQL statements before substitution. Appears only when requested through the `{SQLPage}` syntax. Read-only.
pxTimeElapsed	For the Obj-List method, elapsed seconds taken for the method. Read-only.
pxTotalResultCount	For the RDB-List method, the total number of results returned by the query. This is set only if *pyReturnTotalResultCount* is true. (If *pyReturnTotalResultCount* is false, then this is zero.) Read-only.
pyUseAlternateDb	Set to true if an alternate database, not the primary PegaRULES database, was the source of data for a list view or summary view report.

Performance

This method can return thousands of database rows and create large clipboard pages. Use care to retrieve only the rows (instances) and columns (properties) you truly need.

Execution of this method may cause an alert to be added to the Alert log, of type `PEGA004` - `PEGA007`, `PEGA0025`, `PEGA0027`. or `PEGA0028`. Review the Alert log to understand the frequency and sources of such alerts, and for best performance, alter your application or database to eliminate those that occur frequently. For example, if execution of an Obj-List-View method accesses properties within the Storage Stream column, a `PEGA0025` alert occurs. The alert indicates that exposing one or more columns in the PegaRULES database table that holds instances of the class is likely to improve performance.

©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> **NOTE:** See the Pega Community article *Comparing the Obj-List-View, Obj-Browse and Obj-List methods*.

- Alerts

Obj-Open method

Use the Obj-Open method to open an instance stored in the PegaRULES database or in an external database linked to an external class, and save it as a clipboard page.

After an invocation of Obj-Save without checking the **Write-now**" option, Pega Platform™ reads the instance data from the current Thread, not from the instance in the database. Following any commit, Obj-Open reads instance data from the database. For more information, see the **Troubleshooting** section of Obj-Save method.

The system uses the specified class and key fields to find and open the object and place its data into the specified step page. The system searches up the class hierarchy as appropriate to find the instance. If it finds the specified step page, the system clears any data that is on it and reuses the page. If no existing page has a name matching the specified step page, the system creates a new page.

Except in rare cases, identify in the **Step Page** field the name of a page to be created by this method. If the Step Page field of the step is blank, this method opens the object into the primary page of the current activity, replacing the current contents of the primary page and changing the class of the primary page if necessary to match the class of the object. Typically, this is not desirable.

©2025 Pegasystems Inc

When rule and Function rule alternatives

In some cases, you may need only to confirm whether or not the object exists, and not need access to the properties that make up the object. You can use either of two alternatives, each is faster (less costly) than using Obj-Open. They do not alter your clipboard.

- When — The standard when rule *@baseclass.@pxDoesObjectExist* checks to see whether the object described by the current step page exists. The class of the step page identifies the class of the object. The step page must contain properties and property values that form the key parts of the object to be tested.
- Function — Similarly, the standard function rule *pxDoesObjectExist()* in the Utilities library determines whether a specific object exists in the database. The *pxDoesObjectExist()* function accepts as a single parameter a clipboard page containing the properties and values for the key parts of the object to be tested, and returns true or false. The class of the page identifies the class of the object.

Parameters

This method has four base parameters plus an array of property names and values.

Parameter	Description
OpenClass	Optional. Identify the concrete class of the instance to be opened. If left blank, the class is obtained from the **Pages & Classes** tab of the activity based on the property references in the parameter array. You can't open an instance of a class that corresponds to a view in the PegaRULES database (rather than a table).

©2025 Pegasystems Inc

Parameter	Description
Lock	Select this box to acquire a lock. Lock the instance if you plan to change and save the instance or delete it. Your session cannot lock the object unless your access role includes the ability to update objects of the class. Locking an object from an external class prevents updates by other requestors in the Pega Platform system. Such locking does not prevent changes to the external database — including changes to a row open and locked in PRPC — by other systems.
ReleaseonCommit	Select this box if you selected the **Lock** box and want the system to release the lock when the next Commit method occurs for this instance. In most cases, select this box if you selected the **Lock** box. In the unusual case that you want to retain the lock after Commit (perhaps because additional changes are expected), your activity can later use the Page-Unlock method to release the lock.
LockInfoPage	Optional. As a debugging aid, identify the name of a page to be created by this method to hold the results of the locking attempt, when it fails. The page, of class *System-Locks*, contains the following properties:

©2025 Pegasystems Inc

Parameter	Description
	• *pxOwnerID* — Requestor who holds the lock • *pxExpireDateTime* — Date and time the lock will go soft • *pxUpdateOperator* — *pyUserIdentifier* value for the requestor who holds the lock • *pxLockHandle* — Text string that defines the lock If the Obj-Open fails to acquire a lock, this page is created and can be viewed with the Clipboard tool or the Tracer. By convention, this page is named `LockInfo` in standard rules. When a lock failure occurs during a flow execution, the flow can display the standard harness *Work-.LockInfo* (or a similar harness incorporating the standard section *System-Locks.LockInfo*) to report the situation. If the lock requested but not acquired is held by the same Operator ID in a separate requestor session, that user can click the **Release Lock** button to release the lock.
PropertyName	Enter in this array the name of each property or properties and the associated values that together make up the key of the instance to be opened. Type a period before the property name. Enter at least one property name.

©2025 Pegasystems Inc

Parameter	Description
	Review the **General** tab of the Class form for the class (or of superclasses of this class) to determine which properties form the key. **NOTE:** When acquiring a lock, include the properties that make up the lock key when they differ from the properties that make up the instance key. (This is rare.) Information about the properties that make up locks is on the **Locking** tab of the Class form (or in some cases on the Class Group form).
PropertyValue	For the value, specify either a literal value, a property reference, or a linked property reference. **NOTE:** When this parameter references a property that uses backward chaining, the *StepStatusFail* condition does not return *True* when the backward chaining fails. To ensure that the object exists, use the *pxDoesObjectExist* function rule. For

©2025 Pegasystems Inc

Parameter	Description
	more information, see When rule and Function rule alternatives

Results

The system computes a *pzInsKey* value by concatenating the property values identified in the **Keys** definition on the **Basic** tab of the class rule. If the **OpenClass** parameter identifies an external class, the system sets the value of the *pxObjClass* property on the clipboard to the class name.

If the step page does not exist, a new page is created. (The *pxObjClass* property, part of every instance, identifies its class.) If the **Step Page** field is blank, the system clears and reuses the primary page of the activity.

It reads the persistent object instance from the database and adds all properties of the object to the step page.

If the **Lock** check box is selected and the requestor has permission to update objects of this class, the system locks the instance.

The step page may be an embedded page, or one page in a list or group. In these cases, the page, if it exists, is replaced completely. For example, if the page is seventh in a `Page List`, or “Customer” in a `Page Group`, your application can still reference the page as "7" or “Customer” after the replacement.

To avoid overwriting an existing page

You may want an activity to check whether a page of a certain name exists before you open a new page.

To test for a page, enter the following precondition to the activity step:

©2025 Pegasystems Inc

```
= @PageExists("myPage", tools)
```

where myPage is the name of the clipboard page.

Fail and Warn conditions

These situations cause the method status to be `Fail` or `Warn`:

- If the another requestor has locked the instance, the system opens the object to the clipboard, but the method returns a `Fail` status.
- If the key supplied does not match the key of an instance in the database, the system creates a page of the indicated class that is empty except for the *pxObjClass* property and the key values, and returns a `Fail` status.
- If you selected the **Lock** parameter (on the **Locking** tab of the **Class** form or in some cases the **Class Group** form) but the class does not allow locking, the object is opened, but the method returns a `Warn` status.
- If you selected the **Lock** parameter but at runtime the requestor's access role does not allow the requestor to update objects of that class, the object is opened but without a lock, and the method returns a `Fail` status.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

The Activity-Clear-Status method may be useful following an Obj-Open method.

Restrictions

You cannot use this method in a step that involves iteration.

- Activity-Clear-Status method
- Page-Unlock method
- Obj-Refresh-and-Lock method

PEGA ©2025 Pegasystems Inc

Obj-Open-By-Handle method

The handle of an instance is a unique key, in an internal format, assembled by the system that identifies an instance in the Pega Infinity™ database.

It may differ from the visible key generally used to identify instances, since data record instances can have a unique handle, but more than one instance may share the same visible key. For Rule-resolved instances, the handle includes system-generated identifying values, including the create date and time, that make it unique.

If your use case requires opening a specific instance that is not the instance found by rule resolution, you can reference the *Obj-Open-by-Handle* method in activities if you can determine the unique handle that uniquely identifies which instance to open. Otherwise, use the *Obj-Open* method. For more information, see Obj-Open method.

The *Obj-Open-By-Handle* can only be used for data record instances that have a handle. Not all instances will have a handle. Additionally, any class that is not writable will not have a handle. This includes Abstract classes and classes mapped to views in the database.

When used on a primary page, this method clears any data on the page and reuses its name and position in an embedded `Page List`.

An external class (defined through the Connector and Metadata wizard) may not contain one property that can serve as a handle. To support the Obj-Open-by-Handle method with external classes, a substitute string is assembled from the key columns of the table.

> ⚠ **CAUTION:** Except in rare cases, identify in the **Step Page** field the name of a page to be created by this method. If the **Step Page** field of the step is blank, this method opens the object into the primary page of the current activity, replacing the current contents of the primary page and changing the class of

©2025 Pegasystems Inc

> the primary page to match the class of the object. Typically, this is not desirable.

Function Rule

You can call the standard function Rule *pxDoesHandleExist()* in the Utilities library to determine whether a specific instance exists in the database, as an alternative to creating an activity that uses the Obj-Open method and tests whether the instance was or was not found. The *pxDoesHandleExist* function accepts as a single parameter a text value containing the candidate handle and returns true or false. The clipboard is unchanged. The function execution is faster than using an activity.

Parameters

You define the parameters listed in the following table for the Obj-Open-By-Handle method.

Parameter	Description
InstanceHandle	Enter the handle that identifies the instance to be opened as a literal or a property reference. This corresponds to the value of the *pzInsKey* property for an internal class, and a simulated handle for an external class. Because the handle is lengthy and not easily displayed, it is often easier to identify it by a property reference, rather than attempt to enter the handle by typing. For more information, see the section Capturing a handle in a property reference below.

©2025 Pegasystems Inc

Parameter	Description
Lock	Select this option if you plan to change and save or delete the instance. Your session cannot lock the object unless your access role includes the ability to update objects of the class. You cannot lock an object unless the **Allow Locking?** check box (on the **Locking** tab of the Class form) for the object class is selected. (If the class is part of a class group, the **Allow Locking?** check box on the class named for the class group is applicable.) **CAUTION:** Locking an object from an external class prevents updates by other requestors in the Pega Platform system. Locking does not prevent changes to the external database — including changes to the row open in the Pega Platform — by other systems.
ReleaseOnCommit	Select this option to cause a lock to be released when your changes are completed through the Commit method. In most cases, select this option if you selected the **Lock** box. In the unusual case that you want to retain the lock after Commit (perhaps because additional changes are

©2025 Pegasystems Inc

Parameter	Description
	expected), your activity can later use the Page-Unlock method to release the lock.
LockInfoPage	Optional. As a debugging aid, identify the name of a page to be created by this method to hold the results of the locking attempt when it fails. The page, of class *System-Locks*, contains the following properties: • *pxOwnerID* — requestor who holds the lock • *pxExpireDateTime* — date and time the lock will go soft • *pxUpdateOperator* — *pyUserIdentifier* value for the requestor who holds the lock • *pxLockHandle* — text string that defines the lock If the *Obj-Open-By-Handle* fails to acquire a lock, this page is created and can be viewed with the Clipboard tool or the Tracer. By convention, this page is named *LockInfo* in standard Rules. When a lock failure occurs during a flow execution, the flow can display the standard harness *Work-.LockInfo* (or a similar harness incorporating the standard section *System-Locks.LockInfo*) to report the situation. If the lock not acquired is held by the same user in a separate requestor

©2025 Pegasystems Inc

Parameter	Description
	session, the user can click the **Release Lock** button to relinquish it.
CheckSecondaryStorage	Optional. Select this option to search for the instance in archived Case data in the secondary storage repository. The method retrieves information about the instance from the secondary storage repository but does not reopen the instance. Using the retrieved information, you can review the Case in read-only form or create a clone of the archived instance and save it in the database as a new instance. For more information on Case archival, see Archiving and expunging Case data.

Results

This method accesses the Pega Infinity database and finds the unique instance that matches the handle. It copies the object's properties and values into a new clipboard page (identified in the Step Page column, or the activity's primary page if no step page is specified).

If it finds the specified step page, the system clears any data that is on it and re-uses it. If the step page is not found, the system creates it. If the Step Page field is blank, the system clears and reuses the primary page of the activity.

- If the **Lock** check box is selected and the requestor has permission to update objects of this class, the system locks the instance.
- If the **LockInfoPage** parameter is not blank, the system creates a page of that name and reports results through property values on that page.

©2025 Pegasystems Inc

The *Obj-Open-By-Handle* method can only find and open instances that are committed to the database. If an instance is in a deferred operations list at the time this method is called, the instance will not be found. In this situation, you must either use *Obj-Open* method or perform a Commit operation prior to calling the *Obj-Open-By-Handle* method.

Fail and Warn conditions

The following table lists conditions that cause the method status to be set to `Fail` or `Warn`.

Status	Cause
Warn	The class does not allow locking, but the **Lock** parameter (on the **Locking** tab of the **Class** form or in some cases the Class Group form) is selected.
Fail	The object is opened to the clipboard, but the instance is locked by another requestor.
Fail	The **Lock** parameter is selected, but at, runtime, the requestor's access role does not allow the requestor to update objects of that class.
Fail	The page of the indicated class that is created is empty except for the *pxObjClass* property. This occurs when the key supplied does not match the key of an instance in the database.

Avoid overwriting an existing page

You may want an activity to check whether a page of a certain name exists before you open a new page.

PEGA ©2025 Pegasystems Inc

In the precondition to the activity step, enter the following:

```
= @PageExists("myPage", tools)
```

where myPage is the name of the clipboard page.

You may also want to check whether the page on the clipboard is the same as the one you want. For example, your activity performs an Obj-Open-By-Handle method and the handle is stored in a local variable `Local.strHandle`. After the *PageExists* check, perform the following check:

```
@GetInstanceHandle("myPage", tools) == Local.strHandle
```

If both checks are true, the page on the clipboard is the same as the one you plan to open, so you can skip the step.

Capturing a handle in a property reference

You can obtain a property reference by creating and running a list report (using the Obj-Browse or Obj-List method) earlier in your activity, before the Obj-Open-By-Handle step.

Enter the name of a list Rule (instance of the *Rule-Obj-List* Rule type) in the *RuleObjList* parameter of a step that uses the Obj-List method.

After you execute that step, the results appear in embedded clipboard pages under the *pxResults* page. Each instance listed (pxResults(1), pxResults(2), and so on) includes a property *pyInsHandle* that has the handle as its value.

In some settings, a user selects one element from the results list. Your activity can reference this property in the *InstanceHandle* parameter for the desired instance.

PEGA ©2025 Pegasystems Inc

Checking the method status

The method updates the *pxMethodStatus* property. For more information, see How to test method results using a transition.

The Activity-Clear-Status method may be useful following an Obj-Open-by-Handle method.

Restrictions

The method cannot be used in a step that involves iteration.

Obj-Refresh-and-Lock method

Use this method to assure that the contents of the step page are current and a lock is held on the corresponding object.

- If the object is locked and the lock has not expired, this method has no effect.
- If the object is not locked, this method acquires a lock and if necessary replaces the step page contents with the current value of an object instance.

This is useful if you are not certain whether the contents of the step page are current or whether your requestor session already holds a lock on the page.

> ⚠ **CAUTION:** Locking an object from an external class prevents only updates by other Pega Platform requestors. Locking does not prevent changes to the external database — including changes to a row that is open in PRPC — by other systems.

Parameters

This method has two parameters.

PEGA ©2025 Pegasystems Inc

Parameter	Description
ReleaseonCommit	Select this box if you want the system to release the lock when the next Commit method occurs for this instance. In the unusual case that you want to retain the lock after Commit — perhaps because additional changes are expected — your activity can later use the Page-Unlock method to release the lock.
LockInfoPage	Optional. As a debugging aid, identify the name of a page to be created by this method to hold the results of the locking attempt when it fails. The page, of class *System-Locks*, contains the following properties: • *pxOwnerID* — Requestor who holds the lock • *pxExpireDateTime* — Date and time the lock will expire, or become "soft" • *pxUpdateOperator* — *pyUserIdentifier* value for the requestor who holds the lock • *pxLockHandle* — Text string that defines the lock If the Obj-Refresh-and-Lock fails to acquire a lock, this page is created and can be viewed with the Clipboard tool or the Tracer. By convention, this page is named `LockInfo` in standard rules. When a lock

©2025 Pegasystems Inc

Parameter	Description
	failure occurs during a flow execution, the flow can display the standard harness *Work-.LockInfo* (or a similar harness incorporating the standard section *System-Locks.LockInfo*) to report the situation. If the lock that was requested but not acquired is held by the same Operator ID in a separate requestor session, that user can click the **Release Lock** button to release the lock.

Results

The system checks whether the object on the step page is locked and the lock has not expired. If the lock is held, the method creates the results page as specified in the LockInfoPage parameter, but no further processing occurs.

If the step page does not exist, the system creates a new page.

If no lock is held, the system:

1. Assembles the key to the object instance using information in the class instance.
2. Acquires a lock on the object.
3. Determines whether the current clipboard page is stale, meaning that the PegaRULES database row is newer. To make this determination, it:
4.
    - Compares the last update time stamp of the instance on the step page on the clipboard (property *pxUpdateDateTime*) with the *pxUpdateDateTime* time stamp of the instance on the database. If the times are the same, the clipboard copy is not stale; if they differ, the clipboard copy is stale.
    - If the *pxUpdateDateTime* property has no value in both cases, it performs a similar comparison is based on the *pxCreateDateTime* properties.

©2025 Pegasystems Inc

    - If the instance was saved recently but not yet committed, the comparisons are made with the *pxUpdateDateTime* or *pxCreateDateTime* in the saved copy, not the database row.
5. If the clipboard copy is stale, it reloads the step page from the current database instance.

`DateTime` values are captured to the nearest millisecond. However, for certain database software products, the values are rounded when saved. The comparisons described above consider such rounding. See the Pega Community article *Understanding the Obj-Lock-And-Refresh method* for details.

Fail and Warn conditions

These situations cause the method status to be `Fail` or `Warn` :

- If the instance is locked by another requestor, the method returns a Fail status.
- If the key supplied does not match the key of any instance in the database, the system creates a page of the indicated class that is empty except for the *pxObjClass* property, and returns a `Fail` status.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Restrictions

This method cannot be used in a step that involves iteration.

Do not use this method to open and acquire locks on rule instances.

Debugging

To debug this method, use the Logging Level Settings tool with the following Java class:

©2025 Pegasystems Inc

```
com.pega.pegarules.engine.database.Opener
```

- Page-Unlock method
- Object locking

Obj-Save method

Use this method to request that the system save a clipboard page to the PegaRULES database or, if the page belongs to an external class, an external database.

The Obj-Save method uses properties on the page to derive the internal key under which it will be saved. This method can create a new database instance or overwrite a previous instance with that key.

> **NOTE:** In most situations, it is recommended that you use flow processing or activity APIs such as Save or UpdateWorkObject instead. Using Obj-Save skips common processing such as validation, creating history, and keeping the properties that track last update information current.

You must update the following properties manually if you use the Obj-Save method:

- pxUpdateDateTime
- pxUpdateOperator
- pxUpdateOpName
- pxUpdateSystemID
- pxUpdateOrgUnit (only in the Work- class)
- pxUpdateCounter (only in the Work- class)

If you use the Obj-Save method, you can use apply the pxDefaultUpdateProperties data transform to update the pxUpdateDateTime, pxUpdateOperator, pxUpdateOpName, and pxUpdateSystemID properties.

©2025 Pegasystems Inc

Obj-Save method overview

An object that is stored in the PegaRULES database is persistent. The object is available to other users, and remains after the requestor session that created it ends.

This method does not always cause the object to be written immediately to the database. Often, developers use this method to perform a " deferred write" operation. In such cases, later execution of the Commit method, in the same Thread, completes the database update.

Select the **WriteNow** parameter to force this page (and only this page) to be written to the database as part of the Obj-Save method. Unlike the Commit method, the Obj-Save method does not operate on all previously marked-for-commit objects, only on the current page, and does not release locks.

You can reverse the effect of an Obj-Save method — if it has not yet been committed — with the Obj-Save-Cancel or the Rollback method.

Cautions

- In all but rare situations, do not save a page that contains messages and so is marked as invalid.
- When you use the Obj-Save method in an activity with **Activity Type** of `Trigger`, `Utility`, `Notify`, `Assign`, or `Route`, do not check the **WriteNow** check box. Leave locking and committing changes to work items and assignments to standard activities that support flow execution. See Understanding locking and transactions in flow executions.
- Do not use Obj-Delete method on instances in a class derived from the *Index*-base class. Use only Declare Index rules to change these instances.
- When using Obj-Save (for example, with an external database table), the page cannot contain any property corresponding to an auto-increment database column, even if that column is the primary key.

©2025 Pegasystems Inc

Parameters

This method has three parameters:

Parameter	Description
WriteNow	Normally, leave this box cleared. Select the box to cause the system to write this page's data to the database as this method executes. If selected, when the activity executes, the system performs an immediate write-through (database commit) of the page to the database. Clear the **WriteNow** check box in any activity of type `Trigger`, `Utility`, `Notify`, `Assign`, or `Route`, and in situations where you want to mark a page to be saved, but want the system to perform the database commit later. Do not select this box for activities that support flow execution. The locking and database commit operations for flows are correctly supported by standard activities associated with specific flow shapes.
WithErrors	Select this box if the instance is to be saved even when it contains messages, indicating validation errors. Although allowed, saving invalid instances can seriously affect the results or correct

©2025 Pegasystems Inc

Parameter	Description
	operation of your application. Many activities assume that a newly opened instance from the database is valid.
OnlyIfNew	Select to cause only an insert operation, not insert or update. If the activity is operating in batch mode, it does an insert instead of a defaulted update. The system does not check if the item exists before insertion, so duplicate key exceptions are possible. In such circumstances, ensure that you implement the appropriate logic.

Results

First, the system examines the page's *pxObjClass* property to determine its class. It uses the corresponding *Rule-Obj-Class* instance to find the **Database Table** name, **Lock Key** format, and so on.

The system determines whether a lock is already present on the instance. If the current session does not hold a lock (and the instance belongs to a lockable class) the method fails.

If the **WriteNow** check box is selected, the system then deletes any old instance with that key and adds a new one. No locks are released. If the **WriteNow** check box is cleared, the system adds this page to the set of deferred-write pages for this requestor.

Properties with a null value (or equivalently "") are ignored (not saved) by this method.

Fail and Warn conditions

These situations cause the method status to be `Fail` or `Warn` :

©2025 Pegasystems Inc

- If the page contains messages (validation errors), then unless the **WithErrors** box is selected, processing halts and the page is not saved.
- You cannot save a page that is locked by another requestor.
- You cannot save a page that your session does not hold a lock on (if the page belongs to a lockable class), unless the object is new, never yet saved.
- You cannot save pages of any class derived from the *Code-* base class or the *Embed-* base class. Such pages exist only on the clipboard.

Troubleshooting

As a debugging aid, the standard HTML rule *@baseclass.DeferredOps* interrogates system internals and presents an ordered list of deferred (uncommitted) database operations for the current Thread. Your activity can display this HTML rule using the Show-Page method.

The Pega log entry " `Obj-Save is trying to write to a non-existent column` " indicates that two incompatible schemas are installed for the PegaRULES database. For more information, see Database tables.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

When a deferred — that is, non-immediate — Obj-Save method fails, the next Commit method will also fail unless one of the following is performed successfully before the Commit method is attempted:

- Another deferred Obj-Delete (or Obj-Save) method is performed on the instance
- An Obj-Save-Cancel is performed
- A Rollback method is performed

- Commit method
- Rollback method

PEGA ©2025 Pegasystems Inc

Obj-Save-Cancel method

Use this method to cancel the most recent uncommitted Obj-Save method, so that the instance is not written as part of a later Commit operation. You can also use this method to undo an Obj-Delete that has not yet been committed.

Use the similar Rollback method to cancel all uncommitted Obj-Save and Obj-Delete operations, not only the most recent one.

Parameters

This method has no parameters.

Results

The page identified in the most recent Obj-Save or Obj-Delete method is removed from the set of pages pending a Commit method. When a later Commit method executes, this page is not included in those committed.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Obj-Save method
- Rollback method

Obj-Sort method

Use this method to sort the clipboard pages that are the values of a property of mode `Page List`. You can specify one or more properties to sort on, and whether the sort sequence is ascending or descending for each sort level.

Parameters

This method has two base parameters and an array of two parameters per row:

©2025 Pegasystems Inc

Parameter	Description
PageListProperty	Enter a property reference that identifies a target property of mode `Page List` to be sorted.
Class	Identify the class of the pages within the `Page List` property.
SortProperty	Identify in rows of this array a property reference (starting with a period or a page name) on each page of the `Page List` property that is the basis of sorting. Each property reference must identify a `Single Value` mode property or a single text string. The order of your entries in this array is significant. The first property listed is the major sort.
Descending	Select to cause the method to sort in descending order for the property. Clear the box to sort in ascending order.

Results

The system sorts (reorders) the pages of the indicated `Page List` property. The rest of the clipboard (other than the *pxMethodStatus* value) is unaffected.

Sorting is sensitive to case and to **Type**.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

©2025 Pegasystems Inc

Restrictions

This method cannot be used in a step that involves iteration.

Obj-Validate method

Use this method to apply a validate rule (*Rule-Obj-Validate* rule type) for the object identified on the primary page or step page.

Use the Page-Validate method — not this method — to validate all the properties on a page against the requirements defined in their respective *Rule-Obj-Property* rule instances.

A later Page-Clear-Messages method applied to the same page removes the messages set by this method.

A validate rule (*Rule-Obj-Validate* rule type) can call edit validate rules (*Rule-Edit-Validate* rule type). However, note that the current step page is not the primary page for evaluation of the edit validate rules.

Parameters

This method has two parameters:

Parameter	Description
Validate	Enter the second key part of a validate rule to apply.
OverrideClass	Optional. Enter the name of a class where rule resolution will begin searching for the Validate rule. If blank, rule resolution begins searching for the validate rule on the class of the step

©2025 Pegasystems Inc

Parameter	Description
	page. You can override the class of the step page with a higher or lower class.

Results

The system uses rule resolution to find the validate rule specified by the parameters. It then applies the validation rule to the properties on the step page (or primary page if the **Step Page** column is blank). If any validations fail, the system adds messages to the page.

Ordinarily, if any property fails validation, the page cannot be saved from the clipboard into the database.

Checking the method status

This method does not usually update the *pxMethodStatus* property.

- Testing method results using a transition

Page-Change-Class method

Use this method to change the class of the top-level page or embedded page identified in the Step Page column.

Before changing the class of a page, be sure that all properties referenced on the page are also defined in the target class (the destination class).

Parameters

This method has three parameters:

©2025 Pegasystems Inc

Parameter	Description
ObjClassNew	Identify the target or destination class of the page.
Data Transform	Optional. Identify a data transform (in the target class or a superclass of that class) to apply to the page.
Keep	Optional. Enter 0, 1 or 2 to control the how properties are added to the page. If not specified, 0 is the default. • 0 causes the value of a property already present on the step page to be retained — not overwritten — by the data transform . • 1 causes a property value on the data transform to overwrite a value already present on the step page. • 2 retains the values of properties already on the step page (as with 0), but with special treatment for `Page List`, `Value List` and `Value Group` properties, as described in the table. **Mode** — **Results with Keep 2** `Page List` — First, all data transform `Page List` elements are merged to the step page

©2025 Pegasystems Inc

Parameter	Description
	<table><tr><th>Mode</th><th>Results with Keep 2</th></tr><tr><td></td><td>`Page List`. (This is the same behavior as keep 0). Then, if the step page `Page List` property contains more elements than the data transform `Page List` property, the first element of the data transform list is used as the value for all the remaining target list elements. This differs from processing with **Keep** 0, which ignores the target pages beyond the size of the data transform list.</td></tr></table>

©2025 Pegasystems Inc

Parameter	Description
	<table><tr><th>Mode</th><th>Results with Keep 2</th></tr><tr><td>`Value List` and `Value Group`</td><td>If the step page contains the property but it is empty (contains no elements), the data transform property is copied — including all its elements — into the destination. This differs from the Keep 0 behavior, as existing destination properties may be altered.</td></tr><tr><td>(other modes)</td><td>Same as Keep 0.</td></tr></table>

Results

The system creates a new page of the target class identified in the ObjClassNew parameter.

If a data transform is specified in the Data Transform parameter, the system uses rule resolution to find the rule, and uses it as described above.

©2025 Pegasystems Inc

The system copies properties from the existing step page to the new page, in accord with the **Keep** parameter.

The previous step page (with the previous class) is destroyed.

This method does not validate property values. The page name is unaltered. This method does update the *pxMethodStatus* property. See How to test method results using a transition.

Restrictions

This method cannot be used in a step that involves iteration.

- Data Transforms
- Page-New method

Page-Clear-Messages method

Use the Page-Clear-Messages method to remove all property messages and certain page messages from the page identified in the **Step Page** column of the current step.

> ⚠ **CAUTION:** Use this method with care, because various application-related processing may have added the messages present on the page. Clearing a message before the cause is reported to a user or logged is not advisable.

Your application can clear messages by other means. For example, when a user submits an HTML form to the server containing input values for properties on a page, the server clears all page messages for the page, and also clears property messages for any property with submitted values (that is, any property that was touched by the user before submitting the HTML form.)

To avoid any chance of introducing invalid data into the database, design your activity to re-validate any page that was cleared of messages, before the activity attempts to save the page.

PEGA ©2025 Pegasystems Inc

Parameters

This method uses no parameters.

Results

All messages placed by the Page-Set-Messages method on the step page are deleted.

Messages placed by Constraints rules, edit validate rules, and property type validation are not cleared — they are cleared and the restrictions are detected only when the property value changes.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Page-Copy method

Use this method to copy the contents of a source Clipboard Page to a new or previously created destination Clipboard Page. The Page-Copy method does not alter the source page.

After this method completes, the destination page contains properties copied from the source page, and can contain additional properties set in a Data Transform.

> ⚠ **CAUTION:** Do not use Data Pages in both the CopyFrom and CopyInto parameter because this will cause the identifying parameters on the Data Page, pzPageNameBase and pzPageNameHash, to also be copied. This can lead to unexpected behavior.

Parameters

This method has four parameters:

©2025 Pegasystems Inc

Parameters	Description
CopyFrom	Enter the name of the source page from which to obtain property values to copy. If blank, the system uses the Step Page of the current step, or the primary page of the activity if the Step Page field is blank. If **CopyInto** is left blank, this parameter is required.
CopyInto	Enter the name of the destination page, which may or may not already exist. If it exists, it must have the same class as the source page. The current contents of the destination page are erased before the copy operation occurs. If blank, the system uses the Step Page of the current step, or the Primary Page of the activity if the **Step Page** field is blank. If **CopyFrom** is left blank, this parameter is required. Choose a valid page name. To create a destination page that is a new, last page in a `Page List`, identify a fully qualified `Page List` property name and use the `<APPEND>` keyword as the index.
Data Transform	Optional. Identify a Data Transform to be applied to the destination page before copying data from the source page.
PageList	Optional. To save the name of the new page as the final element in a list, identify a `Value List` property (not a `Page List` property). Use the notation pagename.propertyname. The system appends the name of the new page as the value of the last element of the `Value List` property.

Results

The Page-Copy method works as follows:

1. If the source page cannot be found, the method does not fail. It does not alter any pages. The resulting method status value is `Good`.

©2025 Pegasystems Inc

2. If the destination page (identified in the **CopyInto** parameter value) does not already exist, the system creates a blank page of the same class as the source page. It can be a top-level page or an embedded page.
3. If the destination page does exist (and is of the same class as the source page), it is cleared.
4. If the **Data Transform** parameter is not blank, the system finds the Data Transform using Rule resolution. It applies the Rule to the destination page.
5. Next, the system copies properties and all messages from the source page into the destination page. This copying can create new properties on the destination page or update the value of existing properties created by the Data Transform.
6. During this copy processing, each property is validated against its definition (its *Rule-Obj-Property* instance). If a property value fails validation, the system adds a message to the destination page.
7. If a `Value List` property is identified in the fourth parameter, the system adds the name of the new page as the final element in the list.

Page names are unique within the Clipboard. If an existing page with the **CopyInto** destination name is already present at the same point in the Clipboard structure, the system deletes it and replaces it with an empty page.

Checking the method status

This method updates the *pxMethodStatus* property.

- Page names and reserved pages
- Testing method results using a transition

Page-Merge-Into method

Use the Page-Merge-Into method to merge the contents of two or more pages into one page. Specify the source pages in the MergeFromList parameter.

©2025 Pegasystems Inc

Parameters

The Page-Merge-Into method accepts a list of source pages and one base parameter.

Parameter	Description
MergeFromList	List one or more page names to merge into the step page. Order is significant, because the pages are processed in the sequence that you specify here. Don't include the step page (the destination) as one of the source pages. Identify one or more single pages for this parameter. You can reference a Page List property but you must include an explicit index value.
Keep	Optional. The parameter controls the merge operation. Specify how properties and values are copied from the source pages into the destination page, by entering one of the following values: • 0 - The value of a property that is already present on the step page is retained, not overwritten by a value for that property on a source page. • 1 - The value of a property on a source page overwrites a value for that property that is already present on the step page.

©2025 Pegasystems Inc

Parameter	Description
	• 2 - The values of properties that are already present on the step page are retained (as with 0), but with special treatment for Page List, Value List and Value Group properties, as described below. **Page List** All source Page List elements are merged to the destination Page List with retaining the values of the destination page elements. This is the same behavior as for the **Keep** parameter set to 0. If the destination Page List property contains more elements than the source Page List property, the value of the first element of the source Page List is used for all the remaining target list elements. This behavior is different from that for the **Keep** parameter set to 0. With the **Keep** parameter set to 0, the values of the target list elements that are beyond the size of the source list are kept. **Value List and Value Group** If the destination page contains an empty property (with no elements), the source property is copied, including all its elements,

©2025 Pegasystems Inc

Parameter	Description
	into the destination. This behavior is different from that for the Keep parameter set to 0, as existing destination properties may be altered. **Other modes** Same behavior as for the **Keep** parameter set to 0. • 3 - The value of a property on a source page overwrites a value for that property that is already present on the step page (as with 1), but recursively for Page List, Page Group, Value List, Value Group.

Results

The system finds the destination page named in the **Step page** column in the activity step that calls this method. If not found, it creates a new page with this name.

If any of the pages that are specified in the **MergeFromList** parameter are not found, the merge process ignores them.

It copies properties and values from the source pages into the destination page, retaining or overwriting property values in the destination page according to the value of the **Keep** parameter.

It processes the source pages in the sequence that you entered them in the parameter array. As it copies a property and value, the system validates the value against the property definition (the *Rule-Obj-Property* rule) and marks any property that has an error. It updates the *pxErrorList* property with error messages.

©2025 Pegasystems Inc

Source pages are unaltered.

Checking the method status

This method updates the *pxMethodStatus* property. For more information, see How to test method results using a transition.

Restrictions

This method cannot be used in a step that involves iteration.

Page-New method

Use the Page-New method to create a page on the clipboard. The new page may be a top-level page or an embedded page.

You can identify a data transform to initialize the newly created page. The data transform can set values for one or more properties.

> ⚠ **CAUTION:** Except in rare cases, identify in the **Step Page** field the name of the page to be created by this method. If the **Step Page** field of the step is blank, this method clears the primary page of the current activity and may change its class. This often is undesirable.

Use the Apply-DataTransform method, not the Page-New method, to apply a data transform to an existing page.

Parameters

Identify the name of the page in the **Step Page** column of the step. Choose a valid page name. See Understanding page names and reserved pages.

This method has three parameters:

©2025 Pegasystems Inc

Parameter	Description
Data Transform	Optional. Enter the second key part of a data transform to apply to the new page. Identify a data transform with an Applies To class that is the same as, or a superclass of, the class of the new page.
PageList	Optional. To save the name of the new page as the final element in a list, identify a `Value List` property (not a `Page List` property). Use the notation pagename.propertyname. The system appends the name of the new page as the value of the last element of the `Value List` property.
NewClass	Optional. Identify a class — in most cases concrete class — for the new page. Enter the keyword `$NONE` to create a classless page. If left blank, the system creates the new page using the class of the page specified in the **Step Page** column of the activity step being executed. (It retrieves the class of this page from the array entered in the **Pages & Classes** tab of the Activity form.) If the **Step Page** field is also blank, the system creates the new page of the class that is the first key part of the current activity.

©2025 Pegasystems Inc

The following keywords mean different things in other parts of PRPC. Using them as a page name is not recommended: `param`, `local`, `primary`, `steppage`, `mysteppage`, `top`, `parent`, or `current`.

The following names and patterns are not valid: `pxNode`, `pxRequestor`, `pxThread`, `locate*` `prompt*`, `$*`, `px*`, `pz*`, or `py*`.

To avoid potential confusion, don't use Declare_ as the initial portion of a page name. By convention, such names indicate data pages.

Results

The system creates a new page as specified, using the name in the **Step Page** column of the step. Next, it finds the data transform using rule resolution and applies it to initialize the contents of the new page.

The page name may identify a Data Page rule that has a **Page Type** (on the **Definition** tab) of `Read-Only`. If appropriate conditions are met, the Data Page rule is executed and creates a clipboard page within the User Pages group. That page can thereafter be updated, saved, and processed like other user pages.

> ⚠ **CAUTION:** Page names are unique within the clipboard. If another page with the new name is already present, the system replaces the contents of new page with the contents of the existing page that has the same name.

Checking the method status

This method updates the *pxMethodStatus* property. See How to Test method results using a transition.

To avoid overwriting an existing page

You may want the activity to check whether a page with a certain name exists before you open a new page.

PEGA ©2025 Pegasystems Inc

In the activity step's precondition, enter the following:

```
= @PageExists("myPage", tools)
```

where myPage is the name of the clipboard page.

Restrictions

This method cannot be used in a step that involves iteration. This method cannot be used to create a data page.

- Page names and reserved pages
- Apply-DataTransform method

Page-Remove method

Use this method to delete one or more pages from the clipboard. The contents of the database are not affected.

As a sound development practice, design your activities or processing cycle to remove clipboard pages after they are no longer needed. A large clipboard can affect system performance. You can use the Performance tool to monitor the size (in bytes) of the clipboard.

To remove all User pages, create a Java step containing these two lines:

```
PRThread thread = tools.getThread();
thread.removeAllPages(null);
```

You can use an activity to remove all user pages, either when developing or at appropriate points in an application.

PEGA ©2025 Pegasystems Inc

Parameters

This method uses the page (if any) in the Step Page field plus an optional array of page names.

Parameter	Description
Page	Optional. Enter the name of a top-level or embedded clipboard page to delete.

Results

If the **Step Page** field is not blank, the page identified in that field is deleted. (You can't delete the primary page of the activity.)

In addition, the method finds each page (top-level or embedded) identified in the array and deletes the page.

If a page you specified is not found on the clipboard, the Page-Remove method reports a warning error, but does not fail.

This method does not alter the contents of the PegaRULES database.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Tracking utilization statistics with Performance Analyzer

Page-Rename method

Use this method to rename a top-level page, or to name a primary page.

The page specified in the activity's **Step Page** field is the page to rename (a blank **Step Page** field means the primary page). The page must be listed in the **Pages & Classes** tab of the activity, unless it is the primary page. The **PageNewName** parameter may be the

©2025 Pegasystems Inc

same class as the step page or a different class, but must also be specified in the activity's **Pages & Classes** tab.

You cannot rename an embedded page.

Parameters

This method has one required parameter:

Parameters	Description
PageNewName	Enter a new page name, starting with a letter. Use only letters, digits, and the underscore character.

Choose the page name carefully. See Understanding page names and reserved pages.

Results

The system renames the page identified on the **Step Page** column of the current step to the new name specified in the parameter.

Page names are unique within the clipboard. If another page with the new name is already present, the system deletes it and replaces it with the renamed page.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Defining the pages and classes of a Rule

Page-Set-Messages method

Use the Page-Set-Messages method to add a message to a clipboard page. Like a message associated with a property, a message associated with a page normally

©2025 Pegasystems Inc

prevents the page from being saved into the database. If the page is an embedded page, the message is added to all its parent pages.

A later Page-Clear-Messages method applied to the same page removes the messages set by this method. In addition, page messages are cleared when a workstation user submits an HTML form to the Pega Platform server that includes values for properties on that page.

Parameters

This method has three parameters:

Parameter	Description
Page	Enter a named page from the **Pages & Classes** tab to contain the message. Use the keyword `Primary` for the primary page. This field is optional. If you leave it blank, the system uses the step page, or the primary page if no step page is specified.
Category	Enter a localization category that is the second key part of a field value. The default is *pyMessageLabel*. Otherwise, leave this field blank.
Message	Identify the message to add to the page. You can enter values in this field in one of the following ways: • The name of a *Rule-Message* instance, without quotes. The list of available options in the SmartPrompt is limited

©2025 Pegasystems Inc

Parameter	Description
	to `Error` and blank category messages of the class of the specified **Page**. If the message supports parameters, append a tab-delimited string to the message name. • The third key part of a field value rule where the class of the specified **Page** is the first key part and the value specified in the **Category** field is the second key part. Use this approach when the **Category** field is not blank and the specified field value does not contain property references. Property references can lead to confusion because field values are evaluated each time the page is read, and property values can change over time. • A literal text string within double quotes. Avoid this option if your application supports localization.

Results

The message becomes associated with the page as a page message.

©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Page-Unlock method

Use this method to unlock a page in the unusual situation that the page was locked during the Obj-Open operation but the ReleaseOnCommit parameter was not selected.

Typically, when the Obj-Open method opens and locks an object from the database, the **ReleaseOnCommit** parameter is selected. When a later Commit method executes, the system automatically releases the lock.

However, when the **ReleaseOnCommit** parameter is not selected, the Commit operation has no effect on the lock, which remains in force. This capability facilitates multiple Commit operations on one page without the need to re-acquire the lock.

You can also use this method to release a lock that is no longer needed, regardless of the **ReleaseOnCommit** parameter value.

Parameters

This method has no parameters.

Results

The page identified in the **Step Page** column (or the primary page, if the **Step Page** column is blank) is unlocked.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Commit method

©2025 Pegasystems Inc

- Obj-Open method
- Object locking

Page-Validate method

Use the Page-Validate method to force validation (or revalidation) of the value of all properties on the step page. The system adds a message to each clipboard property that fails validation against the restrictions in its property.

If the page has hundreds or thousands of properties, this method can be expensive in terms of system resources. The properties on the page already set (through the Property-Set method or similar methods) may already be validated. This method validates them again, and also validates any property values not changed since the page was opened.

Unless the content of the page was altered by Java code or by unorthodox means, the Page-Validate method is not necessary.

Use the Obj-Validate method, not the Page-Validate method, to validate specific properties identified in a *Rule-Obj-Validate* rule.

A Page-Clear-Messages method applied later to the page removes the messages set by this method.

Parameters

This method has one parameter.

Parameter	Description
WithErrors	In most cases, leave cleared. If this check box is not selected and the page identified in the **Step Page** of this step already contains messages, this method ends. It does not validate properties or create messages.

©2025 Pegasystems Inc

Parameter	Description
	Select this box to force this method to validate properties even if the page already contains messages.

Results

The system validates each property on the step page against the requirements imposed by that property's definition, such as:

- Confirming that a definition for the property can be found (in a RuleSet and version available to this requestor), based on the property's name and the class of the page.
- Confirming that the property value matches the **Property Mode** and, for `Single Value` properties, the **Type** restrictions appropriate for dates, times, numbers, and Boolean values.
- Checking values against those provided (if any) by the **Table** fields on the **General** tab of the property.
- If the property definition references a property qualifier, confirming that the qualifier value is present and meets its own constraints.
- Comparing the length of values against the maximum defined in the definition.

This checking happens recursively for embedded pages. If any property fails validation, the system adds a message to the property and to the page.

A property with a null value is considered valid.

When a workstation user submits an HTML form to PRPC, previous page messages corresponding to the input are cleared.

©2025 Pegasystems Inc

Checking the results

When validation fails, a messages is added to the page. In a transition, use the when condition rule named *@baseclass.HasMessages* to test for the message. Validation failure does not modify the *pxMethodStatus* property value.

Restrictions

This method cannot be used in a step that involves an iteration.

- Obj-Validate method

Privilege-Check method

Use this method to determine whether the current requestor has access to a specific privilege (*Rule-Access-Privilege* rule type). The privileges that a requestor holds are determined by the contents of the Privilege tab of the set of Access of Rule to Object rules associated with the access roles of the user.

Parameters

This method has two parameters:

Parameter	Description
PrivilegeName	Identify the privilege to check, the second key part of a *Rule-Access-Privilege* rule instance.
PrivilegeClass	Optional. Enter the **Applies To** key part of the privilege. If blank, the system uses the first key part of the activity as the first key part.

©2025 Pegasystems Inc

Results

The system uses rule resolution to locate the privilege rule, and then determines whether the current requestor holds this privilege. It stores the results in the method status property.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Role permissions and privileges in Access Manager
- Understanding Access of Role to Object rules

Property-Map-DecisionTable method

Use the Property-Map-DecisionTable method to evaluate a decision table and save the result as the value of a property.

In the Diagram tab of a flow rule, a decision shape can reference a decision table.

Parameters

This method has three parameters:

Parameter	Description
PropertyName	Optional. Select or enter a target property to contain the results of the decision table evaluation. Enter a property name or property reference to a single value. (You can't use a local variable.) Leave blank when the **Allowed to Return Values** check box on the Decision Table form

©2025 Pegasystems Inc

Parameter	Description
	is not checked, because such decision tables do not return a result value.
DecisionTableName	Enter the name of a decision table, the second key part of a Rule-Declare-DecisionTable instance.
AllowMissingProperties	If selected, the system attempts to evaluate the rule even when one or more properties referenced in the rule are not present on the clipboard.

Results

The system forms a decision table key using the second parameter and the class of the step page or primary page. It uses rule resolution to locate the appropriate decision table to evaluate.

The system then evaluates the decision table, in the context of current property values, and stores the results in the property identified in the **PropertyName** parameter.

If you specify a **PropertyName** property but the decision table does not return any value, the property is set to an empty string value.

Trapping not-found conditions

If the **AllowMissingProperties** parameter is not selected and a needed property is not on the clipboard, the Property-Map-DecisionTable method places an output parameter *DecisionTableInvalidProperty* on the parameter page of the current activity. This output parameter identifies the name and class of the missing property.

This capability can be useful in situations where a user can be prompted for the missing property.

©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Decision tables

Property-Map-DecisionTree method

Use the Property-Map-DecisionTree method in to evaluate a decision tree (*Rule-Declare-DecisionTree* rule type) and store the result as the value of a property.

In the Diagram tab of a flow rule, the Decision shape can reference a decision tree.

Parameters

This method has four parameters:

Parameter	Description
PropertyName	Optional. Identify a target property to contain the results of the evaluation. (You can't use a local variable here.)
DecisionTreeName	Enter the name — second key part — of a decision tree. The system uses the class of the step page (or primary page) as the Applies To key part.
Input	Optional. Enter a literal value or property reference that is the source value for the decision tree evaluation. When provided, this value is used for comparisons in the top-level structure of the true, for evaluations with no left-hand value, such as:

©2025 Pegasystems Inc

Parameter	Description
	`if > "Hello"` When you leave this blank, the system uses the value of the **Property** field on the **Input** tab of the Decision Tree form for the rule.
AllowMissingProperties	Select to cause decision tree processing to continue even if a referenced property is not present on the clipboard. For a `Single Value` property, the system uses the null string as the value of the missing property. Clear to cause Decision Tree processing to halt if processing references a property that is not present on the clipboard. This causes the system to store an output parameter *DecisionTreeStatus* with value "NotFound" on the parameter page of the activity.

Results

The system forms a decision tree key using the second parameter and the class of the step page or primary page. It uses rule resolution to locate the decision tree to be evaluated.

It then takes as input to the decision tree evaluation either the value of the third parameter, or the value of the **Property** field on the **Input** tab of the decision tree. It evaluates the decision tree in the context of this input value and the current clipboard.

PEGA ©2025 Pegasystems Inc

It stores the results in the property you identify in the first parameter. Typically, the flow referencing this decision tree chooses which connector to follow from the decision shape based on this property value.

Trapping not found conditions

If the **AllowMissingProperties** parameter is not selected and a needed property is not present on the clipboard, the Property-Map-DecisionTree method places an output parameter *DecisionTreeInvalidProperty* on the parameter page of the current activity. This output parameter identifies the name and class of the missing property.

This facility can be useful in those situations where a user can be prompted for the missing property. The Pega Community article cited below provides an example.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Pega Community note

See the Pega Community article *How to evaluate a decision tree and handle errors*.

- Decision trees

Property-Map-Value method

The Property-Map-Value method evaluates a one-dimensional map value (*Rule-Obj-MapValue* rule type) defined in the parameter. The method sets the result as a value for a `Single Value` property.

The related method Property-Map-ValuePair works similarly for two-dimensional map values.

In the **Diagram** tab of a flow rule, a decision shape can reference a map value.

©2025 Pegasystems Inc

Parameters

This method has three parameters:

Parameter	Description
PropertyName	Identify a target property to hold the result of this method. (You can't use a local variable here.)
MapName	Enter the name — second key part — of a map value (Rule-Obj-MapValue rule type).
RowInput	Optional. Identify an input value for the row, as a literal constant or `Single Value` property reference. If you leave this blank, the system uses the value of the ROW PROPERTY property, defined on the **Input** tab of the map value.
AllowMissingProperties	Select to cause map value processing to continue even if a referenced property is not present on the clipboard. For a `Single Value` property, the system uses the null string as the value of the missing property. Clear to cause map value processing to halt when processing references a property that is not present on the clipboard. This causes the system to store an output parameter *MapValueStatus* with value "NotFound" on the parameter page of the activity.

©2025 Pegasystems Inc

Results

The system forms a map value key from the value in the second parameter and the class of the step page or primary page. It uses rule resolution to find the map value.

It evaluates this one-dimensional) rule using as input the value you supply in the **RowInput** parameter (or the property identified the **Input** tab of the map value).

It stores the result in the property you identified in the PropertyName parameter.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Map values
- Property-Map-ValuePair method

Property-Map-ValuePair method

The Property-Map-ValuePair method evaluates a two-dimensional map value identified in the parameters. It sets the value that results into a specified property.

In the **Diagram** tab of a flow rule, a decision shape can reference a map value.

Parameters

This method has five parameters:

Parameter	Description
PropertyName	Identify a target property to receive the result of this method. (You can't use a local variable here.)
MapName	Enter the name — second key part — of a map value

©2025 Pegasystems Inc

Parameter	Description
RowInput	Optional. Identify an input value for the row, as a literal or Single Value property reference. If you leave this blank, the system uses the value of the ROW PROPERTY Property , identified on the **Input** tab of the map value.
ColumnInput	Optional. Identify an input value for the column, as a literal constant or `Single Value` property reference. If you leave this blank, the system uses the value of the Column Property property, identified on the **Input** tab of the map value.
AllowMissingProperties	Check to cause map value processing to continue even if a referenced property is not present on the clipboard. For a `Single Value` property, the system uses the null string as the value of the missing property. Clear to cause map value processing to halt if processing references a property that is not present on the clipboard. This causes the system to store an output parameter *MapValueStatus* with value "NotFound" on the parameter page of the activity.

Results

The system forms a map value key from the value in the second parameter and the class of the step page or primary page. It uses rule resolution to find the map value.

©2025 Pegasystems Inc

It evaluates this two-dimensional rule using as input the value you supply in the **RowInput** and **ColumnInput** parameters, or from the properties identified in the **Input** tab of the map value.

It evaluates the rule and stores the result in the property you identified in the **PropertyName** parameter.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Map values
- Property-Map-Value method

Property-Ref method

Use this method to establish links that identify a partner non-reference property for one or more reference properties.

This method links pairs of property references that each identifies a Page mode or Value mode property.

- The left reference identifies a property (or an element of a `Group` or `List` property) that has the **Reference Property** box (on the **Advanced** tab of the **Property** form) selected. This is known as the reference property.
- The right reference identifies a property (or an element of a `Group` or `List` property that does not have the **Reference Property** box on the **Advanced** tab of the Property form) selected. This is known as the non-reference page. (The terms source and target are ambiguous here, as either page may be read or written to.)

After this method executes, changes to the properties on either of these two property references also update the corresponding properties on the other reference.

©2025 Pegasystems Inc

The two references may both be within the same top-level page or may belong to different top-level pages. (If they belong to different top-level pages, both top-level pages must be on the clipboard when either is accessed.)

Ordinarily, use this method in a once-only or initialization activity. The relationships you define with this method remain in force until another Property-Ref method redefines them.

Parameters

This method accepts an array of pairs of property references. Complete at least one row. If you complete more than one row, rows are processed one by one starting at the top.

Parameter	Description
PropertiesRefName	Enter a reference to a property of mode `Single Value`, `Value List`, `Value Group`, `Page`, `Page List`, or `Page Group` that has the **Reference Property** box selected. For a `List` or `Group` property, include a subscript (index) to identify a specific value or page within the list or group. You can use the keywords `<append>`, `<last>`, `<prepend>`, or `<insert>` to identify the item. You can use an expression that resolves to the reference. However, you can't use the `Param` or `Local` keywords here.
PropertiesRefValue	Enter a reference to a property that is to be linked to the property in the

©2025 Pegasystems Inc

Parameter	Description
	PropertiesRefName field. For a `List` or `Group` property, include a subscript (index) to identify a specific page within the list or group. You can use the keywords `<append>`, `<last>`, `<prepend>`, or `<insert>` to identify the page. You can use an expression that resolves to the reference. However, you can't use the `Param` or `Local` keywords here. Referencing provides only one level of redirection; you can't identify a reference property in this field. For Value Mode Properties For properties of mode `Value`, `Value List`, or `Value Group`, choose a property that is of the same mode and type as the property in the **PropertiesRefName** field. References between properties with incompatible types generate validation errors when the reference is updated. For Page Mode Properties Choose a property of mode `Page`, `Page List`, or `Page Group` for which the class in the **PageClass** field (on the **General** tab of the **Property** form) is identical to or derived

©2025 Pegasystems Inc

Parameter	Description
	from the class in the **PageClass** field of the property in the **PropertiesRefName** field. In this field, you can identify a page embedded in the same top-level page as the page identified by the **PropertiesRefName** field, or a page embedded in a different top-level page. If you refer to a page on another top-level page, ensure that your application always has both pages on the clipboard when either of them is accessed. If the **PropertiesRefValue** value identifies a named page or pages as a context for the property, include these page names and classes on the **Pages & Classes** tab of the **Activity** form. You can't specify a non-reference page that is a parent page to the reference page, because doing so would result in looping or infinite redirection. Design the reference and non-reference pages carefully to avoid configurations that could result in looping or infinite redirection.

Results

When executed, this method establishes an enduring relationship between the reference property and non-reference property identified in each row.

©2025 Pegasystems Inc

If an expression appears in a parameter, it is evaluated only once.

Rows in the array are executed in order, so if one reference property is mentioned more than once during the execution, only the last relationship is effective.

If a page identified by either field is not present on the clipboard, it is added as a blank page. (However, if a higher-level page identified as context for the **PropertiesRefValue** page does not exist, this method throws an exception.)

When this method completes, any accesses or property changes to a property value within either page appears to also access or change a corresponding value on the other page. This includes property changes achieved through Property-Set methods, Declare Expression rules, page messages, property messages, and other methods.

Internally, only the non-reference property is maintained; accesses and updates that identify the **PropertiesRefName** property are automatically redirected to the **PropertiesRefValue** page or value.

When you save a clipboard page that includes a reference property, only the reference definition is saved. Saving only the reference definition avoids duplication of data, but also means that the properties on the reference page are usable only when the non-reference page is also present on the clipboard.

Benefits and applications

Reference pages offer two benefits:

- Replacing long, complex, and possibly tedious-to-enter property references with shorter property references.
- Avoiding the memory and processing demands of maintaining duplicate copies of values or pages, whether in the same top-level structure or in distinct structures.

The first benefit is realized by developers, who can type a shorter property reference rather than a longer one.

PEGA ©2025 Pegasystems Inc

The second benefit affects both development and runtime. When an application maintains redundant data by design, developers must ensure that changes to data are propagated as fast as practical to every copy, so that inconsistent data does not affect the outputs of the application. This can be difficult to implement and test, and can affect performance. Using a reference property ensures consistency, avoids copying, and eliminates the memory and table space required for multiple copies.

Clipboard display

On the Clipboard tool display, reference properties appear on the left panel with an annotation in square brackets like `[Refers to ZZZ]`, where ZZZ identifies the non-reference property. For example: `RefPropPage (Data-Party) [Refers to pyWorkPage.pyWorkParty(Customer)]`

You can use the Find function on the Clipboard tool to search through the values on a reference page.

Show-Page results

As with a non-reference page, you can use the Show-Page method to display a reference property for debugging. The resulting XML output includes the properties on the page, with a special attribute `referenceTo` in the root element that identifies the non-reference property. For example, the page in this XML fragment refers to a non-reference page named Grouping.Party(John):

```
<Owner referenceTo=-"Grouping.Party(John)" >
    <PartyID>John</PartyID>
    <HasValidLicense>false</HasValidLicense>
    ...
</Owner>
```

©2025 Pegasystems Inc

Tracer output

For performance reasons, Tracer output does not by default show changes to reference properties. You can enable Tracer output temporarily for testing:

1. Use the Logging Level Settings tool to set the logging level for the following class to INFO:

```
com.pega.pegarules.engine.clipboard.ClipboardPropertyImpl$ReferenceInformation
```

2. Start the Tracer, and click **Settings** to open the Tracer Settings panel.

3. In the **Event Type** field, type `Reference Properties`. Click **Add**, then select the **Reference Properties** check box that appears.

4. Begin the operation to be traced. Tracer output contains a line labeled `Using reference` for each change to the property.

Removing a linkage

A relationship between two properties established through the Property-Ref method lasts indefinitely, or until it is replaced by a different non-reference property through a later execution of the Property-Ref method. If required, however, you can nullify the value of a reference property (linking it to nothing) with a PublicAPI method:

```
ClipboardProperty.resetReferenceInformation();
```

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

©2025 Pegasystems Inc

Performance

Redirection is fast. To review the number of redirections produced by your requestor session, review the **Reference Property redirections** value in the **Rule Execution** area of the Performance tool full details display.

Property-Remove method

Use this method to delete a property or properties and its associated value from the step page or another specified page. This does not affect the property, its definition.

Parameters

The Property-Remove method has an array of parameters.

Parameter	Description
Property	Identify the target property or properties to be deleted. For properties on the step page, enter only a period and the property name. For other properties, use the pagename.propertyname syntax.

Results

The system removes each property and its value from the step page.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Property-Seek-Value method

Use this method to initiate backward chaining computations for the value of a property, based on Declare Expression rules.

PEGA ©2025 Pegasystems Inc

For example, the standard activity *Work-.VerifyProperty* — referenced in the standard flow action *Work-.VerifyProperty* — calls this method.

Parameters

This method has four parameters:

Parameter	Description
GoalProperty	Identify the property to compute. This must be the Target Property key part of a Declare Expression rule.
AlwaysRecompute	Select to force the system to compute the value of the property identified in the **GoalProperty** parameter even when the property already is present on the clipboard and has a non-null value. In most cases, leave this box cleared, so that the **Compute Value** field in the Declare Expression rule applies.
AlwaysForwardChain	Select to cause the system to forward chain after backward chaining, even when the goal-seek attempt failed. If not selected, when goal seeking fails, any property changes made by the goal seek process are not considered for forward chaining. In most cases, leave this box cleared.
MissingReference Property	Optional. Enter a target property, a local variable, or an activity parameter. When backward chaining fails to compute the

©2025 Pegasystems Inc

Parameter	Description
	property identified in the **GoalProperty** parameter, the method stores name of the property that prevented the computation from completing there.

Results

This method uses backward chaining on the declarative network defined through Declare Expression rules to compute or recompute the value of the goal property.

The system uses the **GoalProperty** parameter and class of the step page (or primary page) as the Applies To key part to search for the Declare Expression rule. It then uses the Declare Expression rule to starts backward-chaining computations to obtain a value for the property.

If backward chaining is successful, the result is a new value for the property identified in the **GoalProperty** parameter. (Other properties may also have new values, all consistent with Declare Expression rules and other declarative rules).

If the backward chaining computation is unable to compute a value for the **GoalProperty** property, it may return instead the name of another property (identified in the third parameter) that — if it had a value — could contribute to the computation. Your application may then prompt a user for a value for this property, or use other means to obtain this value.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Declare Expression Rules
- More about Declare Expression Rules

PEGA ©2025 Pegasystems Inc

Property-Set-Messages method

Use this method to associate a text message with a property. The system reads the appropriate property and adds the message to the property. If a property is on an embedded page, the message is added to all its parent pages.

You can provide the entire literal text of the message or reference a message rule key that in turn contains message text (*Rule-Message* rule type).

Some message rules accept fill-in-the-blank parameters, identified in the message text by a number within parentheses. Do not confuse method parameters with message parameters, which appear within the **Message** parameter, and are numbered.

Parameters

This method has three parameters:

Parameter	Description
Category	Enter a localization category that is the second key part of a field value. The default is *pyMessageLabel*. Otherwise, leave this field blank.
Message	Identify the message to add to the property. You can enter values in this field in one of the following ways: • The third key part of a field value rule, where *@baseclass* is the first key part and the value specified in the **Category** field is the second key part. Use this approach when the **Category** field is not blank and the specified field

©2025 Pegasystems Inc

Parameter	Description
	value does not contain property references. Property references can lead to confusion because field values are evaluated each time the page is read, and property values can change over time. • The name of a *Rule-Message* instance without quotes. The list of available options in the SmartPrompt is limited to `Error` and blank category messages of the class of the specified Page. If the message supports parameters, append a tab-delimited string to the message name. • The literal text string within double quotes. If you using a field value (such as when a category is provided), you cannot specify a literal text string, since this results in a validation error when using a property reference or a parameter reference in the **Message** field . If your application supports localization, use one of the alternate options for this parameter.

©2025 Pegasystems Inc

Parameter	Description
Field	Identify the property on which the message is set.

Results

The message is added to the property and its parent pages. This usually prevents the page containing the message from being saved. Associating a message with a property does not directly cause that message to be displayed to a user.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Property-Set-Stream method

Use this method to save the contents of an HTML or XML stream, after stream processing completes, as the value of a `Single Value` property. The property value can then be mapped as output for transmission to an external system.

This method sets the results of stream processing on any stream rule, after all directives or JSP tags are processed, as the value of the property specified.

For example, if your application uses a Service Email rule (*Rule-Service-Email* rule type) to respond to an incoming email message, the response mapping can return an HTML-formatted email message containing clipboard values and HTML markup.

Parameters

This method has four parameters:

Parameter	Description
PropertyName	Enter a `Single Value` target property reference. Identify the property that to

©2025 Pegasystems Inc

Parameter	Description
	receive its value from stream processing. Typically, choose a property that has a `Text` type.
StreamClass	Identify the rule type of the stream, one of the following: • *Rule-Obj-Corr* • *Rule-Obj-HTML* • *Rule-Obj-XML* • *Rule-Corr-Fragment* • *Rule-HTML-Fragment*
StreamName	Enter the **Stream Name** key part of an HTML rule (Rule-Obj-HTML rule type) to be processed to produce the value of the property.
StreamType	Optional. If the **StreamClass** is *Rule-Obj-Corr* or *Rule-Obj-XML*, enter the third key part of the rule instance to be processed.

Results

This method forms a key to a rule of the rule type specified in the **StreamClass** parameter from the value you supply in the third and fourth parameters plus the class of the step page or of the activity. It then uses rule resolution to find the rule.

If the step page does not exist, a new page is created, of the class identified in the **Pages & Classes** tab.

Next, it performs stream processing to interpret and evaluate the JSP tags or HTML directives (including property references) in this rule. It stores the resulting XML or HTML text as the value of the property identified in the first parameter.

PEGA ©2025 Pegasystems Inc

If the Type of the expression doesn't match the Type of the destination, the value is converted following the same scheme as within expressions.

Any property messages previously associated with the property are cleared. The property value is validated against the property rule.

Cautions

- You cannot use this method for a property that is also the target property of a Declare Expression rule. You can't save the Activity form if such a conflict is detected.
- Do not attempt to set the value of a property that has a name starting with pz; changes to such properties are reserved.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Property-Set method

Use this method to set the value of one or more specified properties.

Parameters

The Property-Set method uses an array of parameters. Each row of the array contains a property name or reference (the destination), and a value (the source). Order is significant.

Parameter	Description
PropertiesName	Enter one or more target property references. If you precede the property with a period but no page name, the system expects this property to be on the step page. If the **Step Page** field for this step is blank, the system uses the primary page of the activity.

©2025 Pegasystems Inc

Parameter	Description
	NOTE: Don't use this method to set the value of a property that is the target property of a Declare Expression rule. You can't save the Activity form if such a conflict is detected. For example: .myProperty Primary.myProperty MyPage.myProperty For `Value List`, `Value Group`, `Page List`, and `Page Group` properties, enter a literal, property reference or symbolic index value (subscripts): myList(15) myChild(.Subscript) myGroup("Massachusetts") myGroup(<APPEND>) .myGroup(<LAST>) .myList(<INSERT>2) .myList(<PREPEND>)
PropertiesValue	Enter an expression to compute a value for each property. For example. you can enter: • A literal constant such as "Red" • A reference to the value of another property (*.pyLabel*)

©2025 Pegasystems Inc

Parameter	Description
	• A reference to the value of a parameter variable defined on the current activity's parameter page (`Param.ParameterName`) • An arithmetic operation (3.14159 * .Radius * .Radius) • A function call (@Year("20070430") that returns a non-void value. For more information on function syntax, see Methods for calling a function. • A reference to the value of a local variable defined on the parameter page (`Local.ParameterName`) • A linked property reference (.linkedproperty.instanceproperty)

Results

If the step page does not exist, the system creates a new page of the class identified in the **Pages & Classes** tab.

The system evaluates each expression identified in the **properties values** column of the parameter array and assigns the result to the property or property part you identified in the corresponding **PropertiesName** column.

Any property messages previously associated with the property are cleared.

If the Type of the expression doesn't match the Type of the destination, the value is converted following the same scheme as within expressions.

Each property value is validated against the property rule.

Rows are processed in sequence, so a property that is set in one row (as a destination) may be a source in a later row.

©2025 Pegasystems Inc

> **NOTE:** The standard function *PropertyExists()* can check whether a property is already present on the clipboard.

Cautions

- You cannot use this method for a property that is also the target property of a Declare Expression rule.
- Do not attempt to set the value of a property that has a name starting with pz; changes to such properties are reserved.
- Use the standard activity *Work-.UpdateStatus* — not this method — to change the value of a work item status (*pyStatusWork* property).

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Expressions
- Constants in Expressions

Property-Set-XML method

Use this method to save the contents of an XML Stream rule after stream processing completes as the value of a `Single Value` property. The property value can then be mapped as output and conveyed to an external system.

The target property value is the XML text from an XML Stream rule (*Rule-Obj-XML* rule type), after stream processing. Stream processing evaluates directives and clipboard references.

Parameters

This method has three parameters:

©2025 Pegasystems Inc

Parameter	Description
PropertyName	Identify a `Single Value` target property to receive the value produced by stream processing. If the property is on the step page, precede the property name with a period. If this property is on a different page, include the page name in the reference and ensure that this page appears on the **Pages & Classes** tab of the Activity form.
XMLStream	Enter the **Stream Name** key part of an XML Stream rule to process.
XMLType	Enter the third key part of an XML Stream rule to process.

Results

This method forms a key to an XML Stream rule instance from the value you supply for the second and third parameters and the class of the step page or of the activity. It uses rule resolution to find the XML Stream rule.

If the step page does not exist, a new page is created, of the class identified in the **Pages & Classes** tab.

It then uses stream processing to interpret and evaluate the JSP tags or directives in this rule. It stores the resulting XML text as the value of the property referenced in the first parameter. The property value is validated against the property rule.

©2025 Pegasystems Inc

Cautions

- You cannot use this method for a property that is also the target property of a Declare Expression rule. You can't save the Activity form if such a conflict is detected.
- Do not attempt to set the value of a property that has a name starting with pz; changes to such properties are reserved.
- Use the standard activity *Work-.UpdateStatus* — not this method — to change the value of a work item status (*pyStatusWork* property).

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Property-Validate method

Use the Property-Validate method in an activity to ensure that a property value meets certain requirements including:

- Checking performed by an edit validate rule (*Rule-Edit-Validate* rule type)
- A requirement that a property have a non-null value

You can also provide default values for properties.

If your planned activity can consist entirely of Property-Validate methods, you don't need to create an activity. Create a validate rule (*Rule-Obj-Validate*) instead.

Parameters

This method has an array of parameters, seven per row.

Parameter	Description
PropertyName	Enter a property reference, identifying the property to be validated.

©2025 Pegasystems Inc

Parameter	Description
ValidateAs	Optional. Identify a literal key to an edit validate rule, or enter an expression that returns true or false when applied to the property. If left blank, the edit validate rule referenced in the property itself (if any) is used.
Continue	Select the box to cause the processing of this method to continue after a validation error. Clear the box to cause this method to stop after one validation error is found.
Default	Optional. Provide a value to set if the property identified in the **PropertyName** column of this row has no value.
Required	Select the box to require that the property has a non-blank value. If selected and the property value is blank, the system associates the message `This field may not be blank.` with the property. You cannot override this message text. The value in the **ErrorMessage** parameter does not override this message.
ErrorMessage	Optional. Enter a text string for the message created if the rule defined in the **ValidateAs** parameter fails. If you leave this blank, the system uses a generic message.

©2025 Pegasystems Inc

Parameter	Description
	This text is not used when the **Required** box is selected.
OnlyIfChanged	Leave cleared. Not used.

Results

The system validates the properties identified in the first column one by one, starting with the first row. It stops when all rows are evaluated, or when a property fails validation and the **Continue** check box for that row is not selected.

A property fails validation if the edit validate rule returns false, or if the **Required** parameter is selected but the property is missing or has a blank (null) value. When a property fails validation, the system adds a message to the clipboard page. Ordinarily, pages that contain messages (indicating errors) are not saved into the PegaRULES database.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Property-Set-Corr method

Use this method to save the contents of a correspondence stream (generally HTML text) as the value of a `Single Value` property. The property value can then be mapped as output for transmission to an external system.

The target property value is the contents of a correspondence rule (*Rule-Obj-Corr* rule type), after stream processing. Stream processing evaluates directives and clipboard references.

©2025 Pegasystems Inc

Don't attempt to set the value of a property that has a name starting with pz; these properties are reserved. Don't use this method for a property that is the target property of a Declare Expression rule.

Parameters

This method has three parameters:

Parameter	Description
PropertyName	Enter a `Single Value` target property reference. Identify the property that receives its value from stream processing. Typically, choose a property that has a `Text` type.
corr stream	Enter the second key part — the Correspondence Name — of a correspondence rule.
CorrType	Enter a correspondence type that is the final key part of a correspondence rule.

Results

This method forms a key to a correspondence rule from the value you supply for the second and third parameters and the class of the step page or of the activity. It uses rule resolution to find the correspondence rule.

If the step page does not exist, a new page is created, of the class identified in the **Pages & Classes** tab.

It then applies stream processing to interpret and evaluate the JSP tags or directives (including property references) in this rule. It stores the resulting text as the value of the property referenced in the **PropertyName** parameter.

If the Type of the expression doesn't match the Type of the destination, the value is converted following the same scheme as within expressions.

PEGA ©2025 Pegasystems Inc

Any property messages previously associated with the property are cleared. The property value is validated against the property.

Cautions

- You can't use this method for a property that is also the target property of a Declare Expression rule. You can't save the Activity form if such a conflict is detected.
- Don't attempt to set the value of a property that has a name starting with pz; changes to such properties are reserved.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Publish-Notifications

Use the Publish-Notifications method to publish the top-level scalar and value list properties of a clipboard page to a notification channel.

Parameters

Parameter	Description
ChannelName	Enter the name of a notification channel as displayed on the Notifications Channel landing page.

Results

When the *Publish-Notifications()* is called with a channel name parameter, all the top-level scalar and value list parameters of the step page are published to the specified channel. If the channel was configured to receive a specific set of properties, only those properties are published.

©2025 Pegasystems Inc

In the absence of the channel name parameter, this method publishes the entire step page to any channel that is based on the same class as the step page.

This method logs an error if the publish operation fails.

Checking the method status

This method updates the *pxMethodStatus* property.

For more information on how to use this method and real-time notifications, see the *Real-time applications and push notifications in Pega Platform* article on Pega Community.

- Testing method results using a transition

RDB-Delete method

Use this method to delete a row or rows from an external relational database using SQL. This method operates in conjunction with an SQL statement in the **Delete** tab of a Connect SQL rule (Rule-Connect-SQL rule type) that contains the `DELETE`, `TRUNCATE` or `DROP` SQL statement.

On the **Delete** tab of the Connect SQL rule, use the special data syntax described in Connect SQL rules — Data mapping.

Use Connect SQL rules and the RDB-Delete method only with an external database. Do not use Connect SQL rules or RDB- methods to update the PegaRULES database. Because not all properties in the Pega Platform databases are exposed as distinct database columns, use the Obj-Open and Obj-Save methods, not the similar RDB- methods, with the PegaRULES database to prevent loss of data.

Parameters

This method has four parameters. Three required parameters together identify a Connect SQL rule. The final parameter controls a processing option.

©2025 Pegasystems Inc

Parameter	Description
ClassName	Identify the class of the step page. The system uses this value as the **Applies To** key part of the Connect SQL rule that defines the delete. **NOTE:** Identify the exact class of the rule. The system does not use class inheritance to find the Connect SQL rule.
RequestType	Enter the third key part — **Request Type** — of the Connect SQL rule containing the delete operation.
Access	Enter the second key part — **Package Name** — of the Connect SQL rule (by convention `MSSQL`, `Oracle`, `Db2`, `UDB`, or `Sybase`, but not restricted to these values).
RunInParallel	Select if you want processing for the connector to continue in a separate requestor, in parallel with the current requestor. Clear if you want processing in this requestor session to pause until the connector processing completes. If selected, use the Connect-Wait method later in the current or a later activity to access results from the connector.

©2025 Pegasystems Inc

Results

This method first retrieves the Connect SQL rule identified by the first three parameters, using rule resolution.

It accesses the SQL text in the **Delete** tab of that rule and converts any property, class, or table references in the context of the current clipboard.

It sends the resulting SQL to the external database, which performs the deletions.

When the **RunInParallel** box is selected, the SQL operation occurs in a new requestor session. This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.

Each RDB-Open, RDB-Save, and RDB-Commit method involves an SQL database commit operation. You cannot implement SQL transaction commit operations across multiple Connect SQL rules.

Checking the method status

This method updates the *pxMethodStatus* property.

When a {SQLPage:pagename} directive appears as the first line of the SQL code on the **Delete** tab, a top-level page named pagename is created. If errors occur, properties on that page contain additional error details. (When **RunInParallel** is selected, this page is not available to the original requestor.)

RDB-List method

Use this method to retrieve rows from an external relational database and place the results as embedded pages in a specified step page of class *Code-Pega-List*.

This method references a Connect SQL rule instance, and executes SQL statements stored in the **Browse** tab of that rule instance. The search can do anything you can

©2025 Pegasystems Inc

specify in a SQL statement, such as a `SELECT WHERE` statement. Any constraints on the returned data are in the SQL.

On the **Browse** tab of the Connect SQL rule, use the special data syntax described in Connect SQL rules — Data mapping.

In all but rare cases, use Connect SQL rules only with an external database. Do not use Connect SQL rules or RDB methods for the PegaRULES database(s). Because not all properties in the PegaRULES databases are distinct database columns, use the Obj-Open and Obj-Save methods, not the RDB- methods, with the PegaRULES database to prevent loss of data.

Specify a page of class *Code-Pega-List* as the step page of the RDB-List method. Also identify this page and class on the **Pages & Classes** tab of the Activity form.

Parameters

This method has several parameters. The first three parameters are required; together they identify the key of a Connect SQL rule.

Parameter	Description
RequestType	Enter the third key part — **Request Type** — of the Connect SQL rule containing the search operation.
Access	Enter the second key part — **Package Name** — of the Connect SQL rule (by convention `MSSQL`, `Oracle`, `Db2`, `UDB`, or `Sybase`, but not restricted to these values).
ClassName	Enter a class name, the **Applies To** key part of a Rule-Connect-SQL rule. Identify the exact class of the rule. The system does not use class inheritance to find

©2025 Pegasystems Inc

Parameter	Description
	the *Rule-Connect-SQL* rule instance. This is also the class of the `pxResults` embedded pages which hold the search results. It can be a class that you created to hold the results. This class is distinct from the class which the RDB-List method searches, which you identify in the SQL code in the *Rule-Connect-SQL* rule.
MaxRecords	Optional. Enter a positive integer that is the maximum number of rows to return in the search results.
BrowsePage	Optional. Identify the top-level page into which the search results are to be returned. The `pxResults` pages are embedded in this page. If left blank, the system uses the step page you specified for the RDB-List method in the activity step. If you select this option, do not select the **RunInParallel** option. Parallel operations cannot update the single top-level browse page.
ApplyDeclaratives	Select to cause change tracking for Declare Expression and Declare Constraint rules processing to be applied to the properties returned by this method.

©2025 Pegasystems Inc

Parameter	Description
	You cannot select this box if the **RunInParallel** box is selected.
RunInParallel	Select if you want processing for the connector to continue in a separate requestor, in parallel with the current requestor. When selected, use the Connect-Wait method later in the current or a later activity to access results from the connector. Clear if you want processing in this requestor session to pause until the connector processing completes. You cannot select this box if the **ApplyDeclaratives** box is selected. Do not select this option if you supplied a page name in the BrowsePage field.

Results

This method accesses the *Rule-Connect-SQL* rule instance identified by the first three parameters, using rule resolution. It converts table, class, and property references on the **Browse** tab of this rule using clipboard data, and sends the resulting updated SQL to the external database.

The RDB-List method returns data to the page you specify in the **Step Page** field on the activity form. This can be a previously created page — which is cleared and reused — or a new page created with this method.

©2025 Pegasystems Inc

When the **RunInParallel** box is selected, the SQL operation occurs in a new requestor session. This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.

When the system receives information from the external database, it updates properties on the step page, which has the class *Code-Pega-List*.

If the search is successful, this method also creates a page list page named `pxResults`, which contains pages pxResults(1), pxResults(2), and so on. These embedded pages contain the row results as properties. Depending on the **ApplyDeclaratives** box, the declarative processing may be activated for the properties.

Finally, this method can create a (classless) error page, if specified in the Connect SQL rule.

Properties of Code-Pega-List

Debugging SQL and SQL performance

You can use the DB Trace feature of the Performance tool to view in depth the SQL operations of a Connect SQL rule.

Execution of this method may cause an alert to be added to the Alert log, of type `PEGA004` - `PEGA007`, `PEGA0025`, or `PEGA0028`. Review the Alert log to understand the frequency and sources of such alerts, and for best performance, alter your application or database to eliminate those alerts that occur frequently.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

When a {SQLPage:pagename} directive appears as the first line of the SQL code on the **Browse** tab, a top-level page named pagename is created. If errors occur, properties on

©2025 Pegasystems Inc

that page contain additional error details. (When **RunInParallel** is selected, this page is not available to the original requestor.)

- About Connect SQL rules
- Connect-Wait method
- Connect SQL form - Data Mapping
- Tracking utilization statistics with Performance Analyzer

RDB-Open method

Use this method to retrieve a single row (record) of data from an external relational database and add the retrieved data into a specified clipboard page as property names and values.

Use this method in conjunction with a Connect SQL rule that contains SQL `SELECT` or `EXECUTE` statements in the **Open** tab. Define the SQL statements so that the database returns exactly one row. (To search the external database with criteria that may sometimes find no rows, or find more than one row, use the RDB-Browse method, not the RDB-Open method.)

On the **Open** tab of the Connect SQL rule, use the special data syntax described in Connect SQL rules — Data mapping.

Except in rare cases, identify in the **Step Page** field the name of the page to be created by this method. If the **Step Page** field of the step is blank, this method clears the primary page of the current activity and may change its class. Typically, this is not desirable.

Use Connect SQL rules and the RDB-Open method only with an external database. Do not use Connect SQL rules or RDB methods to updated the PegaRULES database(s). Because not all properties in the Pega Platform databases are distinct database columns, use the Obj-Open and Obj-Save methods, not the RDB- methods, with the PegaRULES database to prevent loss of data.

PEGA ©2025 Pegasystems Inc

Parameters

This method has four parameters. Three required parameters together identify a Connect SQL rule. The final parameter controls a processing option.

Parameter	Description
OpenClass	Enter the **Applies To** key part of the Connect SQL rule that defines the open SQL. This must match the class of the step page. Identify the exact class of the rule. The system does not use class inheritance to find the *Rule-Connect-SQL* rule instance.
RequestType	Enter the third key part — **Request Type** — of the Connect SQL rule containing the open operation.
Access	Enter the second key part — **Package Name** — of the Connect SQL rule (by convention `MSSQL`, `Oracle`, `Db2`, `UDB`, or `Sybase`, but you are not restricted to these values).
RunInParallel	Select if you want processing for the connector to continue in a separate requestor, in parallel with the current requestor. When selected, use the Connect-Wait method later in the current or a later activity to access results from the connector. Clear if you want processing in the current requestor session to wait until the connector processing completes.

©2025 Pegasystems Inc

Results

This method uses the three required parameters and rule resolution to retrieve a Connect SQL rule.

Next, the system processes the SQL text on the **Open** tab of the Connect RDB rule, performing any keyword evaluations and clipboard substitutions. It then sends the resulting SQL text to the external database. When the Pega Platform receives the results from the external database, it places the results into the single clipboard page identified in the **Step Page** column of the step.

The system chooses the class of the results from the value of the *pxObjClass* property, if any returned from the database. In the common case that the RDB-Open method is used with an external database (rather than the PegaRULES database), the opened instance has no *pxObjClass* property: the system then determines a concrete class for its results from either:

- The **Applies To** key part of this Connect SQL rule, or
- `CLASS` instructions within the SQL statement. If the SQL statement contains more than one CLASS directive, the most specific concrete class is used.

If the step page exists, the system changes the class of the step page to the class determined by the above algorithm. If the step page does not exist, the system creates a new page with a that class.

If the external database request returns no rows, or more than one row, the step page is not updated and the method returns a fail status.

When the **RunInParallel** box is selected, the SQL operation occurs in a new requestor session. This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.

PEGA ©2025 Pegasystems Inc

> ⚠ **CAUTION:** Each RDB-Open, RDB-Save, and RDB-Delete method involves an SQL database commit operation. You cannot implement SQL transaction commit operations spanning two or more Connect SQL rules.

Debugging SQL and SQL performance

You can use the DB Trace feature of the Performance tool to view in depth the SQL operations of a Connect SQL rule.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

When a {SQLPage:pagename} directive appears as the first line of the SQL code on the **Open** tab, a top-level page named pagename is created. If errors occur, properties on that page contain additional error details. (When **RunInParallel** is selected, this page is not available to the original requestor.)

Restrictions

This method cannot be used in a step that involves iteration.

- About Connect SQL rules
- Connect-Wait method
- Connect SQL form - Data Mapping
- Tracking utilization statistics with Performance Analyzer

RDB-Save method

Use this method to save the contents of a clipboard page into a row of a relational database. The system saves the properties on the specified step page to the specified table in the database.

PEGA ©2025 Pegasystems Inc

This method operates in conjunction with a Connect SQL rule that contains SQL statements such as `INSERT`, `UPDATE`, and `CREATE` statements on the **Save** tab.

On the **Save** tab of the Connect SQL rule, use the special data syntax described in Connect SQL rules — Data mapping.

Use Connect SQL rules and the RDB-Save method only with an external database. Do not use Connect SQL rules or RDB methods to update the PegaRULES database. Because not all properties in the Pega Platform databases are distinct database columns, use the Obj-Open and Obj-Save methods, not the RDB- methods, with the PegaRULES database to prevent loss of data.

Parameters

This method has four parameters. Three required parameters together identify a Connect SQL rule. The final parameter controls a processing option.

Parameter	Description
ClassName	Enter the class of the step page. The system uses this as the **Applies To** key part of a *Rule-Connect-SQL* rule. **NOTE:** Identify the exact class of the rule. The system does not use class inheritance to find the *Rule-Connect-SQL* instance.
RequestType	Enter the third key part — **Request Type** — of the Connect SQL rule containing the save operation.
Access	Enter the second key part — **Package Name** — of the Connect SQL rule (by convention

©2025 Pegasystems Inc

Parameter	Description
	`MSSQL`, `Oracle`, `Db2`, `UDB`, or `Sybase`, but not restricted to these values).
RunInParallel	Select if you want processing for the connector to continue in a separate requestor session, in parallel with the current requestor session. When selected, use the Connect-Wait method later in the current or a later activity to access results from the connector. Clear if you want processing in this requestor session to pause until the connector processing completes.

Results

This method uses rule resolution to find and open the *Rule-Connect-SQL* instance identified by the three parameters. It then processes the SQL text in the **Save** tab using current clipboard values. Finally, it sends the converted SQL to the external database, which performs the Save.

When the **RunInParallel** box is selected, the SQL operation occurs in a new requestor session. This temporary requestor has the same access group and clipboard as the original. At most 10 such child requestor sessions can be active at once.

Each RDB-Open, RDB-Save, and RDB-Commit method involves an SQL database commit operation. You cannot implement SQL transaction commit operations across two or more Connect SQL rules.

PEGA ©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

When a {SQLPage:pagename} directive appears as the first line of the SQL code on the **Save** tab, a top-level page named pagename is created. If errors occur, properties on that page contain additional error details. (When **RunInParallel** is selected, this page is not available to the original requestor unless the dynamic system setting `isPropageteExceptionFromChildToParent` is added to the Pega-RULES ruleset and set to `true`. For more information, see Configuring dynamic system settings.)

- About Connect SQL rules
- Connect-Wait method
- Connect SQL form - Data Mapping
- Tracking utilization statistics with Performance Analyzer

Requestor-Stop method

Use this method to halt processing of the requestor and log off the requestor. This method operates on the `pxRequestor` page.

To log off browser-based users, a preferred approach is to call the standard activity *Code-Security.LogOff* with parameter `pzAuth` set to `guest`. Your application does not need to use the Requestor-Stop method directly except in unusual cases.

Parameters

This method uses no parameters.

Results

When all processing associated with the current request is complete, the system checks the property *pxRequestor.pxStopPending*, and takes appropriate action. The Requestor-Stop method takes action after the current client input message is completely

©2025 Pegasystems Inc

processed (so as not to delete the client connection out from under the running activity). If a user tries to send input again, the user receives an HTML display that indicates the connection is no longer valid.

Checking the method status

This method is always successful. You can't check for errors from this method.

Rollback method

Use this method to cancel or withdraw any previous uncommitted changes to the PegaRULES database (and to external databases accessed from an external class) from the current Thread. All pending Obj-Save and Obj-Delete methods are canceled.

Parameters

This method has no parameters.

Results

All uncommitted database operations are withdrawn. If an instance is locked and the **ReleaseOnCommit** box was selected, the lock is released. Locks on instances where the **ReleaseOnCommit** box was not selected are retained.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Commit method
- Obj-Save-Cancel method

Show-Applet method

Do not use this deprecated method. It is replaced by the Show-Page method.

©2025 Pegasystems Inc

See Show-Page method.

Show-Applet-Data method

Do not use this deprecated method. It is replaced by the Show-Property method.

See Show-Property method.

Show-HTML method

Use the Show-HTML method to cause the activity to process an HTML rule and send the resulting HTML to a user for display by a browser. Known as stream processing. this may involve the interpretation of JSP tags (or the older directives), which can access the clipboard to obtain property values, or can insert other HTML rules, and so on.

> **NOTE:** Use the *@baseclass.Show-Harness* activity to display a user form defined by a harness.

Parameters

This method has three parameters:

Parameter	Description
HTMLStream	Identify the **Stream Name** key part of an HTML rule (*Rule-Obj-HTML* rule type) to be processed. The system uses the class of the step page as the **Applies To** key part of the HTML rule.
HTMLFrame	Optional. Leave blank in most situations. If you use the `transid` option in a URL JSP tag, the system uses the frame name to verify the transaction ID parameter

©2025 Pegasystems Inc

Parameter	Description
	`pzTransactionID`. If you enter a value for HTMLFrame, use both the `transid` and the `frame` options in the URL JSP tag. (As a best practice, use the name of the primary page as the frame name.) Technically, this frame name is only an assertion to the server engine as to which HTML frame will receive the output, to enable a later redisplay if required. The run-time destination frame is controlled through JavaScript routines in the portal and may differ from this value. When designing and implementing multiThread interactions for a browser-based user session, it is a best practice to have each Thread that uses the Show-HTML method send its HTML to a specific named HTML frame on the browser window, to avoid confusion.
ReadOnly	Select this check box to cause the system to present any HTML `INPUT` elements on any form as display-only, so that the user receiving the HTML display cannot enter or update data. Clear this check box to allow user input, the normal situation.

©2025 Pegasystems Inc

Results

The system uses rule resolution to access the HTML rule referenced in the first parameter. It uses stream processing to evaluate and process all the JSP tags (or directives) it contains. It sends the resulting processed HTML code using HTTP or HTTPS protocol to the user who ran the activity.

The system stores the processed HTML as the value of the *pxThread.pxHTMLStream* property on the Thread page.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

NOTE:

- You can't use the Show-HTML method in an activity that has an **Activity Type** (on the **Security** tab) of `Utility`. Utility shapes in a flow do not produce any user interaction.
- The Show-Page, Show-Stream, Show-Property and Show-HTML methods each send an HTTP message corresponding to an entire HTML document to the user's browser. They can't be used in combination to assemble a document. Use Show-Page and Show-Property only for debugging.
- To review the HTTP messages produced by the Show-HTML method, install an HTTP debugger such as Fiddler, an open-source Windows-based proxy (or "sniffer" that logs all HTTP traffic between your computer and the Internet.

Show-Page method

Use the Show-Page method to send an XML representation of the step page to a user's browser session, typically as an aid to debugging.

©2025 Pegasystems Inc

Parameters

This method uses no parameters.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Restrictions

This method cannot be used in a step that involves iteration.

> **NOTE:**
>
> The Show-Page, Show-Stream, Show-Property and Show-HTML methods each send an HTTP message corresponding to an entire document to the user's browser. They can't be used in combination to assemble a document.
>
> Use Show-Page in an activity that supports the Autocomplete control, or for debugging

Show-Property method

Use the Show-Property method to send the value of a single scalar property on the clipboard page identified in the **Step Page** column to a user's browser session, as an aid to debugging.

Parameters

This method has one required parameter.

©2025 Pegasystems Inc

Parameter	Description
Property	Enter an expression, such as a property name or reference to a `Single Value` property, or a linked property reference. The property must not be a Java Object.

Results

The system sends the value of the expression or property to the user's browser.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

> **NOTE:** The Show-Page, Show-Stream, Show-Property and Show-HTML methods each send an HTTP message corresponding to an entire document to the user's browser. They can't be used in combination to assemble a document. Use Show-Property only for debugging.

Show-Stream method

Use the Show-Stream method to apply stream processing to an HTML rule, correspondence rule, or XML Stream rule, and send the results to a browser-based requestor session.

Parameters

This method has five parameters:

Parameter	Description
StreamClass	Optional. Identify the class of a rule type derived from the *Rule-Stream* class. such as:

©2025 Pegasystems Inc

Parameter	Description
	• *Rule-Obj-Corr* • *Rule-Obj-HTML* • *Rule-Obj-XML* If blank, the default value is *Rule-Obj-HTML*. Do not use this method with list view or summary view rules, which are deprecated.
StreamName	Enter the second key part of the rule to be processed. To locate the rule using rule resolution, the system uses the class of the of the step page as the **Applies To** class of the rule.
StreamType	Optional. If you selected *Rule-Obj-Corr* or *Rule-Obj-XML* as the **StreamClass** value, enter here the third key part of the rule (the **Correspondence Type** or **XML Type**).
FrameName	Optional. Leave blank in most situations. If you use the `T` (or `TRANSID`) option in a URL directive, the system uses the frame name to verify the transaction ID. If you enter a value for **HTMLFrame**, use both the `T` and the `F` options in the URL directive. (As a best practice, use the name of the primary page as the frame name.) When designing and implementing multiThread interactions for a browser-based user session, it is a best practice to have each Thread that uses the Show-Stream

©2025 Pegasystems Inc

Parameter	Description
	method send its HTML to a specific, named HTML frame on the browser window, to avoid confusion.
Read-only	Select this box to cause the system to present any HTML `INPUT` elements on any form as display-only, so that the user receiving the HTML display cannot enter or update data.

Results

The system uses the class of the step page (or of the primary page) of the activity with rule resolution to find and access the rule referenced.

It uses stream processing to evaluate and process all the JSP tags that the rule instance contains. It sends the resulting processed HTML by HTTP or HTTPS protocol to the user who ran the activity.

The system stores the processed HTML as the value of the *pxHTMLStream* property on the `pxThread` page.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

NOTE:

- The Show-Page, Show-Stream, Show-Property and Show-HTML methods each send an HTTP message corresponding to an entire document to the user's browser. They can't be used in combination to assemble a document. Use Show-Page and Show-Property primarily for debugging.

©2025 Pegasystems Inc

- To review the HTTP messages produced by the Show-Stream method, install an HTTP debugger such as Fiddler, an open-source Windows-based proxy that logs all HTTP traffic between your computer and the Internet.

Start-Validate method

Use this method only when building custom rule types, to start the internal processing that develops rule references. Use this method as the first step in an activity of **Activity Type** `Validate` when the class of the activity is a *Rule-* class and rule referencing is to be available.

Parameters

This method uses no parameters.

Results

Starts internal processing to develop a list of other rules that reference this rule.

- End-Validate method

StringBuffer-Append method

Use this method to manipulate local variables of type `String Buffer` for the current activity that are contained in a Java StringBuffer object. This method appends a text string to the current contents of the buffer.

Local variables are defined on the **Parameters** tab of the Activity form.

Parameters

This method accepts an array of one or more variable names and values.

©2025 Pegasystems Inc

Parameter	Description
Buffer	Identify the name of a local variable.
Value	Enter an expression (such as a literal value, a property reference, a linked property reference, or a function call) the produces a text value to append.

Results

The text string entered in the **Value** column is appended to any existing text in the StringBuffer, and associated with the name you supplied in the **Buffer** column.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Defining the local variables for an activity
- StringBuffer-Insert method
- StringBuffer-Reset method

StringBuffer-Insert method

Use this method to manipulate local variables of type `String Buffer` for the current activity that are stored in a Java StringBuffer object. This method inserts a string into the buffer at a specific byte offset position.

Local variables are defined on the **Parameters** tab of an activity.

Parameters

This method has the following parameters:

©2025 Pegasystems Inc

Parameter	Description
Buffer	Enter the name of a local variable to be stored in the StringBuffer.
Offset	Indicate the index (offset) position within the buffer. Enter an integer constant, or an expression that evaluates to an integer value.
StrInsert	Enter an expression (such as a literal value, a property reference, a linked property reference, or a function call) the produces a text value to append.

Results

The system inserts the text into the `StringBuffer`, starting at the indicated byte position, and associates it with the named local variable.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Defining the local variables for an activity
- StringBuffer-Append method
- StringBuffer-Reset method

StringBuffer-Reset method

Use this method to clear a local variable of type `String Buffer` of an activity, setting its value in the Java StringBuffer to the null string. Local variables are defined on the **Parameters** tab of the activity form.

©2025 Pegasystems Inc

Parameters

This method has one required parameter.

Parameter	Description
Buffer	Identify the local variable to be reset.

Results

The system updates the Java StringBuffer object to associate the empty string with the named local variable.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

- Defining the local variables for an activity
- StringBuffer-Append method
- StringBuffer-Insert method

TaskStatus-Set method

Use this method in activities with an **Activity Type** of `Utility` that return a value that a flow rule can examine to make decisions. The system can test the output of this method to determine which connector to take.

For example, a utility named Direction could analyze geographic information and return one of four values, North, South, East,. and West. Four connector lines leading from this utility shape in the flow diagram can correspond to these four values.

Do not confuse the output of this method (the property *Code-Pega-Thread.pxTaskStatus* on the `pxThread` page, returned to a flow, to be tested by connectors) with the method status (a property on the `pxResults` page indicating whether method execution was successful). All but a few methods set the method status.

©2025 Pegasystems Inc

The value you set through this method remains until the next execution of the TaskStatus-Set, or a decision shape is processed, or the user completes a flow action. Advancing to another utility shape or testing the value does not clear the value.

Parameters

This method has two parameters:

Parameter	Description
TaskStatus	Enter a literal text value to be returned to in the flow instance following the activity. The system sets this value in the *pxTaskStatus* property.
TaskStatusInfo	Optional. Enter a short explanation for the status.

Results

This method sets the text value supplied in the **TaskStatus** parameter into the *pxTaskStatus* property on the pxThread page. This property value is arbitrary and unedited. The flow can examine this value to make a decision.

Also, this method sets the text value supplied in the **TaskStatusInfo** parameter into the *pxTaskStatusInfo* property.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Text-Normalize method

Use this method to apply a parse normalize rule (*Rule-Parse-Normalize* rule type) to convert a source text string to a normalized form.

©2025 Pegasystems Inc

Parameters

The Text-Normalize method has three parameters:

Parameter	Description
Name	Enter the second key party of a parse normalize rule. The system uses the class of the step page as the first key part.
Source	Enter a property reference or linked property reference containing the source text to be normalized.
Target	Optional. Enter a property reference to receive the resulting output text. If blank, the results are stored in the property identified in the **Source** parameter.

Results

The system uses rule resolution to locate a parse normalize rule, using the class of the step page and the **Name** parameter as key parts.

Next, it applies the normalize rule to the text in the **Source** value and stores the normalized result in the **Target** property value. If the **Target** parameter is not blank and does not match the **Source** parameter, the property in the **Source** parameter is not altered.

Execution of the rule may make no changes, one change, or many changes. Matching occurs only against the input source text, not recursively against intermediate results.

The method fails if the rule, the **Source** property, or the **Target** property cannot be found.

PEGA ©2025 Pegasystems Inc

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

Thread-Clear method

Use this method to clear the Thread pages for your current session, except for any pages specified in the **KeepPages** parameter.

Parameters

The Thread-Clear method has a single fixed parameter and an array parameter.

Parameter	Description
EmptyTaskLists	Optional. To clear the three activity lists (Activity, Worklist, and Current), set this parameter to `Y`. If blank, the lists are not cleared.
KeepPages	Optional. Enter any page names to retain.

Results

If the **EmptyTaskLists** parameter is selected, all three activity lists are cleared.

Thread pages are deleted, except the `pxThread` page and those pages listed explicitly in the second parameter array.

After this method deletes the pages, it continues with the next activity step.

Checking the method status

This method updates the *pxMethodStatus* property only if it fails. See How to test method results using a transition.

©2025 Pegasystems Inc

Thread-Stop method

This method is deprecated for new development.

Use this method to delete the pages of the current requestor, and optionally log off the requestor.

To log off browser-based users, a preferred approach is to call the standard activity *Code-Security.LogOff*, with parameter `pzAuth` set to `guest`. Your application does not need to use the Thread-Stop method directly.

Parameters

The Thread-Stop method has one parameter:

Parameter	Description
SignoffRequestor	If set to `Y`, then when this method executes, the system logs the requestor off after stopping this thread, by stopping all other threads for the requestor and clearing the requestor information. Default is `N`.

Results

All user pages for the current Thread are deleted. If **SignoffRequestor** is set to Y, then the requestor (and hence all other Threads) are deleted after processing of the current user interaction completes. The `pxThread` page is not deleted.

Checking the method status

No status is returned.

- Requestor-Stop method

©2025 Pegasystems Inc

Wait method

Use this method to pause a thread for a specified time interval. This method causes the current thread to wait a specified time interval (at least) before continuing with activity processing. (This method performs a Java *sleep()*.)

Use this method only when truly necessary and for short intervals, as system resources are consumed during the wait. Do not create activities that may loop indefinitely on a wait condition.

Parameters

The Wait method has two optional parameters. Enter a value for at least one of them. If you specify both parameters, the system adds the two parameters together so the resulting wait interval is WaitSeconds plus WaitMillSeconds.

Parameter	Description
WaitSeconds	Enter a length of time to wait in seconds as an integer value, or a numeric expression or property reference.
WaitMillSeconds	Enter a length of time to wait in milliseconds as an integer value from 0 to 999, or a numeric expression or property reference.

Results

This method causes both the system and the client to wait for the specified number of seconds and milliseconds.

Checking the method status

This method updates the *pxMethodStatus* property. See How to test method results using a transition.

PEGA ©2025 Pegasystems Inc

Wait-For-Requestor method

This method is deprecated. See the equivalent method Connect-Wait.

Rules

This book contains reference topics for rules in Pega Platform.

- **About Connect BPEL rules**
- **About Connect Robot rules**
- **About Calendar data instances**
- **Divisions**
- **Organization data instances**
- **Organization Unit data instances**
- **Summary views**
- **OAuth 1.0 clients**
- **Web Service Security profile**
- **Flow markers**
- **About SpellChecker Properties data instances**
- **Word templates**
- **Service Java rules (to be deprecated)**
- **Service JSR94 rules (to be deprecated)**
- **Service MQ rules**
- **Creating an SMS account**

PEGA ©2025 Pegasystems Inc

- Creating a web service policy data instance
- How to call through Java-based services
- Registry Java Property Transform data instances - Completing the Save As form
- About Broadcast data instances
- Circumstance definitions
- Circumstance templates
- Function Rule
- Function Alias Rules
- Library rules
- Messages
- Property aliases
- Property Qualifier rules
- About Static Content Bundle rules
- Text File rules
- Defining URL patterns for Cases
- Function Alias rules — Completing the Create, Save As, or Specialization form
- About Image Content data instances
- Integration Connectors landing page
- Integration Services landing page

PEGA ©2025 Pegasystems Inc

- Connected apps landing page
- About Collection Rules
- About Connect Cassandra rules
- About Connect CMIS rules
- About EJB connector rules (deprecated)
- About Connect HBase rules
- About Connect Java rules
- About Connect SQL rules
- Atom Servers
- Connect Request Processor data instances
- About SOAP Server data instances
- Web service policy data instances
- Map eForm rules
- About Parse Normalize rules
- Converting from directives to JavaServer Pages tags
- Toggling features on and off
- Class Management landing page
- List rules
- About Edit Validate rules
- Properties

PEGA ©2025 Pegasystems Inc

- **JavaServer Pages tags**
- **Fields that support the Global Resource Settings syntax**
- **Integration-Connectors category**

About Connect BPEL rules

The following tabs are available on this form:

- Methods
- Deployment

> **NOTE:** The rule type Rule-Connect-BPEL is deprecated. Use Service SOAP rules instead.

The Business Process Execution Language (BPEL) integration interface enables you to define your Pega Platform system as one partner in a BPEL process. The BPEL integration interface is layered on top of the SOAP integration interface. You use SOAP service and connector rules to define the operations and a BPEL service or connector to package them as a BPEL Web service.

- Package the BPEL Web service as a Connect BPEL rule when your Pega Platform system starts the BPEL process.
- Package the BPEL Web service as a Service BPEL rule when an external partner starts the BPEL process, sending a request to your Pega Platform system.

Access

Use the Application Explorer or **Configure** > **Integration** > **Connector Definitions & Simulations** to list the Connect BPEL rules in your application.

Select **Configure** > **Integration** > **Connectors** > Connector and Metadata wizard to start the Connector and Metadata wizard to create a Connect BPEL rule.

©2025 Pegasystems Inc

Use the Records Explorer to list all Connect BPEL rules that are available to you.

Category

Connect BPEL rules are instances of the *Rule-Connect-BPEL* class. They belong to the Integration-Connectors category.

- **Connect BPEL form – Completing the Methods tab**
- **Connect BPEL form - Completing the Deployment tab**

Connect BPEL form – Completing the Methods tab

When a BPEL request is made, the request is sent to a specific instance of the BPEL process. When the BPEL interaction is asynchronous, the process must use correlation property values to match responses to request. Use the Methods tab to specify how replies are matched to requests by defining a correlation property.

> **NOTE:** The rule type Rule-Connect-BPEL is deprecated. Use Service SOAP rules instead.

Field	Description
Connection	
Asynchronous	Select to make this connection asynchronous.
Correlation Property	Use this section to configure the correlation properties for the BPEL connector rule.
Name	Enter a name for the correlation parameter.
Type	Select the data type of the parameter from the list.

©2025 Pegasystems Inc

Field	Description
Services	This section lists all the SOAP service rules for which the service package and service class match the values specified in the service package and service class fields for this BPEL connector rule.
Method	The third key part (Service Method) of the name of the SOAP service rule.
RuleSet	The RuleSet the SOAP service rule was created in.
Version	The version of the RuleSet the SOAP service rule was created in.
Correlation Parameter	Select the property that represents the correlation property for this SOAP service rule. The list contains the properties on the SOAP service rule's Request tab with a data type compatible with the one you selected in the Correlation Property section.
Connects	This section lists all the SOAP connector rules with a Namespace value that matches the value you specified in the Namespace field for this BPEL connector rule.
Method	The third key part (Request Type) of a SOAP connector rule.
Applies To	The class rule that the SOAP connector rule applies to.
RuleSet	The RuleSet the SOAP connector rule was created in.

©2025 Pegasystems Inc

Field	Description
Version	The version of the RuleSet the SOAP connector rule was created in.
Correlation Parameter	Select the property that represents the correlation property for this SOAP connector rule. The list contains the properties on the SOAP connector rule's Request tab with a data type compatible with the one you selected in the Correlation Property section.

Connect BPEL form - Completing the Deployment tab

> **NOTE:** The rule type Rule-Connect-BPEL is deprecated. Use Service SOAP rules instead.

Use the **Deployment** tab to generate WSDL files that you can use to help design the external parts of the BPEL process. Be sure to configure the correlation property on the Methods tab and save the BPEL connector rule before you click the **Generate WSDL** button.

Click the button to generate two WSDL documents:

- A standard WSDL 1.1 file that defines the SOAP services and connectors as operations.
- A WSDL file with BPEL extensions that defines the Connect BPEL rule as a BPEL partner.

About Connect Robot rules

Connect Robot is a Pega-supplied rule that you can use to import automation packages from Pega Robotic Studio into your Pega Platform development environment. When

©2025 Pegasystems Inc

you use the Connect Robot rule, you can use robotic desktop automations as data sources for data pages.

> **NOTE:** To use this feature, download and install the Pega Robotic Automation Importer component. For more information, see *Using Pega Robotic Automation Importer component* on Pega Community.

When you select a robotic desktop automation as a data source for a data page, Pega Platform automatically generates and displays the list of automations that are available to your application. Each robotic automation is represented by an instance of the Connect Robot rule. You can view an automation to understand its inputs, outputs, and implementation.

The process of importing automation packages into the Pega Platform development environment proceeds as follows:

- In Robotic Studio, an automation developer creates a solution that contains, for example, two automations, and then publishes the package to the Pega Platform development server.
- A REST Service accepts the package data and performs the following tasks:
    1. Creates a component. The component name is the package name.
    2. Creates an unlocked ruleset (*01-01-01*) inside the component.
    3. Creates two automation rules in the new ruleset.
    4. Generates data records that describe the automation package, version, and configurations.
    5. Associates each data record with the new ruleset.
    6. Locks the ruleset.
    7. Packages and closes the component.
    8. Attaches the component to the application that is authenticated.

After the REST Service process is completed, you can select the new robotic desktop automation as a data source for data pages.

©2025 Pegasystems Inc

- Attended robotic automation as a data page source

About Calendar data instances

Use this data class to define the business days and holidays for an organization. Calendar data instances can be referenced in organization, work queue, and Operator ID data instances, and in service-level agreements.

The following tabs are available on this form:

- Schedule

If your application and business operations depend on business day calculations, be sure to create calendar instances in advance for each future year (and every district, country, or jurisdiction) needed by your application. Business calendar functions such as *isBusinessDay()* fail with a Java exception when a date more than one year in the future is referenced and no calendar instance is found.

Where referenced

You can associate a calendar with operators (in the Operator ID data instances), organizations, and work queues.

References to calendars identify only the first key part. The system uses a context-determined date as the (starting) second key part.

Maintained by

Administrators maintain calendar data instances. To access calendar instances on your system, select **Configure** > **Org & Security** >**Tools** > **Calendars**.

Category

The *Data-Admin-Calendar* class contains calendar data instances. They are part of the Organization category.

©2025 Pegasystems Inc

- More about Calendar data instances
- Creating Calendar data instances

More about Calendar data instances

How the system finds a calendar

The second key part of a calendar instance identifies a start date for its use. The system computes the end date of a calendar as one year after the last holiday defined on the Schedule tab.

References to calendars identify a name, but not a start date. When supplied with a date and a calendar name, the system finds the Calendar data instance with that name and the latest Start Date key part not greater than the supplied date.

For example, a system contains four calendars, each indicating that December 31 is a business holiday, with these keys:

- ThorrUS 20040101
- ThorrUS 20050101
- ThorrUS 20050701
- ThorrUS 20060701

If the system needs to access a calendar named ThorrUS with a supplied date of 20050715, it uses the third calendar listed, even though the second calendar also covers July 15, 2005.

If this system needs a ThorrUS, calendar value beyond December 31, 2007, a Java exception is thrown, and a log message is added similar to the following:

```
The calendar obj corresponding to BusinessDay[3570] is null.
```

©2025 Pegasystems Inc

Functions for business days

Review the *Pega-RULES:BusinessCalendar* library for useful functions that use calendar data instances.

Availability to operators after change

Changes made to the calendar referenced in an Operator ID data instance take effect the next time that operator logs in.

Related Pega Community articles

See the Pega Community article *How BusinessCalendar functions AddDays() and AddTime() treat holidays, weekends, and time zones*.

About Calendar data instances

- About Calendar data instances
- Creating Calendar data instances

Creating Calendar data instances

Create a new calendar data instance by selecting `Calendar` from the `Organization` category.

The two-part key to a calendar data instance consists of the calendar name and starting date:

Field	Description
Calendar Name	Specify a name for this calendar. Begin the name with a letter, and use only letters, numbers, the ampersand character, and hyphens.
Starting Date	Specify a starting date in the form YYYYMMDD, for example 20070101.

©2025 Pegasystems Inc

- **Completing the Schedule tab**

- About Calendar data instances
- More about Calendar data instances

Completing the Schedule tab

Use the Schedule tab to define business days and regular working hours in your organization.

By default, information on this tab supports date and time computations only. Calendar data instances do not affect when users can log in or what they can do when logged in, unless your application implements such constraints.

> **NOTE:** Keep Closed Days complete and up-to-date. Computations for service levels, escalation, urgency, timeliness, due dates, and other properties often depend on correct holiday schedules.

Field	Description
Standard Time Zone	Select a time zone for the calendar. The system uses this value when comparing the current date and time with absences that are scheduled for the operator. This value does not affect other standard processing, though you can access it in your application processing. For example, select `America/New York` for the United States Eastern Time zone.

©2025 Pegasystems Inc

Field	Description
	These time zone codes follow the Olson TZ database, used in the ICU4J 3.4.4 library and most UNIX systems. If another calendar data instance has the same first key part as this calendar name, this field must match the Standard Time Zone value of the other calendar.
Calendar Locale	Specifies the locale in which the starting date of the calendar is parsed. This setting is relevant for locales that use non-Gregorian calendars, such as the Thai Buddhist or Japanese Imperial calendars
Schedule	Record the regular weekly work schedule that applies to this calendar.
Business	Select any combination of the seven check boxes to indicate normal business days.
Start	For each weekday, enter the local time at which the workday begins. Use the HHMMSS format without punctuation. For example, enter 201515 for 15 minutes and 15 seconds after 8 P.M. These time values are available for the *isBusinessTime(* and *AddTime(* standard functions in the *Pega-RULES:BusinessCalendar* library. The functions are used in service-level agreement calculations that are based on business days. For example, if an assignment created at 3:15 P.M. has an

©2025 Pegasystems Inc

Field	Description
	associated goal of three hours, but the workday is from 9:00 A.M. to 5:00 P.M., the goal time becomes 10:15 on the next workday. The default value is 080000.
End	For each weekday, enter the local time at which the workday ends. Use the HHMMSS format without punctuation. Enter a time that is later than the start time. The default value is 170000.
Closed Days	Optional. Enter in this array all dates when your business is closed for holidays or any other reason, other than weekdays not selected. Use the date format YYYYMMDD. Order is not significant. The system converts your input to a display format (MM DD YYYY).

- About Calendar data instances
- Creating Calendar data instances
- More about Calendar data instances

Divisions

A division is the second-level element of your organization. You can identify rulesets and versions that all members of that division are allowed to access.

©2025 Pegasystems Inc

The system offers a three-level organization structure. The top level is known as the organization; this middle level as divisions, and the lowest level as organization units.

The installation process creates a single division named *Administration@pega.com*.

- **Creating a division**

Related concepts

- Organization landing page

Related tasks

- Creating a division

Creating a division

Define the second-level element of your organization by creating a division.

1. In the header of Dev Studio, click Create > **Organization** > **Division**.
2. In the **Organization Name** field, specify the name of an existing organization data instance, a key to an Organization instance.
3. In the **Division Name** field, enter the name of this division, for example, Finance. Begin the name with a letter and use only letters, numbers, the ampersand character, and hyphens. Division identifiers are limited to 32 characters.
   As a best practice, limit the division name to no more than eight characters in length. When you use App Studio, the division name becomes one segment of several class names.
4. Click **Create and open**.
5. **Optional:** Identify an access group to be available to users in this organization. As a user or other requestor belonging to this division signs on, the system adds the RuleSets and versions in the application rules of that access group to the user's RuleSet list.

©2025 Pegasystems Inc

   a. In the **Defined by** list, select `an Application-based Access Group`. The selection `an Explicit RuleSet list` remains available for Division data instances created in Version 4 systems. However, this approach is deprecated for new development.
   b. In the **Access Group Name** field, identify one or more access groups that contains one or more access roles. Mark one access group as the initial default by selecting a radio button. In specific, unusual situations, this access group is used at sign-on, for operator IDs where no access group is present on the Access tab. (This can occur only for operators created in version 6.1SP1 or earlier.)
6. **Optional:** Determine to whom an assignment is routed or an email is addressed.
   a. In the **Cost Center** field, enter the cost center for the division. This field is not referenced by any standard rules but appears in the organization chart.
   b. In the **Cost Center Manager** field, enter the operator ID of a user of the application who can receive assignments or email. This field is not referenced by standard rules but appears in the organization chart.
7. Click **Save**.

> **What to do next:**
>
> If the Associated RuleSet is `[none]`, the associated RuleSet is set to the RuleSet from the current application of the logged-in user. This association assists with application packaging. You can update the associated RuleSet by using the field in the top left of the form.

- Organization landing page
- Divisions

Organization data instances

Complete an organization data instance for each organization that uses your system. Use the Organization form to specify an access group available to all members of an

©2025 Pegasystems Inc

organization, a calendar of work days and holidays, a top-level class, and other aspects of the organization.

The following tabs are available on this form:

- Definition
- Chart

A three-level organization hierarchy is supported that consists of organizations at the top level, divisions at the second level, and organization units at the third level. Each user (operator ID instance) is associated with an organization, division, and unit.

- **Organizations – Completing the Create or Save As form**
- **Organization form – Definition tab**
- **Organization form – Chart tab**
- **More about Organization data instances**
- **Organization landing page**
- Organization landing page

Organizations – Completing the Create or Save As form

Create a new organization data instance by selecting `Organization` from the `Organization` category.

An organization data instance has one key:

Field	Description
Organization	Enter an identifier for your organization, such as BUXR. Use only letters, digits,

©2025 Pegasystems Inc

Field	Description
	ampersand, underscores, and period characters. Organization identifiers are limited to 32 characters. As a best practice, limit the organization name to four characters — the length of many company stock ticker symbols. When you use App Studio, the organization name becomes one segment of several class names.

You can also create an organization instance using the organization chart, available on the **Chart** tab and on the Organizational Chart gadget on the Organization landing page.

- Organization data instances
- More about Organization data instances

Organization form – Definition tab

Optionally, you can identify an access group to be available to users in this organization. As a user or other requestor belonging to this organization signs on, RuleSets and versions in the application rules of that access group are added to the user's RuleSet list.

Organizational Access

Field	Description
Defined by	Select `an Application-based Access Group`.

©2025 Pegasystems Inc

Field	Description
	The choice `an Explicit RuleSet list` remains available for Organization data instances created in Version 4 systems. However, this approach is deprecated for new development.
Access Group Name	Optional. Displayed when `Application-based Access Group` is selected. Identify one or more access groups that contain one or more access roles. Mark one access group as the initial default by selecting a radio button. This access group is used for an operator signing on in specific, unusual situations: when the operator signing on meets both of these conditions: • No access group is present on the **General** tab of the **Operator ID** form for the operator signing on. (This condition can arise only for operator ID instances created in PRPC Version 6.1 SP1 and earlier.) • No access group is present in the Organization Division data instance associated with the operator signing on. (If these conditions are not met, the access group you identify here has no effect.)

©2025 Pegasystems Inc

Details

You can identify the controller or Chief Financial Officer of your organization and a default calendar for business day calculations.

Field	Description
CFO/Controller	Optional. Enter the operator ID of the Chief Financial Officer or Controller for your organization, if you plan to route email, assignments, or directed Web access messages to this person in your application.
Default calendar name	Optional. You can identify a calendar name, the first key part of a Calendar data instance, for business day calculations involving this organization. At runtime, when evaluating a calculation involving dates, system uses the first date in a calculation and this key to locate a calendar instance (*Data-Admin-Calendar* class) that identifies holidays and work days. (That may be a different calendar data instance than the one selected based on the current date.)
Top level class name	Optional. Identify a top-level class that applies to all operators in this organization. When an operator belonging to this organization uses App Studio, this class is the default top-level class. When you create an organization in App Studio, the system uses the **Organization Name** value to create the top-level class and enters the name here.

©2025 Pegasystems Inc

Field	Description
	See top-level class. Your applications can include classes that inherit — by pattern inheritance — from this class, to indicate they are used only by this organization. Using directed inheritance, your classes can inherit rules from the *Work-* base class, *Data-* base class, or other classes. However, some applications support more than one organization, and the system does not require that classes you create be derived from an organization **Top Level Class**.
Default RuleSet	Optional. Identify a RuleSet that applies to all operators in this organization. (For an organization data instance created during installation, this field is completed automatically.) Enter the RuleSet that contains the top-level class entered in the previous field. The ruleset version created by the New Application wizard identifies your own organizational ruleset as a prerequisite to the ruleset version it creates. The Localization wizard and Field Value inspector also use this optional value. These tools derive a target language-specific ruleset from the developer's organizational ruleset. The value in this field does not affect a user's RuleSet list. To make the rules in this RuleSet

©2025 Pegasystems Inc

Field	Description
	available to all operators in the organization, include this RuleSet and a version in the access group identified on the **Access** tab. If blank, the requestors that are associated with this organization do not have an organizational ruleset.

RuleSets

For Organization data instances with `an Explicit RuleSet list`. This approach is deprecated for new development.

Field	Description
RuleSet Name	Deprecated. Displayed when `explicit RuleSet list` is selected. Select a RuleSet or RuleSet and RuleSet Version. Order is significant.

Results

To review your RuleSet list:

- From Dev Studio, click the down arrow next to your name to access the Operator menu, and then select *Profile*.
- From a *WorkManager* portal, click the Dashboard bar, and then click the link containing your name at the top of the navigation panel.

Organization form – Chart tab

You can interact with the organization chart presented on this tab to maintain divisions, maintain organization units, and add operators.

PEGA ©2025 Pegasystems Inc

The chart is also available on the Division and Organization Unit rule forms, and in the **Organizational Chart** landing page tab on the Organization landing page. To access the tab, select Dev Studio > **Org & Security** > **Organization** > **Organizational Chart**.

Field	Description
Organizational Structure	The diagram presents the current structure as a three-level tree: organization, division, and organization units in your current organization (as displayed in your operator profile). To display all organizations in your system, click **Display All**. Click a plus sign to expand any node. Organizations are the first (top) level (AlphaCorp.com in this example). The chart includes a text description of the organization. By right-clicking a node, you can use a pop-up panel to: • Add nodes at the peer level (for example, select an organization and create a new one). • Add children nodes (for example, select an organization and create a new division belonging to it). • Add operators to organizations, divisions, or organization units. • Open an organization, division, or organization unit form.

©2025 Pegasystems Inc

Field	Description
	• View a list of operators associated with each node. As you make your modifications, use the **Refresh** button to update the display. To add an organization using the chart: 1. Select an organization node on the chart. 2. Right-click to open the pop-up panel, and select **Add Node**. 3. Enter an organization name in the **Organization** field and click **OK**. The instance appears on the chart. 4. To open the rule form to add or modify the information, double-click the instance, or select the instance on the chart, right-click, and select **Open**.

More about Organization data instances

The system offers an organization structure that has three levels. The top level, defined by this form, is the organization; the middle level defines divisions, and the lowest level is organization units.

Work queues

As a best practice, create a work queue named default@organization for each organization in your system, and have someone review the assignments in that work queue from time to time. Certain standard routing activities use this work queue as a last resort when no other suitable destination can be found for an assignment.

©2025 Pegasystems Inc

Organizational Setup landing page tab

You can use the **Organization Setup** landing page tab to create an organization structure for your company or for experimentation and learning. See Org & Security category — Organization landing page.

- Organization data instances
- Organizations – Completing the Create or Save As form

Organization landing page

Access the Organization landing page by clicking **Configure** > **Org & Security** > **Organization**.

The following tabs are available.

- Operators
- Organizational Chart

Operators tab

Use this tab to display a list of operators in the system and information about them.

Column	Description
Full Name	The full name of this user (up to 50 characters). This text is present in the navigation panel of standard portal displays. A user can have more than one operator ID.
Operator ID	The key part of an operator ID instance.
Job Position/Title	Optional. The user's job title or position.
Access Group	The name of an access group (a *Data-Admin-Operator-AccessGroup* instance) for this user.
Last sign on	The date and time when this operator most recently logged on using HTTP or HTTPS

©2025 Pegasystems Inc

Column	Description
	access to the Pega Platform portal using Microsoft's Internet Explorer Web browser on a Windows workstation, or from a browser presenting a Pega composite application.

- To add an operator ID instance, click **New...**.
- To migrate the data into an Excel spreadsheet, click **Excel**.
- Use the **Search Text** field to locate a user either by name or by operator ID. Use autocomplete to filter the text you want to search on.

Organizational Chart tab

This tab enables you to view the organizational structures and operators in the system. You can create or modify existing structures and add operators.

Pega Platform supports a three-level organizational hierarchy. The top level is known as the organization; the middle level as divisions, and the lowest level as organization units. Each user (operator ID instance) is associated with an organization, division, and unit.

This tab displays a tree chart representing the organizations in your system. You can use the chart to:

- Add nodes at the peer level (for example, add a division belonging to the same organization)
- Add children nodes (for example, add organization units reporting to a division)
- Add operators to the organization
- Open an organization, division, or organization unit rule form
- View a list of operators associated with each node

This chart is also available on the **Chart** tab in organization, division, and organization unit rule forms.

©2025 Pegasystems Inc

Viewing the chart

Click a plus sign to expand any node. Organizations are presented at the top level, divisions at the first indented level, and organization units at the second indented level. Organization units can have child units reporting to them. These child units are for charting purposes only. Their relationships are not referenced during rule processing or used in reports.

The Description column contains the short description text in the node's rule form. The Cost Center and Manager columns use the values entered in the Cost Center and Cost Center Manager sections on the **Accounting** tab on division and organization unit rule forms.

Double-click any node to open its rule form.

Working with the chart

You can perform operations on a node by selecting it on the chart and right-clicking. This opens a pop-up menu, which displays these options:

- *Add Node* -- Add a node at the same level. Enter the node name in a pop-up panel and click **OK**. The new node appears on the tree and a new instance is created. By default, the description text is the same name you entered.
- *Add Child* -- Add a child node beneath a parent node. Enter the node name in the pop-up panel and click **OK**. The new node appears beneath the parent node on the tree and a new instance is created.
- *View Operators* -- Display the list of operators associated with a node (identified in the **Organizational Unit** field on the **Work Settings** tab of the operator ID instance). Click an item to open the instance.
- *Add Operator* -- Add an operator instance to an organization. Enter the required information in the pop-up window and click **OK**. For more information, see Creating an operator ID. You can also create operators in the **Operators** tab.
- *Open* -- Open a node's rule form. Alternatively, double-click a node in the chart.

PEGA ©2025 Pegasystems Inc

> **NOTE:** As you make modifications, use the Refresh button to update the display.

Organization Unit data instances

A three-level organization structure is supported. The top level is known as the organization; the middle level as divisions, and the lowest level as units. Each user (Operator ID instance) is associated with an organization, division, and unit. Use this form to specify the organizational units that report to the divisions previously defined.

The following tabs are available on this form:

- Accounting
- Chart

Class

The *Data-Admin-OrgUnit* class contains organization unit data instances. They are part of the Organization category.

- **More about Organization Unit data instances**
- **Organization Units – Completing the New or Save As form**
- **Organization Unit form – Accounting tab**
- **Organization Unit form - Chart tab**

More about Organization Unit data instances

Where referenced

The system uses organization unit information in work queue data instances, and in management reports as levels of aggregation and detail.

©2025 Pegasystems Inc

LDAP-based authentication

If your company has an LDAP-compliant directory service, you can define a model operator ID data instance for any organization unit that is supported. An authentication service data instance links the organization unit and the directory service.

- Authentication services
- Organization Unit data instances
- Organization Units – Completing the New or Save As form

Organization Units – Completing the New or Save As form

Create a new division data instance by selecting `Organization Unit` from the `Organization` category.

An organization unit data instance has three key parts:

Field	Description
Organization Name	Specify the name of an organization that this unit belongs to, a key to an existing Organization instance.
Division	Specify the name of a division within that organization that this unit belongs to. (This is the second key part of a Division data instance.)
Unit	Specify the name of a unit within that division that this instance defines. Enter the name of this division, for example, Finance. Begin the name with a letter and use only letters, numbers, the ampersand character, and hyphens. Unit identifiers are limited to 32 characters.

PEGA ©2025 Pegasystems Inc

You can also create an organization unit instance using the organization chart, available on the **Chart** tab and on the **Organizational Chart** tab on the Organization landing page.

- Organization Unit data instances
- More about Organization Unit data instances

Organization Unit form – Accounting tab

Completing this tab is optional. Applications can use this manager information to route an assignment, or address an email.

Field	Description
Reporting To	
Org Units	Optional. Leave blank unless the organization unit reports to one or more units. This value is automatically filled if you add a child organization unit in the chart. Add units or modify their values to correctly portray their relationships within a division. For example, a unit may report to two units within the same division. These values are used in the organization chart display only and not in rule processing. **CAUTION:** Use caution when deleting organization units that have child units. Doing so makes them orphans, and they then do not appear on the chart. Revise or

©2025 Pegasystems Inc

Field	Description
	remove the **Reporting To** values to display their correct relationships.
Staff	
Manager	Optional. Select the operator ID of a user of this application who can receive assignments or email as a manager of this organization unit.
Model User	Optional. To indicate that users associated with this organization unit are authenticated by an LDAP-based directory service, identify a single operator ID data instance that services as a model or template for operators in this unit. See More about Organization Units.
Cost Center	
Number	Optional. Enter a text string that identifies this organization unit in an enterprise accounting system. This code may be referenced in the single sign-on interface defined through an Application ID data instance (*Data-Admin-AppID* class), or in a router activity. This value appears in the organization chart. If you use this property for single sign-on, choose **Cost Center Number** values that are unique to each organization unit, system-wide.

©2025 Pegasystems Inc

Field	Description
Manager	Optional. Enter the operator ID of a user who, as manager of the cost center, can receive assignments or email. This value appears in the organization chart.

- Organization Unit data instances
- Organization Units – Completing the New or Save As form
- More about Organization Unit data instances

Organization Unit form - Chart tab

You can interact with the organization chart presented on this tab to maintain divisions, maintain organization units, and add operators.

The chart is also available on the Division and Organization rule forms, and on the **Organizational Chart** tab on the Organization landing page. To access the tab, select **Configure** > **Org & Security** > **Organization** > **Organizational Chart**.

Field	Description
Organizational Structure	The diagram presents the current structure as a three-level tree: organization, division, and organization units in your current organization (as displayed in your operator profile). To display all the organizations in your system, click **Display All**. Click a plus sign to expand any node. Organization units are presented at the second indented level (Development, Finance, and so on in this example). Organization units can have child nodes (QA and Research in this example), which

©2025 Pegasystems Inc

Field	Description
	represent reporting relationships (and are not used in rule processing). These units indent further at each reporting level. By right-clicking a node, you can use a pop-up panel to: • Add nodes at the peer level (for example, select division and create a new one belonging to the same organization). • Add children nodes (for example, select a division and create a new organization unit reporting to it). • Add operators to organizations, divisions, or organization units. • Open an organization, division, or organization unit form. • View a list of operators associated with each node. The chart includes a text description of the unit, and the cost center and manager associated with the instance (if this information was entered in the **Number** and **Manager** fields on the **Accounting** tab). As you make your modifications, use the **Refresh** button to update the display. To add an organization unit using the chart:

©2025 Pegasystems Inc

Field	Description
	1. Select an organization node on the chart. 2. Right-click to open the pop-up panel, and select **Add Node**. You can also select a division node and then select **Add Child**. 1. Enter an organization unit name in the **Unit Name** field and click **OK**. The instance appears on the chart. 2. To open the rule form to add or modify the information double-click the instance, or select the instance on the chart, right-click, and select **Open**.

- Organization Unit data instances
- Organization Units – Completing the New or Save As form
- More about Organization Unit data instances

Summary views

Use the Summary View rule form to define a custom or personal query. This rule generates HTML that provides an interactive display or chart.

> **NOTE:** The summary view rule is deprecated. Re-create the custom summary view reports that you need as report definitions. The last version of Internet Explorer that summary view rules support is Internet Explorer 11.

©2025 Pegasystems Inc

OAuth 1.0 clients

The OAuth standard framework enables secure, delegated access to services over HTTPS. OAuth provides tokens (sometimes called "valet keys") that provide access to your data that is hosted by a specific service provider. Each token grants access to a specific site for specific resources, for a defined period of time. A later version of this protocol, OAuth 2.0, uses a different data instance type.

Your applications can act as an OAuth 1.0 consumer and client. As a result, your application can access private resources stored in external websites like LinkedIn, which support the OAuth 1.0 protocol.

Create an OAuth Client data instance to allow your application to securely access Web Service APIs such as those provided by LinkedIn.com, Twitter.com, and Flickr.com, without storing or disclosing individual users' names and passwords. For example, you can query your LinkedIn account for information about your connections to help you build relationships with customers with whom you connect through your application.

Activities that submit information to the application that you want to connect to and receive tokens back from use data stored in the OAuth Client data instance.

Each external application handles OAuth in its own way. Consult the API guide for the application that you want to work with. At a high level, the process involves these steps:

1. Get a request token. This is a temporary token that the application that you want to connect to uses to authenticate you. When you obtain a request token, you also get a token secret.
2. Obtain user authorization from the application that you want to connect to, specifying the permissions (read, write, delete) that you want to use.
3. Exchange the request token for an access token, which your application stores in a map value rule for reuse.
4. After you get the access token, use it to make authenticated requests to the application you want to connect to, through its provided API.

PEGA ©2025 Pegasystems Inc

Web Service Security profile

Use a Web Service Security profile (WS-Security) to determine how to authenticate a web service message.

In the WS-Security profile, you can define a SOAP message that verifies whether the message comes from a known sender and that the message was not modified during transmission. This configuration provides content integrity and confidentiality for SOAP requests and responses.

Enable Web Services Security on a SOAP connection to securely send messages to and from your application.

For more information, see Using WS-Security to enhance SOAP messages.

Flow markers

A flow marker saves test data and decisions that advance a flow execution to a specific point (mark). It allows you to jump directly to a specific point in the flow rule without having to input the same information every time to reach that point. In automated unit testing runs, the flow markers are used to advance the execution of a flow to that point.

The following tabs are available on this form:

- Inputs
- Results

Ordinarily, flow markers are created automatically as you test flows. They are not typically created by using a rule form. However, you can open and review the **Flow Marker** rule forms.

Flow marker rules belong to a RuleSet and version, but they do not appear in the **Application Explorer** display.

©2025 Pegasystems Inc

Access

Use the **Test Cases** tab of the rule form for the associated flow to see the saved flow markers for that flow.

Where referenced

Flow marker rules are referenced:

- In the **Test Cases** tab of a flow's rule form.
- During playback of a flow's recorded test case.

Category

Test case flow markers are instances of the *Rule-AutoTest-Case-FlowMarker* class. They belong to the SysAdmin category.

- **Flow Marker form – Understanding the Inputs tab**
- **Flow Marker form — Understanding the Results tab**

Flow Marker form – Understanding the Inputs tab

The Inputs tab shows you the flow data that is saved in the flow marker from a unit testing run of a flow. This data represents the inputs and conditions at the point where the flow marker was created.

This tab is best used to examine the initial clipboard pages so that you can review the initial properties and values that were used when this flow marker was saved. If you change values here and then save, inconsistent results from using this flow marker might occur. Instead, to update a saved flow marker with new input data, the best practice is to play back the associated flow test case and overwrite this flow marker during the playback.

PEGA ©2025 Pegasystems Inc

Data Transform
: Read-only. The name of the data transform that was used to create a test page in the unit test run of the flow (typically, *pyDefault*).

Initial Pages
: When testing rules, sometimes clipboard pages are created, loaded, or copied before the rule can run. These pages are saved when you create a flow marker, so that you don't need to recreate the initial conditions before jumping to that flow marker in the execution of the flow. Use the controls in this section to view those pages and examine the initial properties and values.

Input Parameters
: For review only. A list of the test data that was used — the input properties and their values. If you make changes here and save them, inconsistent results might occur when this flow marker is used.

Flow Marker form — Understanding the Results tab

The **Results** tab shows you the results that were captured at the point in the flow when this flow marker was saved.

This tab is best used to examine the results' clipboard pages so that you can review the expected properties and values captured when this flow marker was saved. If you change values here and then save, inconsistent results from using this flow marker might occur. Instead, to update a saved flow marker, the best practice is to play back the associated flow test case and overwrite this flow marker during the playback.

Result Value
: The value that must be returned when you jump to the flow marker for the results to be considered expected.

Show Result Pages
: The clipboard pages that held the results when the flow marker was created. This section allows you to view the resulting property and values up to the point the flow marker was created.

PEGA ©2025 Pegasystems Inc

About SpellChecker Properties data instances

Use the SpellChecker Properties form to control operation of the spelling checker features of the Pega Platform, for a specific locale (language variant).

The following tabs are available on this form:

- Dictionaries
- Properties

> **NOTE:** Spellchecking is not available when a Pega Platform application is accessed by using Pega Web Mashup.

Where referenced

When an application user clicks the SpellCheck button ( ) on a user form, the system uses information in the SpellChecker Properties instance (based on the locale of each user) to determine which dictionaries to apply, how to operate the spellchecking algorithm, and other details.

Access

Use the Records Explorer to list SpellChecker Properties data instances in your system.

Category

A Spellchecker Properties data instance is an instance of the *Data-SpellChecker-Properties* class. They are part of the SysAdmin category.

- **Spellchecker Properties form – Completing the Dictionaries tab**
- **Spellchecker Properties form - Completing the Properties tab**
- Spell checker

©2025 Pegasystems Inc

Spellchecker Properties form – Completing the Dictionaries tab

List the dictionaries that apply to the locale identified in the key of this instance. List at least one dictionary. At runtime when a spellcheck operation is requested, the system searches dictionaries in the order they are listed on this tab.

Dictionaries included with the Pega Platform itself are in locked RuleSets and cannot be altered.

Field	Description
Main	Compressed dictionaries are searched in the order they appear in this list. Enter at least one compressed dictionary.
Name	Select a compressed dictionary, a reference to a binary file rule with `webwb` as the first key part and `clx` as the third key part. Use SmartPrompt to see dictionaries available to your RuleSet list including six initially installed compressed dictionaries.
Supplemental	Supplemental dictionaries are searched in the order they appear on this list. Enter at least one supplemental dictionary.
Name	Select a supplemental dictionary, a reference to a text file rule with `webwb` as the first key part and `clx` as the third key part. Use SmartPrompt to see supplemental dictionaries available to your RuleSet list

©2025 Pegasystems Inc

Field	Description
	including six compressed dictionaries provided at installation.
User	Optional. Complete this array to reference custom dictionaries and allow users to add words to them.
Name	Select a user-maintained dictionary, a reference to a text file rule with `webwb` as the first key part, `clx` as the third key part, and that belongs to a RuleSet version that is not secured. Users who hold the *@baseclass.AddToDictionary* privilege are provided the opportunity, during spellchecking operations, to add a word to any dictionary listed here. (The standard Access of Role to Object rule named *@baseclass.SysAdmin4* includes this privilege, but you can make this privilege available to other access roles in your application.) To create a user-maintained dictionary, copy and rename the text file rule named *webwb.user.tlx*, and use the HTML editor to add words, one per line.
Default?	If this tab includes a user-maintained dictionary, select to mark one user-maintained dictionary as the default for added words.

©2025 Pegasystems Inc

Spellchecker Properties form - Completing the Properties tab

Spellchecker Properties form – Completing the Properties tab

Settings on this tab determine:

- Which elements of the text are spellchecked,
- Which are ignored, and
- How suggestions appear

Field	Description
General	Three check boxes control overall operation of spellchecking.
Allow Accented Capitals?	If not selected, any word containing an accented capital letter is marked as misspelled. Ordinarily, select this for best performance, except for French Canadian spellings with the French dictionary.
Is Case Sensitive?	Select to cause dictionary comparisons to be case sensitive so that `FROM` (or `From` or `fROM`) in the form does not match `from` in a dictionary. Clear to force case-insensitive comparisons.
Strip Possessives?	Select to cause a trailing 's or s' to be stripped from words before dictionary comparisons. (Dictionaries do not contain possessive forms of words.)

©2025 Pegasystems Inc

Field	Description
Ignore	Select a box to cause the spellchecker to skip over certain words.
Mixed Case?	Select to ignore words containing an unusual mix of uppercase and lowercase characters, such as `RuleSet`.
Digits in Word?	Select to ignore words containing both letters and digits, such as `Win2K`.
Capitalized Word?	Select to ignore words beginning with a capital letter, such as `Pega`.
All Caps in Word?	Select to ignore words containing only capital letters, such as `IBM`.
Domain Names?	Ignore words that appear to be Web domain names, such as `pega.com`.
Non-Alpha Word?	Ignore words that contain no alphabetic characters, such as $123.45.
Flag as Bad Word	Select to cause the spellcheck function to flag words meeting these criteria as misspelled.
Mixed Case?	Select to mark words containing an unusual mix of uppercase and lowercase letters, such as RuleSet. (This box is ignored if the Mixed Case? box in the Ignore group is selected.)

©2025 Pegasystems Inc

Field	Description
Digits in Word?	Select to mark words containing both letters and digits, such as Win2K. (This box is ignored if the Digits in Word? box in the Ignore group is selected.)
Not Capitalized?	Select to mark words for which a dictionary contains a capitalized form (such as Chicago), but the word does not match (for example, chicago or chicagO). (This box is ignored if the Capitalized Word? box in the Ignore group is selected.)
Double Word?	Select to mark a repeated word as misspelled.
Examine as Separate Words	
Contracted Words?	Select to cause each part of a contraction to be checked separately when the whole contraction is not found in any dictionary. For example, French and Italian allow such ad hoc contractions.
Hyphenated Words?	Select to cause each part of a hyphenated word to be checked separately when the whole hyphenated word is not found in any dictionary. The entire word is correct only if each part is correct.
Split Words?	Leave blank in most cases. Select to cause the spellchecker to attempt to split invalid words into sub-words that are valid. This feature is useful in languages such as German and Finnish, which allow ad hoc concatenation of words.

PEGA ©2025 Pegasystems Inc

Field	Description
Suggestions	
Level of Quality	Choose a value between 1 and 100 to limit the depth of search for suggested words. Enter 100 for maximum search, which may take several seconds to complete.
Give Split Word Suggestions?	Select to instruct the spellchecker to attempt to split a misspelled word into two valid words when searching for suggested words. For example, a suggestion for "thereis" is "there is."
Initial Number of Suggestions?	Specify the number of initial suggestions to display for a misspelled word in the rich text editor. The rich text editor displays in a number of locations within the Pega Platform, including correspondence, paragraph rules, and attachments.
Maximum Number of Suggestions?	Specify the maximum number of suggestions to display for a misspelled word in the rich text editor. These suggestions display when a user clicks **More Suggestions** in the spell checker within the rich text editor. The rich text editor displays in a number of locations within the Pega Platform, including correspondence, paragraph rules, and attachments.

Word templates

A Word template is a supporting Rule type that enables you to incorporate Microsoft Word documents (`.docx`) into your application to support correspondence generation

©2025 Pegasystems Inc

and creation of dynamic document. The Word templates can include standard Word features such as images, styles, macros, and formulas, and support Clipboard data merging at run time.

The following Pega Platform™ capabilities use the Word template Rules:

User-edited Correspondence

When a Correspondence Rule is configured with the **User Edit** option set to **Must Edit**, and triggered by the *Work-.AttachAndHold* Flow Action, users can open and edit the document in Microsoft Word on their local machine. The updated document is then saved as a file attachment in the Case and can optionally be sent as an email attachment.

Automated document generation

A Word template can be merged at run time with Clipboard data to automatically generate a Word document. The merge process replaces field codes in the `.docx` file with values from properties, referenced Correspondence Rules, or file attachments. The generated document can be stored in a property or attached to the Case.

Legacy Word merge with Microsoft Silverlight

In earlier Pega Platform applications, Word merge functionality relied on Microsoft Silverlight to support client-side document merging and previewing. Silverlight is no longer supported in modern browsers, and this approach is not recommended for new implementations.

Supported field codes

If you want a Word document to support the Clipboard merge function, include the following directives as field codes. In Microsoft Word, select **Insert** > **Text group** > **Quick Parts** > **Field** from the menu, then insert the code using the following syntax:

Syntax	Description	Sample	Result

©2025 Pegasystems Inc

`{pega:reference propertyreference }`	Inserts the value of a scalar property from the Clipboard page.	`{pega:reference .Customer.Name}`	Inserts the customer's full name, such as "John Smith", from the *.Customer* page.
`{pega:insert corr="Name" }`	Inserts the content of a Correspondence Rule, with `Name` as the second key part.	`{pega:insert corr="ClaimApprovalNotice"}`	Inserts the HTML content of the correspondence Rule *ClaimApprovalNotice* , such as a formatted approval message.
`{pega:attach CategoryName }`	Inserts the HTML content from the most recent file attachment of the specified category.	`{pega:attach Receipts}`	Inserts the contents of the latest file attached under the "Receipts" category, for example, a scanned invoice or receipt.
`{pega:binary Name }`	Inserts the content of another Word template stored as a `Rule-Template-Word` Rule, using the format `AppliesToClass.Purpose` .	`{pega:binary Work-Insurance.ClaimSummaryTemplate }`	Inserts the entire content of the Word template Rule named `ClaimSummaryTemplate` in the *Work-Insurance* class, such as a pre-formatted claim summary.

©2025 Pegasystems Inc

Sample scenario

In a scenario in which you generate a claim settlement letter for a customer, your Word template might look like this:

```
Dear {pega:reference .Customer.Name},

We have reviewed your insurance claim (Claim ID: {pega:reference .ClaimID}) submitted on {pega:reference .SubmissionDate}.

{pega:insert corr="ClaimDecisionNotice"}

Please find the attached receipts for your reference: {pega:attach Receipts}

A detailed summary is included below: {pega:binary Work-Insurance.ClaimSummaryTemplate}
```

Word template with field codes

Result after merging:

```
Dear John Smith,

We have reviewed your insurance claim (Claim ID: CLM123456) submitted on April 12, 2025.

<HTML content from the ClaimDecisionNotice Rule inserted here>

Please find the attached receipts for your reference: <HTML content of the most recent receipt file inserted here>

A detailed summary is included below: <Content from ClaimSummaryTemplate inserted here>
```

Result during merge

Related tasks

- Attaching content to a Case
- Creating a Correspondence Rule

Related information

©2025 Pegasystems Inc

- Clipboard tool

Service Java rules (to be deprecated)

Using the Java Business Delegate pattern and the Java Session Facade pattern, the Java service interface makes service rules available to external, Java-based applications as if the service rules were public methods of a Java class.

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

To use Java service rules, the Pega Platform must be deployed as an enterprise application, not as a Web application. When the Pega Platform is deployed as an enterprise application, two EJBs (*PRServiceStateless* and *PRServiceStateful*) implement the session facade. They provide the public interface for remote and local access to Pega Platform Java and EJB services.

Java services generally process service requests synchronously. That is, they immediately perform their requested processing and return a configured return value while the calling application waits. However, you can configure Java services to process service requests asynchronously, which means the service queues the request for asynchronous execution and the calling application calls back later for the results. Additionally, you can configure synchronous Java services to check for specific error conditions that you expect will be temporary — work item locks, for example — and then queue service requests that fail for those reasons for another attempt at a later time.

Access

Use the Application Explorer to see Service Java rules in your application. Use the Records Explorer to list all Service Java rules that are available to you.

©2025 Pegasystems Inc

Category

Service Java rules belong to the Integration-Services category. A Service Java rule is an instance of the *Rule-Service-Java* rule type.

- **Service Java rules - Completing the Create, Save As, or Specialization form**
- **Unit testing a Service Java rule**
- How to call through Java-based services

Service Java rules - Completing the Create, Save As, or Specialization form

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

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

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

PEGA ©2025 Pegasystems Inc

This information identifies the key parts and options that apply to the record type that you are creating.

> **NOTE:** Create a Service Package data instance before creating a Service Java rule; the name of the service package is the first key part of a collection of Service Java rules.

Create a Service Java rule by selecting `Service Java` from the `Integration-Services` category.

A Service Java rule has three key parts:

Field	Description
Customer Package Name	Select the name of the service package (instance of the *Data-Admin-ServicePackage* class). The service package must exist before you can create the service rule. See About Service Package data instances. If your application is to process requests from this service asynchronously through a background agent, define a Service Request Processor data instance (*Data-Admin-RequestProcessor-Service* class) with this Customer Package Name value as key.
Customer Class Name	Enter the name of the service class. This name is an arbitrary identifier; it is not an instance of *Rule-Obj-Class* but must be a valid Java identifier. See How to enter a Java identifier.

©2025 Pegasystems Inc

Field	Description
	The service package name and service class name group service rules.
Identifier	Enter an identifier for the service method. This name is a string; it is not an instance of the *Rule-Method* class. The service method named describes what the service rule does. See How to enter a Java identifier.

- Service Java rules (to be deprecated)

Unit testing a Service Java rule

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

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

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

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

To run a unit test, complete the following steps:

1. Save the rule form.
2. Start the Tracer by clicking **Actions** > **Trace**. For more information, see Tracing services.

©2025 Pegasystems Inc

3. Click **Actions** > **Run**.
4. Complete the form as described in the following table, and then click **Execute**.

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

©2025 Pegasystems Inc

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

- Service Java rules (to be deprecated)

Service JSR94 rules (to be deprecated)

Java Specification Request 94 (JSR 94) describes an Application Programming Interface (API) for rules engines defined by the Java Community Process (JCP). The JSR 94 integration interface makes Pega Platform processing available to external Java applications through JSR 94 invocations to JSR94 service rules.

PEGA ©2025 Pegasystems Inc

> **NOTE:** Service JSR94 rules are no longer being actively developed, and are being considered for deprecation in upcoming releases. Using Service JSR94 rules does not follow Pega development best practices. Consider other implementation options instead.

This service type enables the Pega Platform to be embedded into, and called directly from, any Java program.

Access

Use the Application Explorer to see Service JSR94 rules in your application. Use the Records Explorer to list all Service JSR94 rules that are available to you.

Development

Consult the standards documents on the Java Community Process website for a definition of the API methods and the Test Compatibility Kit. The Pega Platform works with JSR 94 version 1.0 jar files.

For a comprehensive description of the development process and a sample, see article *Building JSR94 Services* (August 2005), a document available from the Integration pages of the Pega Community.

Category

Service JSR94 rules belong to the Integration-Services category. A Service JSR94 rule is an instance of the *Rule-Service-JSR94* class.

- **Service JSR94 rules - Completing the Create, Save As, or Specialization form**
- **Unit testing a Service JSR 94 rule**
- How to call through Java-based services

PEGA ©2025 Pegasystems Inc

Service JSR94 rules - Completing the Create, Save As, or Specialization form

> **NOTE:** Service JSR94 rules are no longer being actively developed, and are being considered for deprecation in upcoming releases. Using Service JSR94 rules does not follow Pega development best practices. Consider other implementation options instead.

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

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

This information identifies the key parts and options that apply to the record type that you are creating.

> **NOTE:** Create a Service Package data instance before creating a Service JSR94 rule; the name of the service package becomes the first key part of a group of related Service JSR94 rules. The class and method name key parts are considered "external" and unrelated to the Pega Platform class and methods, for flexibility.

PEGA ©2025 Pegasystems Inc

Create a Service JSR94 rule by selecting `Service JSR94` from the `Integration-Services` category.

A Service JSR94 rule has three key parts:

Field	Description
Customer Package Name	Select a service package name (instance of the *Data-Admin-ServicePackage* class); this service package groups related Service JSR94 rules. Choose a name already defined by a Service Package data instance. See About Service Package data instances.
Customer Class Name	Enter the name of the service class. This is an arbitrary text value that is a Java identifier, not an instance of *Rule-Obj-Class* class. See How to enter a Java identifier. The service package name and service class name group service rules.
Identifier	Enter an identifier for the service method. See How to enter a Java identifier. This name is an arbitrary text value; it is not an instance of *Rule-Method*. The service method named describes what the service rule does.

- Service JSR94 rules (to be deprecated)

©2025 Pegasystems Inc

Unit testing a Service JSR 94 rule

Use the unit testing feature to verify that the operations of a Service JSR 94 rule function correctly before you add the external client to your testing process.

> **NOTE:** Service JSR94 rules are no longer being actively developed, and are being considered for deprecation in upcoming releases. Using Service JSR94 rules does not follow Pega development best practices. Consider other implementation options instead.

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

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

To run a unit test, complete the following steps:

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

Field	Description
Requestor Context	Select a radio button to define the requestor session is to be used in the test: • Use current requestor context — Use your current requestor session (including your RuleSet list, privileges, and current clipboard).

©2025 Pegasystems Inc

Field	Description
	• Initialize service requestor context — Create a new requestor session based on the *APP* requestor type and, if the service package requires authentication, another Operator ID instance.
Authentication User ID	If you selected Initialize service requestor context, and the service package instance for the service requires authentication, enter the Operator ID to be used to test the service.
Authentication Password	If you selected Initialize service requestor context, and the service package instance for the service requires authentication, enter a password for the Operator ID.
Enter Request Data	Select a radio button to define the source of request data values for this test: • Specify individual request values — This option appears only if the JSR94 input parameters are scalar values, such as strings, numbers, or booleans. • Invoke Initialization activity — A test activity creates values for the JSR94 input parameter values.
Input Parameter Values	If you selected Specify individual request values for the previous field, enter in the Value field a literal constant value for each input method parameter declared on the

©2025 Pegasystems Inc

Field	Description
	Request tab. Enter a value that corresponds to the Java data type listed.
Activity	If you selected Invoke Initialization activity, enter here the Activity Name key part of an activity that creates JSR94 input parameters. The system assumes the Applies To class of the activity matches the Primary Page Class value on the Service tab. If the activity applies to a different class, enter the class name, a period, and the activity name.

- Service JSR94 rules (to be deprecated)

Service MQ rules

Use a package of Service MQ rules to let external systems and users access the Pega Platform data and processing using the IBM WebSphere MQ middleware messaging standard. Each Service MQ rule identifies an activity providing the service. The external system passes an MQ message to the Pega Platform, which runs the activity and then may return an output MQ message.

Pega supports Websphere MQ versions up to and including version 9.

When the Pega Platform is deployed as an enterprise application, you must use a message-driven bean (MDB) as the listener for message-based service rules (JMS and MQ). As a result, although you can use WebSphere MQ as a JMS service provider, use JMS MDB listeners with JMS service rules, rather than MQ listeners with MQ service rules, when the Pega Platform is deployed as an enterprise application.

MQ services generally process service requests synchronously. That is, they immediately perform the requested processing and return a configured response while

©2025 Pegasystems Inc

the calling application waits. However, you can configure MQ services to process service requests asynchronously, which means the service queues the request for asynchronous execution and the calling application calls back later for the results. Additionally, you can configure synchronous MQ services to check for specific error conditions that you expect will be temporary — work item locks, for example — and then queue service requests that fail for those reasons for another attempt at a later time.

For information about creating MQ services, see the following Pega Community articles:

- *How asynchronous service processing works*
- *Configure a service to process requests asynchronously*
- *Configure a service to queue a failed service request for another attempt*

Access

Use the Application Explorer to see Service MQ rules in your application. Use the Records Explorer to list all Service MQ rules that are available to you.

Category

Service MQ rules belong to the Integration-Services category. A Service MQ rule is an instance of the *Rule-Service-MQ* rule type.

- **Service MQ rules - Completing the Create, Save As, or Specialization form**
- **Unit testing a Service MQ rule**

Service MQ rules - Completing the Create, Save As, or Specialization form

Service MQ rules - Completing the Create, Save As, or Specialization form

PEGA ©2025 Pegasystems Inc

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

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

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

This information identifies the key parts and options that apply to the record type that you are creating.

> ⓘ **NOTE:** Create a Service Package data instance before creating a Service MQ rule; the name of the service package becomes the first key part of a group of related Service MQ rules. The class and method name key parts are considered "external" and unrelated to the Pega Platform class and methods, for flexibility.

Create a Service MQ rule by selecting `Service MQ` from the `Integration-Services` category.

PEGA ©2025 Pegasystems Inc

A Service MQ rule has three key parts:

Field	Description
Customer Package Name	Select the name of a service package (instance of the *Data-Admin-ServicePackage* class); this package groups related Service MQ rules. Choose a name already defined through a Service Package data instance. See About Service Package data instances. If your application is to process requests from this service asynchronously through a background agent, define a Service Request Processor data instance (*Data-Admin-RequestProcessor-Service* class) with this Customer Package Name value as key.
Customer Class Name	Enter a class name to logically group related methods. This name may or may not refer to the Pega Platform class that the activity belongs to, but must be a valid Java identifier. See How to enter a Java identifier.
Identifier	Enter a Java identifier that describes the Pega Platform activity being called. See How to enter a Java identifier.

- Service MQ rules

Unit testing a Service MQ rule

Use the unit testing feature to verify that the operations of a Service MQ rule function correctly before you add an external component to your testing process.

PEGA ©2025 Pegasystems Inc

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

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

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

To run a unit test, complete the following steps:

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

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

©2025 Pegasystems Inc

Field	Description
	authentication, another Operator ID instance.
Authentication User ID	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter the Operator ID to be used to test the service
Authentication Password	If you selected Initialize service requestor context , and the service package instance for the service requires authentication, enter a password for the Operator ID.
Enter Request Data	Select a radio button to define the source of request data values for this test: • Specify individual request values — This option appears only if the Message Data of the Request tab contains only scalar values, such as strings, numbers, or Boolean values. • Invoke Initialization activity — A test activity creates values for the request message values.
Message Header Values	If you selected Specify individual request values for the previous field, enter in the Value field a literal constant value for each Header Field row on the Request tab. Enter a value that corresponds to the Java data type listed.

©2025 Pegasystems Inc

Field	Description
Message Buffer Values	If you selected Specify individual request values for the previous field, enter in the Value field a literal constant value for each Message Data row listed on the Request tab. Enter a value that corresponds to the Java data type listed.
Activity	If you selected Invoke Initialization activity , enter here the Activity Name key part of an activity that creates the information from a test MQ request message. The system assumes the Applies To class of the activity matches the Primary Page Class value on the Service tab. If the activity applies to a different class, enter the class name, a period, and the activity name.

Creating an SMS account

You can use short message service (SMS) accounts for sending a one-time password in multifactor authentication.

1. Open the instance list for *data-smsaccount* and click **Create**.
2. In the **Account Name** field, enter a name for the account, and click **Create and open**.
3. In the **Select SMS provider** list, select the messaging service provider. Currently, Amazon Simple Notification Service (SNS) is supported.
4. In the **Key ID** field, enter the SMS Key.
5. In the **Secret Key** field, enter the secret key.
6. In the **Sender Region** list, click the sender region.
7. In the **Sender ID** field, enter the sender ID.
8. Click **Save**.

©2025 Pegasystems Inc

- Security policies
- Security policies settings

Creating a web service policy data instance

Create a web service policy (WS-Policy) data instance to allow a SOAP connector to securely communicate with an annotated web service.

1. In the header of Dev Studio, click **Create** > **Integration-Resources** > **Web Service Policy**.
2. Enter a short description and web service policy name for this data instance.
   The name is arbitrary and is used only to distinguish instances from one another. Begin the name with a letter, and use only alphanumeric characters.
3. Click **Create and open**.
4. In the **Service** section, attach the policy xml attached to the binding element in the WSDL. A policy attached to the binding element of the endpoint policy subject is placed in the service policy section of the source view tab. There are two views for the service.
   - **Source View** – The text form of the policy document. You can directly edit the text. Click **Generate/Update Design View** to produce a Design View of the current policy document.
   - **Design View** – A graphical tree view of the policy XML document. You can edit the fields in the display to enter configuration information. Click **Update Source View** to write the changes you have made in the Design View into the policy XML document.
5. In the **Input Message** section, attach the policy xml attached to the input element of the endpoint policy subject in the WSDL.
   Use the Source View and Design View panes as described above for the Service section.
6. In the **Output Message** section, attach the policy xml attached to the output element of the endpoint policy subject in the WSDL.

©2025 Pegasystems Inc

Use the Source View and Design View panes as described above for the Service section.

7. Click **Keystore** to go to the Keystore tab.
8. In the **Keystore** field, select the keystore data instance to use with this policy. See Keystores.

   Leave the **Truststore** field blank. This field is not used for WS-Policy implementations. (It is used for SSL.)
9. Click **Save**.

- Web service policy data instances

How to call through Java-based services

How to call Pega Platform through Java-based services

Pega Platform provides three Java-based service interfaces (EJB, Java, and JSR94) that you can use to request services using a signature-based or an engine API invocation-based invocation style:

- Signature-based means the external application invokes the Pega Platform service rules through the proxy EJB or Java archive generated by the service package. The proxy publishes the method signatures of the service rules. A client application sends the input to the service rule by calling a method in the proxy — a method with parameters that match the service rule. The proxy manages type validation, constructs the URI for the service rule, and so on.
- Engine API invocation means the external application invokes the Pega Platform service rules through a (generic) business delegate, directly through the *PRService* EJBs, or through an instance of the PegaRULES engine. The client application calls one Java method no matter which service rule is targeted in the Pega Platform. The parameters provided in the invocation identify the service rule and provide the input for the service rule. Unlike the signature-based style, no compile-time validation of method names and parameters does not occur, because you are not using a proxy compiled by the Pega Platform.

©2025 Pegasystems Inc

Service EJB rules and Service Java rules support both invocation styles. Service JSR94 rules support the engine API invocation style only, in accord with the JSR 94 specification.

Which service rule type and invocation style is best depends on your situation and needs. For example:

- If the external calling application is an enterprise application, use Service EJB rules with a generated proxy.
- If your organization seeks to comply with the JSR 94 specification, use Service JSR94.
- For a POJO (plain old Java object) interface to the Pega Platform, use Service Java rules, with or without a generated proxy.

For more information, see *Building EJB Services* and *Building JSR-94 Services*, two documents available on the Integration area of the Pega Community.

Registry Java Property Transform data instances - Completing the Save As form

A Registry Java Property Transform has a single key part, the Java class name.

To create a Registry Java Property transform, open an existing instance of this class and use the Save As toolbar button.

Field	Description
Java Class Name	Enter the fully qualified Java class name of the Java class that is to be implemented as a `Single Value` property by the Import JavaBeans wizard. For example, `java.util.Date`.

©2025 Pegasystems Inc

About Broadcast data instances

Create a broadcast data instance to define the text of short messages, also known as welcome messages, news messages, or messages-of-the-day. These messages appear in the navigation panel of User portals when the Dashboard is open.

The following tabs are available on this form:

- Message

Typically, a manager in the organization unit maintains the message for that unit. A developer can create a distinct message for each unit in an organization.

For composite portals, the standard section *@baseclass.News* provides this capability.

Changing the message

Users of Dev Studio can update any broadcast data instance, using the form. Manager updates depend on access roles:

- If you have a SysAdmin access role and use the Manager composite portal, you can alter the text of the broadcast message for the organization unit you belong to without using the Broadcast form.
- If you have a manager access role and use the *WorkManager* portal, you can alter the text of the broadcast message for the organization unit you belong to without using the Broadcast form.

On the Dashboard navigation panel, click the **Create** or **Edit** button appearing in the broadcast message. A pop-up window appears. Enter your brief message--no more than 70 characters--directly into the window and click **Save**. The new message replaces any existing broadcast message for the unit; older messages are not saved.

Users in Dev Studio can update the message for any organization unit:

1. Use the Records Explorer to list instances of the broadcast data class (in the Technical category).

©2025 Pegasystems Inc

2. Open the Broadcast form for the organization unit, or create a new one if necessary. Enter the organization, division, and unit as key parts.
3. Edit the Broadcast instance and save it.

The next time a user in the organization unit logs in, the portal News area displays the updated message.

Category

The *Data-Broadcast* class holds data instances. They are part of the Technical category.

- **Broadcast form - Completing the Message tab**
- **Broadcast data instances - Completing the New or Save As form**

Broadcast form - Completing the Message tab

Broadcast form – Completing the Message tab

Description
Enter the text of the message that will appear in the navigation panel. The updated message appears the next time users in that organization unit log in, or the next time they refresh the portal window. Enter no more than 70 characters of text. The new message replaces any existing message. Older messages are not saved. (After saving, click the Refresh button to refresh your display.)

Broadcast data instances - Completing the New or Save As form

Broadcast data instances – Completing the New or Save As form

©2025 Pegasystems Inc

Create a Broadcast data instance by selecting `Broadcast` from the `Technical` category.

A Broadcast data instance has a key that identifies an organization unit. Choose values that form the three-part key of an organization unit.

Field	Description
Organization Name	Enter an organization name.
Division Name	Enter a division name within that organization.
Unit Name	Enter the name of a unit within that division.

About Broadcast data instances

- About Broadcast data instances

Circumstance definitions

A circumstance is an optional qualification available for supported rule types and is built upon the base rule.

Using circumstances in your application allows you to easily support a variety of use cases. For example, you might want a different data transform to execute depending on a customer's geographic location (for example, when `.StateCode` = "MA"). Instead of maintaining the logic for all cases in one large rule, start with a base rule and extend (or specialize) it as needed. At runtime, the Pega Platform automatically selects the correct version to execute as part of its rule resolution process.

The following figure illustrates example logic for evaluating geographic location and other properties in a case.

©2025 Pegasystems Inc

	Apply this Definition			
	State	Channel	Claim Amount	
			>=	<=
if	MA	Email	750.	
or if	NC	Letter	500.	
or if	CA	Email	250.	
or if	CT	Letter	100	1000

You can create a circumstance of a base rule by a single value property, template (multiple properties), or date.

> **NOTE:** If you have upgraded to Pega 7.2, all date circumstance rules will be upgraded automatically and can be modified, except the rules circumstanced by date range and as-of date. These rules continue to run without issues but must be manually upgraded to use the new date circumstancing options.

- Creating a circumstance definition

Circumstance templates

Use a circumstance template rule to identify properties for multivariate circumstanced rules. A circumstance template rule contains an array of properties such as `.State`, `.Channel`, and `.ClaimAmount` that reference property values in one or more circumstance definition rules. These rules define the value combinations for each of the properties defined in the circumstance template rule.

Where referenced

Circumstance template rules are referenced by multivariate circumstance-qualified rules.

Multivariate circumstancing enables you to specialize a rule by more than one feature. For example, an insurance company may be required to collect, for every state, claims that exceed a specific amount and the method by which the claims were conveyed (letter, phone, email, and so on).

©2025 Pegasystems Inc

Create a circumstance template rule before you create a circumstance definition rule. For instructions, see these Pega Community articles:

- *How to create a circumstance template rule*
- *How to create a circumstance definition rule*
- *How to create a rule with multiple circumstance properties.*
- *How multivariate circumstancing works*

- Creating a circumstance template

Function Rule

A Function Rule defines a Java method that is accessed through Rule resolution. By using Rule resolution at run time, a Function can express different behavior depending on the application Ruleset list. You can invoke Functions from Expressions, from activities, and from Java.

Functions are defined within a library; both the library and its Functions must be defined in the same Ruleset. A Function Rule defines the parameters and the Java code that is invoked when calling the Function.

After you define a Function, you can define one or more Function aliases to customize the parameter descriptions and default values.

> ⚠ **CAUTION:** The Java code in a Function Rule is restricted to computations and limited types of side effects. You cannot perform stateful processing, use member variables, or extend classes in a Function. Consider hosting complex Java processing in a dedicated reusable service with a REST API, and integrate your application with that service by using the Connect REST Rule.

PEGA ©2025 Pegasystems Inc

Predefined Functions

Pega Platform includes many predefined Functions. To see if a Function you need has already been defined, you can use the Expression Builder to list the Functions that are defined in each library.

Offline Functions

Java Functions are not supported in offline mode. If you require an offline Function, you must create its JavaScript equivalent.

- **Creating a Function**

Creating a Function

You can create a Function to run Java code that you define. By putting your logic in a reusable Function, you can reduce the amount of custom code in your application.

1. In the Dev Studio header, click Create > **Technical** > **Function**.
2. On the **Create** form, enter values that define the context of the Function.
   a. In the **Add to ruleset** field, select the name and unlocked version of a Ruleset that stores the Function.
   b. In the **Library** field, press the Down Arrow key and select the library to contain the Function. The library must be defined in the same Ruleset as the Function.
3. Define the input parameters for the Function.
   a. In the **Parameters** section, click **+ Add new**.
   b. In the **Name** field, enter a unique identifier that you can reference from your Java code.
   c. In the **Java type** field, enter a primitive data type, a reference data type, or a Pega data type that your library supports. Do not use `PublicAPI` or `ClipboardPage`, because your application manages these types internally. Use `Activity` or `Page`, respectively.

©2025 Pegasystems Inc

> **NOTE:** You can define two or more Function Rules in one library with the same name, Ruleset, and version if their input parameters have differently ordered Java types. Standard Java overloading applies.

> **CAUTION:**
> - Review the number, order, and Java types of your input parameters, because you cannot change them after the Function has been created.
> - For input and output Java data types, always use primitive Java types and not boxed types. Using primitive types ensures that your Function Rule can be called from anywhere, including Expressions.

4. Click **Create and open** to open the Function Rule form.
5. In the **Output** section, in the **Java data type** field,either enter the return type: a primitive data type, a reference data type, or Pega data type that your library supports, or if there is no return type, enter `void`.
6. In the **Input parameters** section, customize the Function's input parameters.
   - In the **Name** field, you can override the name that you provided on the Create form.
   - If the Java type is String, then in the **Pega type** field, you can choose a data type so that additional validation is performed at run time. For other Java types, leave this field blank.
7. Complete various fields that are used for documentation purposes. These values make it easier for other developers to understand your Function, but they do not affect run-time behavior.
   - In the **Input parameters** section, enter values for the **Description** , **Page class** , and **In/out** fields. The **Page class** field is used only for ClipboardPage parameters.
   - In the **Output** section, enter values for the **Pega type** and **Page class** fields.

©2025 Pegasystems Inc

    - In the **Pega type** field, enter a value only if the **Java data type** field is set to `String` or a primitive data type.
    - In the **Page class** field, enter the class of the return value only if the **Java data type** field is set to `ClipboardPage` .
  - In the **Classification** section, enter values for the **Usage type** and **Description** fields.
  - To help with debugging, in the **Rule type parameters** section, enter one or more Rule Types.
8. Click **Save**.

- Creating a data object
- Creating library rules
- Function Rule

Function Alias Rules

A Function alias Rule has one of two types: Java function or SQL expression.

Depending on the alias type, different fields appear on the **Reference** and **Generated Output** tabs.

Java functions

Function aliases of this type provide natural-language descriptions and prompting for input parameters to an underlying function rule.

You need to create and test the function before defining a function alias rule that references it.

You cannot define an alias for a function that returns a complex Java type.

SQL expressions

Function aliases of this type provide compact packaging of complex SQL expressions that can be used to query a database. They are defined in the *Embed-UserFunction* class.

PEGA ©2025 Pegasystems Inc

You can use one of the SQL function aliases or configure a new one with the Calculation Builder.

Where referenced

Java alias functions can be used in collections, decision trees, when condition rules, Declare Expression rules, and constraints rules.

SQL alias functions rules support the development of report definitions.

Category

Alias function rules are instances of the *Rule-Alias-Function* class. They belong to the Technical rule category.

Library rules

A library rule is a container for a set of related function rules (the *Rule-Utility-Function* rule type). Create a library rule to hold custom functions for your application, which can supplant and extend standard functions.

The following tabs are available on this form:

- Packages
- Static Variables

- **Check-out** and **Restore** operations are not available for library rules, even for rulesets with the Use check-out? check box selected.
- Library rules are always available. You cannot change the availability of a library rule to `No/Draft Mode`, `Blocked`, `Withdrawn`, or `Final`. (However, you can change the availability of functions in the library, and then recompile the library.)

Where referenced

A library rule is the first key part of a function rule. Each reference to a function rule (a call or invocation) identifies the library to which the function rule belongs. You can

©2025 Pegasystems Inc

reference function rules in expressions and in Java code. Expressions are widely used in many rule types to compute property values.

Your system contains several standard libraries and hundreds of standard functions. For example, the standard library rule *DateTime* (in the Pega-RULES ruleset) contains several standard functions for date and time format conversions.

Access

Use the Records Explorer to list all library rules in your system.

Category

Library rules belong to the Technical category. A library rule is an instance of the *Rule-Utility-Library* rule type.

- **Creating library rules**
- Expressions

Creating library rules

Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a library rule in Dev Studio by clicking Create > **Technical** > **Library**.

Key parts

A library rule has two key parts:

©2025 Pegasystems Inc

Field	Description
Identifier	Enter a name for this library. Begin the name with a letter and use only letters and digits. The name must be a valid Java identifier; see How to enter a Java identifier. • Do not choose a name that matches a standard library unless you intentionally plan to override functions in that standard library. • The library name `Default` is deprecated and should not be used.
Ruleset Name	Select a ruleset name. A library is available to all versions of a ruleset. However, the functions in the library might differ from ruleset version to ruleset version. **NOTE:** You cannot create a new library rule in a branch ruleset.

Rule resolution

When searching for library rules, the system filters candidate rules based on a requestor's ruleset list of rulesets (but not versions).

NOTE: Time-qualified and circumstance-qualified rule resolution features are not available for library rules. The class hierarchy is not searched.

About Library rules

©2025 Pegasystems Inc

- Library rules

Messages

A message is dynamic text that communicates errors, warnings, or other types of information to users. By using messages instead of static text in your user interface, you can provide contextual assistance to users that supports localization.

> Message format

Each message has a definition that contains text and references to input parameters. In the following message definition, references to input parameters are in braces. The numbers correspond to the order in which you define the input parameters on the Message form.

```
The {1} must be later than {2} and at least 30 days before {3}.
```

When you call a message, you pass a tab-delimited string that starts with the message name and provides run-time values for the input parameters. You can pass literal values or property references to a message.

The following strings are example inputs to the DateRange message above:

- `DateRange\tstart date\ttoday\tthe end date`
- `"DateRange\tstartdate " + .StartDate + " \tJanuary 1, 2006\ttoday"`
- **Creating a message**
- **Finding messages in your application**
- **Creating a custom message category**
- **Displaying a warning message in a rule**

©2025 Pegasystems Inc

- **Standard functions for messages**
- **Updating legacy messages and related rules**
- Constellation in traditional UI

Creating a message

You can create a message to display text with run-time values to users. By using messages instead of static text, you can avoid maintaining a set of strings for each possible context and user locale in your application.

1. In the header of Dev Studio, click Create > **Technical** > **Message**.
2. In the **Label** field, enter text that describes the purpose of the message.
3. In the **Apply to** field, press the Down Arrow key and select the class that defines the scope of the message.
   The class controls which rules the message can use. It also controls which rules can call the message.
4. In the **Add to ruleset** field, select the name and unlocked version of a ruleset that stores the message.
5. **Optional:** To change the default identifier for the message, click **Edit**, and then provide a unique value in the **Identifier** field.
6. Click **Create and open** to open the Message form.
7. **Optional:** To include run-time values in your message, define input parameters.

   Do not define more than one **Raw** input parameter because the tab delimited items in this type can be interpreted as values for other input parameters. Always put this type of parameter in the last position of the parameter list.

   For more information about input parameters, see Defining the input parameters of a rule.

8. Click the **Details** tab.

©2025 Pegasystems Inc

9. In the **Category** list, select an option based on the type of information in the message and the expected response from the user.
10. **Optional:** If the **Category** field is set to **Warning**, provide information that applications use to compute their compliance scores.
    a. In the **Severity** list, select the degree to which an application is adversely affected by the conditions that trigger the warning.
    b. In the **Application Impact** list, select the aspect of the application that is affected the most by the conditions that trigger the warning.
11. If the **Category** field is set to **Error**, in the **Classification** field, select the type of error message.
    The classification is required for error messages used by automations; otherwise, it is optional. To create a new classification, create a field value of type *pyErrorClassification*.
12. In the **Message** field, enter the text of the message.

    You can reference input parameters by using the following format:

    ```
    Rule {1} was processed on the {2} thread with data from page
    {3}.
    ```

13. Click **Save**.
14. Click **Actions** > **Preview** to view your assembled message with default input parameter values.

> **What to do next:**
>
> You can use your message in validate rules, standard functions that support messages, and activities.

- Copying a Rule or data instance
- Creating a custom message category
- Messages

©2025 Pegasystems Inc

Finding messages in your application

You can use the Records Explorer to find the messages that your application defines. By understanding which messages exist and when they are called, you can quickly decide whether you need to create a new message.

1. In the navigation panel of Dev Studio, click **Records**.
2. Expand the **Technical** section and then click **Message** to open a list of message rules in your application.
3. Click the **Category** column to sort the messages.
4. Click a message to open the Message form.
5. On the **History** tab, review the information in the **Description** and **Usage** fields to understand the purpose of the message.
6. Click **Actions** > **View references** to display a list of rules that call this message.
7. Click a rule to open it, and then review the conditions that cause the rule to call the message.

- Creating a message
- Messages

Creating a custom message category

You can extend the standard categories for messages. By defining custom categories, you can create and classify messages in a way that is specific to your business.

1. Create a field value for your custom category.
   a. Click Create > **Data Model** > **Field Value** to open the Create form.
   b. In the **Label** field, enter `[custom category name]`.
   c. In the **Field Name** field, enter `pyCategory`.
   d. In the **Apply to** field, enter `Rule-Message`.
   e. Click **Create and open**.
2. Create a field value for the caption of your custom category.
   a. Click Create > **Data Model** > **Field Value** to open the Create form.

©2025 Pegasystems Inc

   b. In the **Label** field, enter `[custom category name]`.
   c. In the **Field Name** field, enter `pyCaption`.
   d. In the **Apply to** field, enter `Rule-Message`.
   e. Click **Create and open**.
3. To prompt users for more input when they select your custom category, create a custom section.
   a. Find and open the standard *Rule-Message.pyCustomDetails* section by searching for it or by using the Application Explorer.
   b. Click **Save as** > **Specialize by circumstance** to open the Specialization form.
   c. Click **Property and Date**.
   d. In the **Property** field, enter `pyCategory`.
   e. In the **Value** field, enter the name of your custom category.
   f. Click **Create and open** to open the Section form.
   g. Update the section with references to properties in the *Rule-Message* class, so that it collects information that is relevant to your custom category. For example, you can add a text box to a layout in the section and then associate it with a property that stores text.
   h. Click **Save**.
4. To validate the values that users provide at run time, create validation logic that is specific to your custom category.
   a. Find and open the standard *Rule-Message.pyValidateFields* edit validate rule by searching for it or by using the Application Explorer.
   b. Click **Save as** > **Specialize by circumstance** to open the Specialization form.
   c. Click **Property and Date**.
   d. In the **Property** field, enter `pyCategory`.
   e. In the **Value** field, enter the name of your custom category.
   f. Click **Create and open** to open the Edit Validate form.
   g. Add criteria to the rule that evaluates the properties that your custom section references.
   h. Click **Save**.

©2025 Pegasystems Inc

Result:

You can test your changes by creating a message and verifying that your custom category is one of the available options.

- Creating a message
- Creating Sections
- Messages

Displaying a warning message in a rule

You can display a warning message in a rule based on criteria that you define. By calling attention to configurations that do not follow best practices at design time, you can help users improve their compliance score before they take their application to production.

1. Open a warning message that matches your use case by doing one of the following:
   a. Find an existing message. The message **Category** must be set to **Warning**.
   b. Create a warning message. For more information, see Creating a message.
2. On the Message form, take note of the message definition and the type of input parameters that the message supports, because you need this information when you call the message from another rule.
3. Find and open the rule that detects the warning conditions, by searching for the rule or using the Application Explorer.
4. Update the rule to call the standard *pxAddGuardrailMessage* function. The syntax for calling a function can vary, based on the type of rule that you are updating.
5. Pass a parameters to the function, by using a tab-delimited string that begins with your message name and ends with values for input parameters. For examples of correctly formatted strings, see Messages.
6. Click **Save**.

©2025 Pegasystems Inc

Result:

At run time, your application displays warning messages in the header of a rule form. Users can choose to follow the guidance in the message to resolve the warning, provide an explanation that justifies the warning, or ignore the warning.

- Creating a custom message category

Standard functions for messages

You can use standard functions to evaluate messages in your application. By understanding the errors and warnings that are associated with your application, you can debug issues quickly and improve your guardrail compliance score.

The following standard functions are supported:

- *getRuleMessage(tools, messageName)* — Returns the localized text of a message.
- *pyGetUnlocalizedRuleMessage(className, ruleMessageName)* — Returns the original text of the message.
- *hasMessages(stepPage)* — Determines whether a clipboard page has messages associated with it.
- *hasMessagesOnGivenProperty(myStepPage, myPropertyReference)* — Determines whether a property has messages associated with it.

- Creating a message
- Messages

Updating legacy messages and related rules

You can update your legacy messages so that they are compatible with the latest definition of the *Rule-Message* class. By providing values for the new properties that all

©2025 Pegasystems Inc

messages must support, you can ensure that your legacy messages and the rules that rely on them continue to function as expected.

> **NOTE:** The following steps apply only to messages that were created in Pega Platform version 7.1.6 or earlier.

1. Find the messages that are in your application.
2. In the **Instances of Message** list, click the **Category** column to sort the results.
3. Ensure that all messages are assigned a category, because blank values in user prompts can lead to confusion or unexpected results. Repeat the following steps for all messages that do not have a category defined:
   a. Click a message in the list to open the Message form.
   b. In the **Category** list, select an option based on the type of information in the message and the expected response from the user.
   c. Click **Save**.
   d. Close the Message form to return to the list of results.
4. Click the **Class Name** column to sort the results.
5. Review the messages in the *@baseclass* class and decide whether they can stay in this class or require a different scope. You can change the scope of a message by creating a new version in a different class. For more information about creating rules, see Creating a specialized or circumstance rule.
6. Update the activities that are affected by the latest definition of the *Rule-Message* class.
   a. In the search tool, select the **All Content** and **Contains** options to find and open activities that use the *addWarning* function, *Property-Set-Messages* method, or *Page-Set-Messages* method.
   b. On the **Steps** tab of the Activity form, replace references to *addWarning* with *pxAddGuardrailMessage*, because this function is deprecated.
   c. Expand the steps that set a message.
   d. For each step, verify that the class of the page in the **Page** field is the same class as the message the activity sets, because your application uses this

©2025 Pegasystems Inc

class to find the correct version of the message at run time. If there is no value in the **Page** field, you can provide one or leave it blank to instruct your application to use the step page at run time.

e. Click **Save**.

- Creating a message
- Messages

Property aliases

A property alias provides an alternative name for a property in a function alias that is more meaningful to business users.

> **NOTE:** This rule type is deprecated.

Fields and controls

The following table describes the fields and controls that you can use on the Property Alias form.

Field	Description
Alias Reference	Enter a period followed by the property name (second key part) of the property being aliased. Identify a `Single Value` property in the class that is the first key part of this property alias rule (or a subclass). You can create multiple property alias rules that reference a single property.
Resulting Property Type	Optional. Specify a Type value other than the Type in the property, to cause automatic

©2025 Pegasystems Inc

Field	Description
	conversion (typecasting) of the value. Only certain typecast conversions are supported.
Category	Optional. To group several property alias rules for purposes of user selection, select a category value. Select `Expression` as the Category if these conditions are met: • This property is computed by a Declare Expression rule from other properties in the same class or a higher class. • The property is not exposed as a column in the database, and you want to make the property available in reports. The system maintains values for the Category field as field value rules. To add a category, create a field value rule using: • `Rule-Obj-Property-Alias` as the first key part • `pyCategory` as the second key part • The category value as the third key part

Property Qualifier rules

A property qualifier rule can supplement information in a property with additional capabilities or restrictions. Property qualifier rules are referenced in a property in the Property Qualifiers section of the property's Advanced tab.

PEGA ©2025 Pegasystems Inc

The following tabs are available on this form:

- Definition
- Table Edit

For example, the *pyDecimalPrecision* property qualifier rule affects how the system rounds arithmetic computations that involve the properties that reference it.

Access

Use the Records Explorer to list all property qualifier rules available to you.

Category

Property qualifier rules are part of the Data Model category. A property qualifier rule is an instance of the *Rule-Obj-Property-Qualifier* rule type.

- **Property Qualifier rules - Completing the New or Save As form**

Property Qualifier rules - Completing the New or Save As form

Property Qualifier rules - Completing the Create, Save As, or Specialization form .

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a property qualifier rule by selecting `Property Qualifier` from the `Data Model` category.

Key parts:

A property qualifier rule has two key parts:

©2025 Pegasystems Inc

Field	Description
Apply to	Select a class to which this rule applies. Choose *@baseclass* to make the qualifier available to properties of any class. The list of available class names depends on the Ruleset that you select. Each class can restrict applying Rules to an explicit set of Rulesets as specified on the **Advanced** tab of the class form.
Identifier	Enter a qualifier name that starts with a letter and includes only alphanumeric characters.

Rule resolution

About Static Content Bundle rules

Static content bundle rules define a group of JavaScript files or Cascading Style Sheet (CSS) files, identified as text file rules, that the server transmits to workstation browsers as a single unit or "bundle". Used appropriately, static content bundles can reduce HTTP network traffic and reduce user response times.

The following tabs are available on this form:

- Definition

Create a static content bundle rule if your application contains multiple text file rules containing JavaScript or CSS files that are always, or almost always, needed together to support a browser display or function. See More...

Where referenced

Static content bundle rules are referenced by:

©2025 Pegasystems Inc

- An HTML rule, HTML fragment rule, or other stream rule that contains the <pega:static > JSP tag.
- Harness rule form Scripts and Styles tab.

Access

Use the Records Explorer to list all static content bundle rules available to you.

Category

Static content bundle rules are instances of the *Rule-File-Bundle* class. They are part of the Technical category.

- **Static Content Bundle rules - Completing the New or Save As form**
- **Static Content Bundle rules - Completing the Definition tab**
- **More about Static Content Bundle rules**

Static Content Bundle rules - Completing the New or Save As form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a static content bundle rule by selecting `Static Content Bundle` from the `Technical` category.

Key parts

A static content bundle rule has two key parts:

©2025 Pegasystems Inc

Field	Description
Identifier	Enter a name for this bundle. Start the name with a letter and use only letters, digits, and underscore characters.
Bundle Type	Select `script` if the bundle is to contain text file rules containing JavaScript code. Select `styles` if the bundle is to contain text file rules containing CSS styles.

Rule resolution

When searching for static content bundle rules, the system filters candidate rules based on a requestor's RuleSet list of RuleSets and versions.

Time-qualified and circumstance-qualified rule resolution features are not available for static content bundle rules. The class hierarchy is not relevant to rule resolution of static content bundle rules.

About Static Content Bundle rules

- About Static Content Bundle rules
- More about Static Content Bundle rules

Static Content Bundle rules - Completing the Definition tab

Complete this tab to list the text file rules in this bundle. A bundle must contain only JavaScript files (where the third key part is `js` or `htm`, or only CSS files (where the third key part is `css`)

At runtime, the system uses normal rule resolution with the requestor's RuleSet list to locate the text file rules in a bundle. The rules you list here can belong to various RuleSets and versions.

PEGA ©2025 Pegasystems Inc

Include these files

Field	Description
	List the distinct members of this bundle. Order is not significant.
App Name	Identify the directory name that forms the first key part of a text file rule to be included in this static content bundle (typically `webwb`).
File Name	Identify the middle key part of a text file rule.
File Extension	Identify the final key part of a text file rule, typically `js` , `htm` , or `css` .

About Static Content Bundle rules

- About Static Content Bundle rules
- Static Content Bundle rules - Completing the New or Save As form
- More about Static Content Bundle rules

More about Static Content Bundle rules

Static content bundle rules define a group of JavaScript files or Cascading Style Sheet (CSS) files, identified as text file rules, that the server transmits to workstation browsers as a single unit or "bundle". Used appropriately, static content bundles can reduce HTTP network traffic and reduce user response times.

When to create a static content bundle rule

Create a static content rule if your application RuleSets contain multiple text file rules that are needed together to render or support a browser window. You can continue to update the text file rules (in the same or higher versions).

PEGA ©2025 Pegasystems Inc

For example, if your application includes a JavaScript function ALPHA that calls other functions BETA and GAMMA, you can choose to send all three files to the browser as soon as it requests any one.

Static content can be incorporated in any stream rule, supporting gadgets, and portal rules.

Creating a bundle rule

To bundle multiple text files rules into a single static content response:

1. Identify the static files (all of a single type, CSS or JS) to be included
2. Create a bundle rule using the Static File Bundle form.
3. Create an HTML fragment rule that uses the static JSP tag (and one or more bundle JSP tags) to identify the bundle.
4. Update a stream rule to include the HTML fragment, with the <pega:include > JSP tag or by another means.
5. Test. Use Fiddler2 or similar HTTP debugging tools to confirm that a browser receives the entire bundle in a single response.

Whenever possible, maximize use of auto-generated HTML code even when adding a static file bundle to a stream rule. For example, you can identify an HTML fragment in a portal rule by name, letting PRPC auto-generate the <pega:include > JSP tag.

- About the static JavaServer Page tag
- About Static Content Bundle rules
- Static Content Bundle rules - Completing the New or Save As form

Text File rules

Use a text file rule to hold and identify static HTML, XML, Cascading Style Sheet, DHTML, and JavaScript script files used in an application. This rule provides the security, inheritance versioning, and deployment benefits of rule resolution to the file. (A static file in this context is one that does not change each time it is served. The contents of

©2025 Pegasystems Inc

static files may differ from RuleSet version to RuleSet version, and may be updated after a Rule-File- rule is updated.)

The following tabs are available on this form:

- Main

A text file rule containing HTML must define an entire static page with pure HTML, containing no reference to the Pega Platform JSP tags or directives. In contrast, an HTML Fragment rule can contain JSP tags or directives.

Where referenced

Any rule type that contains XML or HTML code can reference a text file rules, using the pega:include JSP tag.

In addition, text file rules for styles (CSS) or JavaScripts (JS) can be referenced in a static content bundle rule (*Rule-File-Bundle*) to reduce the number of HTTP interactions between a browser session and the Pega Platform server. Such text file rules can also be included in a static content bundle with the `static` JSP tag.

The first time a requestor asks for this text file, the Pega Platform extracts the text file from the rule into an appropriate Web server directory on the server.

Access

Use the Records Explorer to list all text file rules available to you.

Category

Text file rules are instances of the *Rule-File-Text* class. They are part of the Technical category.

- **Text File rules - Completing the Create, Save As, or Specialization form**

©2025 Pegasystems Inc

Text File rules - Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a Text File rule by selecting `Text File` from the `Technical` category.

Key parts

A Text File rule has three key parts:

Field	Description
App Name (Directory)	Enter the name of the Web server directory to place this file when it is extracts from the database. Subdirectories specific to the RuleSet and version are created to hold the extracted text file.
Identifier	Enter the name you choose for this file. Choose a name that is valid for both UNIX and Windows files. Use only lowercase letters.
File Type (extension)	Enter an extension for this type of file, with no period. For example enter css for Cascading Style Sheets, js for JavaScript, or htm for (static, pure) HTML. Enter htc for Dynamic HTML (DHTML) files. The list of Windows file types that you can enter here is defined through a standard

©2025 Pegasystems Inc

Field	Description
	map value named *Rule-File-Text.RuleFileTextTypes*. You can add to this list by overriding that rule.

Rule resolution

When searching for text file rules, the system filters candidate rules based on a requestor's RuleSet list of RuleSets and versions.

Time-qualified and circumstance-qualified rule resolution features are not available for text file rules. The class hierarchy is not relevant to rule resolution of text file rules.

Defining URL patterns for Cases

Share your work items with users of the same application by defining URL patterns for the work items in your application, for example, for Cases or other work items. Simple, meaningful URLs make collaboration more efficient for users. For example, they can bookmark a Case for quick access, and then send a direct link to the Case to other users.

Before you define URL patterns for work items, your application must contain a URL Mappings Rule. Each application can have only one URL Mappings Rule, whose name is always *pyDefault*. You define URL patterns by editing this Rule.

> **NOTE:** This procedure focuses on configuring URL patterns for Cases. Configuring URL patterns for other objects, such as reports and landing pages, requires advanced configuration and creating your custom activities.

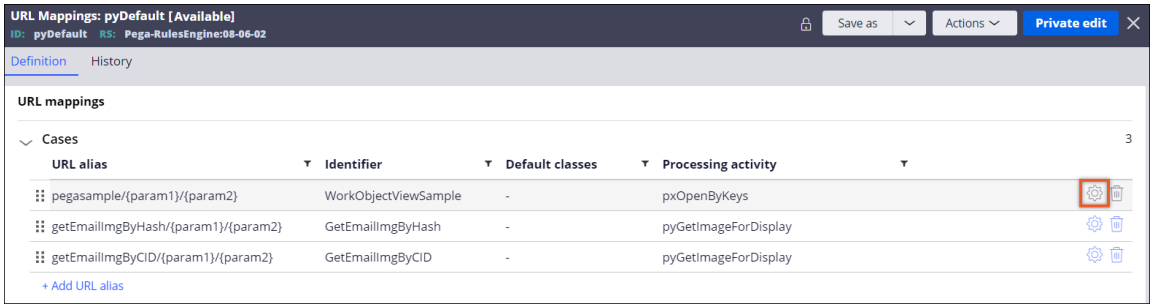
1. Create a URL Mappings Rule in your application:

©2025 Pegasystems Inc

> **NOTE:** If your application already contains a URL Mappings Rule, go to step 2.

   a. In the header of Dev Studio, click **Create** > **Technical** > **URL Mappings**.
   b. In the **Context** section, in the **Add to ruleset** list, select the name and version of a Ruleset to store the URL Mappings Rule.
   c. Click **Create and open**.
   d. Click **Save**.

2. In the navigation pane of Dev Studio, click **Records**.
3. Expand the **Technical** category, and then click **URL Mappings**.
4. In the list of URL Mappings instances, click *pyDefault*.
   Each application has only one valid URL Mappings Rule, whose name is always *pyDefault*.
5. In the **Cases** section, start configuring a URL pattern:
   - To modify an existing pattern, click the **Switch to edit mode** icon, as shown in the following figure:



Switch to edit mode icon on the URL Mappings form

   - To add a URL pattern to a work item, click **Add URL alias**.
6. In the **Identifier** field, enter a name for the URL alias that uniquely identifies it in all patterns in the Rule.

> **For example:**
>
> Enter `OpenWorkForReview`.

©2025 Pegasystems Inc

7. **Optional:** To generate a valid URL for an object, apply the URL pattern to the object:
   a. Select the **Map path elements for URL generation** checkbox.
   b. In the **Class** field, specify the class of the object for the properties mapping.
8. In the **Path element type** column, select **Constant**.
   The first path element must be a constant.
9. In the **Value** field, enter text that is a constant element in the URL pattern.

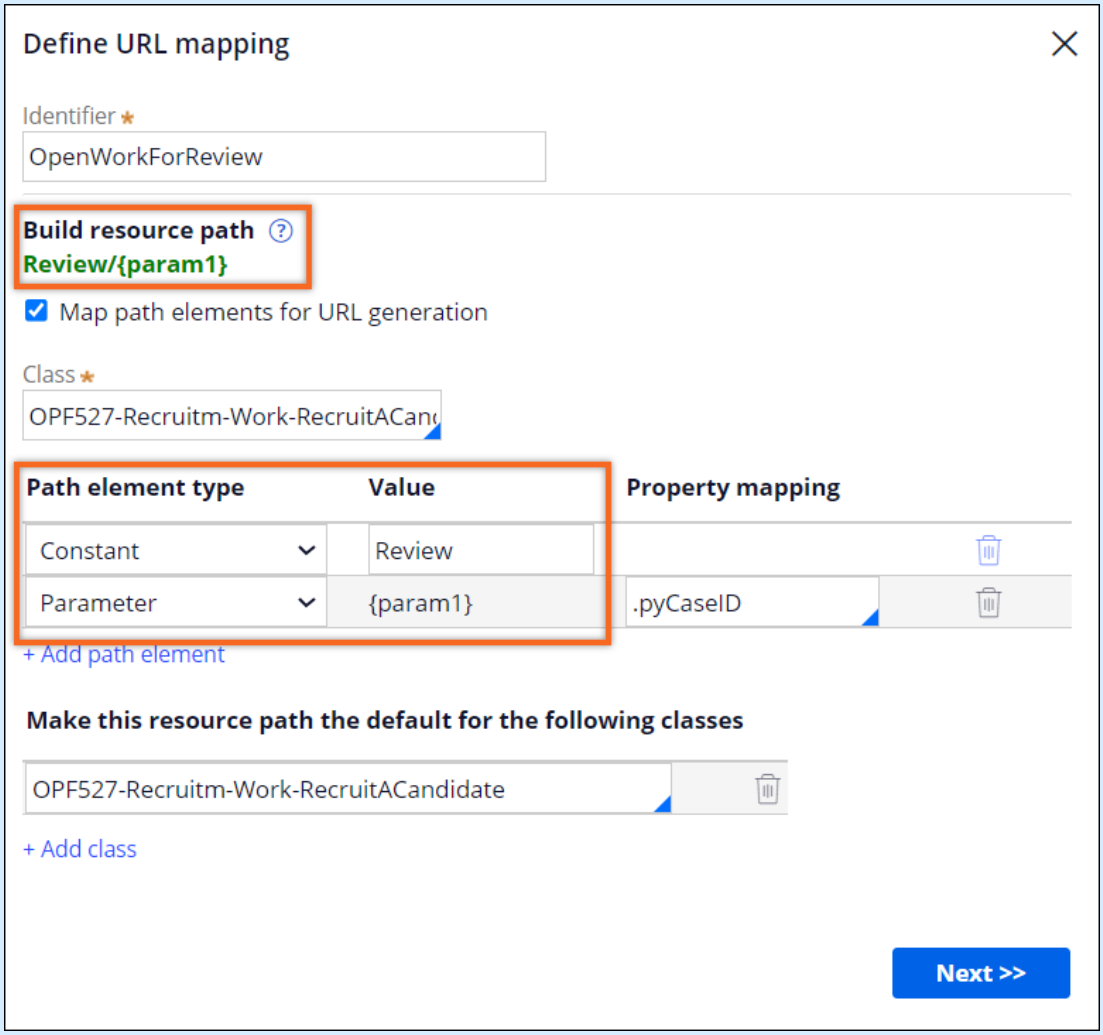
   For example:

   Enter `Review`.

10. **Optional:** To add more path elements to the URL pattern, click **Add path element**.
    a. To define text, in the **Path element type** column, select **Constant** and then, in the **Value** field, enter the text.
    b. To define a parameter, in the **Path element type** column, select **Parameter**.
    c. If you selected **Map path elements for URL generation** in step 7, in the **Property mapping** column, enter a clipboard property to map to the URL path element.

NOTE: You can preview the URL pattern that you create below the **Build resource path** label, as shown in the following figure:

©2025 Pegasystems Inc



Preview of a resource path and the configured path elements

11. **Optional:** To make this URL pattern the default for objects in specific classes, in the **Make this resource path the default for the following classes** section, click **Add class**, and then in the text field, enter the name of the class.
The class can be the same class that you entered in step 7, or any of the child classes.

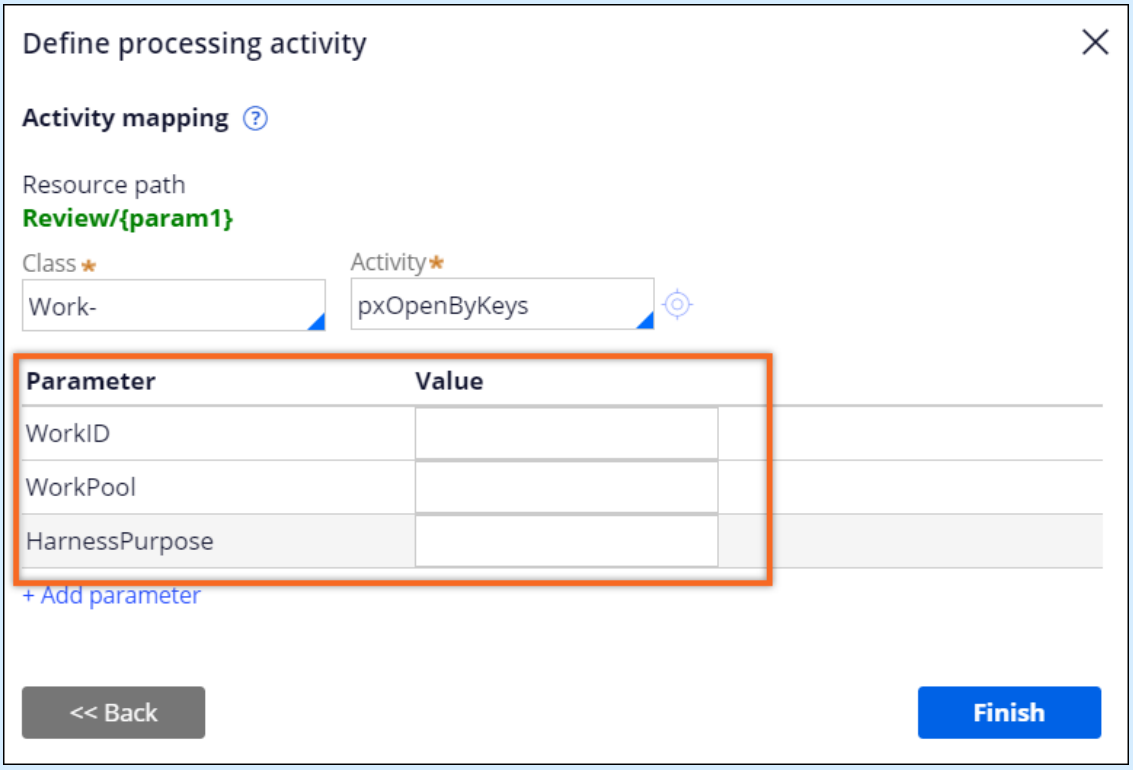
> **NOTE:** The option is available only if you selected **Map path elements for URL generation** in step 7.

©2025 Pegasystems Inc

12. Click Next.
13. In the **Define processing activity** dialog box, in the Class field, enter `Work-`, and then, in the Activity field, enter `pxOpenByKeys`.

Result:

The system identifies activity parameters and displays them in the table, as shown in the following figure:



Mapping an activity to a URL mapping Rule

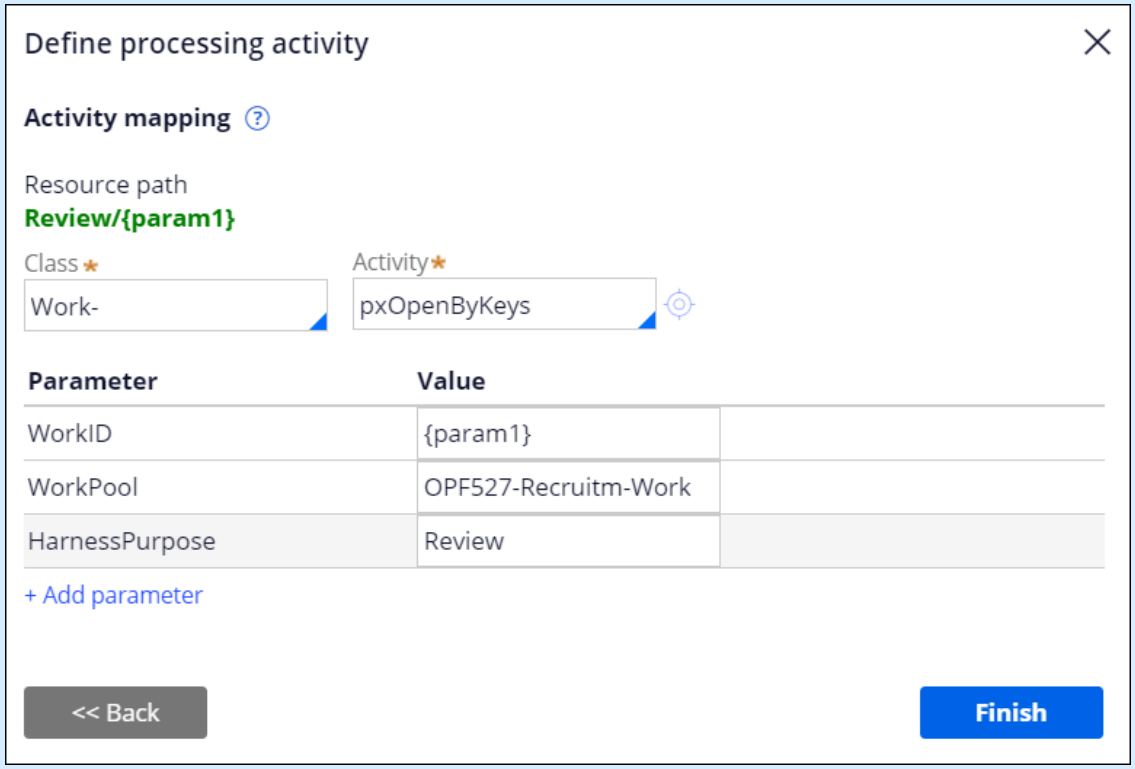
14. In the **Value** column, enter the values for the processing activity parameters. Map your URL path elements (parameters) to the processing activity parameters.

For example:

In step 10, if you defined a URL path element as the `{param1}` parameter, in the Value field for an activity parameter, enter `{param1}`. For example, In the Value field for the `WorkID` parameter, enter `{param1}`. As a result, because `{param1}` is assigned to the `pyCaseID` property, at run time the

©2025 Pegasystems Inc

URL contains the Case ID. The following figure shows parameters with sample values:



Mapping a processing activity to the URL mapping

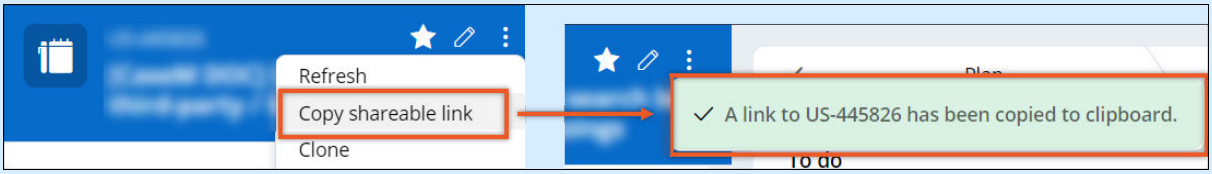
15. **Optional:** To pass an additional parameter to the activity that the current activity invokes, click **Add parameter**, and then repeat step 14.
16. Click **Finish**.
17. **Optional:** To add a URL pattern to another category, expand the category, and then repeat steps 5 through 16.
18. On the **URL Mappings** form, click **Save**.

NOTE: If you make any changes to an activity referenced in a URL Mapping, you must re-save the individual URL Mapping. This step is necessary irrespective of any change in the individual URL Mappings rule.

©2025 Pegasystems Inc

For example:

You can use the mappings to generate URLs that follow the patterns that you define by configuring a **Copy shareable link** button in a case. The following figure shows the results of the configuration at run time:



Copying a simplified URL for a case

For more information, see Theme-Cosmos: Open work object in Full portal on Copy sharable Link.

Related information

- https://support.pega.com/support-doc/url-mappings-troubleshooting-faqs

Function Alias rules — Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

1. Create a function alias rule by selecting **Function Alias** from the **Technical** category.

Key parts:

2. From the **Apply to** field, select a class.
   Many function alias rules use *@baseclass*.

©2025 Pegasystems Inc

To create an SQL function alias rule, select or enter `Embed-UserFunction`. Specifying any other class creates a Java function alias rule.

3. In the **Identifier** field, specify a name for the rule.
   Begin the name with a letter and use only letters, numbers, the ampersand character, and hyphens.

The following options listed in the **Additional creation options** area (referred to as Quick Create options) are are optional. Selections made here can be used as a starting point for the rule you are creating:

4. **Optional:** From the **Alias type** field, select an alias type to preset the Java function name, template and input parameters for your newly created Functional Alias rule:
   - To set the Java function name to `compareTwoValues()`, choose **Compare two values**.
   - To set the Java function name to `compareTwoValues()`, choose **Is Property Equal** and update the template to include a specified property.
   - To set the Java function name to `dateIsInRange()`, Choose **Date within Date Range**.
   - For all SQL function alias rules and for most Java function alias rules, choose **Other**.
5. In the **Property** field, specify the property name that will be used as the `{lValue}` in the `compareTwoValues()` function.
   This field is only available when selecting an alias type of **Is Property Equal To**.

Result:

When searching for instances of this rule type, the system:

- Filters candidate rules based on a requestor's ruleset list of rulesets and versions

©2025 Pegasystems Inc

- Searches through ancestor classes in the class hierarchy for candidates when no matching rule is found in the starting class
- Finds circumstance-qualified rules that override base rules with the same key

Time-qualified resolution features are not available for this rule type.

- Function Alias Rules

About Image Content data instances

Image Content instances provide an alternative to binary file rules, to store images that:

- are widely used across multiple applications
- look the same for all users, and
- are rarely updated.

For example, you can use an Image Content data instance to identify each division in an organization.

NOTE: When uploading image files to Pega Platform™, ensure that any embedded Exif metadata is removed or sanitized before upload. Exif data can contain sensitive information such as GPS coordinates, timestamps, and device details, which may pose privacy or security risks if exposed. Pega does not automatically strip Exif metadata from uploaded images. It is recommended to preprocess images using trusted tools or libraries to remove metadata before storing them in the application.

The following tabs are available on this form:

- Main

©2025 Pegasystems Inc

Where referenced

You can reference data content image instances in two ways. The first is to write out a reference to the image cleartext URL in HTML code:

```
<img src="../datacontent/Image/filePath/fileName.fileType">
```

This URL is consistent and reusable.

The second way is by using the contentURL JSP tag, which will generate the same cleartext URL as if one had written it as a cleartext URL:

```
<img src="<pega:contentURL name="fileName.fileType path="/filePath/" classType="I
mage" />">
```

The contentURL JSP tag creates a constant, reusable cleartext URL.

For example: To reference an image with the file name "profilepicture," the file path "/operatorimages/" and the file type "jpg," you could do this in one of two ways:

<img src="../datacontent/Image/operatorimages/foo.jpg">

or

<img src="p:contentURL name="foo.jpg" path="/operatorimages/" classType="image"/>"/>

Both of these references generate the same cleartext URL for the image.

Category

An Image Content data instance belongs to the *Data-Content-Image* class. They are part of the Technical category.

©2025 Pegasystems Inc

Related tasks

- Configuring the Attach content control

Image Content form - Completing the New or Save As form

Image Content data instances – Completing the New or Save As form

Create an Image Content data instance by selecting `Image Content` from the `Technical` category.

An Image Content instance has three key parts:

Field	Description
File Name	Enter the name of this file. Choose a name suitable for UNIX or Windows or both.
File Type (extension)	Enter the extension for this type of file. Only the following types are allowed: Portable Network Graphics (.png), Graphics Interchange Format (.gif), or Joint Photographic Experts Group (.jpg or .jpeg).
File Path	Enter a name starting and ending with a forward slash, and containing only forward slashes, digit, and letter between slashes. You can choose names that group related image content instances, for example: /employees/ /2009/ /logos/2008/

©2025 Pegasystems Inc

About Image Content data instances

- About Image Content data instances

Image Content data instances - Completing the Main tab

Image Content data instances – Completing the Main tab.

Complete this tab to:

- Upload an image from your workstation or network into the data instance.
- Download an image from the data instance to your workstation.

Image Preview

Field	Description
Image Preview	The form contains a preview of the image and indicates the width and height of the image in pixels.
Upload File	Click to upload into this data instance an image on your workstation or local area network. Locate the file, and click **Upload File**. A message box confirms when the upload completes. Save the Image Content form.
Download File	Click to download the image associated with this rule to your workstation or local area network.

References to images

Reference image content in HTML for display using a cleartext URL or the contentURL JSP tag.

©2025 Pegasystems Inc

Integration Connectors landing page

Access the Integration Connectors landing page by clicking **Configure** > **Integration** > **Connectors**.

The **Connectors** tab helps you define and manage interfaces between a Pega Platform application (acting as a client) and external systems (acting as servers).

At the top of the display are the following buttons:

- **New Connector Data Source** — Click to identify a new data source. Select from the options presented:
  - **Create SOAP Integration** — Select to create a new SOAP integration. See Creating SOAP integration.
  - **Create REST Integration** — Select to create a new REST integration. See Creating REST integration.
  - **Create SAP Integration** — Select to create a new SAP integration. See Creating SAP integration.
  - Connector and Metadata wizard — Select to open the Connector Wizard to review existing connectors. Click New at the top right to open the Connector and Metadata wizard, where you can specify a new connector. See Connector and Metadata wizard help.
- **Disable Simulations** — Developers create connector simulators to test connect rules in flows before the real data source is implemented, or when it is not available. See Creating connector simulators. Click this button to disable existing simulations.
- **Enable Simulations** — If you have disabled simulations by clicking the "Disable" button, as described above, the button label changes to "Enable Simulations". Click to enable existing simulations.
- **Clear Simulations** — Click to remove all simulations from the system.

PEGA ©2025 Pegasystems Inc

The tab displays a count of connectors by category, and how many simulations in each category are active. Click a category name to see a list of connectors for that category, and click an entry in the list to display the connector rule form.

The following rule types can implement a connector:

Connector Type	Description
BPEL	About Connect BPEL rules
Cassandra	About Connect Cassandra rules
CMIS	About Connect CMIS rules
dotNet	Connect dotNet rules
EJB	About Connect EJB rules
File	Connect File rules
HBase	About Connect HBase rules
HTTP	About Connect HTTP rules
Java	About Connect Java rules
JCA	Connect JCA rules
JMS	About Connect JMS rules
MQ	About Connect MQ rules
REST	About Connect REST rules
SAP	About Connect SAP rules
SAPJCo	Connect SAPJCo rules
SOAP	About Connect SOAP rules
SQL	About Connect SQL rules

NOTE: The rule type Rule-Connect-BPEL is deprecated. Use Service SOAP rules instead.

©2025 Pegasystems Inc

- **Integrating your application with external resources**
- **Data mapping in services and connectors**

Integrating your application with external resources

Most applications require access to data or computations that are provided by another system, or applications need to respond to requests from other systems for data or computations. The approaches, technologies, and facilities that support accessing data are collectively known as integration services.

Integration services in the *Pega-IntSvcs* ruleset include:

- Connector capabilities, which allow your application (in the role of client), to request data or services from another system (in the role of server).
- Service capabilities, which allow your application (as a server) to respond to requests it receives from another system (a client).

Connectors

Rule types provide direct support for the following protocols and technologies:

- ATOM syndicated feeds — About Connect ATOM rules
- Big Data
  - Connect Cassandra rules
  - Connect HBase rules
- BPEL — About Connect BPEL rules

> **NOTE:** The rule type Rule-Connect-BPEL is deprecated. Use Service SOAP rules instead.

- Content management systems
- Microsoft .Net
- Enterprise JavaBeans

©2025 Pegasystems Inc

- File output
- HTTP messages (no SOAP wrapper)
- Java classes
- Java Common Connector Interface
- Java Message Services — About Connect JMS rules
- IBM MQSeries messaging — About Connect MQ rules
- WSDL-based Web services — About Connect SOAP rules
- Relational databases through SQL — About Connect SQL rules
- Robotic desktop automations - About Connect Robot rules

Services

Similarly, rule types for services cover the following protocols and technologies:

- BPEL — The rule type Rule-Service-BPEL is deprecated. Use instead.
- Microsoft COM
- Microsoft .Net
- Enterprise JavaBeans
- Incoming email — About Service Email rules
- Input files — About Service File rules
- HTTP message (no SOAP wrapper)
- Plain old Java — About Service Java rules
- Java Message Service messaging — About Service JMS rules
- JSR-94 API — Service JSR94 rules (to be deprecated)
- IBM MQSeries messaging — About Service MQ rules
- REST — About Service REST rules
- JSR-168 portlets
- Web Services — About Service SOAP rules

How connectors work

Connector interfaces consist of a call or outgoing message (known as the request), followed by a return or arriving message, known as the response. You can parse, convert, and map data in either direction to or from the clipboard.

PEGA ©2025 Pegasystems Inc

Arriving information can be an XML file format (and accepted by the Parse XML rule), in a fixed record structure (accepted by the Parse Structured rule), or a text file with input fields separated by a tab character or other specific characters (accepted by the Parse Delimited rule).

Your flows can include Integrator shapes, which execute activities that use connector rules to gather data or request processing from another system.

For most connector types, you can simulate the operation of a connector before you build the connector. This allows your application development and testing to proceed when the external system is unavailable or is difficult to test with. See Creating connector simulators.

Mapping and Resources

Several rule types facilitate two-way mapping between property values (on a requestor clipboard) and the messages, records, or structures used by the external system or technology. These rule types belong to the Integration-Mapping category. See Data mapping in services and connectors — Concepts and terms.

Resource identifiers, such as URLs, port numbers, user names, and passwords, might vary between a development or test environment and a production system, and might change during the operation of a production system. Such information is usually better stored in data instances, rather than rules. The data classes belong to the Integration-Resources category.

Data mapping in services and connectors

Pega Platform uses the term data mapping for the task of converting data received from, or sent to, an outside source or system into the formats, values, and objects needed within an application.

Most system interfaces, known as connectors and services, require data mapping in both directions, for the request and the response.

PEGA ©2025 Pegasystems Inc

Typically the source (of outgoing data) and the destination (of incoming data) is a property on a clipboard page. For a developer, establishing the relationships among the external formats and structures and internal formats and structures can be laborious, exacting, and tedious. When dealing with legacy external systems, differences in character sets, the endian orientation of binary values, the representation of numbers and dates or times, and so on add to the complexity of data mapping.

> **NOTE:** For more information about data mapping of SOAP messages, see Data Mapping XML, a document available from in the Integration area of Pega Community.

Pega Platform offers several features that can simplify and speed this development task, including:

- Eleven property modes and eleven type values, accommodating needs for dates, times, numbers, and aggregates — arrays or repeating groups.
- A flexible and powerful multilevel clipboard structure that supports embedded pages many levels deep.
- A guided process for the development of connectors and conversion of WSDL files, database schema, and XML schema. See Using the Connector and Metadata wizard .
- Direct execution of SQL statements, including stored procedures, through Connect SQL rules.
- Automatic parsing of XML in arriving XML documents. See About Parse XML rules.
- Automatic parsing of HTTP Post messages (sometimes called Common Gateway Interface query strings).
- Data conversion routines provided by Edit Input rules. See About Edit Input rules.
- Two parse rules exist for converting fixed-format records or string values into distinct property values. For simple cases, see About Map Structured rules. For more complex cases, see About Parse Structured rules.
- A similar parse rule for converting tab-delimited input files into individual properties. See About Service File rules and About Parse Delimited rules.

PEGA ©2025 Pegasystems Inc

- Extensible libraries of functions for mapping, one each for inbound mapping (the MapTo library) and outbound mapping (the MapFrom library)

Integration Services landing page

The Integration Services landing page helps you create and review services in your application. Access it by selecting the **Configure** > **Integration** > **Services** menu item. Click the **Service Wizard** button to launch the Service Wizard.

The Service Wizard creates service rules and supporting resource data instances for service types. Work items are created using the prefix `pxW-`.

- Integrating your application with external resources
- Mapping an endpoint to a cross-origin resource sharing policy

Connected apps landing page

You can establish a connection between your application and a service provider for automated cases processing using the Connected apps landing page. For example, you can use Box to store attachments for inbound and outbound automated email processing. The applications listed on the Connected apps landing page use OAuth 2.0 and have a grant type of authorization code.

You can also disconnect applications from the service provider on this landing page. Disconnected apps are not authorized to access the service provider and the applicable use cases will fail. For example, automated email processing of attachments will fail.

If you are manually processing cases, you can connect to a service provider at runtime and do not need to connect using the Connected apps landing page. For example, you can connect to Box when attaching or storing a document to Box.

You can view the Connected apps landing page either as a table or a grid by clicking the appropriate icon in the upper right corner. The Connected apps landing page has the following information:

PEGA ©2025 Pegasystems Inc

Column heading	Description
Name	Specifies the name of the profile. Click on the name to view or edit the profile.
Description	Displays the profile description.
Service provider	Displays the service provider. For example, Box. Google, and so on.
Authorization expires	Displays the amount of time until the app's authorization token expires. This field is blank if a user has not connected.
Status	Displays the status of the app, either **Connected** or **Not connected**.
Refresh token	If a refresh token is being used to refresh the authorization token, displays **enabled**, otherwise, the field is blank.
Action	Displays either of the following options: • **Connect** - Click to connect to the service provider and authorize the app. • **Disconnect** - Click to disconnect from the service provider and deauthorize the app.

- Configuring a REST connection

About Collection Rules

Use a collection to model business logic and implement common Rule engine patterns. Collections define an ordered sequence of Rules, the conditions under which they execute, and postprocessing steps known as response actions. The business-friendly options on this form help you to quickly develop flexible solutions that are easily understood by various audiences.

PEGA ©2025 Pegasystems Inc

The following tabs are available on this form:

- Rules
- Pre / Post Actions

Where referenced

Collections can be called by a Collect instruction in an activity or a step in another collection Rule. Collections can also be associated with specifications.

Access

Use the Application Explorer to access collections that apply to a specific class in your application. To list all available collections, see Records Explorer.

Enable the **Declare Collections** setting to view collections output in a Tracer session.

You can view the generated Java code of a Rule by clicking **Actions** > **View Java**. You can use this code to debug your application or to examine how Rules are implemented.

Category

Collection Rules are instances of the *Rule-Declare-Collection* class. They are part of the Decision category.

- **Collection rules**
- **Completing the Basic/Advanced Collection tab**
- **Completing the Preamble and Stop tab**
- **Completing the Specifications tab**
- **Using conditions in a collection**
- **Setting the context of a collection step**

PEGA ©2025 Pegasystems Inc

- **Using response actions in a collection**
- **Rule type**
- **Unit testing a collection**

Collection rules

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a Collection rule by selecting `Collection` from the `Decision` category.

Key parts:

This rule has two key parts:

Field	Description
Apply to	Select a class for this rule.
Identifier	Enter a name for this collection rule. Begin the name with a letter and use only letters, numbers, the ampersand character, and hyphens.

Rule resolution

When searching for instances of this Rule Type, the system uses full Rule resolution which:

- Filters candidate Rules based on a requestor's Ruleset list of Rulesets and versions
- Searches through ancestor classes in the class hierarchy for candidates when no matching Rule is found in the starting class

©2025 Pegasystems Inc

- Finds circumstance-qualified Rules that override base Rules
- Finds time-qualified Rules that override base Rules

- About Collection Rules
- Defining the pages and classes of a Rule
- Viewing Rule history

Completing the Basic/Advanced Collection tab

Complete the fields on this tab to define the sequence of steps the collection processes at run time. Each step can call a supported rule, set property values, or stop collection processing.

Adding a step

The Rules tab is divided into two main columns of information: a list of steps and a set of configuration options related to each step. When you select a step in the first column, the second column dynamically updates to display only the configuration details that are relevant to that step.

1. Click **Add step** to insert a step into the existing list.
2. Click **[Edit Description]** and enter text in the **Step Description** field.

   > **Result:**
   >
   > The text you provide is used in the **Collection step** list on the Specifications tab.

3. Click **Submit**.
4. Hover over the step number.
5. When the pointer becomes a drag handle, drag and drop the step to a different position in the list.

©2025 Pegasystems Inc

Configuring a step

You can now configure the step. You cannot select another step or save the collection until all required fields in the configuration column are populated.

1. Select the step to load the relevant configuration options.
2. Complete the fields in the **Condition** section to define criteria that must be satisfied before the step is run.
3. Complete the fields in the **Context** section to set the scope of the step.

   > **Result:**
   >
   > The context you select restricts the available options in other parts of the form such as the **Action** and **Response actions** sections.

4. Complete the fields in the **Action** section to select an action performed by this step. You can call a rule, set properties, or stop collection processing.
5. Complete the fields in the **Response actions** section to define sets of tasks that run after the step completes processing.

> **Result:**
>
> As you add configuration details in each section, the step column automatically displays a business-friendly label. While the names used in these labels are different for each step, the keywords are universal. You can interpret any set of labels using the following format:
>
> `"When [condition criteria are met] run [this action] on [the specified context]".`
>
> If you do not see the expected label for a particular step, refresh the collection or select another step in the list.

©2025 Pegasystems Inc

Understanding run-time-behavior

The system evaluates each step on the Rules tab in sequential order. Disabled steps are ignored.

Each step follows the same processing logic:

1. Compares the criteria in the Condition section against values on the primary page of the collection. If the criteria is met, the step continues. Otherwise, the next step in the list is run.
2. Switches the context to the page specified in the Context section.
3. Calls the rule or method specified in the Action section. It waits for the rule or method to complete processing before running the response actions.
4. Evaluates the condition for each response action. If the condition is true, the set of tasks associated with the response action is run. Otherwise, the next response action is evaluated.

At the conclusion of step processing, the system returns control to the calling activity or collection.

You can insert additional steps into the run-time sequence by defining actions on the Pre/Post Actions tab.

Completing the Preamble and Stop tab

Complete the fields on this tab to define pre-actions and post-actions that are evaluated before and after the collection is processed.

The **Before This Collection** and **After This Collection** sections display numbered lists of pre-actions and post-actions, respectively. Each action can contain one or more tasks. A task calls a supported rule or method.

1. Add an action by performing the following actions:
    a. Append an action to the existing list by clicking **Add pre-action** or **Add post-action**.

©2025 Pegasystems Inc

Because pre-actions and post-actions use identical fields and options, the procedures for creating and configuring them are the same.

b. From the drop-down list, select a condition.
c. Hover over the action number.
d. When the pointer becomes a drag handle, drag and drop the action to the required position in the list.

You cannot save the collection until all required fields are populated.

2. Add a task to an action by performing the following actions:
   a. Append a task to the existing list by clicking **Add task**.
   b. Open the **Update task** modal by clicking **Edit**.
   c. From the **Type** drop-down list, select an option.
      Options include supported rules and methods.
   d. Complete the additional fields that are required for your selected rule or method.

      For example, when you select an action type of **Activity**, you must enter the name of the activity to call and any applicable parameter values to pass at run time.

      Refer to the list of supported rules and methods for more details.

   e. Click **Submit**.
   f. Hover over the **Drag** icon next to the task label.
   g. When the pointer becomes a drag handle, drag and drop the task to the required position in the list.

What to do next:

You can now click on the task label to open the rule it calls. When a task calls a method, the label is read-only.

PEGA ©2025 Pegasystems Inc

Completing the Specifications tab

Complete the fields on this tab to create a link between a specification and this collection. Rules that are linked to a specification are implementations.

You can view implementations on the Specifications tab of the Application Profile landing page or the Implementations tab of the Specification form. Specifications and their linked implementations can be included in documents generated by the Document Application tool.

Linking a specification to a collection

The list of linked specifications is displayed in a table format. Each row represents an association between a specification and a step in the collection or the collection itself.

The table supports many-to-many relationships. Each specification can link to more than one step and each step can have more than one linked specification.

1. Ensure that the collection is checked out or in an unlocked ruleset.
2. Click the **add** icon to append a specification to the list.
3. Enter a specification name in the **Name** field or use the autocomplete to prompt you with a list of names.
4. Select a step description from the **Collection step** drop-down list. Alternatively, keep the default value `N/A` to indicate that the specification is linked to the collection rule itself.
5. Click **Save**.

You can click any specification name to open it. The full set of linked implementations is available on the Implementations tab of the Specification form.

Click the **delete** icon to remove a specification from the list.

Automatic link management

©2025 Pegasystems Inc

During application development, you can delete steps in a collection and modify step descriptions. You can withdraw or delete specifications.

In all of these cases, the Specifications tab of the Collection form stays in sync. Labels in the **Collection step** field are automatically refreshed and links to invalid steps are removed from the table.

If a collection step does not specify a value in the **Step Description** field, the step label is used to populate the **Collection step** field.

Using conditions in a collection

Add conditions to your collection to support multiple paths of execution.

At run time, a condition evaluates an expression or the results of a function alias. If the returned value equals true, the condition is met and the step or action that immediately follows is processed.

All conditions use the primary page of the collection. They do not use context page of the step.

Where supported

You can use a condition to control the execution of:

- A step on the Rules tab.
- An iteration within a step when the context is set to Page List or Page Group.
- An action on the Pre/Post Actions tab.
- A response action for a step on the Rules tab.

Options and run-time behavior

The Collection form displays conditions as a drop-down list. You can select one of the following options.

©2025 Pegasystems Inc

Option	Description
Always	The step or action runs each time the collection is called. This is the default value.
When	The step or action only runs when the specified criteria is met. Additional icons appear when you select this option: • Click the **Down arrow** icon to select a function alias from the list that models your condition criteria. • Click the **Gear** icon to launch the Expression Builder. • Click the **Open** icon to open a rule. Steps that use this condition appear as skipped in Tracer results when the specified criteria is not met.
Disabled	This step is ignored when the collection is called. This option is only available for conditions on steps on the Rules tab. Steps that use this condition do not appear in Tracer results.

- About Collection Rules
- Completing the Preamble and Stop tab

©2025 Pegasystems Inc

- Using conditions in a collection
- Completing the Basic/Advanced Collection tab
- Setting the context of a collection step
- Application debugging by using the Tracer tool
- Collection rules
- Defining the pages and classes of a Rule
- Viewing Rule history

Setting the context of a collection step

Each step in a collection has a context, or primary page, that determines the rules and data it can reference. The context of a collection step is analogous to the step page of an activity step.

The context you select can restrict the available options in the Action section and Response actions section for a step.

Supported context types

Before setting the context of a step, ensure that you have populated the Pages & Classes tab.

In the Context section, select an option in the **Execute on** list. The following types are supported:

Option	Description
Current page	Use the primary page of the collection when this step is run. This is the default option.
List/group	Use a Page List or Page Group as the primary page when this step is run.

©2025 Pegasystems Inc

Option	Description
	The **List/group** field appears when this option is selected. Enter the name of a Page Group or Page List property. Specifying an invalid Page Group or Page List in this field results in an error upon saving the form. At run time, the collection iterates over the Page List or Page Group. For each element it finds in the list, it calls the rule or method specified in the Actions section, followed by any response actions. You can optionally define a condition to evaluate before each iteration.
Other page	Use the specified page as the primary page when this step is run. The **Page** field appears when this option is selected. Enter the name of a Single Page property.

NOTE: To see the class of a specified page property, click the **List/group** or **Other page** field and press the Down Arrow key.

- About Collection Rules
- Collection rules
- Defining the pages and classes of a Rule
- Viewing Rule history

©2025 Pegasystems Inc

Using response actions in a collection

Use response actions to call rules or methods after a collection step completes processing.

The Response actions section appears for each step on the Rules tab that does not call the **Stop** method. It displays an ordered list of response actions. Each response action can contain one or more tasks. A task calls a supported rule or method.

- About Collection Rules
- Rule type
- Using conditions in a collection
- Collection rules
- Defining the pages and classes of a Rule
- Viewing Rule history

Adding a response action

1. Click **Add response action** to append a response action to the existing list.
2. Select a condition from the drop-down list.
3. Hover over the response action **drag** icon.
4. When the pointer becomes a drag handle, drag and drop the response action to the required position in the list.

> **Result:**
>
> You can now add tasks to the response action. You cannot save the collection until all required fields are populated. To remove a response action, click the **delete** icon.

Adding a task to a response action

1. Click **Add task** to append a task to the existing list.
2. Click **Edit** to open the **Update task** model.

©2025 Pegasystems Inc

3. Select an option from the **Type** drop-down list. Options include supported rules and methods.
4. Complete the fields that are required for your selected rule or method.

   For example, when you select a response action type of **Activity**, you must enter the name of the activity to call and any applicable parameter values to pass at run time. The available options in these fields are limited by your step context.
5. Click **Submit**.
6. Hover over the **drag** icon next to the task label.
7. When the pointer becomes a drag handle, drag and drop the task to the required position in the list.

> **Result:**
>
> You can now click on the task label to open the rule it calls. When a task calls a method, the label is read-only. To delete a task, click **Remove**.

Referencing parameter page properties

You can use properties on the parameter page to make decisions about further processing. As a best practice, add a condition to your response action that uses a function to evaluate the value of the property.

Some parameter page properties are exclusive to response actions. They represent data about the rule called in the current collection step. The system automatically updates the values of these properties when the next collection step is run.

Use the `param` keyword to reference these properties in your response action:

- *pxCalledRuleKey* — The unique key, or pzInksKey, that identifies the called rule.
- *pxCalledRuleType* — The object class of the called rule. For example, an activity has an object class of *Rule-Obj-Activity*.
- *pxCalledRuleAppliesToClass* — The class to which the called rule applies.
- *pxCalledRuleRuleSet* — The name of the ruleset that contains the called rule.

PEGA ©2025 Pegasystems Inc

- *pxCalledRuleRuleSetVersion* —The version number of the ruleset that contains the called rule.
- *pxCalledRuleUpdateDateTime* — The most recent time the called rule was modified and saved.
- *pxReturnValue* — The output or results returned by the called rule. Only a subset of the supported rule types sets the value of this parameter.

Rule type

Rules and methods supported by a collection

Call a rule or method from a collection to help model business decisions in your application. For example, you can call a decision tree in a step and then pass the results to a function for further analysis.

This functionality is available in:

- The Action section of each step on the Rules tab.
- The **Update task** modal, which can be launched from:
  - The Response actions section on the Rules tab.
  - The Before This Collection section on the Pre/Post Actions tab.
  - The After This Collection section on the Pre/Post Actions tab.

The context of a collection step can restrict the available options you see when you call a rule or method.

Calling a rule or method

Select a rule type or method from the **Type** drop-down list.

The majority of the options prompt you for the name of the rule to call in the **Name** field. If applicable, additional fields appear for you to pass parameter values to the rule or method. Collections always pass parameters by value.

©2025 Pegasystems Inc

Rules that support return values automatically set the value of `param`. *pxReturnValue* when they complete processing. You can use the **Copy return value to** field to save this result and then evaluate it in a later step or response action.

> **NOTE:** The Collection form reuses assembler logic for activities. Many options in the table below are equivalent to calling an activity method or instruction.

Rule type	Guidance
Activity	Equivalent to a Call instruction in an activity.
Collection	Equivalent to a Collect instruction in an activity.
Data Transform	Equivalent to the Apply-DataTransform method.
Decision Table	Equivalent to the Property-Map-DecisionTable method. Use the **Copy return value to** field to store the value of `param`. *pxReturnValue* in the specified property. This option is only available for a step in a collection.
Decision Tree	Equivalent to the Property-Map-DecisionTree method. Use the **Copy return value to** field to store the value of `param`. *pxReturnValue* in the specified property.

©2025 Pegasystems Inc

	This option is only available for a step in a collection.
Expression	Use the **Target property** field to call a declare expression with a key part that matches the specified property. Use the **Copy return value to** field to store the value of `param`. *pxReturnValue* in the specified property. You can only call declare expressions that use a calculate value of **When applied by a Rule Collection**. All other calculate values are not supported.
Function	Click the **Down arrow** icon and select a function alias. This list is populated with function aliases that return a void or boolean value. Enter values in the parameter prompts for the selected function alias. You can pass additional parameter values when the **Get params** button is visible. Use the **Copy return value to** field to store the value of `param`. *pxReturnValue* in the specified property. Your list of referenced rules is updated when the underlying function has parameters mapped to rule types.

©2025 Pegasystems Inc

Map Value	Equivalent to the Property-Map-Value method. Use the **Copy return value to** field to store the value of `param`. *pxReturnValue* in the specified property. This option is only available for a step in a collection.
Set Property	Equivalent to the Property-Set method.
Stop	Select an option in the **Scope** drop-down list to stop one or more collections in the call stack. Two stop methods are supported: • **This collection rule** — Stops the current collection only. The post-actions specified on the Pre/Post Actions tab are processed before control returns to the calling collection or activity. • **Collection processing** — Stops this collection and all other collections in the call stack. If the system finds an activity as it traverses the call stack, the stop method exits. You cannot call the stop method in a response action or post-action.
When	Enter the name of a when condition in the **Name** field.

©2025 Pegasystems Inc

	Use the **Copy return value to** field to store the value of `param`. *pxReturnValue* in the specified property. This option is available only for a step in a collection.

You can enter a period followed by the name of a property in the **Name** field. At run time, the system calls the rule name that matches the value of the specified property.

- About Collection Rules
- Completing the Preamble and Stop tab
- Completing the Basic/Advanced Collection tab
- Collection rules
- Defining the pages and classes of a Rule
- Viewing Rule history

Unit testing a collection

You can test a collection individually, before testing it in the context of the application that you are developing. You specify a test page for the rule to use, provide sample data as the input, run the rule, and examine the results. Additionally, you can convert the test run into a Pega unit test case.

1. In the navigation pane of Dev Studio, click **Records** > **Decision** > **Collection**, and then select the collection that you want to test.
2. Click **Clipboard** to open the Clipboard and examine the pages that are generated by the unit test. For more information, see Clipboard pages created by the Run Rule feature.

- Unit testing individual Rules
- Running a unit test Case
- Viewing test Case results

©2025 Pegasystems Inc

- Exporting a list of test Cases

About Connect Cassandra rules

Connect Cassandra rules support interactions between Pega Platform and Apache Cassandra data stores by performing reading and writing operations on a single row identified by the unique row ID.

The following tabs are available on this form:

- Service
- Methods

Access

To list the Connect Cassandra rules in your application, use the Application Explorer or click **Configure** > **Integration** > **Connector Definitions & Simulations**.

To list all Connect Cassandra rules that are available to you, use the Records Explorer.

Development

Connect Cassandra rules are available in applications that have access to the functionality provided by the Pega-BigData ruleset.

Before you can connect to Apache Cassandra data store, upload the Cassandra client jars into the application container with the Pega Platform. You need different jars for particular versions of Apache Cassandra data store, so the jars are not in the Pega Platform database.

To create a Connect Cassandra rule directly in your application, click Create > **Integration-Connectors** > **Connect Cassandra**.

Where referenced

Connect Cassandra rules can be referenced in activities.

©2025 Pegasystems Inc

Category

A Connect Cassandra rule is an instance of the *Rule-Connect-Cassandra* rule type. Connect Cassandra rules belong to the Integration-Connectors category.

- **Connect Cassandra rules - Completing the Create, Save As, or Specialization form**
- **Connect Cassandra form - Completing the Service tab**
- **Handling connector exceptions**
- **Connect Cassandra rules - Completing the Mappings tab**

Connect Cassandra rules - Completing the Create, Save As, or Specialization form

Connect Cassandra rules support interactions between the Pega Platform and Apache Cassandra data stores.

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

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

PEGA ©2025 Pegasystems Inc

This information identifies the key parts and options that apply to the record type that you are creating.

To create a Connect Cassandra instance:

1. In the header of Dev Studio, click **Create** > **Integration-Connectors** > **Connect Cassandra**.
2. On the **Create** form, enter values in the fields to define the context of the rule.
   a. In the **Label** field, enter text that describes the purpose of the rule.
   b. **Optional:** To change the default identifier of the rule, click **Edit**, and then provide a unique value in the **Identifier** field.
   c. Select the **Context**.
   d. In the **Apply to** field, press the Down Arrow key and select the class that the rule applies to.
   e. In the **Add to ruleset** field, select the name and version of a ruleset that stores the rule.
3. Click **Create and open**.

- About Connect Cassandra rules
- Viewing Rule history

Connect Cassandra form - Completing the Service tab

Complete the **Service** tab to identify the external system with which this connector communicates.

> **NOTE:** The connection details may change over time, or when you move between a test system and a production system. After the ruleset version containing this connector instance is locked, you cannot update the connection fields. If your application needs to override the connection details, complete the *EndPointURL* parameter of the Connect-Cassandra method in the connector activity that calls this connector instance.

©2025 Pegasystems Inc

1. Configure connection with the Cassandra node.
   a. In the **Host** field, enter the server IP address of the Cassandra node.
   b. In the **Port** field, enter the port of the Cassandra node. The default port is *9042*.
   c. In the **Response timeout** field, enter the number of milliseconds to wait for the server response. Enter zero or leave it empty to wait indefinitely.
      This field is disabled and ignored at run time when the **Intended for** field is set to queuing (response will not be available).
2. **Optional:** Configure authentication settings.
   This connector can establish secure and non-secure connections.
   a. Select the **Use authentication** check box.
   b. In the **Authentication profile**, select a *Data-Admin-Security-AuthenticationProfile* instance that contains details required by the remote service (user name and password using BASIC authentication).
3. **Optional:** Configure security settings.
   a. Select the **Use secure connection** check box.
   b. Specify a truststore and a keystore.
4. **Optional:** Configure error handling settings.
   a. **Status value property**: Select a property to hold the status code of the *Connect-Cassandra* method. The literal value `Good` identifies success. You can use the standard property *@baseclass.pyStatusValue*.
   b. **Status message property**: Select a property to hold status descriptions or return values from the external system returned by the *Connect-Cassandra* method. You can use the standard property *@baseclass.pyStatusMessage*. This field is disabled and ignored at runtime when the **Intended for** field is set to queuing.
   c. **Error handler flow**: Identify a flow instance to route work items that cannot continue in its current flow because this connector instance was unable to connect to the external system. The default *Work-.ConnectionProblem* flow provides one approach for handling such problems. See handling connector exceptions.

PEGA ©2025 Pegasystems Inc

5. **Optional:** Configure processing options.
   You can allow connector requests from this Connect Cassandra instance to be processed asynchronously in parallel with the calling activity, or in the background by an agent. The mode of execution of each connector request is determined by the **Execution mode** parameter of the *Connect-Cassanra* method, as enabled by the values in these two fields.
   a. **Intended for**: Select the execution mode.
      - **immediate execution**: The connector executes synchronously; the calling activity waits for results.
      - **queuing (response will not be available)**: The connector executes asynchronously, with calls placed in a queue. The calling activity continues without waiting for results. When selected, queuing occurs only if also specified in the *Connect-Cassandra* method.
   b. **Request processor**: If you selected queuing in the **Intended for** field, in this field identify a Connect Request Processor data instance that defines the classes and other characteristics of the queued connector request.
6. Click **Test connectivity**. Pega Platform verifies the connectivity by constructing the URL based on the host and port settings but it does not confirm the ability to authenticate with the service.
7. Click **Simulations** to create a simulator for this connect rule. See Connect Simulators.

Handling connector exceptions

Through an Integrator shape on a flow rule, a flow execution can call an external system such as a relational database, Enterprise JavaBean, or Web service. The activity referenced in the Integrator shape references a connector rule (for example, a Connect SOAP rule) that controls the communication and data exchange with the external system.

A tested connector interface may fail or time out, causing work item processing in the flow rule to stop. To detect and analyze or solve such issues, you can create in your application a flow rule for connector exceptions. Failure of an Integrator shape causes

PEGA ©2025 Pegasystems Inc

the designated flow rule to start. The flow rule can send out email correspondence, attempt retries, skip over the integrated step, or send an assignment to someone.

To use this functionality, specify the second key part — Flow Name — of a flow rule in the **Error Handler Flow** field on a connector rule form. The Applies To key part of the calling flow is used as the first key part when retrieving the exception flow rule.

If you leave the **Error Handler Flow** field blank, a connector problem causes the flow execution to fail and is reported only as Java exceptions in the Pega log of type: `com.pega.pegarules.pub.services.ConnectorException`.

The standard flow rule *Work-.ConnectionProblem* provides a default approach to exception handling.

When you accept the default, a connector exception causes the following processing:

1. The original flow execution is paused. The *ConnectionProblem* flow is called with seven parameters:
    - Operator ID of a user to notify (not used)
    - The Applies To and Flow Name key parts of the original flow execution
    - The name of the shape in which the problem arose
    - A ConnectorException type
    - An error message
    - An assignment type (Work queue or Worklist), indicating where to place a resulting assignment.
2. For all the exception types other than `ResourceUnavailable`, processing continues in a FlowProblems flow. Use the Flow Errors gadget in the Processes landing page to address flow errors.
3. If the exception type is `ResourceUnavailable`, up to five retries are attempted, at intervals determined by a service-level agreement.
4. A developer accessing the work queue can cancel the assignment (ending both the *ConnectionProblem* flow and the original flow) or attempt to restart the original flow, perhaps after taking corrective action.

PEGA ©2025 Pegasystems Inc

This flow may use a work queue named IncompleteConnections@org.com. You can override this default with a flow rule (in your application's work classes) of the same name, or override the rules it calls. For example, your exception flow can send email notifications to an appropriate user.

Connect Cassandra rules - Completing the Mappings tab

Use the Mappings tab to map the table fields in the data source to the Pega Platform™ properties. Defining mappings assumes that you have configured the connection settings in the **Service** tab.

Mapping Source

The mapping source section displays the tables available through the connection defined in this connect rule.

- Select the data source's keyspace.
- Depending on the keyspace selected in the previous step, select one of the available tables.

Data Mapping

Define how to map columns in the selected table to properties. Mapping for primary key columns listed in this section is mandatory.

- Type the Cassandra column name, for example `customerid`.
- Select a corresponding Pega Platform property, for example *.CustomerID*.

Connect-Cassandra supports single-value column types and one collection column type: `Map`.

In order to create a mapping for a **Map** column:

©2025 Pegasystems Inc

- Type the Cassandra column name followed by a colon and a name of a key that will be stored in the **Map** column, for example: `address:street` or `address:zipcode` which can be used to define a mapping for *street* and *zipcode* keys in the **address** column of type `Map`.
- Select a property used to store a value for the corresponding key in a **Map** column.

About Connect CMIS rules

Use a Connect CMIS rule to define a connection to an enterprise content management (ECM) system repository that supports the OASIS CMIS standard.

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

Access

To list the Connect CMIS rules in your application, use the Application Explorer or click **Configure** > **Integration** > **Connector Definitions & Simulations**.

To list all Connect CMIS rules that are available to you, use the Records Explorer.

Development

To configure a CMIS connection to a content management system repository, create a Connect-CMIS rule and select a CMIS provider and a server type (Atom or SOAP).

Where referenced

Connect CMIS rules can be referenced by an application rule and can be used to store and version work item attachments in the content management system.

PEGA ©2025 Pegasystems Inc

Category

A Connect CMIS rule is an instance of the *Rule-Connect-CMIS* rule type. Connect CMIS rules belong to the Integration-Connectors category.

- **Creating a Connect CMIS Rule**

- Integration-Connectors category
- Viewing Rule history

Creating a Connect CMIS Rule

Create a CMIS rule to define a connection to an enterprise content management (ECM) system repository that supports the OASIS CMIS standard. The CMIS standard enables your applications to target different ECM repositories uniformly with a common interface. Using content from external systems guarantees that users access the most up-to-date versions of files.

For example, in a loan application you can source the Terms and Conditions document from an external storage system in which the files are regularly updated so that CSRs who manage loan requests use the most up-to-date version of the necessary document.

Content Management Interoperability Services (CMIS) is an open standard that allows different enterprise content management systems to inter-operate over the Internet by defining an abstraction layer for controlling diverse document management systems and repositories that use web protocols. CMIS provides a vendor-neutral, language-independent API and query language for working with ECM repositories.

1. In the header of Dev Studio, click **Create** > **Integration-Connectors** > **Connect CMIS**.
2. On the **Create** form, enter values in the fields to define the context of the rule.
   a. In the **Label** field, enter text that describes the purpose of the rule.

©2025 Pegasystems Inc

b. **Optional:** To change the default identifier of the rule, click **Edit**, and then provide a unique value in the **Identifier** field.
c. In the **Context** section, in the **Apply to** field, press the Down arrow key and select the class to which the rule applies.
Typically, you select *CMIS-ServiceData*. You can also select a class that is derived from *CMIS-ServiceData*.
d. In the **Add to ruleset** field, select the name and version of a ruleset that stores the rule.

3. Click **Create and open**.
4. On the **Service** tab, in the **Endpoint** section, in the **CMIS provider** list, select the provider:
   - To select a provider that is already configured, choose a provider from the list.
   - To configure a custom CMIS provider, select **Other**.
5. In the **Server type** section, select either the Atom or SOAP server type protocol, depending on the configuration of your CMIS provider.
6. In the **Server URI** field, enter or modify the URI for your CMIS provider host.
7. In the **Authentication profile** list, select or create an authentication profile that has the credentials required to connect to your CMIS provider.

   For more information see Creating an authentication profile, Authenticating requests in connectors, Application URL patterns for various authentication service types, and Authentication.

8. In the **Connection** section, in the **Timeout** field, enter the number of milliseconds after which this rule times out with a failure message if no response arrives from the external system.
   By default, the time-out setting is 30,000 milliseconds. Enter `0` to wait indefinitely.
   The value of the **Timeout** field cannot contain a negative number.
   For more information, see Configuring dynamic references to external systems by using the Global Resource Settings feature, Fields that support the Global Resource Settings syntax, and Configuring a GRS for a connector endpoint.

PEGA ©2025 Pegasystems Inc

9. In the **Service** section, click **Select a CMIS repository**, and then select a repository from the specified server.
   The **Repository ID** field displays the unique identifier for the selected repository.
10. Click **Submit**.
11. On the rule form, click **Save**.
12. **Optional:** To test whether Pega Platform can connect to the external system, in the **Testing and simulations** section, click **Test connectivity**.

    The system presents test results in a separate window, identifying the parameters used in the test, the steps attempted, and the outcome of each step.

13. **Optional:** To create a simulator for this connect rule, click **Simulations**.

    For more information, see Creating connector simulators.

- Creating Rules
- Creating a Rule specialized by circumstance
- Creating a rule or data instance
- Viewing Rule history
- Sourcing attachments from external storage
- Integrating your application with external resources

About EJB connector rules (deprecated)

Use Connect EJB rules to define and configure a connector to an Enterprise JavaBean on a local or remote system. The services are deployed as Enterprise JavaBeans (EJBs) within a Java Enterprise Edition (Java EE) application server, such as IBM's WebSphere or Oracle WebLogic.

> **NOTE:** Beginning in Pega Platform 8.6, EAR deployments of Pega Platform are deprecated. In Pega Platform 8.7 and later, EAR deployments are no longer supported. As a result, in Pega Platform 8.7 and later, Connect EJB rules are

©2025 Pegasystems Inc

> also no longer supported. For more information on the alternatives for EAR-related rules, see Deprecation of EAR deployments in Pega Platform 8.6.

The following tabs are available on this form:

- Service
- Create
- Remote

Access

To list the Connect EJB rules in your application, use the Application Explorer or click **Configure** > **Integration** > **Connector Definitions & Simulations**.

To list all Connect EJB rules that are available to you, use the Records Explorer.

Development

You can use a wizard to generate classes, properties, and Connect EJB rules, rather than creating them one by one through rule forms. To use the wizard, select **Configure** > **Integration** > **Connectors** > Connector and Metadata wizard .

At runtime, the system locates the local or remote services and connects to them using the Java Naming and Directory Interface (JNDI) protocol.

To debug your application, you can simulate an EJB connector when the external system is unavailable or lacks a test environment.

Where referenced

Connect EJB rules can be referenced in an Integrator shape of a flow rule.

PEGA ©2025 Pegasystems Inc

> **NOTE:** As of Pega 7.1.8, this rule is deprecated for data pages. Existing data pages that reference the rule will still work, but you cannot reference it when you create or edit a data page.

Category

A Connect EJB rule is an instance of the *Rule-Connect-EJB* class. Connect EJB rules belong to the Integration-Connectors category.

- **Connect EJB rules - Completing the New or Save As form**
- **Connect EJB form - Completing the Service tab**
- **Connect EJB form - Completing the Create tab**

Connect EJB rules - Completing the New or Save As form

Use a Connect EJB rule to define and configure a connector to an Enterprise JavaBean on a local or remote system. The services are deployed as Enterprise JavaBeans (EJBs) within a Java Enterprise Edition (Java EE) application server, such as IBM's WebSphere or Oracle WebLogic.

> **NOTE:** Beginning in Pega Platform 8.6, EAR deployments of Pega Platform are deprecated. In Pega Platform 8.7 and later, EAR deployments are no longer supported. As a result, in Pega Platform 8.7 and later, Connect EJB rules are also no longer supported. For more information on the alternatives for EAR-related rules, see Deprecation of EAR deployments in Pega Platform 8.6.

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance

©2025 Pegasystems Inc

definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

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

This information identifies the key parts and options that apply to the record type that you are creating.

To create a Connect EJB instance:

1. In the header of Dev Studio, click **Create** > **Integration-Connectors** > **Connect EJB**.
2. On the **Create** form, enter values in the fields to define the context of the rule.
   a. In the **Label** field, enter text that describes the purpose of the rule.
   b. **Optional:** To change the default identifier of the rule, click **Edit**, and then provide a unique value in the **Identifier** field.
   c. Select the **Context**.
   d. In the **Apply to** field, press the Down Arrow key and select the class that the rule applies to.
   e. In the **Add to ruleset** field, select the name and version of a ruleset that stores the rule.
3. Click **Create and open**.

Connect EJB form - Completing the Service tab

Use the **Service** tab to identify the properties of the remote EJB service. Complete the **Service** tab first.

©2025 Pegasystems Inc

> **NOTE:** Beginning in Pega Platform 8.6, EAR deployments of Pega Platform are deprecated. In Pega Platform 8.7 and later, EAR deployments are no longer supported. As a result, in Pega Platform 8.7 and later, Connect EJB rules are also no longer supported. For more information on the alternatives for EAR-related rules, see Deprecation of EAR deployments in Pega Platform 8.6.

As you work with the form, the information on this tab is used to connect to the JNDI server and automatically complete fields on the **Create** and **Remote** tabs.

JNDI server

Field	Description
Server name	Select the JNDI server (instance of *Data-Admin-Connect-JNDIServer)* that identifies the EJB container with the bean you want to connect to. To connect to an EJB through its local interface, select the JNDI server named `localEJBs`. To connect to a remote EJB through an EJB reference rather than its JNDI name, select the JNDI server named `remoteEJBrefs`. This field supports the Global Resource Settings syntax (= `PageName.PropertyName`). The default property name for this field is `pyJNDIServerKey`.
Lookup name	Select a key to use to look up the EJB home object of the remote EJB in the JNDI server.

©2025 Pegasystems Inc

EJB home

Field	Description
Java class	Enter the name of the EJB Home Interface class to use to create an instance of the EJB client stub. If the JNDI Server and Lookup Name values are valid, this value is automatically filled in.
Create method	Optional. Enter the type-signature of the EJB Home Interface method that creates an instance of the EJB client stub (for example: `create(String, int)`). This field is not required if you select the `Require use of a valid object handle` option of the EJB **Remote/Initialize Object** field.

EJB remote

Field	Description
Initialize object	Select how to initialize the EJB client stub: • **Create a new object every time** — perform a JNDI lookup and use the specified Create Method of the EJB Home Interface class to create an instance of the client stub each time this connector is called. • **If possible, use a valid object handle** — try to restore the client stub instance using the value contained in the **Object handle property** field. If the object handle value is blank or no longer valid,

©2025 Pegasystems Inc

Field	Description
	then perform a JNDI lookup and use the specified Create Method of the EJB Home Interface class to create an instance of the client stub. • **Require use of a valid object handle** — try to restore the client stub instance using the value of the property identified in the **Object handle property** field. If the object handle value is blank or no longer valid, report an error using the status properties. Use this option if the remote method being called must be preceded by an earlier remote method call.
Java class	Enter the name of the EJB Remote Interface class that defines the remote methods that can be called on the EJB client stub instance. If the JNDI server and lookup names are valid, this value is filled in automatically.
Remote method	Enter the type-signature of the EJB Remote Interface method that is called each time this connector is called. For example: `getListValue(int)`
Finalize object	Select how to finalize the EJB client stub: • **Remove object after method call** — call the client stub's Remove method after the remote service method is called.

©2025 Pegasystems Inc

Field	Description
	• Keep alive and store object handle — store the client stub's object handle value in the specified **Object handle property** after the remote service method is called. Use this option if you intend to call multiple remote methods on a single EJB instance, either for performance reasons, or to maintain the session state between method calls.
Object handle property	Optional. Select a Single Value property for storing and/or recovering the serialized object handle of the EJB client stub instance. This field is not required if you select both the **Create a new object every time** option of the EJB **Remote/Initialize Object** field and the **Remove object after method call** option of the EJB **Remote/Finalize Object** field.

Error handling and test

Field	Description
Error handling	
Status value property	Optional. Select a property for reporting the status of the remote service request. You can use the standard property *@baseclass.pyStatusValue*. The literal value `Good` identifies success.

©2025 Pegasystems Inc

Field	Description
Status message property	Optional. Select a property for providing more information about errors that occur when processing the remote service request. You can use the standard property *@baseclass.pyStatusValue*.
Invocation exception property	Optional. Select a property of mode `Java Object` to contain a Java exception, if this connection throws any exceptions at runtime.
Error handler flow	Optional. Identify a flow rule that is to be started when this connector rule is started by a flow rule but fails. The default flow rule *Work-.ConnectionProblem* provides one approach for handling such problems. See Handling connector exceptions.
Test connectivity	Click this button at any time after you have completed the information on this tab to confirm that Pega Platform can connect to the JNDI server identified and can validate the Home Interface and Remote Interface. **NOTE:** The Test connectivity button does not work if either the `localEJBs` or `remoteEJBrefs` JNDI server is specified in the Server name field. The system presents test results in a separate window, identifying the parameters

©2025 Pegasystems Inc

Field	Description
	used in the test, the steps attempted, and the outcome of each step.
Generate test client	As a debugging aid, if your Pega Platform is hosted on a Java EE-platform (not Apache Tomcat) you can generate a stand-alone Java client program that calls the EJB. 1. Save the rule. 2. Copy a vendor-supplied `EJB.jar` library to the server and make sure that this library appears in the server's `$CLASSPATH` Environment variable. 3. If you changed the `$CLASSPATH`, stop and restart the Pega Platform. 4. Open the Connect EJB rule. 5. Click **Generate test client**. This produces a Java source file in the `ServiceExport` directory.
Simulations	Click to create a simulator for this connect rule.

Connect EJB form - Completing the Create tab

Use the **Create** tab to specify the data values that the remote system passes as input parameters to the `create()` method of the EJB home object. Pega Platform maps data from the clipboard to parameter values of the create method.

> **NOTE:** Beginning in Pega Platform 8.6, EAR deployments of Pega Platform are deprecated. In Pega Platform 8.7 and later, EAR deployments are no longer

©2025 Pegasystems Inc

> supported. As a result, in Pega Platform 8.7 and later, Connect EJB rules are also no longer supported. For more information on the alternatives for EAR-related rules, see Deprecation of EAR deployments in Pega Platform 8.6.

Complete the **Service** tab before completing this tab. The list of parameters and their data types are automatically filled in.

Parameters

Field	Description
Data type	The form contains the external data type of each parameter of the EJB Create method. Primitive Java types (`int` , `boolean` , `char` , and so on), Java Strings, and arrays of primitive types are supported.
Map from	Select how the system creates parameter values for the request. The choice you make for this field depends on the Data Type value: • **Clipboard** — The system maps the data value to the scalar property specified by the **Map from key**. • **Constant** — The system sends a fixed text value. • **XML Stream** — The system composes a stream of XML text formatted by an XML Stream rule. • **HTML Stream** — The system composes a stream of HTML text formatted by an HTML rule.

©2025 Pegasystems Inc

Field	Description
Map from key	Identify a source for the Map From value: • If you map from **Clipboard**, enter a property reference that is the source of the data. • If you map from **Constant**, enter a literal text string. To include blank spaces in your text string, surround the string with double quotes. • If you map from **XML Stream**, enter the Stream Name key part of the XML Stream rule that is to assemble the outgoing XML document. To find the rule at runtime, the system uses the Applies To class of this Connect EJB rule as the class of the XML rule, and the literal constant `MapFrom` as the final key part. Enter only the Stream Name key part. • If you map from **HTML Stream**, enter the Stream Name key part of the HTML rule that is to generate the data stream. To find the rule at runtime, the system uses the Applies To key part of this Connect EJB rule as the Applies To key part of the HTML rule.

©2025 Pegasystems Inc

About Connect HBase rules

Connect HBase rules support interactions between Pega Platform and Apache HBase data stores by performing reading and writing operations on a single row identified by the unique row ID.

The following tabs are available on this form:

- Service
- Mappings

> **NOTE:** There is a new form of the HBase rule. While you cannot create any new instances of the Pega 7.1.7 HBase rule, legacy instances in your application remain functional. For details see the Service tab.

Access

To list the Connect HBase rules in your application, use the Application Explorer or click **Configure** > **Integration** > **Connector Definitions & Simulations**.

To list all Connect HBase rules that are available to you, use the Records Explorer.

Development

Connect HBase rules are available in applications that have access to the functionality provided by the Pega-BigData ruleset.

Before you can connect to Apache HBase data store, upload the HBase client jars into the application container with the Pega Platform. You need different jars for particular versions of Apache HBase data store, so the jars are not in the Pega Platform database.

To create Connect HBase rules directly in your application, click Create > **Integration-Connectors** > **Connect HBase**.

©2025 Pegasystems Inc

Where referenced

Connect HBase rules can be referenced in activities.

Category

A Connect HBase rule is an instance of the *Rule-Connect-HBase* rule type. Connect HBase rules belong to the Integration-Connectors category.

- **Connect HBase rules - Completing the Create, Save As, or Specialization form**

Connect HBase rules - Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

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

This information identifies the key parts and options that apply to the record type that you are creating.

To create a Connect HBase instance:

©2025 Pegasystems Inc

1. In the header of Dev Studio, click **Create** > **Integration-Connectors** > **Connect HBase**.
2. On the **Create** form, enter values in the fields to define the context of the rule.
   a. In the **Label** field, enter text that describes the purpose of the rule.
   b. **Optional:** To change the default identifier of the rule, click **Edit**, and then provide a unique value in the **Identifier** field.
   c. Select the **Context**.
   d. In the **Apply to** field, press the Down Arrow key and select the class that the rule applies to.
   e. In the **Add to ruleset** field, select the name and version of a ruleset that stores the rule.
3. Click **Create and open**.

- About Connect HBase rules
- Viewing Rule history

About Connect Java rules

Use a Connect Java rule to define a connector interface between your application and an external Java application or a Java class.

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

The following tabs are available on this form:

- Service
- Constructor
- Method

©2025 Pegasystems Inc

Pega Platform offers various ways to interact with existing Java code, but using a Java connector is a best practice in most situations. See *Using existing Java code — A comparison of design alternatives for information about design options* on the Pega Community.

Access

To list the Connect Java rules in your application, use the Application Explorer or click **Configure** > **Integration** > **Connector Definitions & Simulations**.

To list all Connect Java rules that are available to you, use the Records Explorer.

Development

Ensure that the external Java class or .jar file are on the Pega Platform class paths. For instructions, see the Pega Community article *About the Process Commander Class Paths*.

Typically, use Connector and Metadata wizard when building a Java connector. The wizard uses the Java Reflection API to introspect the class or JavaBean's public constructors and public methods and generates the appropriate connectors, connector activities, classes, and properties.

To use the wizard, select **Configure** > **Integration** > **Connectors** > Connector and Metadata wizard.

If you create a Java connector rule manually, complete the **Service** tab first.

Simulation

To debug your application, you can simulate a Java connector when the external class is unavailable or lacks a test environment.

©2025 Pegasystems Inc

Where referenced

Connect Java rules can be referenced in an Integrator shape of a flow rule, or on the **Definition** tab of a Data Page rule, as the source of data for the contents of the data page.

Category

A Connect Java rule is an instance of the *Rule-Connect-Java* class. Connect Java rules belong to the Integration-Connectors category.

- **Connect Java rules - Completing the Create, Save As, or Specialization form**

Connect Java rules - Completing the Create, Save As, or Specialization form

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

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

- Creating Rules
- Creating a Rule specialized by circumstance

©2025 Pegasystems Inc

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

This information identifies the key parts and options that apply to the record type that you are creating.

To create a Connect Java instance:

1. In the header of Dev Studio, click **Create** > **Integration-Connectors** > **Connect Java**.
2. On the **Create** form, enter values in the fields to define the context of the rule.
   a. In the **Label** field, enter text that describes the purpose of the rule.
   b. **Optional:** To change the default identifier of the rule, click **Edit**, and then provide a unique value in the **Identifier** field.
   c. Select the **Context**.
   d. In the **Apply to** field, press the Down Arrow key and select the class that the rule applies to.
   e. In the **Add to ruleset** field, select the name and version of a ruleset that stores the rule.
3. Click **Create and open**.

- About Connect Java rules
- Viewing Rule history

About Connect SQL rules

Use Connect SQL rules when your application needs to run complex Structured Query Language statements such as joins or stored procedures to update or extract information from an external database.

The following tabs are available on this form:

- Open
- Delete
- Save

©2025 Pegasystems Inc

- Browse

As an alternative, if your application needs to perform a simple browse, open, delete, or update operation against an external database, use the Connector and Metadata wizard to configure external table access. When external table access is configured for an external table, you can use the Obj-Open, Obj-Browse, and other Obj- methods to interact with that table through a class that represents it and you do not need to create SQL connector rules.

Access

Use the Application Explorer to list the Connect SQL rules in your application. Use the Records Explorer to list all Connect SQL rules that are available to you.

Development

To configure an SQL connection to an external database, use the following elements along with the Connect-SQL rule:

- An activity that references one or more of four standard methods for operating on relational databases — RDB-Open, RDB-List, RDB-Delete, and RDB-Save. The names of the methods match the names of the tabs in the connector rule. An activity that uses RDB-Open to call a connector rule runs the SQL statement on that connectors Open tab, and so on.
- A Database instance and one or more Database Table instances that provide access to the tables or views of the external database.
- A JDBC library that provides access to the external database software.
- Special data mapping syntax to associate properties with database columns.

Where referenced

Connect SQL rules can be referenced in an Integrator shape ( ) of a flow rule .

©2025 Pegasystems Inc

> **NOTE:** As of Pega 7.1.8, this rule is deprecated for data pages. Existing data pages that reference the rule will still work, but you cannot reference it when you create or edit a data page.

Category

Connect SQL rules are instances of the *Rule-Connect-SQL* class. They belong to the Integration-Connectors category.

- **Connect SQL form - Data Mapping**
- **Connect SQL rules - Completing the Create, Save As, or Specialization form**
- **Connect SQL form - Completing the Open tab**
- **Connect SQL form - Completing the Delete tab**
- **Connect SQL form - Completing the Save tab**
- **More about Connect SQL rules**

Connect SQL form - Data Mapping

You can enter standard SQL into the Connect SQL form. In addition, employ the syntax described here to cause Pega Platform to make runtime substitutions to the SQL before sending it to the external database.

These keywords and notations provide a two-way mapping between Pega Platform features (the clipboard, Database Name instances, and Database Table instances) and the tables, rows, and columns of the external database.

> **NOTE:** In the Oracle implementation of Structured Query Language, column names use only uppercase letters unless you use the AS clause. To match a column name that contains lowercase letters, use AS:

©2025 Pegasystems Inc

```
Select MarketPrice as ".MarketPriceprop"
```

where MarketPrice is the Oracle column name and MarketPriceprop is the property name.

Referencing properties

A property on the clipboard can be a source of string values (as in a `SELECT` statement or `INSERT` statement) or a destination for data values. (Your source or destination property reference must be a single string.)

Similar to the HTML `reference` directive, use the curly brace characters to reference a property or part of a property.

```
{.property} or {page.property}.
```

For example:

```
WHERE zipcode={myPage.zipcode}
```

When used as a source of a string value, Pega Platform converts this into an SQL prepared statement variable, approximately equivalent to:

```
WHERE zipcode='02142'
```

In situations where the quote characters are not needed or not appropriate, use the `Asis` keyword, described below.

Use the optional `Decimal` or `Double` qualifier to indicate that a column being mapped to corresponds to the `java.sql.Types` constants `DECIMAL` or `DOUBLE` respectively. For example:

©2025 Pegasystems Inc

```
{pyWorkPage.pyEffortActual Decimal }
```

NOTE:

The Type value of a Pega Platform property value (such as `Integer`, `Text`, `DateTime`, or `True or False`) is not relevant here. In both retrieval operations (such as `SELECT`) and update operations (`INSERT`, `UPDATE`), the datatype of the column in the external database is what matters.

References to elements of `Page List` and `Value List` properties must contain integer constant index values such as 7 or 23 or keywords such as `<LAST> or <CURRENT>`. You can't reference a property as an index value.

Likewise, references to elements of `Page Group` and `Value Group` properties must contain constant subscript values such as "VA" or "Customer" or keywords such as `<APPEND> or <CURRENT>`. You can't reference a property as a subscript value.

You cannot use the `Param` keyword in these property references.

The `ARRAY`, `IMAGE`, and `CLOB` SQL types are not supported.

In and Out keywords for stored procedures

Optionally, follow the property name by `out` to indicate that this property is the result (output) of a stored procedure. Similarly, use the keyword `in` to indicate that this property value is a parameter to a stored procedure:

```
{.zipcode out }
```

Use both keywords when the parameter is both an input and an output.

©2025 Pegasystems Inc

```
{.zipcode in out }
```

Identifying a database table or a Pega Platform class

Literal table name

When the SQL `from` clause identifies a table name literally, the system assumes that this table is the database associated with the Applies To class of the this connect SQL rule. For example, if the Applies To class is Data-HR-KB-Employee and the SQL statement includes

```
select ... from dependents
```

the system looks in sequence for database table instances named Data-HR-KB-Employee, Data-HR-KB-, Data-HR- and Data- to identify a database, and then uses the table named dependents in that database.

Alternatively can use the `Table` keyword or the `Class` keyword to identify a database table to use, and search through all the instances of all the classes stored in that table The system uses information in the class rules, database table instances, and database instances to identify the external database and a table (or view) within the external database.

> **NOTE:** For the best performance and ease of debugging use the `Class` keyword rather than the `Table` keyword.

Class keyword

You can identify a Pega Platform class. At runtime, the system uses the class name to determine a Database Table instance, and from that instance, a database table. The syntax for the `Class` keyword is similar to the `Table` keyword. With `Class` keywords, use the format:

©2025 Pegasystems Inc

```
{class:ClassName}
```

for a direct reference to a fixed, known class. or:

```
{class: {page.property}
```

for an indirect reference that at runtime becomes the class.

For example,

```
{class:Data-Customer-Address-Details}
```

Table keyword

To use `Table` keywords, follow the format:

```
{Table:TableName}
```

for a direct, static reference to one Database Table instance, or:

```
{Table: {page.property}}
```

for an indirect reference to a property that, at runtime, contains in its value a Database Table name. For example,

```
{Table: {CustomerMaster.OracleAddressTable}}.
```

> ⚠ **CAUTION:** If your system contains two or more databases that by coincidence all contain a table specified in the `Table` keyword, do not use the `Table`

©2025 Pegasystems Inc

> keyword. You cannot control which database is used; results are indeterminate and may vary from run to run. Use the `Class` keyword instead.

For example, if both the PegaRULES database and a separate EMPLOYEE database contain a table named pr_other, the syntax

```
{Table: pr_other}
```

is ambiguous and may apply to either database.

Using SELECT statements

Use this syntax when entering a `SELECT` statement:

```
SELECT columnname AS ".propertyname" or "page.propertyname".
```

For example, if the property named zip is to contain the value of the column named Postal:

```
SELECT Postal AS ".zip".
```

If the database column name is the same as the property name, you can omit the AS clause. Specify:

```
SELECT columnname.
```

You can also use:

```
SELECT {all}
```

©2025 Pegasystems Inc

to select all the columns in the table. Use it to replace a `SELECT * from ...` statement.

For example:

```
SELECT {all} from {class: Data-Customer-Summary}
```

retrieves all the columns from the table identified in the Database Table instance named Data-Customer-Summary.

SQL Comments

Use the characters /* and */ to surround comments within your SQL statements. Text between these is ignored. For example:

```
/* This is a comment. */
```

Don't attempt to nest comments within comments.

Comments can document your rules, but are stripped from the SQL code text sent to the database software. Use the ASIS keyword — not comments — to send Oracle query hints.

Calling stored procedures

Include the optional `out` keyword to capture the results of a stored procedure in a `Single Value` property. Use the optional `in` keyword to identify a property that supplies an input parameter to a stored procedure. If omitted, `in` is the default.

For example, the SQL used to call a stored procedure can look like this:

```
\{call my_stored_proc({Values.In1}, {Values.Out1 out}, {Values.InOut1 in out }) \}
```

©2025 Pegasystems Inc

where the first parameter is an input parameter, the second is an output parameter, and the third is an input/output parameter.

- You can't specify the datatype of an out parameter. Consider returning a text value that is then converted.
- Stored procedure execution is not available for Db2/S390 databases. (However, stored procedures can be executed in UDB and in Db2/AS400) .
- To process result sets from a UDB SQL stored procedure, include in the UDB SQL stored procedure source the optional clause 'DYNAMIC RESULT SETS N', where N is the number of result sets that are returned.

Resultsets from Oracle stored procedures

To support the output of resultsets, a `RESULTSET` keyword is available. Include the `RESULTSET` keyword inside curly braces as any parameter for a stored procedure in the Browse tab textarea:

```
\{call procedure({resultset})\}
```

This keyword causes the system to process the output of the stored procedure as a resultset which is used to populate the *Code-Pega-List.pxResults* `Page List` property. One page is created for each result in the resultset.

To create multiple resultsets in one procedure, place a *{resultset}* marker for each parameter for each example that returns a resultset. For example:

```
\{
    {resultset} = call testFunc(
    {dataPage.OutProperty1 out},
    {dataPage.inkey1},
    {resultset},
    {dataPage.inkey2},
    {dataPage.OutProperty2 out},
```

©2025 Pegasystems Inc

```
    {dataPage.inkey3},
    {resultset})
\}
```

Note that the example above uses a `{resultset}` marker as a return value.

For an example of this feature, see the article *How to process a resultset returned from Oracle stored procedures* on Community.

Multiple resultsets from SQL Server or Sybase stored procedures

A stored procedure in a Microsoft SQL Server, IBM UDB, or Sybase database may execute two or more SELECT statements and return multiple resultsets. Include the `MULTISET` keyword anywhere in the Browse tab textarea, in curly braces:

```
{multiset}
```

Case and location are not important for the keyword.

The Connect SQL rule constructs a different clipboard structure for the results when this keyword is used. The `pxResults` property is a `Page List`; each page is of class *Code-Pega-List* and contains an integer property *pxResultCount* as well as another `Page List` property *pxResults*, which contains the pages belong to the application class.

You can affect the clipboard structure of MULTISET results by setting two Boolean properties *pyRDBSuppressUpdateCounts* and *pyRDBSingleMultisetResultSet* on the `pxRequestor` page. See the article *How to return multiple return sets in Rule-Connect-SQL operations using the MULTISET keyword* on Community.

> **NOTE:** For Oracle databases, calling stored procedures that return any result set is supported, so *MULTISET* is not supported.

©2025 Pegasystems Inc

The Asis keyword

You can include the value of a `Single Value` property or a text literal into the SQL using the `Asis` keyword. For a property, use the syntax:

```
{Asis: property-name}
```

In contrast to a simple property reference such as { .LastPrice }, the system does not place spaces or quotes around the value. For example, you can search for last names that start with "Smit" using:

SELECT A, B from {Class:CustomerMaster} where LastName LIKE '{Asis:Mypage.LastNamePrefix}%'

when value in LastNamePrefix is the four characters Smit.

For literal text, use the syntax:

```
{Asis: "/* literal text */"}
```

This facility can be used for passing "hints" to Oracle databases.

Case in the keyword is not significant; `asis`, `ASIS`, and `AsIs` are equivalent.

Use this feature only when necessary, as it can be costly in terms of database operations and performance.

Capturing SQL statements and status responses

To include SQL debugging and return status information from the database software, enter a line at the top of your SQL code in the format:

```
{SQLPage:mySQLPage}
```

©2025 Pegasystems Inc

Enter the name of page on which the system records the error messages. The page can have a class or be a classless page.

When the Connect SQL rule executes, the system creates a page with this name and adds two properties:

- *pxSQLStatementPre* — The SQL you entered in the Connect SQL form.
- *pxSQLStatementPost* — The SQL you entered in the Connect SQL form tab with the references resolved to show their values, so you can verify the references.

If errors occur, additional properties appear on this page:

- *pxRDBError* — An error message from Pega Platform.
- *pxRDBSQLCode* — A result code from the database access driver.
- *pxRDBSQLVendorError1* — An error code from the database vendor.
- *pxRDBSQLVendorError2* — An error code from the database vendor.
- *pxRDBSQLVendorMessage1* — An error message from the database vendor software.
- *pxRDBSQVendorMessage2* — An error message from the database vendor software.

When the RDB-List method executes, the system adds error messages if any to the browse page. With the RDB-Open, RDB-Save, and RDB-Delete methods, the system returns a method status, but error messages are not visible unless you use the {SQLPage } mechanism.

Mapping of Dates, Times, and DateTime properties

Some database vendors provide non-standard SQL datatypes for dates and times. If the external database you are connecting to contains vendor-specific date or time formats, you may need to map them to a `Text` mode property of the appropriate length (rather than a `Date`, `Time`, or `DateTime` mode property), or may need to cast (convert) the values.

©2025 Pegasystems Inc

Output of integers and DateTime values

When performing an INSERT SQL operation, you can use the keywords `Integer` or `DateTime` with a property to cause the property to be formatted appropriately.

The `Integer` modifier causes the value of the property to be passed as an integer. For example:

```
Insert into MyTable (MyIntColumn) values ({.MyIntProperty Integer})
```

The `DateTime` modifier causes the value of the property to be passed as a `DateTime` value. For example:

```
Insert into MyTable (MyDateColumn) values ({.MyDateProperty DateTime})
```

NOTE:

- You can use the `Integer` and `DateTime` keywords with the `in` keyword or with both `in` and `out`. You cannot use these keywords with only the `out` keyword.
- You can use the `Integer` and `DateTime` keywords regardless of the Type of the property, if the value conforms to the internal representation of an integer or `DateTime`.

Dynamic SQL

The syntax elements described in this topic are most often used to parameterize an SQL statement, changing the comparisons for a SELECT statement, providing data to insert, and so on.

©2025 Pegasystems Inc

If required, your application can generate an SQL statement or portion of an SQL statement dynamically, and store the statement as a property value. For example, if a WHERE clause is to vary from case in more ways that can be handled with simple value substitutions, your SQL statement can become:

```
SELECT Property1 WHERE Table1 WHERE {ASIS:.MyDynamicSQL }
```

Use care to ensure than values in the property MyDynamicSQL are valid and appropriate.

Connect SQL rules - Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

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

This information identifies the key parts and options that apply to the record type that you are creating.

Create a Connect SQL rule by selecting **Connect SQL** from the **Integration-Connectors** category.

PEGA ©2025 Pegasystems Inc

A Connect SQL rule has three key parts:

Field	Description
Apply to	Select a class associated with the database table on which you want to operate.
Package Name	Enter a value that indicates any SQL language variant. For example, you can enter: • `MSSQL` for Microsoft SQL Server • `Oracle` for Oracle. • `Db2` for IBM Db2 • `UDB` for IBM Db2 Universal Database (UDB) • `Sybase` for Sybase **NOTE:** This key part is required but unrestricted. As a best practice, use this key part to record which flavor of SQL is present in the rule.
Request Type	Enter a name for this rule. Begin the name with a letter and use only letters, numbers, the ampersand character, and hyphens.

- About Connect SQL rules
- Viewing Rule history
- More about Connect SQL rules

©2025 Pegasystems Inc

Connect SQL form - Completing the Open tab

You can use the **Open** tab to enter a SQL statement that opens a single row in the database and copies its column values as properties onto a clipboard page. To execute the SQL statement on this tab, an activity uses the RDB-Open method.

As a best practice, include only one SQL statement on this tab. If your situation requires multiple statements, place one statement each in separate Connect SQL rules. Your activity can cause the statements to be executed in the desired sequence. If your situation requires many statements executed in sequence, consider using a stored procedure.

> **NOTE:** As an alternative, if your application needs to perform a simple operations against an external database, it is best practice to use the External Database Table Mapping wizard or the Connector and Metadata wizard or the to configure access to the rows as an external class. When such external access is configured for an external table, you can use the Obj-Open, Obj-Browse, and other Obj- methods to interact with that table through a class that represents it; you do not need to create SQL connector rules. However, you can still use the **Open** tab to enter the SQL statement that saves an instance to the database.

If the `SELECT` statement when executed causes no rows to be retrieved, or more than one row to be retrieved, the RDB-Open method produces a fail status, and no properties are added to the step page. Complete the **Browse** tab, not the **Open** tab, if you want to search the database in a way that may result in none, one, or multiple matches.

> **NOTE:** The IBM Db2 data type `DATE` does not directly correspond to any Pega Platform property type. Use syntax similar to the following as a workaround:

©2025 Pegasystems Inc

```
SELECT TIMESTAMP(RPT_DATE, '00.00.00') AS ".RPT_DATE"
```

Properties retrieved by browsing an external database are not tracked for changes by Declare Expression processing.

Field	Description
Open SQL	Follow the guidelines in Data Mapping in SQL. Enter a single SQL statement directly into the text box. Typically this tab contains a `SELECT` statement, which may include the `WHERE` and `ORDER BY` clauses. You can use an `EXECUTE` statement (or equivalently a `CALL` statement for Oracle) on this tab. The results of the RDB-Open method are properties and values added to the step page (of the step that contains the RDB-Open method) for the selected row. (If the **Step Page** column of that step is blank, the system uses the primary page of the activity.)
Error Handler Flow	Optional. Identify a flow rule that is to be started when this connector rule is started by a flow rule but fails when the Integrator shape throws a `ConnectorException` exception. The default flow rule *Work-.ConnectionProblem* provides one approach for handling such problems.
Test Connectivity	After you save this rule, you can click to confirm connectivity to the database. No database operations are performed. See More about Connect SQL rules.

©2025 Pegasystems Inc

Connect SQL form - Completing the Delete tab

Connect SQL form – Completing the Delete tab

As a best practice, include only one SQL statement on this tab. If your situation requires multiple statements, place one statement each in separate Connect SQL rules. Your activity can cause the statements to be executed in the desired sequence. If your situation requires many statements executed in sequence, consider using a stored procedure.

> **NOTE:** As an alternative, if your application needs to perform a simple operations against an external database, it is best practice to use the External Database Table Mapping wizard or the Connector and Metadata wizard or the to configure access to the rows as an external class. When such external access is configured for an external table, you can use the Obj-Open, Obj-Browse, and other Obj- methods to interact with that table through a class that represents it; you do not need to create SQL connector rules. However, you can still use the Delete tab to enter the SQL to delete a row or rows from a relational database. The SQL statement in this tab operates in conjunction with an activity that uses the RDB-Delete method.

Field	Description
Delete SQL	Follow the guidelines in Data Mapping in SQL. Enter an SQL statement directly into the text box. You can use a `DELETE` statement, `TRUNCATE` statement, or `DROP` statement in this tab.
Error Handler Flow	Optional. Identify a flow rule that is to be started when this connector rule is started by a flow rule but fails when the Integrator shape throws a `ConnectorException`

©2025 Pegasystems Inc

Field	Description
	exception. The default flow rule *Work-.ConnectionProblem* provides one approach for handling such problems.
Test Connectivity	After you save this rule, you can click to confirm connectivity to the database. No database operations are performed. See More about Connect SQL rules.

Connect SQL form - Completing the Save tab

Enter an SQL statement to be executed by an activity that includes the RDB-Save method.

As a best practice, include only one SQL statement on this tab. If your situation requires multiple statements, place one statement each in separate Connect SQL rules. Your activity can cause the statements to be executed in the desired sequence. If your situation requires many statements executed in sequence, consider using a stored procedure. (Pega Platform uses prepared statements and submits the SQL code in a single call to database software. Some database software versions may allow multiple statements in one call, but other software versions do not; use of multiple statements on this tab reduces portability and flexibility.)

> **NOTE:** As an alternative, if your application needs to perform a simple operations against an external database, it is best practice to use the External Database Table Mapping wizard or the Connector and Metadata wizard or the to configure access to the rows as an external class. When such external access is configured for an external table, you can use the Obj-Open, Obj-Browse, and other Obj- methods to interact with that table through a class that represents it; you do not need to create SQL connector rules. However, you can still use

©2025 Pegasystems Inc

the **Save** tab to enter the SQL statement that saves an instance to the database.

Field	Description
Save SQL	Follow the guidelines in Data Mapping in SQL. Enter an SQL statement directly into the text box. You can use an `INSERT` statement, an `UPDATE` statement, or a `CREATE` statement on this tab. For an INSERT statement, ensure that the property values that define the primary key have appropriate values (unless the primary key is computed, as through auto-increment.) Some external database tables may include one or more auto-increment columns. Do not reference such columns in an INSERT statement or attempt to change values for these columns with an UPDATE statement.
Error Handler Flow	Optional. Identify a flow rule that is to be started when this connector rule is started by a flow rule but fails when the Integrator shape throws a `ConnectorException` exception. The default flow rule *Work-.ConnectionProblem* provides one approach for handling such problems. See Handling connector exceptions.
Test Connectivity	After you save this rule, you can click to confirm connectivity to the database. No

©2025 Pegasystems Inc

Field	Description
	database operations are performed. See More about Connect SQL rules.

More about Connect SQL rules

Use Connect SQL rules with an external database. Except in unusual situations, use the Obj-Browse method or Obj-List-View method rather than Connect SQL rules and RDB methods for the PegaRULES database.

When you need to access objects in a PegaRULES database, the RDB- methods and Connect SQL rules may not work, because not all the properties in the PegaRULES database are distinct database columns. Use the Obj-Open and Obj-Save methods, not the RDB- methods, with the Pega Platform database.

Debugging

Use the Clipboard tool to see the final SQL statement sent to the database and the results. Use the Tracer tool with breakpoints to pause processing before or after the RDB method executes.

For advanced and detailed debugging, use the DB Trace feature of the Performance tool.

Using views for SQL involving multiple tables

As a best practice, restrict the SQL statements to operate on a single table, aligned with a database table value and the Applies To key part of the Connect SQL rule.

If your SQL needs to operate on multiple tables, consider implementing a view within the database that presents the columns. This approach places some responsibility for good database design and performance with the database administrator and database software.

PEGA ©2025 Pegasystems Inc

Testing connectivity

After you save this rule, you can click **Test Connectivity** to confirm that Pega Platform can find, log into, and communicate with the database. This button does not cause the SQL to be processed and does not alter the contents of the database.

The system presents test results in a separate window, identifying the parameters used in the test, the steps attempted, and the outcome of each step.

How this rule works

At runtime, the system interprets the keywords in the context of your clipboard, Database Table and Database information to form the final SQL. It then executes the SQL using standard database access capabilities.

Each operation involves the SQL statement on one tab of the rule. To determine which database to access, the system uses this search algorithm:

1. Uses the database table associated with the Class of a `{Class }` directive in the source SQL statement.
2. When the SQL contains no `{Class}` directive but contains a `{Table}` directive, searches through *Data-Admin-DB-Table* instances for a match on the table name, and uses the database identified in the referenced *Data-Admin-DB-Name*.
3. Uses the database table associated with the Applies To key part of the Connect SQL rule.

> **NOTE:** Algorithm 2. can produce uncertain results if two tables in separate databases have the same name. Use `{Class}` rather than `{Table}` when feasible.

Data retrieved from a relational database is saved in clipboard pages.

©2025 Pegasystems Inc

Your activity can perform two or more RDB methods, for example first to open one record (row) in the database and later to save the record with updated values. The activity can first execute the RDB-Open method, referencing the first instance. The row data is then on the clipboard; the activity can modify the data (using Property-Set and other methods). The same activity can then execute the RDB-Save method, to store the updated row into the database.

Each RDB-Delete and RDB-Save method execution commits the changes to the external database. You cannot combine the database commit operations of multiple statements.

Notes on date and time values

In its internal PegaRULES database, Pega Platform stores date and time information adjusted to GMT time, based on the time zone of the server. The PegaRULES database contains few dates without a time and zone, and few times without a zone and a date. This approach facilitates date and time comparisons and arithmetic regardless of the time zone of users or external systems that interact with your system.

If the external database accessed through Connect SQL rules contains dates without a time zone, or times without a date and time zone, analyze these values carefully. Use caution before making comparisons of external dates and times with date/time values within Pega Platform.

For example, the Oracle `DATE` data type requires a time. If you store a Pega Platform property of mode `Date` (which has no time) into an Oracle `DATE` column, the time portion of the value is defaulted to 00:00:00 GMT, which may not be appropriate. As a workaround, store `Date` values into Oracle columns of data type `VARCHAR`.

Debugging SQL and SQL Performance

You can use the DB Trace feature of the Performance tool to view in depth the SQL operations of a Connect SQL rule.

PEGA ©2025 Pegasystems Inc

Limitations

The database you reference in the Connect SQL rule must be the default database of the database user account identified in the Database data instance.

Connect SQL rules can access an IBM Db2/390 database, with two restrictions:

- Do not execute stored procedures
- Do not use the `SELECT AS` statement with a column name

Connect SQL rules can access an IBM Db2 Universal Database (UDB) database, except for database columns with a `DATE` data type. Use syntax similar to the following as a workaround:

```
SELECT TIMESTAMP(RPT_DATE, '00.00.00') AS ".RPT_DATE"
```

NOTE: SQL connectors cannot process columns of data type `ARRAY`, `IMAGE`, or `CLOB` (Character Large Object).

Microsoft does not supply a JDBC driver for Microsoft Access databases, although third-party facilities are available. As a workaround, some Pega Platform applications have converted Microsoft Access databases to SQL Server databases, or exported the data to CSV files that are then imported using Service File rules.

Standard rules

Standard Connect SQL rules with *Assign-* as the Applies To key part support the service-level agreement operations of the *Pega-ProCom* agent. Other standard Connect SQL rules support operation of the Automated Test facility and usage reporting.

Your system also contains a single standard Connect SQL rule named *PegaSample-Task.CollectData*. This rule is marked as unavailable and is not part of a working example. It is provided to illustrate the rule form.

©2025 Pegasystems Inc

Atom Servers

> **NOTE:** This form is deprecated. You should use the *Connect CMIS* form instead. Rules created previously will still work, but you cannot create new ones.

Atom server data instances provide a means to store connection details for an external Atom server.

Fields and controls

The following table describes the available options on the **Environment** tab of the Atom Server form.

Field	Description
Atom Server URI	Enter the URI of the external Atom server. Do not include details pertaining to the individual Atom feed. This field supports the Global Resource Settings syntax (= *PageName.PropertyName*). The default property name for this field is `pyAtomServerURI`. For more information, see Using the Global Resource Settings feature.
User ID	Enter the user ID of the account that your system uses to connect to the Atom server. This field supports the Global Resource Settings syntax (= *PageName.PropertyName*). The default property name for this field is `pyAtomServerUserID`. For more

©2025 Pegasystems Inc

Field	Description
	information, see Using the Global Resource Settings feature.
Password	Optional. If required, enter the password of the user account specified in the User ID field. This field supports the Global Resource Settings syntax (= *PageName.PropertyName*). The default property name for this field is `pyAtomServerPassword`. For more information, see Using the Global Resource Settings feature.
Test Connectivity	After you complete this tab and save the Atom Server data instance, click to confirm that your system can connect to the Atom server. The system presents test results in a separate window, identifying the parameters used in the test, the steps attempted, and the outcome of each step.

Connect Request Processor data instances

Define a Connect Request Processor data instance to support an asynchronous HTTP connector or SOAP connector. This data instance defines the queuing and dequeing options for the connector.

The following tabs are available on this form:

©2025 Pegasystems Inc

- Queuing Options
- Dequeuing Options

Access

Use the Application Explorer to list the Connect Request Processor data instances in your system.

Category

The *Data-Admin-RequestProcessor-Connect* class contains Connect Request Processor data instances. This class is part of the Integration Resources category.

- **Connect Request Processor – Completing the New or Save As form**

Connect Request Processor – Completing the New or Save As form

Create a Connect Request Processor data instance by selecting **Connect Request Processor** from the Integration-Resources category.

A Connect Request Processor instance has the following two key parts:

Field	Description
RuleSet Name	Although you are defining a data instance and not a rule, specify the name of a ruleset that contains SOAP connector rules that can reference this instance.
Request Processor Name	Specify a name for this request processor. Begin the name with a letter, and use only alphanumeric characters, ampersands, and hyphens.

- Connect Request Processor data instances

©2025 Pegasystems Inc

About SOAP Server data instances

> **NOTE:** This form is deprecated. You should use the *Connect CMIS* form instead. Rules created previously will still work, but you cannot create new ones.

SOAP server data instances provide a means to store connection details for an external SOAP server. Connect-CMIS rules reference this information when connecting to external CMIS instances.

Fields and controls

The following table describes the available options on the **Environment** tab of the SOAP Server form.

Field	Description
SOAP Server URI	Enter the Uniform Resource Identifier for the SOAP service you are connecting to. This field supports the Global Resource Settings syntax (= *PageName.PropertyName*). The default property name for this field is `pyAtomServerURI`. For more information, see Using the Global Resource Settings feature.
User ID	Optional. For SOAP servers that require authentication, enter the user ID of the account that your system uses to connect to the SOAP server. This field supports the Global Resource Settings syntax (= *PageName.PropertyName*).

©2025 Pegasystems Inc

Field	Description
	The default property name for this field is `pySOAPServerUserID`. For more information, see Using the Global Resource Settings feature.
Password	Optional. For SOAP servers that require authentication, enter the password of the user account specified in the User ID field. This field supports the Global Resource Settings syntax (= *PageName.PropertyName*). The default property name for this field is `pySOAPServerPassword`. For more information, see Using the Global Resource Settings feature.
Test Connectivity	After you complete this tab and save the SOAP Server data instance, click to confirm that your system can connect to the SOAP server. The system displays test results in a separate window, identifying the parameters used in the test, the steps attempted, and the outcome of each step.

Web service policy data instances

SOAP connectors can use web service policy data instances to securely communicate with a web service policy (WS-Policy) annotated web service. A WS-Policy annotated web service exposes its security policy by embedding a WS-Policy element in the WSDL. The WS-Policy contains a set of assertions that represent the capabilities and

©2025 Pegasystems Inc

requirements of a web service. The SOAP connector can understand these assertions and apply necessary security to the payload accordingly.

WS Policy data instances are referenced on the Advanced tab of Connect SOAP rules.

Map eForm rules

The SmartForms capability enables your applications to work with PDF files. Your application can extract field data from incoming PDF forms and can generate PDF forms and documents.

The following tabs are available on this form:

- Mapping
- Pages & Classes

Map eForm rules specify the relationship between the form fields in a PDF form and properties in your application. EForm File rules hold uploaded PDF form files that your application uses as templates when generating new PDF documents. When you create a map eform rule, you specify an eForm file rule *(Rule-File-eForm*) to which the mapping applies.

Where referenced

Two standard activities accept the name of a map eForm rule as an input parameter:

- *@baseclass.ExtractDataFromEForm*
- *@baseclass.GenerateEForm*

You can also reference map eform rules with the Connect-File method, to write a PDF form to an output file.

Access

Use the Application Explorer to access map eForm rules in your application. Use the Records Explorer to see all map eForm rules available to you.

PEGA ©2025 Pegasystems Inc

Development

For information about configuring your application to use these activities to process incoming PDF forms or to generate a PDF form, see the Pega Community article *Working with PDF Forms and Documents*.

To simplify the process of creating an eForm and its corresponding map eForm rule, use the Create eForm Rules wizard. To start the wizard, choose Create eForm rules from the Wizards area of the Integration slice. See About the eForm Rules accelerator.

Category

Map eForm rules are instances of the *Rule-Map-Eform* class. They are part of the Integration-Mapping category.

- **Map eForm rules - Completing the Create, Save As, or Specialization form**
- **Map eForm Rules - Completing the Mapping tab**
- **Map eForm rules - Completing the Pages & Classes tab**
- **More about Map eForm rules**

Map eForm rules - Completing the Create, Save As, or Specialization form

1. About
2. New
3. Mapping
4. Pages & Classes
5. More...

Create a map eForm rule by selecting Map eForm from the Integration-Mapping category.

©2025 Pegasystems Inc

Map eForm Rules - Completing the Mapping tab

A map eform rule creates a two-way mapping. When the Pega Platform extracts data from an incoming PDF form, it uses this rule to determine into which properties to write the field values. When a new PDF form is generated, the map is used to determine into which form fields to write property values from the clipboard.

Use the Mapping tab to specify which eForm file rule holds the PDF form the mapping represents and to define the mappings between form fields and properties.

Field	Description
Source Form	
File	Enter the second key part — the File Name — of the eForm file rule this mapping applies to. The first key part, the Applies To class, is the same as the Applies To class for this map eForm rule.
Extension	Enter the third key part — the File Type (extension) — of the eForm file rule this mapping applies to.
Field/Property Mapping	
Field Name	Enter the name of a form field from the PDF form specified as the source form.
Property	Select the property that matches the form field specified in the Field Name field. If you want to map field values to properties in classes other than the Applies To class, first create an entry for that class on the Pages & Classes tab. Then you can use the usual property syntax

©2025 Pegasystems Inc

Field	Description
	(pagename.propertyname) to reference the property. You can't call a function or enter a general expression here.

Map eForm rules - Completing the Pages & Classes tab

To map PDF field data to or from properties that do not belong to the Applies To class of the map eForm and eForm file rule, create a page entry for the appropriate class on this tab. Then, return to the Mapping tab and use the page name in the property reference (pagename.propertyname).

Field	Description
Page Name	Optional. Enter the name of a clipboard page referenced on the Mapping tab.
Class	Optional. Select the class of that page.

About Parse Normalize rules

Parse normalize rules operate on a text string and perform string substitutions, to produce a normalized output text string. Normalization can replace multiple variants of a word or other text with a uniform, normalized value. Such normalization can make later operations on the normalized string simpler or faster.

For example, the original text string may contain a date or amount in any one of a variety of formats. If the desired date format is YYYY-MM-DD (for example 2009-10-12), parse normalize rules can search for dates in other formats and replace the value with the normalized format.

For more information about creating parse normalize rules, see Parse Normalize rules - Completing the Create, Save As, or Specialization form.

PEGA ©2025 Pegasystems Inc

Where referenced

Parse normalize rules are referenced and executed in activities, by the Text-Normalize method.

Access

Use the Application Explorer to access parse normalize rules that apply to the work types in your application. Use the Records Explorer to list all parse normalize rules available to you.

Category

Normalize rules are part of the Integration-Mapping category. A parse normalize rule is an instance of the *Rule-Parse-Normalize* rule type.

- **Parse Normalize rules - Completing the Create, Save As, or Specialization form**
- Text-Normalize method

Parse Normalize rules - Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Based on your use case, you use the Create, Save As, or Specialization form to create the record. The number of fields and available options varies by record type. Start by familiarizing yourself with the generic layout of these forms and their common fields using the following articles:

- Creating Rules

PEGA ©2025 Pegasystems Inc

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

This information identifies the key parts and options that apply to the record type that you are creating.

Create a Parse Normalize rule by selecting **Parse Normalize** from the **Integration-Mapping** category.

A Parse Normalize rule has two key parts:

Field	Description
Apply to	Select the name of the class to which this rule applies. The list of available class names depends on the Ruleset that you select. Each class can restrict applying Rules to an explicit set of Rulesets as specified on the **Advanced** tab of the class form.
Identifier	Enter a name for this rule. Begin the name with a letter and use only letters, numbers, and hyphens.

- About Parse Normalize rules

Converting from directives to JavaServer Pages tags

Pega Platform provides automated conversion facilities that translate most directives into equivalent JavaServer Pages (JSP) tags. Use of JSP tags rather than directives is recommended for new applications and upgraded older applications. Use of JSP tags is required for applications which are to be localized.

PEGA ©2025 Pegasystems Inc

Benefits

Use of JSP tags rather than directives offers these benefits:

- Processing is more efficient.
- Developers familiar with JSP technology are immediately productive
- Your application can incorporate JSP tag libraries (taglibs) already developed or acquired from other parties.
- Properties and pages named in the XML or HTML code are validated as you save the rule.
- The Referencing Rules display for properties includes the JSP-style stream rules that reference the property. (In contrast, such information is less complete for directive-based stream rules.)

To simplify this transition, your application can use directives in some stream rule instances and JSP tags in others. (Do not attempt to use both forms in a single rule.)

In many cases, you can use the automated conversion described here to convert directives to the equivalent JSP tag forms. If you can save the JSP-converted form of the rule without errors, the updated rule operates identically to the original directive-based version.

Procedure

These instructions apply to these rule types, which are derived from the *Rule-Stream* class:

- *Rule-HTML-Fragment*
- *Rule-Obj-HTML*
- *Rule-Obj-XML*

To update a rule containing directives to contain equivalent JSP tags:

1. Open the rule.

PEGA ©2025 Pegasystems Inc

2. Use the **Save As** toolbar button to make a new copy in another ruleset, or in the same ruleset with a higher version number.
3. Confirm that the **Generate As** field is set to `HTML`.
4. The **Convert to JSP** button displays. Click the button.
5. Pega Platform parses the HTML or XML text and automatically replaces the directives with equivalent JSP tags. It changes the **Generate As** field value to `JSP`. Respond to any errors reported by updating the **Pages & Classes** tab, information on other tabs, or the HTML or XML source code as appropriate, and save again.

Restrictions and notes

Certain constructs and Version 02-02 PublicAPI calls are not permitted in stream rules that depend on JSP tags. These conditions are detected and reported as errors when you attempt to save the rule.

1. A JSP tag cannot reference a rule that does not yet exist or is in a ruleset or version not on your ruleset list. (You can reference a rule that exists but is marked as unavailable.)
2. Page names that display in the HTML or XML text must also display on the Pages & Classes tab.
3. Some Version 02-02 PublicAPI methods including the following are not supported. These are rare.
   - *thisRefPush()*
   - *thisRef()* — use *tools.getActive()*
   - *targetPush()*
   - *inputPermitted()*
   - *getOpposit()*
   - *getIdent()*
   - *forEachPageInList()*
   - *basePagePush()*
   - *basePage()*
   - *appendLookup()*

©2025 Pegasystems Inc

4. In the attributes of the <pega:when > tag, you cannot reference both a JavaScript and another when tag option (such as a when condition rule or the `$mode-input` option). Also, complex expressions that use the two Java operators `||` and `&&` may fail to convert or operate correctly at runtime.
5. A circumstance-qualified or time-qualified rule derived from the *Rule-Stream* class (for example, control rules) and the corresponding base rule must both use JSP tags, or both use directives. If you convert a rule that is a base rule for qualified rules, convert all the qualified rules also.

Toggling features on and off

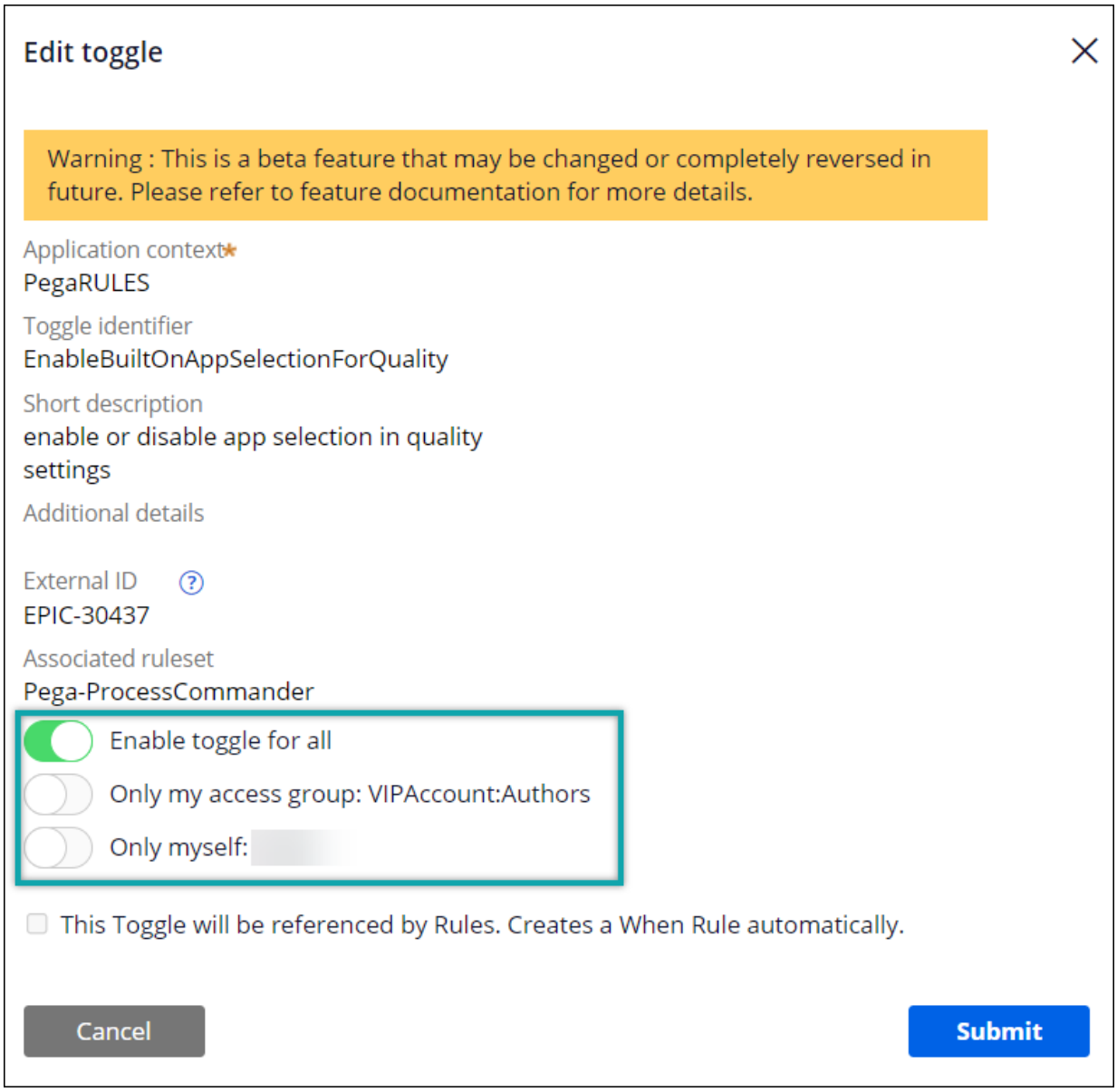
To enable or disable features for testing or to control which users have access to a feature, use feature toggles. You can also migrate a toggle to other applications by changing the associated Ruleset. As a result, you deliver an experience best tailored to users' needs.

The data instances and When Rules for toggles are associated with branched Rulesets in branched environments, and with application Rulesets in Rules-based development environments.

The Toggle Management landing page lists all the feature toggles available in your instance of Pega Platform™, however, you can only switch toggles for features that are present in your application stack.

1. In the header of Dev Studio, click **Configure** > **System** > **Release** > **Toggles**.
2. On the Toggle Management landing page, in the list of toggles, identify the feature toggle that you want to switch.
3. In the row of the feature toggle, click the **Edit** icon.
4. In the **Edit toggle** dialog box, switch the toggle on or off.
   You can enable the feature for all users of the application, a current Access Group, or a current operator, as shown in the following figure:

©2025 Pegasystems Inc



Toggling features on and off

5. Click **Submit**.

Class Management landing page

On the Class Management landing page, you can generate the module version report for a single node (current or remote) or for the whole cluster. The report lists the classes in the system with the module version. You can use this report to debug discrepancies in the system.

©2025 Pegasystems Inc

To access the Class Management landing page and generate the module version report, you must have the *pzSystemOperationsObserver* or *pzSystemOperationsAdministrator* privilege.

Access the Class Management landing page by clicking **Configure** > **System** > **Release** > **Class Management**.

- Identifying loaded module versions

List rules

A list rule identifies a set of properties that are to be included in a clipboard page assembled by the Obj-List method. This rule also supports simple tabular reporting (involving no calculations or sorting) and the Data Table editor.

The following tabs are available on this form:

- Fields

> **NOTE:** Local data storage replaced data tables.

Where referenced

List rules are referenced in activities, as a parameter to the Obj-List method.

Access

Use the Application Explorer to access list rules that apply to the work types in your application. Use the Records Explorer to list all list rules available to you.

Alternatives

©2025 Pegasystems Inc

> **NOTE:** This rule type is deprecated. As a best practice, use the Obj-Browse method.

Category

List rules are part of the Technical category. A list rule is an instances of the *Rule-Obj-List* rule type.

- **Creating list rules**
- **More about List rules**

Creating list rules

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a list rule by selecting `List` from the `Technical` category.

Key parts:

A list rule has two key parts:

Field	Description
Apply to	Select a class that this list rule applies to. Choose: • A concrete class to report only on instances of that class, stored as rows in a database table.

©2025 Pegasystems Inc

Field	Description
	• An abstract class that is parent of concrete classes, to define a rule that at run time can retrieve instances of a single class that is derived from the abstract class. • A concrete class that corresponds to a view of the PegaRULES database, to report on the rows of that view.
Identifier	Enter a list name that is a Java identifier. Begin the name with a letter, and use only letters, numbers, and hyphens. See How to enter a Java identifier.

More about List rules

Use list rules and the Obj-List method for simple selection, search, and reporting needs in your application.

Unit testing

After you save the List form, click **Actions** > **Run** to cause the system to create a top level user page named `pyQueryResultPage` of the *Code-Pega-List* class and to display the list as a report in a new browser window. You can export this display to Microsoft Excel.

The `pxResults` pages within the `pyQueryResultPage` contains an embedded page for each list entry.

Standard activity

Use the standard activity *Rule-Obj-List.ShowReport* to execute a list rule and display the results in a new window.

©2025 Pegasystems Inc

Method

To evaluate a list rule, execute the Obj-List method in an activity. The Obj-List method sends results of its search to the clipboard, and you can then use HTML to display these results.

For a class that contains many instances, each of which contains many properties, the Obj-List method can return only a filtered selection of instances, and only selected properties for each selected instance. This improves performance and reduces the clipboard size (and memory demand) compared with retrieving all properties. Pages containing only a portion of the properties of an instance are known as shortcut pages.

About Edit Validate rules

Use the Edit Validate form to define a Java routine that tests the validity of an input value in an activity that processes user input. The activity calls the Property-Validate method, which applies the test defined in an Edit Validate rule.

Typically, the input value arrives from a user who completes an HTML form, but it may also arrive through an interface with an external system.

Normally, if user input fails such processing, the input is rejected and a red X mark appears next to the input field in error. Messages may convey more about the error and the suggested remedy. The user can change the input and resubmit the form.

Where referenced

A property (of mode `Single Value`, `Value List` or `Value Group`) may reference an edit validate rule on the Advanced tab. Use the Property-Validate method in an activity to execute the edit validate rule when a value is submitted through a user input form.

Edit validate rules can also appear in the Validate As parameter of the Property-Validate method, and in validate rules).

PEGA ©2025 Pegasystems Inc

Access

Use the Records Explorer to list all edit validate rules available to you.

Category

Edit validate rules are instances of the *Rule-Edit-Validate* class. They are part of the `Data Model` category.

- **Edit Validate rules- Completing the New or Save As form**

Edit Validate rules- Completing the New or Save As form

Edit Validate rules - Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create an Edit Validate rule by selecting `Edit Validate` from the `Data Model` category.

Key parts

An Edit Validate rule has a single key part:

Field	Description
Identifier	Enter a name that is a valid Java identifier, not more than 64 characters long. See How to enter a Java identifier.

©2025 Pegasystems Inc

Rule resolution

When searching for edit validate rules, the system:

- Filters candidate rules based on a requestor's RuleSet list of RuleSets and versions
- Finds circumstance-qualified rules that override base rules
- Finds time-qualified rules that override base rules

The class hierarchy is not relevant to rule resolution of edit validate rules.

Properties

A property provides a name and characteristics for data in an object. Because a property definition is a rule, it shares the benefits of versioning, inheritance, and access control that the Pega Platform provides to all rules.

The following tabs are available on this form:

- General
- Advanced
- Completing the General tab
- Completing the General tab — Value modes

A property has one of eleven modes, arranged in three groups. The **General and Behavior** tab layouts vary according to the mode and the **Table Type** field selected. See:

- Completing the General tab — Choosing a property mode
- Completing the General tab — Value modes
- Completing the General tab — Java Object modes
- Completing the General tab — Tables

Your system includes hundreds of standard properties useful to your application. Review the relevant existing properties before creating new properties. Consult documentation and use the Pega Platform facilities to look for the ones you need.

PEGA ©2025 Pegasystems Inc

> **NOTE:** For a `Single Value` property, choose the wording of the Description with care. This helps you and others select the right property when using the Application Explorer, entering expressions, and elsewhere.

Access

Use the Application Explorer to access the properties available to you.

Category

Properties are part of the Data Model category. A property rule is an instance of the *Rule-Obj-Property* rule type.

Database table

Properties are normally stored in the PegaRULES database as rows of the `pr4_rule_property` table.

- **Creating a property**
- **Properties - Completing the Create, Save As, or Specialization form**
- **Property form - Completing the General tab**
- **Property form - Completing the General tab for Page modes**
- **Property form: Completing the General tab - Value modes**
- **Property form - Completing the General tab for Java Object modes**
- **Property form: General tab — Configuring validation**
- **Property form - Completing the Advanced tab**
- **More about Properties**
- **Implementing and using the TextEncrypted property type**

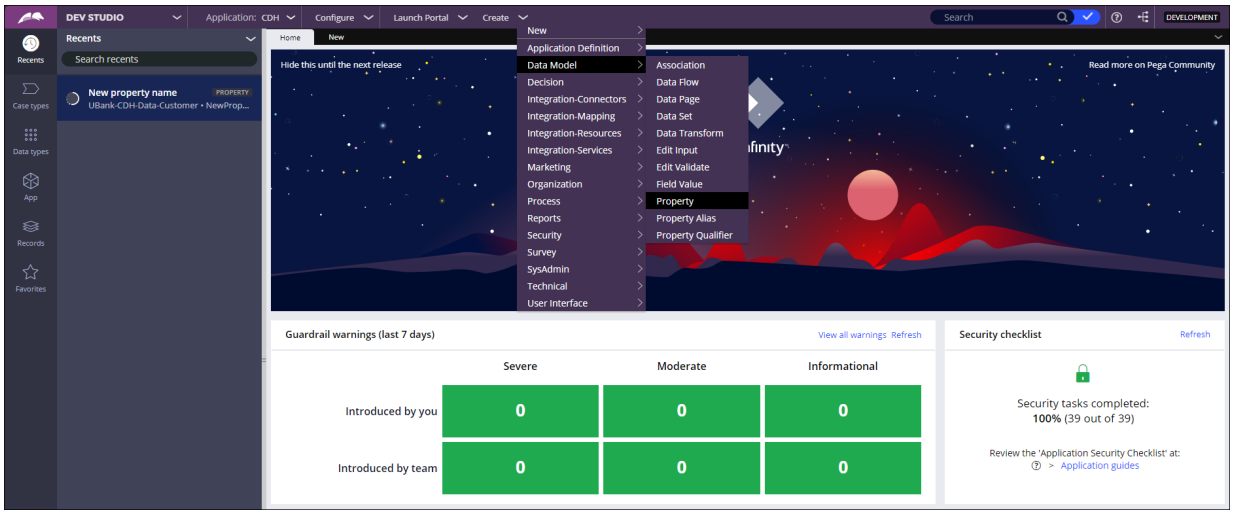
PEGA ©2025 Pegasystems Inc

Creating a property

Create a property by using the property Rule form. A property provides a name and characteristics for data in an object. Because a property definition is a Rule, it shares the benefits of versioning, inheritance, and access control that the Pega Platform™ provides to all Rules.

> **Before you begin:**
>
> Log in to Dev Studio using an account with system admin privileges.

1. In the header of Dev Studio, click **Create** > **Data Model** > **Property**.



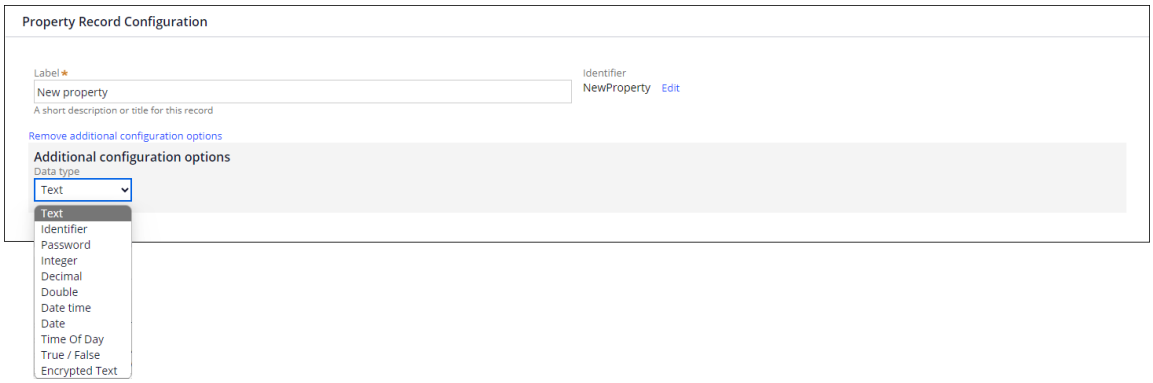
The Create menu

2. Configure the **Property Record Configuration** section:
   a. In the **Label** field, enter the property name.
   b. **Optional:** Next to the Identifier field, click **Edit** to change the property identifier.

©2025 Pegasystems Inc

> **NOTE:** The default identifier for the property is set automatically based on the property name.

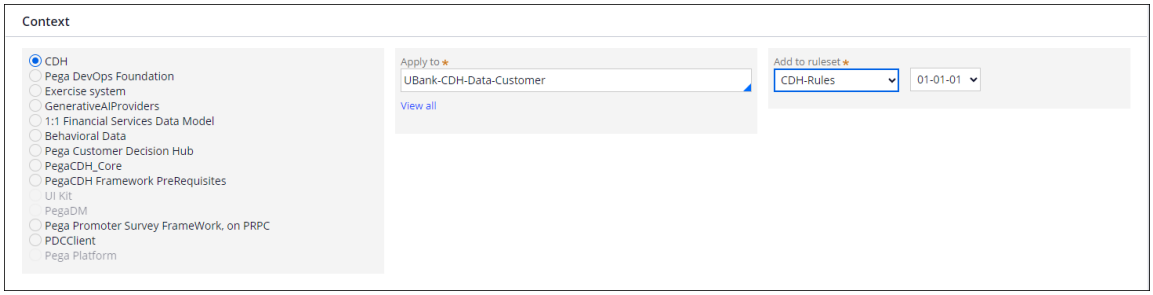
c. **Optional:** Click the **View additional configuration options** and then in the **Data type** list select the data type of the property.
You can select one of the following single value data types:



The Property Record Configuration section

For more information, see Property Type.

3. Configure the **Context** section:
   a. Select the context of your application.
   b. In the **Apply to** field, enter the appropriate class name.
   c. **Add to ruleset** list, select a Ruleset name and version.

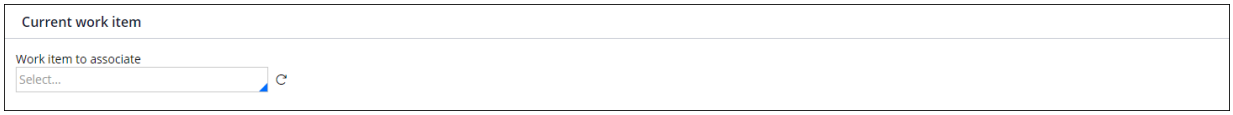


The Context section

For more information, see Table 1.

©2025 Pegasystems Inc

4. In the **Current work item** section, in the Work item to associate field, select the work item associated with the property.



The Current work item section

5. Click Create and open.
6. Configure the General tab.
   Depending on the data type of the property there are different options available. For more information, see:
   - Property form - Completing the General tab
   - Property form - Completing the General tab for Page modes
   - Property form: Completing the General tab - Value modes
   - Property form - Completing the General tab for Java Object modes
7. Configure the Advanced tab.
   For more information, see Property form - Completing the Advanced tab.
8. **Optional:** On the Specifications tab, click Add to associate specifications to the property.
9. **Optional:** Configure the History tab.
   a. In the **Documentation** section, add information that describes the property ant its use.
   b. In the **Custom fields** section, click the Add field link to add custom fields to the property.
10. Click Save.

What to do next:

To reuse the created property in App Studio, mark it as a relevant record. As a result, you can use the property as a field in your Data Model or Views while

©2025 Pegasystems Inc

> developing applications in App Studio. For more information, see Marking a record as relevant.

Related tasks

- Properties - Completing the Create, Save As, or Specialization form

Related reference

- Property form - Completing the General tab
- Property form - Completing the General tab for Page modes
- Property form: Completing the General tab - Value modes
- Property form - Completing the General tab for Java Object modes
- Property form: General tab — Configuring validation
- Property form - Completing the Advanced tab
- More about Properties
- Implementing and using the TextEncrypted property type

Properties - Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create a new property by selecting `Property` from the `Data Model` category.

- Use **Action** > **Create** rather than Save As to create most properties, because after you save a new property, you cannot change the Mode or certain other values on the form.
- To create several properties in a single class quickly, use the Create Properties wizard, available from the right-click menu on the Application Explorer.

©2025 Pegasystems Inc

Key parts:

A property has two key parts:

Field	Description
Apply to	Select the name of the class in which you are creating the property. For a property of mode `Java Property`, enter *Embed-Java-* or a class derived from *Embed-Java-*. The list of available class names depends on the Ruleset that you select. Each class can restrict applying Rules to an explicit set of Rulesets as specified on the **Advanced** tab of the class form.
Identifier	Enter a name that starts with a letter, and use only letters, numbers, and hyphens, forming a Java identifier. See How to enter a Java identifier. Avoiding duplicate names As a best practice for good runtime performance, choose property names that are distinct and unique throughout your application. For technical reasons, having several properties named Amount in one inheritance path, even if these have different RuleSets, can slow the initial assembly of rules that reference Amount. Choosing more explicit names — such as PrincipalAmount,

©2025 Pegasystems Inc

Field	Description
	AmountAtRisk, TotalOrderAmount — will both prevent this performance impact and also help ensure that developers use the appropriate property. When you click **Create and open** or **Save As**, and a property of the same name already exists in the inheritance path, a warning message appears asking if you want to create the property anyway. Properties that are duplicates of other property instances have an effect on caching, which can have a negative effect on system performance. Notes on names Property names may contain up to 64 characters. However, if this property is of mode `Single Value` and may prove useful for searching and selection in your application, choose a name not longer than the maximum column name length allowed by the database software supporting the PegaRULES database. This avoids issues that may arise later if the property is to become exposed as a database column. See exposed property. Except when overriding a standard rule in an upwards-compatible way, choose a name that does not start with px or py. Property names that start with these letters are

©2025 Pegasystems Inc

Field	Description
	reserved. In your application, however, use of standard properties with names starting with px and py is encouraged. Your application can set and test values in these properties. The pz prefix identifies standard properties that support Pega Platform internal operations. Do not directly alter these in your application. Don't choose a name that matches a reserved page name or keyword, such as `Top`, `Parent`, `Local`, `Param`, or `Primary`. See Understanding page names and reserved pages.

Additional creation options

Field	Description
Data Type	Select a data type to create a `Single Value` property and accept default values for all fields except Type .
Access When	Select a when rule if you are creating a `TextEncrypted` property type. The access when rule test and automatic encryption (or decryption) occur only as users interact with a form. In all other cases, your application must explicitly call the functions.

Reporting

©2025 Pegasystems Inc

If you are creating a top-level `Single Value` property in a concrete class, you can choose to optimize ("expose") the property as a database column now, before the system contains any values for the property.

If you are uncertain, leave these blank; you can expose the property as a column, and populate the column with values later. (This feature is available only on systems with a production level of 1 or 2, requires specific database account capabilities, and requires that you hold the *SchemaPropertyOptimization* privilege. See Platform generated schema changes.)

Field	Description
Optimize this Property for reporting on (class)	Select the top check box to expose the property for the class only.
Optimize this Property for reporting on (class) and its descendent classes	Select the second check box to expose the property for the **Apply to** class and all classes derived from this class.

Property form - Completing the General tab

The **General** tab for properties has three sections: Property Type, Data Access, and Display and Validation.

Property Type

A property has one of eleven modes. Each is defined in the table below. Three are string-based modes, three are page-based modes, and five are modes used with Java objects. The appearance of the General and Behavior tabs depends on the mode chosen.

Use the Property Mode selection box to choose a mode. The property mode determines how the system represents the property data. After you select a mode, consult these topics to complete the General tab fields:

PEGA ©2025 Pegasystems Inc

- Completing the General tab for Value modes
- Completing the General tab for Page modes
- Completing the General tab for Java Object modes

Mode	Description
`Single Value`	Select `Single Value` to create a property that can contain a single text string value. Text, numbers, dates, Boolean values, and amounts are stored in `Single Value` mode properties. The internal value of a `Single Value` property is String, not Single Value. `Single Value` property values correspond to objects of the `Java.lang.String` Class. For example, use this mode to create a property that can hold a birth date. The standard `Single Value` property named *Work-.pyID* holds the system-assigned unique identifier for a work item.
`Value List`	Select `Value List` to create a property that can contain an ordered list of single values. If you create a Value List property named LetterSent, you can use it to hold a list of values. For example, you can use an activity or an HTML form to set LetterSent(1) to Mom and LetterSent(2) to Dad.

PEGA ©2025 Pegasystems Inc

Mode	Description
	After setting these values, the LetterSent `Value List` contains LetterSent(1) and LetterSent(2). An example of a standard `Value List` property is *Work-.pyOpenWindows*.
`Value Group`	Select `Value Group` to create a property that can contain an unordered group of single values. (Rare.) If you create a Value Group property named Birthday, you can use it in your future configuration to hold a group of values. For example, you can use an activity or an HTML form to set Birthday(Bob) to July and Birthday(Mary) to September. Birthday(Bob) and Birthday(Mary) become elements of the Birthday Value Group.
`Page`	Select `Page` to create a property that can contain an embedded page as its value. For example, a Page mode property named Originator can contain information about the originator of a work item, such as the originator's name, account number, and so on. An example of a standard `Page` property is *Work-.pyIndexes*.

©2025 Pegasystems Inc

Mode	Description
`Page List`	Select `Page List` to create a property that can contain an ordered list of embedded pages. For example, a Page List property named Contacts can contain the pages Contact(1), Contact(2), Contact(3), and so on. *Work-.pyFlowActionCosts* is an example of a standard `Page List` property.
Page Group	Select **Page Group** to create a property that can contain an unordered group of embedded pages, each identified by a string subscript value. For example, the standard Page Group property named *Work-.pyWorkParty* can contain *pyWorkParty(Lawyer)*, *pyWorkParty(Customer)*, *pyWorkParty(Staff)*, and so on. **NOTE:** Subscript values must be a valid Java identifier, starting with a letter. As a best practice, use only letters and digits in the subscript.
`Java Object`	Some properties with this mode can be saved as persistent data in the PegaRULES database. If the Java class of the Java object instance implements the `Serializable`

©2025 Pegasystems Inc

Mode	Description
	interface, and the value can be validated, the Pega Platform can persist instances of the Java class with other clipboard data.
`Java ObjectList`	**NOTE:** Select `Java ObjectList` to indicate that this property can contain an array of references to instances of Java objects. The first element in the array is identified by the 1 subscript, not the 0 subscript.
`Java Object Group`	**NOTE:** Select `Java Object Group` to indicate that this property can contain an unordered group of references to instances of Java objects.
`Java Property`	Select `Java Property` to indicate that this property represents a single property of a Java class that defines an external Java object. Typically, properties of this mode are generated by the Import JavaBeans wizard when you use it to import a Java class and implement in the Pega Platform the Data Model of an external Java object.

©2025 Pegasystems Inc

Mode	Description
`Java Property List`	Select `Java Property List` to indicate that this property represents an array property of a Java class that defines an external Java object. Typically, properties of this mode are generated by the Import JavaBeans wizard.

CAUTION: If you update an existing property and change the Property Mode, additional steps might be necessary.

Data Access

Use this section to configure automatic data access and persistence settings.

For all property types except Page and PageList

Section	Controls
Data Access	Select a Data Access Type. The options are: • **Manual** – At run time, the user adds data to this property through the UI. Data transforms and other rules may be required to support this workflow. • **Automatic reference to class instance (linked)** – This property holds the value of a key to an Operator ID, Organization, Access Group, or other data object that presents data from that object anywhere in the system.

©2025 Pegasystems Inc

Section	Controls
	If you choose **Automatic Reference to class instance (linked)**, further fields appear: • Provide a linked class in the Linked Class field • Select a Source Property in that class to populate the property you are editing. The keys for the linked class then appear. If there is only one key, the value of your property becomes the key used to open the object at run-time. If there is more than one key, the source properties become input fields where you may enter values for the keys or properties that will contain the appropriate values at run time.
Data Persistence	The system saves the property data with the work item for which it is used by default. Check the "Don't save property data (transient)" check box to prevent the system from saving the data with the work item.

For Page and PageList properties

Section	Controls
Data Access	Select a data access option from those provided:

©2025 Pegasystems Inc

Section	Controls
	• **Manual** – At run time, the user adds data to this property through the UI. Data transforms and other rules may be required to support this workflow. • **Refer to a data page** – This is the recommended option. When data page parameters change and a reference to a non-parameter value is made, the system loads a new data page. Reads from and writes to the property occur on the new data page. • **Copy data from a data page** – Only use this option if you need to copy the data into your case. When parameters to the data page change and a non-parameter value is referenced, a new data page is loaded and copied into this property. Interactions with the property happen on the copied data. **NOTE:** To ensure that the property is editable through DX API (both v1 and v2), select **Manual** or **Copy data from a data page**. The **Refer to a data page** option only supports read-only usage. If you choose "Refer to a data page", further fields appear:

©2025 Pegasystems Inc

Section	Controls
	• For a Page property, you must choose a Data Page whose structure is "Page". • You can check the Load each page in this page list individually check box to have the system retrieve each embedded page in the PageList separately. You must specify a data page whose structure is "page". • Select a data page in the Data Page field, or click the **Edit** icon to the right of the field to create one. • If fields appear below the Parameters title, specify for each parameter a value, or a property that, at run time, contains the value to assign to the parameter. The parameters are passed to the data page for keyed page access. For more information, see Creating a Data Page. If you select Copy data from a data page , you can select a data transform in the Data Transform field that appears. This allows you to copy only a subset of the results on a data page into the property. To create a new data transform, specify a name and then click the **Edit** icon to the right of the field. **NOTE:** When you select **Copy data from a data page**, be aware of the

©2025 Pegasystems Inc

Section	Controls
	following behavior if the value for a parameter is missing: • If you are copying to a property of page structure, the property behaves as if no relevant reference exists and retrieves no data from the data page. The property displays any data the user has set manually. Once there are values for the keys, the property autopopulates from the data the data page provides and overrides any existing data the user has provided. • If you are copying to a property of page-list structure, the property behaves as if no filters are applied and copies the complete list of data (.pxResults) provided by the data page. Once there are values for the keys, any additional list elements that were created earlier are removed and the data in any existing list-elements is overridden.
Data Persistence	• For property types with manual data access selected, the system by default saves the property data with the case for which it is used. Check the Don't save property data (transient) check

©2025 Pegasystems Inc

Section	Controls
	box to prevent the system from saving the data with the work item. • For Interact directly with automatically loaded data pages , the property is a reference.The user can choose at run time to save or not save parameters to the data page. • If you select Copy automatically loaded data page into this property , the system follows your direction. There is no way to discard property data at the time of save. Select "Interact directly with automatically loaded data pages" if you wish to discard property data when saving.

Display and validation

In this section, you configure which control displays the property and, optionally, validation of input data.

To configure the UI control:

1. In the **UI Control** field, specify the control that displays the property.
2. If the control takes parameters, click the **Parameters** link and provide values for the parameters in the **Parameters** dialog.

For information about configuring validation tables, see Property form General tab — Configuring validation.

©2025 Pegasystems Inc

Other ways to associate values with properties

You can use other methods to associate a value with a property, including:

- Use a Declare expression.
- Use a "set" action in a data transform.
- Apply the value directly from the UI.
- Apply the results of a decision table or a decision tree.

It is also possible to use any of the Property-Set methods in an activity, but it is not recommended to use this method.

> **NOTE:** The values of many standard properties, such as *pyUserName* (which holds the identifier for the current user), are set automatically at log-in.

Property form - Completing the General tab for Page modes

Property form: Completing the General tab — Page modes

1. About
2. New
3. General
4. Advanced
5. More...

Complete the General tab fields that appear when you select a property mode of `Page`, `Page List`, or `Page Group`. You can use such properties in activities to create embedded pages on the clipboard.

For more information, see Properties.

©2025 Pegasystems Inc

Property form: Completing the General tab - Value modes

Use this topic to Complete General tab fields that appear when you create a property of mode `Single Value`, `Value List` or `Value Group`. For information on choosing a property mode, see Completing the General tab.

Selecting the Property Type

The Property Type determines the kind of data that values of the property represent, and can affect the format and allowable characters in the value.

> **NOTE:** Choose the Property Type carefully. After you save a property, you cannot later change its type except in a few special cases, even if no instances or pages have a value for the property.

Type	Description
`Text`	Select `Text` to allow the property to hold text strings that can contain tabs, carriage returns, and line feeds. When you use an activity or an HTML form to set the value of the property, the value can contain any graphic characters in addition to tabs, carriage returns, and line feeds. Use the `Text` Property Type if a user may enter the value in a text area of an HTML form.
`Identifier`	Select `Identifier` to allow the property to hold a text string that does not contain

©2025 Pegasystems Inc

Type	Description
	double quote, tab, carriage return or line break characters. **NOTE:** Do not confuse the `Identifier` type with the more restricted Java identifier values. To limit the values of a property to those that the Pega Platform accepts as Java identifiers, set the Type to *Identifier* and also set the Validate field (on the Advanced tab) to *JavaIdentifier*. See How to enter a Java identifier.
`Password`	Select `Password` to allow the property to hold a password. The values of properties with this Property Type are always encrypted when stored in the database. When on the clipboard, the values may appear as clear text (not encrypted), until the page containing the property is validated. The value can contain only legal graphic characters. Tabs, carriage returns, and line feed characters are not allowed. **NOTE:** Do not confuse the `Password` type with the

©2025 Pegasystems Inc

Type	Description
	`TextEncrypted` type, which has encrypted values even when on the clipboard; clear text values are visible only if a *Rule-Access-When* rule evaluates to true. See Implementing and using the TextEncrypted type. You can prevent any text field from echoing when typed into an HTML form using the `PASSWORD` input type: `<INPUT TYPE="PASSWORD" NAME=...>` This HTML does not cause any encryption. This native HTML feature is not related to the `Password` Type.
`Integer`	Select `Integer` to allow the property to hold an integer. When you specify the value in your later configuration, use any sequence of digits for the value. You can precede the digits with a plus sign (+12) or a minus sign (-12).
`Double`	Select `Double` to allow the property to hold a 64-bit floating point value. Do not use the Double property type to represent a precise value. Double properties are represented through Java double primitives, which are

©2025 Pegasystems Inc

Type	Description
	implemented in double-precision 64-bit IEEE 754 floating point format, and provide minimal precision. For precise decimal numbers, use a Decimal property type instead.
`Decimal`	Select `Decimal` to allow the property to hold a decimal quantity. Your property can reference the standard property qualifier named *pyDecimalPrecision* to fix the number of digits the system keeps after the decimal position in values of this property.
`DateTime`	Select `DateTime` to allow the property to hold a UTC (Coordinated Universal Time) value. The value is essentially the same as Greenwich Mean Time (GMT), used for the synchronization of computers on the Internet. Internally, a `DateTime` value is stored in the format YYYYMMDDTHHMMSS.QQQ GMT where T and GMT are literal characters, YYYY represents a year, MM is a month, DD is a day, HH is an hour value between 00 to 23, MM is a minute value between 00 and 59, SS is a second value between 00 and 59, and QQQ is a millisecond value between 000 and 999.

©2025 Pegasystems Inc

Type	Description
	Users need not enter `DateTime` values in this format, as shortcuts and alternative entry formats are available.
`Date`	Select `Date` to allow the property to hold a calendar date. Internally, a date is stored as an eight-digit text value, in the format YYYYMMDD. For example 20171123 represents November 23, 2017. No time zone is implicitly associated with a date. Application users are not required to enter `Date` values in this format, as many shortcuts and alternative entry formats are available. As a best practice, use `DateTime` rather than `Date` for properties that are part of work items, assignments, or other persistent objects in your application. Avoid using `Date` property types in persistent objects, such as work items or assignments, even if your application is used in only a single time zone, unless a time zone value is also stored in another property of the object.
`TimeOfDay`	Select `TimeOfDay` to allow the property to hold a time of day at the seconds level, not associated with a time zone. Internally, a `TimeOfDay` value is stored as six digits in the format HHMMSS, with two

©2025 Pegasystems Inc

Type	Description
	digits for the hour, two digits for the minute, and two digits for the second. For example, 000000 represents midnight and 120000 identifies noon. The value 235959 identifies the second before midnight. Application users are not required to enter `TimeOfDay` values in this format, as shortcuts and alternative entry formats are available. As a best practice, use `DateTime` rather than `TimeOfDay` for properties that are part of work items, assignments, or other persistent objects in your application.
`TrueFalse`	Select `TrueFalse` to allow the property to hold values representing true or false. When you specify the value in your later configuration, you can enter `true`, `false`, or any of these values: • 1, -1 or Y for true • 0 or N for false A `TrueFalse` property with a null value ("") can be saved into the PegaRULES database, but this is not a desirable practice. To avoid possible confusion and errors, choose a value "true" or "false" when initializing a `TrueFalse` property using a data

©2025 Pegasystems Inc

Type	Description
	transform or through other means, as appropriate to the business situation.
`TextEncrypted`	Select this type to implement encryption for the value. See Implementing and using the TextEncrypted type. The `TextEncrypted` type is differnt from the `Password` type. For `TextEncrypted` values, clear-text values are visible only if a *Rule-Access-When* rule evaluates to true.

Using Linked properties

For a Single Value mode property of type `Text`, you can link the property to a single instance of a different, concrete class. The information on this tab describes how the Pega Platform forms a key to that instance from the value of this property and possibly the value of other properties. The other concrete class is known as the target.

In many rules, use of linked properties can improve runtime performance by eliminating multiple requests to the PegaRULES database to retrieve the target instance. Use of a linked property can also eliminate the need for an activity to open the target instance.

The relationship defined by the property and the target fields is similar to an SQL JOIN or foreign-key relationship among database tables. The relationship allows the Pega Platform to retrieve (for display only, without a lock) the target object at run time.

In property references on sections, harnesses, and flow actions, you can include values of properties from the linked object, using property references of the form:

`.LinkPropertyName.TargetPropertyRef`

©2025 Pegasystems Inc

where .LinkPropertyName is the Property Name field of the linked property, and PropertyRef is any property reference in the Linked Class class. These values must be read-only.

Do not reference properties of the target in reports, activities, or data transforms.

Field	Description
Linked Property	Select to indicate that this property is a linked property, with values that correspond to all or part of the internal key of a unique instance of a concrete class.
Linked Class	For a linked property, identify a concrete class, one instance of which is to be accessed through the value of this linked property. **CAUTION:** Do not link to a rule (any instance of a concrete class derived from the *Rule-* base class.)
Linked Mapping	Identify one or more properties (in the Source Property column) that at run time contain values that together define all parts of a key to a single, unique instance of the objects in the Linked Class class. To assist in defining this mapping, the form identifies the properties that form the key to the linked class in the Linked Class Key column. If each value of the linked property forms the entire key to the instances of the linked class, enter the property name (the value you

©2025 Pegasystems Inc

Field	Description
	entered in the New dialog) in the Source Property column. If the values of the linked properties form only part of they key to instances of the linked class, complete the Source Property column by identifying the properties that supply the other key parts.

Example

Organization Divisions (instances of the *Data-Admin-OrgDivision* class) have two key parts, defined by properties *pyOrganization* and *pyOrgDivision*.

Property *Work-.pyOrigDivision* is a linked property that identifies, as source properties, two other (not linked) properties available on every work object:

- *Work-.pyOrigOrg* contains a value identifying an organization
- *Work-.pyOrigDivision* contains a value identifying the second key part of a division within the *pyOrigOrg*

As a result, any section derived from a *Work-* class can display (read-only) the Division Cost Center using the notation:

.pyOrigDivision.pyDivisionCostCenter

Using linked properties can reduce or eliminate the need to create activities in your application.

Runtime operation

At run time, the system retrieves the target object to evaluate any property reference within it. The retrieved pages become part of the requestor's clipboard and appear in the Clipboard display within a grouping named Linked Property Pages .

PEGA ©2025 Pegasystems Inc

If a second request for the same page occurs soon after a first request, the page may be still on the clipboard, eliminating a database access. All linked property pages are removed from the clipboard when your requestor session performs a Commit operation (which need not be related to the form containing information from the linked page).

If debugging is necessary, you can enable the Tracer options Linked Page Hit and Linked Page Miss . These options cause Tracer output to include a row for each hit and miss, identifying the referring page (containing the linked property) and the page containing the target object.

Completing the Control fields

These fields determine the presentation of the property when it appears on an HTML form. (By definition, the read-write mode is never used for special properties; only the read-only mode is relevant.)

For developers: If a property's Control field specifies an auto-generated control, in FormBuilder-based forms, the system displays the property using the *Default* control when displaying that property in the FormBuilder-based form. Examples of this behavior are when such properties are displayed in the Decision Table rule form, Decision Tree rule form, and the Run Rule form. In harness-based forms, the system displays the property using the specified auto-generated control.

Field	Description
Control	Optional. For `Single Value` properties, identify a control (auto-generated or non-auto-generated) to control the read-only presentation and read-write presentation of this property. Dozens of standard controls are available: • If there is no value, the system uses the control named *Default*.

©2025 Pegasystems Inc

Field	Description
	• Choose *Default-NonNLS* to use default presentations that do not reflect operator locale settings. • Choose *PromptSelect* (for a drop-down list) or *RadioButtons* (for a rectangular array of radio buttons) in most cases when Table Type has `Local List`, `Remote List`, or `Prompt List` selected. • Choose *PromptFieldValue* when Table Type has `Field Value` selected. • *ShowTextEncryptedPropertyValue* is the default if the property Type is `TextEncrypted.` • Do not choose *SmartLabel*, *StartSmartField*, *SmartInfoBehavior*, *SmartInfo*, *ListView*, *ListToList* or *Chart*; these are pseudo-controls with special, restricted use. When you create a new single value property, the system enters an auto-generated control in the Control field based upon the property type as follows: • `Checkbox` — *pxTrueFalse* • `Date` — *pxDateTime* • `DateTime` — *pxDateTime* • `Decimal` — *pxNumber* • `Double` — *pxNumber* • `Integer` — *pxInteger*

©2025 Pegasystems Inc

Field	Description
	• `Password` — *pxPassword* • `Text` — *pxTextInput* • `TimeOfDay` — *pxDateTime* **NOTE:** This field is not meaningful for `Value List` or `Value Group` properties. Leave blank.
Configure	If the control requires parameters and you want to specify them in this Property form, click the button and enter values as prompted. When completing a Field panel (for a cell in a harness, section, or flow action), you can override the parameter values entered here with values that apply to that cell only. This button does not work with auto-generated controls. You can customize an auto-generated control only in a section or harness that contains the control. To customize the control, specify the rule in the Control field on the Cell Properties panel and use the **Gear** icon next to the field to open the control's Parameters dialog.

©2025 Pegasystems Inc

Using the Table Type for input value validation

A property of mode `Single Value`, `Value List` or `Value Group` can use "table validation" to constrain the property's input value to one of a set of specified acceptable values, or to a pattern of values. At run time, the system verifies that the value matches one on the list, and validates the value is appropriate input for the property.

To use table validation:

1. Select `Single Value`, `Value List`, or `Value Group` for the Property Mode field. (You cannot use table validation for properties of any other modes.)
2. Select one of the values in the Table Type drop-down list (other than the `None` value). The appropriate fields for that type appear in the form.
3. Complete the fields appropriate for the selected Table Type value. See Property form General tab — Completing the Table fields for descriptions of the fields.

See Property form General tab — Completing the Table fields — Basics for details about how the system compares the set of acceptable values to the input values.

After you have specified the table validation you want and completed the appropriate fields:

1. In most cases, set the Control field to:
   - *PromptSelect*, to get a runtime presentation of a drop-down list of the acceptable values
   - *RadioButtons* to get a runtime presentation of a rectangular array of radio buttons for the acceptable values; this control requires parameters
   - *PromptFieldValues* when using the `Field Value` table type
2. Save the Property form and click the Preview toolbar button to test the presentation and values of the property.

©2025 Pegasystems Inc

> ⚠ **CAUTION:** If you update an existing property and revise the table details, additional steps may be necessary. See Revalidation is necessary after certain property updates in More about Properties.

Field	Description
Table Type	Select `None`, `Local List`, `Field Value`, `Class Key Value`, `Remote List`, or `Prompt List`: • `None` — No table validations apply. • `Local List` — The set of acceptable values is provided directly in the General tab of the Property rule form. See ../../../../platform/reference/rules/property-form-general-tab-configuring-validation.html#Property_form__General_tab___Configuring_validation__locallist. • `Field Value` — The set of acceptable values corresponds to a set of field value rules. See ../../../../platform/reference/rules/property-form-general-tab-configuring-validation.html#Property_form__General_tab___Configuring_validation__fieldvalueval. • `Class Key Value` — The set of acceptable values is defined using the keys of a known class (a class other than this property's class). See ../../../../platform/reference/rules/property-

©2025 Pegasystems Inc

Field	Description
	form-general-tab-configuring-validation.html#Property_form__General_tab___Configuring_validation__keyvalueval. • `Remote List` — The set of acceptable values consists of the elements of a Value List, Page List, Value Group or Page Group embedded within a single object. See ../../../../platform/reference/rules/property-form-general-tab-configuring-validation.html#Property_form__General_tab___Configuring_validation__remotelistval. • `Prompt List` — Acceptable values are enumerated as with a `Local List` , but for each internal value, a different localized value appears on input or output. See ../../../../platform/reference/rules/property-form-general-tab-configuring-validation.html#Property_form__General_tab___Configuring_validation__promptlistval.

Using the Preview toolbar button

For `Single Value` properties, you can preview the presentation and other aspects of the property.

©2025 Pegasystems Inc

Click the Preview toolbar button to see an approximate presentation of the property as it will appear in both input and display at run time. For properties not marked as Special , you can preview the representation of a specific value:

1. Type or select the value in the Input: area.
2. Click **Show output** to see the resulting display-only representation, in the Output: area.

The preview presentation does not reflect the *pyMinimumLength* property qualifier (when referenced on the Qualifiers tab) unless the Edit Input field contains *pyMinimumLength-Enforce*.

> **NOTE:** You cannot preview a *TextEncrypted* property if your system does not include a cipher algorithm.

Property form - Completing the General tab for Java Object modes

Property form - Completing the General tab for Java Property and Java Object modes

> **NOTE:** Use this table to complete the General tab fields that appear for a property that has a mode of `Java Object`, `Java Object List`, `Java Object Group`, `Java Property` or `Java Property List`. Such properties are typically used in conjunction with JSR 94 services (*Rule-Service-JSR94* rule type), EJB services (*Rule-Service-EJB*), Java service rules (*Rule-Service-Java*), or EJB connectors (*Rule-Connect-EJB*), and Java connectors (*Rule-Connect-Java*).

A `Java Object` property on the clipboard holds a reference to an instance of a Java object. A `Java Property` property represents a single property in the Java class that defines an instance of a Java class.

©2025 Pegasystems Inc

Properties of the Java Property modes support the Java pages feature that enables your application to attach external Java objects to clipboard pages, where they can be processed by any rule type. Properties of the `Java Object` modes can be manipulated within the Java steps of activities only.

For information about the Java pages feature, see *Working with the Java Pages Feature*, a document on the Integration area of Pega Community.

For basic information about the General tab and choosing a property mode, see Completing the General tab.

For properties of mode Java Object, a clipboard page containing this property can be saved into the database if the following are true: the Java Class field specifies the name of a Java class, the Serialize check box is selected, and the Java class specified implements the Serializable interface.

> **NOTE:** If the Java class does not implement the Serializable interface, a null value is stored for the property in the database.

Property form: General tab — Configuring validation

Providing validation of input data is a best practice to prevent invalid data from entering a work item or the business process. One way of doing that is to specify a set of acceptable values for a field. At runtime, the system compares the input value to the specified set, verifies whether the value has a match, and potentially formats the value (as appropriate) for the property.

> **NOTE:** For basic information about the General tab and choosing a property mode, see Completing the General tab — Choosing a mode.

A property of mode `Single Value`, `Value List` or `Value Group` can use table validation to constrain an input value to match one of the specified values or a pattern

©2025 Pegasystems Inc

of values. You define a table of data that describes what values are valid for this property.

To use table validation:

1. Select one of the values in the Table type drop-down list (other than the `None` value). The appropriate fields for that type appear in the form.
2. Complete the fields appropriate for the selected Table type value. These fields define the set (table) of acceptable values. See the following sections for details of the fields used for each type of table validation.

> **CAUTION:** If you update an existing property and revise the table details, additional steps may be necessary.

Basics

> **NOTE:** Table validation is part of property and page validation. If an input value for a property doesn't have a match in the set of table elements, the property name and the invalid value remain on the clipboard, but during validation the system adds a message and error text to the clipboard page that contains the property. The presence of the message ordinarily prevents the system from saving the clipboard page (which contains invalid data) into the database. The message does not prevent an activity from reading or using the (invalid) property value.

Although the value is compared with the table elements a case-insensitive way, the system corrects the case. For example, if the values recorded in a `Local List` on this tab are all lowercase, a user can enter at runtime the value in uppercase, mixed case, or lowercase letters. The system accepts a matching value but converts it to lowercase.

Display-only option

©2025 Pegasystems Inc

> **NOTE:** This check box appears when the selected Table type is `Field Value`, `Class Key Value`, `Remote List`, or `Prompt List`.

Local List table validation

Select `Local List` as the Table Type when your property value is permanently limited to a modest number of known values in a single language. (To accomplish a similar presentation with localized values, specify `Prompt List` instead of `Local List`.)

If there are to be a large list of values that are expected to change frequently, rather than having to maintain a long list here, use the field value table validation method instead.

Typically when using this option, the Control field is set to either the *PromptSelect* or *RadioButtons* standard control, so that at runtime, a selection list displaying the specified acceptable values is presented to the user.

You cannot use a `Local List` if the Property Type field is `Password`, `TrueFalse`, `Date`, or `DateTime`.

> **NOTE:** When `Local List` is selected, a Table Values array appears. In this array, specify those values that are acceptable input for this property. Enter each value as a literal. Order is not significant.

Using `Local List` is the simplest way to show a list to the user at runtime for the user to make a selection. Localization of the values is not supported for a `Local List`. The value displayed at runtime is the value stored in the clipboard.

©2025 Pegasystems Inc

Field	Description
Table values	You need not leave a blank entry to allow the property value to be empty. If you enter a value in the first row, it appears at runtime as the default value, and might be too easy for users to overlook. To encourage users to make a non-blank selection at runtime, leave the first entry blank. To require users to make an explicit choice, leave the first row entry and mark the corresponding field on a user form as required. **NOTE:** To encourage users to make accurate, fast selections, list the values in a meaningful order, such as alphabetically, or in a natural sequence (for example S, M, L, XL, XXL).

NOTE: If, at runtime, a user input or other processing assigns a non-null value to the property that is not on this list, the value is accepted but, when the page is later validated, the system adds an error message to the page on the clipboard.

Field Value table validation

Select `Field Value` as the Table Type when the set of valid values:

©2025 Pegasystems Inc

- Contains more than a few hundred values or the values are expected to change frequently, and
- The values are available at runtime within the keys of field value rules

For this method, you specify only the first two key parts of the set of field value rules. At runtime, the system locates the actual literal values by using the specified two key parts to locate the field value rules that contain the values. If a match for the input value is not found among the field values, the validation fails and the system returns an error.

Typically when using this option, the Control field is set to *PromptFieldValue*, to present a user with a fixed list of selection values to select from at runtime.

Field	Description
Field Value Description	
Class	Select a class that is the first key part (Applies To class) of one or more field value rules, where the third key part of the field value rule (the Field Value on the Field Value rule form) defines an acceptable input value for the property. (The class specified here need not match the Applies To key part of this property.)
Field name	Enter the second key part of one or more field value rules that have the specified Class as their first key part. This key part of the field value rule is usually a Single Value property in that class. Its name does not have to match the name of the property you are defining.

©2025 Pegasystems Inc

> **NOTE:** The field value — the third key part — of a field value rule is limited to 64 characters.

Field value examples

For example, the standard property *Work-.pyRootCause* uses field value rules to define possible root causes for an exception-oriented work item.

The first key part of the field value rules that provides the acceptable values is *Work-* (the Applies To class for these rules). The second key part is the literal *pyRootCause* (the Field Name for these rules).

The final key part of these field value rules varies to provide specific values. Recommended values for the *pyRootCause* property are `Conduct`, `Duplicate`, `Facilities`, `Finance`, `Policy`, `Product`, `Service`, and `User Error`. These values are predefined through standard field value rules.

You can add field values as the application evolves, and the values can be updated without having to edit the *pyRootCause* property itself.

In comparison, the standard property *Work-.pyStatusWork* also is restricted to values defined by field value rules. However, for this property, an activity and a control enforces the restriction, rather than table validation.

Reviewing field value rules for the property

After you save the property, you can open the Property Explorer from the Property form's Advanced tab to locate and review the list of existing field value rules that match the specified field value class and field name:

1. Click the Advanced tab. Click the **Explore** button at the bottom of the tab to start the Property Explorer in a new window
2. On the Property Explorer, click the Field Values tab.

©2025 Pegasystems Inc

Class Key Value table validation

Use `Class Key Value` to use a concrete class, rather than field value rules, as the source of acceptable values for the property. `Class Key Value` specifies a class to validate the input values against.

When you select `Class Key Value`, Table Description fields appear. Complete them to identify a range of keys in a class other than the Applies To class of the property you are defining.

Field	Description
Table Description	
Validation class	Select a concrete class that stores the set of acceptable values. Depending on the number of keys defined for the class (in the Keys array of the class's rule form, certain conditions must be met: If class has: — Then: One key value — Corresponds to the valid value for the property you are defining. Two key values — The first key value must act to group values for the second key. The second key value corresponds to a valid value for

©2025 Pegasystems Inc

Field	Description
	the property you are defining.
	Three key values: The first key value must correspond to a class (the standard property *pyClassName*). The second key value must act to group values for the second key. The third key value must act to group values for the third key.
Subset name (when appropriate)	Optional if the Validation class has only one key. Required if the Validation class has more than one key. Enter the value of the key in the Validation class that represents the subset that contains the list of acceptable values. For example, if a class `Data-CustomerTerritory` has key parts for state code and county name, to restrict input

©2025 Pegasystems Inc

Field	Description
	values for a property at runtime to county names in New York state, enter `Data-CustomerTerritory` as the Validation class and "NY" as the Subset Name . If you leave this field blank, the system uses the Property Name of the property you are defining in the Property form as a default. (The Property Name is the second key part of this property.)

Class Key Value examples

Single key example: The standard class *Rule-Obj-Class* is used to validate values entered into the Applies To field in the Create form when creating a new rule. To validate that input entered into the Applies To field is a valid, existing class, the property associated with that field in the New dialog specifies *Rule-Obj-Class* as the Validation class . The *Rule-Obj-Class* class has one key defined in its Keys array.

Two key example: A class `Data-CustomerTerritory` has two keys defined in its Keys array — state code and county name. To restrict input values for a property at runtime to county names in New York state, specify `Data-CustomerTerritory` as the Validation class and "NY" as the Subset Name .

Three key example: The standard class *Rule-Obj-FieldValue* used to validate properties has three keys defined in its Keys array. The first is *pyClassName*, the second is *pyFieldName* and is used to group the values for the third key (*pyFieldValue*), which are the allowed input values.

The `Class Key Value` validation method is a generalization of the `Field Value` method. For example, a Pega Platform system has fifty-one (51) standard field value rules that identify codes for the fifty United States and the District of Columbia. These

PEGA ©2025 Pegasystems Inc

field value rules have *Data-Party* as the first key part (the Applies To) and *pyState* as the second key part (the Field Name). The values for the third key part of these field value rules range from `AK` (for Alaska) to `WY` (for Wyoming).

Two standard properties, *Data-Party.pyHomeState* and *Data-Party.pyWorkState*, use these field values through the Class Key Values method. Those properties specify:

- Rule-Obj-FieldValue for the Validation Class
- pyState for the Subset Name

At runtime, if your system contains other field value rules in this key range (in an application RuleSet) such as PR for Puerto Rico, the system uses them as acceptable values for *pyHomeState* and *pyWorkState* also.

Remote List table validation

Choose `Remote List` when there are a large number of objects to be validated against (such as countries of the world) that you might not want to enter as individual values, or when the data changes frequently, but is available at runtime as the keys of a certain known class.

Whereas Class Key Value specifies a class to validate the input values against, Remote List specifies a particular instance of a class to validate against.

After you select `Remote List`, complete the fields that appear.

Field	Description
Class of instance	Select the class of the instance. After entering the class, you can skip to complete the Keys of instance fields and then return to complete the Reference to list or group field.
Reference to list or group	Select a property that has one of the List or Group modes — `Value List`, `Value Group`, `Page List`, or `Page Group`.

©2025 Pegasystems Inc

Field	Description
Reference in each to scalar property	Optional if the Reference to list or group is `Value List` or `Value Group`. Specify either an index value, a reference to one element of an aggregate property, or the `Use-Group-Name` keyword. Your choices depend on the Property Mode of the property in the Reference to list or group field. **Mode** / **Choices** Page List or Page Group: Identify a property that contains a value that is an index that identifies the specific page that contains the list. Value List: Leave blank. Value Group: Leave this field blank to edit against each value in the `Value Group`, or enter the keyword `Use-Group-Name` to indicate that the index values of the `Value`

©2025 Pegasystems Inc

Field	Description
	<table><tr><th>Mode</th><th>Choices</th></tr><tr><td></td><td>`Group` are the allowed values.</td></tr></table>
Reference to localized value	Leave blank.
Keys of instance	Click the **Add a row** icon to add a key. This array identifies the instance that contains the property value that contains the set of values. Identify keys in the correct order, matching the class definition.
Key field	Enter the name of a property that is a key part of the class.
Value	Enter the value for this key part for the instance.

Prompt List table validation

Select `Prompt List` and complete the associated fields to provide a mapping between a list of text values used in the property value internally and corresponding text values presented on input or output. For example, the text values presented at runtime can be human-readable labels that end users see and choose from, while the corresponding stored values can be shorter, easy-to-machine-reference values.

When you use `Prompt List` as the Table type , specify *PromptSelect* for the Control . At runtime, a selection list displaying the Prompt value values is presented to the user. If the values have been localized using the Localization wizard, the appropriate translated values are presented.

`Prompt List` tables are used when you want the actual value stored in the clipboard to be different from the value presented to the user 'on the glass' at runtime. This choice allows users to see information from the Prompt value column on both input

©2025 Pegasystems Inc

and display, while the system internally maintains the corresponding value from the Standard value column. For example, when the Standard value specifies a short value like EDate, the corresponding Prompt value can be a more human-readable value like Date to complete this task. The Standard value is the value stored in the clipboard, and the Prompt value is the value displayed to the user at runtime.

This feature is most useful to support localization. The Localization wizard localizes the values specified in the Prompt value field. When the wizard runs, it creates the appropriate field value rules for the Prompt Value entries, and then those field value rules are used to hold the translated text.

Field	Description
Prompt values	Click the **Add a row** icon to add a prompt. This array identifies the mapping of the internal values to the runtime-presentation ('on the glass') values.
Standard value	Enter a value. The system stores this value internally as the property value.
Prompt value	Specify the runtime-presentation alternative value. Use the SmartPrompt to select from available field value rules.

Property form - Completing the Advanced tab

Optional. Complete this tab to:

- Define a length in characters (for text properties)
- Prohibit this property from being the target property in Declare expression rules
- Define edits and validation criteria
- Recommend to your database administrator that the property is to be optimized as a separate database column

©2025 Pegasystems Inc

The format and contents of this tab depend on the Property Mode value selected on the General tab.

Basics

Field	Description
Max length	Optional. This field is visible only for `Value` mode properties of type `Password`, `Text`, or `Identifier`. You can restrict the value to a specific maximum number of characters in internal storage. Enter the number here. If a user or activity attempts to assign a longer value to the property, the clipboard does contain the longer value, but also has an associated message. **CAUTION:** If you leave this blank, Pega Platform™ does not limit the length of values of the property. However, operational considerations and system performance considerations might apply. For example, database software supporting the PegaRULES database might limit the size of the column to no more than 255 characters, or 1023 characters, or another value; this applies if the property is to be exposed. Even when the property is not exposed as a database

©2025 Pegasystems Inc

Field	Description
	column, sizes larger than 1,000 bytes could adversely add to the memory demands on the server and JVMs, depending on how many instances contain the large property and how often they are accessed on requestor clipboards. **NOTE:** When updating or overriding a property that has a Max Length defined, you can enter a smaller Max Length value but not a larger value.
Expected length	Optional. Enter the typical length in characters expected for the property being defined, as entered into a form. This does not affect the number of characters that can be stored internally in the property value. If you associate the standard control *FixedSizeForInput* with this property, the system uses this value to define the size of HTML input textboxes. Use the similar *FixedSize* control to apply this limit to both input and display presentations of the property. This field is visible only for `Value` mode properties.

©2025 Pegasystems Inc

Field	Description
Override sort function	Optional. Use to this field in special situations, to compare and sort values of this property in ways other than the natural sort sequence for the Type . If you leave this blank, the system uses a built-in sort order based on the Type. Identify here a function rule in the *Pega-RULES:Sort* library or a custom function in a custom Sort library. For a custom function, enter the ruleset name, a colon, and the sort function name. This field is visible only for `Single Value` mode properties.
Access when	Optional. This field is meaningful only when the Type is `TextEncrypted` and the mode is `Single Value`. Identify the second key part of an access when rule (*Rule-Access-When* rule type) that controls whether or not the clear-text or encrypted value appears in reports and forms. Design the access when rule to be true for those operators, situations, and times when the clear text of the property value is to appear, and false otherwise. See Implementing and using the TextEncrypted type.

©2025 Pegasystems Inc

Field	Description
Edit input	Optional. To convert data entered by a user into another format, identify an edit input rule to perform the conversion. This field is visible only for `Value` mode properties. If this property is to hold text that is composed using the *RichTextControl* control and so might contain HTML markup such as `<b>` or `<h2>`, select *pxFilterRichText*. This standard edit input rule deletes potentially insecure markup from the input value while retaining the legitimate display markup.
Use validate	Optional. Validation of a Value mode property always includes testing its value against the formats allowed by the Type and its maximum length. To provide stronger validation for this property, identify an edit validate rule. When users complete and submit a form containing a value for this property, the system first applies the edit input rule. This rule may transform the input text. Next, it applies the edit validate rule, which may cause a message to be added to the property indicating that the property value has failed validation.

©2025 Pegasystems Inc

Field	Description
	NOTE: To restrict this value to a (Pega Platform) Java identifier, enter *JavaIdentifier*. **NOTE:** Many standard edit validate rules are emulated by an equivalent JavaScript function that can be run on the client workstation before users submit a user form or flow action form. See client-side validation. This field is visible only for `Value` mode properties.
Column inclusion	Optional. If this `Single Value` property is to belong to a top-level object (not to an embedded page) in any concrete class, indicate whether this property is to appear in the Database Class Mappings gadget and Modify Schema wizard as a candidate for an exposed column: • Leave blank if you do not anticipate any benefit in making this property an exposed column. • Choose `Required` or `Recommended` to record your intent that a database administrator extend a database table schema to expose this property. These

©2025 Pegasystems Inc

Field	Description
	choices cause this property to appear as a candidate for column exposure in the Database Class Mappings gadget and Modify Database Schema wizard. The other choice `Optional` is equivalent to blank. **NOTE:** Your choice in this field only conveys advice to a database administrator; it does not cause the schema to change. You can update a property to revise this selection at any time. Using the Database Class Mappings gadget, you can compare the values in this field with the current schema, identifying properties marked as `Required` which are not exposed. To start this gadget, select Configure > Data Model > Classes and Properties > Database Class Mappings. See Property optimization using the Property Optimization tool and Data Model category — Classes and Properties landing page. **NOTE:** If the property is not a `Single Value` property, or is not at the top level of an instance

©2025 Pegasystems Inc

Field	Description
	of any concrete class, this field is not meaningful. **NOTE:** If this is a transient property (the Transient Property check box is selected), select Optional or leave this field blank. A transient property cannot be exposed ("optimized").
Do not save property data	Select to cause the value of this property to be omitted (equivalently, set to null) when any clipboard page containing this property is committed to the PegaRULES database. Similarly, the property has no value when an instance containing this property is opened to the clipboard from the PegaRULES database. Marking a property as transient, when appropriate, improves performance by reducing the size of the Storage Stream property value (the *pzPVStream* property and column). While on the clipboard, the property can have values of any length. Select only for properties — Single Value or aggregate — that have values thatdo not need to be saved in the database.

©2025 Pegasystems Inc

Field	Description
	If you mark a property as transient, but saved instances of objects already exist in the PegaRULES database, the values of this property are removed only when and if those objects are later opened and then recommitted. **NOTE:** The value of a transient property is not deleted when a requestor session containing a clipboard value for a transient property is passivated. The current value of the property is included in the saved passivation record, and restored when the requestor session is later activated. **NOTE:** You cannot expose a transient property as a database column. Because all values of the column would be empty, this is not useful. Similarly, you cannot mark a property as transient if Pega Platform determines that it is part of an external database table.
Optimized for classes	A list of the classes containing this property that have values as an exposed column in the PegaRULES database. This read-only list appears only for `Single Value` properties

©2025 Pegasystems Inc

Field	Description
	that have been exposed through the Property Optimization tool. See Property optimization using the Property Optimization tool.
Cannot be declarative target	Select to prevent this property from being the target of a Declare Expression rule. **NOTE:** This restriction applies to Declare Expression rules created after you save the Property form; it is not applied retroactively. This restriction is meaningful only for `Single Value`, `Page`, or `Java Property` properties.
Cannot be included as input field	Select to turn on the *pySpecial* restriction for this rule. If selected, users cannot directly enter a value for this property on an HTML form. This restriction improves system security and reduces the chance of errors that can occur when one property is in some cases computed and in other cases directly input. Such properties are sometimes called special properties.

©2025 Pegasystems Inc

Field	Description
	NOTE: Choose a naming convention for special properties, to help everyone on your team remember them. (Standard special properties have a property name starting with the letters px.)
Allow use as reference property in activities	Select to make this property a reference property, a property that can link to a source property. Reference properties can reduce the need for duplicating data values on multiple pages or objects, and can simplify lengthy property references. Use the Property-Ref method in an activity to establish the contents of a reference property. **NOTE:** If this property is of mode `Page` or `Page List` and the `Page Class` is a concrete class, you can in some cases use auto-population rather than use the Property-Ref method.
Cannot be localized in UI controls	Select to exclude this property from being localized if the property will not appear in the UI nor have a field value. Excluding such properties increases localization performance.

©2025 Pegasystems Inc

Property Qualifiers

Optional. Complete this array to associate one or more property qualifier rules (*Rule-Obj-Property-Qualifier* rule type) with this property.

Each property qualifier you list here provides additional information at runtime about the property, and may alter the behavior in computations or presentation of the property.

> **NOTE:** You can provide a value for the qualifier on this tab, thereby defining a permanent value. Or when desired, you can leave the value blank and use Java methods in the PublicAPI to set or test values dynamically.

> **CAUTION:** If you update an existing property that references the *pyDecimalPrecision* qualifier, additional steps could be necessary.

Field	Description
Qualifier	Optional. For a permanent qualifier, enter the value in a pop-up dialog box: • For the standard qualifier *pyDecimalPrecision*, enter a literal constant integer such as 2 to indicate how many decimal places are to be presented when this property value appears at runtime. (Use *pyDecimalPrecision* only with properties with a `Decimal` type.) • For the standard qualifier *pyMinimumLength*, enter a literal constant minimum length, such as 10.

©2025 Pegasystems Inc

Field	Description
	• For the standard qualifiers *pyUnits* and *pyCurrency*, identify an application property (found at runtime on the same page as this property) that holds the units or currency code applicable to this property. **NOTE:** To allow the value of a qualifier to be set dynamically and temporarily at runtime, leave this field blank.

Property Explorer

Click **Explore** after you save the form if you completed the Table Type field on the General tab and want to preview the presentation of choices for this property when presented in input mode. This display also identifies any Declare Expression rules that reference the property, and dependencies among expressions.

More about Properties

Standard properties—meaning all the properties in the *Pega-RULES*, *Pega-IntSvcs*, *Pega-WB*, and *Pega-ProCom* RuleSets—have names that start with px, py, or pz.

These three prefixes are reserved. You cannot create new properties using these prefixes; however, you can override these standard properties with a custom property of the same name (without changing the property mode or type).

©2025 Pegasystems Inc

Prefix	Meaning
px	Identifies properties that are special, meaning that the values cannot be input by user input on an HTML form.
py	Properties with names that start with py are not special, meaning that values can be input by users on an HTML form.
pz	Properties with names that start with pz support internal system processing. Users cannot directly manipulate `pz` properties. Your application may examine these values, but do not set them. The meaning of values may change with new product releases.

Updating properties

If you update or override a property of mode `Single Value`, `Value List` or `Value Group`, you can change the type, but only to a narrower type. This does not cause any runtime conversions of property values.

The following table shows the changes that are allowed.

To type	`DateTime`	`Date`	`Time of Day`	`Integer`	`Decimal`	`Double`	`True or False`	`Text`
From type								
`DateTime`	✓	✓						

©2025 Pegasystems Inc

`Date`		✓						
`Time of Day`			✓					
`Integer`				✓				
`Decimal`					✓			
`Double`				✓	✓	✓		
`True or False`							✓	
`Text`	✓		✓	✓	✓	✓	✓	✓

Revalidating after property updates

If you override or update a property, you must ensure that saved instance data (if any) conforms to the revised property definition. This might require a once-only conversion activity.

In addition, if you override or update a property by changing any of the following fields, certain rule types that reference the property might not automatically reflect the change at runtime.

- **Property Mode** on the General tab
- **Property Type** on the General tab
- Table types or contents on the General tab
- **Expected Length** on the Advanced tab

©2025 Pegasystems Inc

- **Page Class** on the General tab, for page list or page group properties
- For properties with Property Type of `Decimal` and the *pyDecimalPrecision* qualifier, the decimal precision

The impacted rule types are validate rules and most stream rules (harness, section, paragraph, flow action, HTML, and HTML Fragment). Use the following procedure to update rules that reference the updated property.

- Open and save the rules that reference the property, if the rules are not in a locked ruleset.
- If the rules are in a locked ruleset and they are non-stream rules such as activities, data transforms, and so on, reassemble the rules either using the reassemble API or by truncating the assembled classes table.

  To use the reassemble API:

  1. Obtain the *pxinsid* and *pxobjclass* for the rules by viewing and searching the XML for the calling rule.
  2. In Admin Studio, click **Resources** > **APIs**.
  3. In the **System management** section, click the reassemble API.
  4. In **Parameters** section, enter the pxObjclass and pxinsid.
  5. Click **Try it out!**
  6. If the above procedure does not work:
     a. Create the `fua/invalidation/filter/PropertyChange` `prconfig.xml` setting and set it to `false`.
     b. Clear the temporary directory and caches, and restart the server(s).

  To truncate the assembled classes table:

  1. Truncate `pr_assembledclasses`.

     If you do not want to truncate the entire assembled classes table, you can delete the targeted stream assemblies that reference this property as follows:

©2025 Pegasystems Inc

Delete from `pr_assembledclasses` where `pzpackage in ('com/pegarules/generated/corr', 'com/pegarules/generated/flowaction', 'com/pegarules/generated/html', 'com/pegarules/generated/html_fragment', 'com/pegarules/generated/html_harness', 'com/pegarules/generated/html_paragraph', 'com/pegarules/generated/html_property', 'com/pegarules/generated/html_section' , 'com/pegarules/generated/xml')`

2. Delete the PRGenJava directory on all nodes.
3. Restart all nodes.

- For stream rules, use the procedure for truncating the table above.

Password types and encryption

Properties of type `Password` correspond to the HTML element <INPUT TYPE="PASSWORD" >, and so do not echo when typed into a browser form. However, on the clipboard, the value of a property of this type may be in clear text or encrypted, depending on how the value was set.

Normal assignments to a `Single Value` or `Java Property` property of type `Password` by the Property-Set similar methods create a clear-text value on the clipboard. To store an encrypted value, use the property of type `TextEncrypted`.

Deleting properties

Do not delete a property that is referenced in other rules. Check whether a property is the target of a Declare Expressions rule before deleting it.

Large property value storage

Your application does not restrict the length of property values. However, you can set the size limit for a property value by setting the Max length field on the Advanced tab of the Property form. Long properties, especially if you have thousands of them in memory, can affect performance, including memory and database traffic. The JVM

PEGA ©2025 Pegasystems Inc

memory limit for the system must be large enough to handle the number of long properties that are processed.

In addition to performance considerations, you might not be able to expose properties with long values as database columns, depending on your database software's aggregate size limit for exposed columns. If a property exceeds your database's aggregate size limit, making the property too long to expose as a database column, consider dividing the property and storing it as a number of shorter properties.

If your property is exposed, set the property value size in the Max length field to equal the maximum size of the database column to ensure that your data fits into the column.

Implementing and using the TextEncrypted property type

`Single Value`, `Value List`, and `Value Group` properties can be encrypted by using the `Password` and `TextEncrypted` encryption types. Both types produce encrypted or hashed values for the property value within the PegaRULES database, and both types offer some degree of security within the user interface. Another encryption type, `PropertyEncrypt`, can be used for all properties when your implementation uses attribute-based access control.

> **NOTE:** The `PropertyEncrypt` encryption type is preferred to the `TextEncrypted` type described here, as it takes effect more broadly and can be used for all property types.

If you have a legacy implementation that uses the `TextEncrypted` type, consider these factors:

- The `Password` type requires no advanced configuration or Java skills to set up. Several standard properties implement this type; for example, the property *Data-*

©2025 Pegasystems Inc

Admin-Operator-ID.pyPwdCurrent. Pega Platform applies the one-way MD5 algorithm to this value, which is never sent to any external system.

- The `TextEncrypted` type requires one-time Java coding of encryption Java functions of your choice to implement a Public API Interface.
- On Pega Platform forms, the display of a value of a `Password` property is a string of asterisks, for all users, in all situations. The Password value is never decrypted. In contrast, the value of a `TextEncrypted` property can appear in clear text or as asterisks, depending on the runtime outcome of an access when rule (*Rule-Access-When* rule type). Thus your application can make the clear-text value visible to specific users, or on certain reports, or during specified time periods.
- `Password` properties are initially added to the clipboard as unencrypted, clear text values. The system computes the hashed value only as the page is committed to the PegaRULES database. Thereafter, the hashed value appears in both the clipboard and the database row. Properties for passwords for the Operator IDs, rulesets, and ruleset versions are of mode `Password`.
- `TextEncrypted` values are always encrypted on the clipboard and in server-to-database network messages.

Implementing the TextEncrypted type

1. Choose and apply the type of cipher for your implementation of Pega Platform. Choose from platform cipher or custom cipher.

2. Create one or more properties that use the `TextEncrypted` type. Complete the General tab:

- Set the Type to `TextEncrypted`.
- Set the Control field to *ShowTextEncryptedPropertyValue*.
- In the Access When field, enter the second key part (When Name) of a Rule-Access-When rule that determines whether the property value appears as decrypted cleartext or asterisks. The Rule-Access-When rule can depend on any properties in the Applies To class, plus properties on the `pxRequestor` page or other clipboard pages.

©2025 Pegasystems Inc

3. Reference the property normally in other rules.

> ⚠ **CAUTION:** If you optimize a `TextEncrypted` property for reporting, make the exposed column size greater than the number of characters of your longest cleartext value to avoid truncation of the exposed values. The required size of a `TextEncrypted` property depends on your cipher. In most cases, for 64 characters of cleartext, 255 characters are adequate for the encrypted value.

Comparisons

You can use a `TextEncrypted` property in expressions, testing for equality or inequality only. Call the standard function rule *encryptPropertyValue()* to encrypt the comparison value (a constant, a property value, or computed text value) before the comparison. For example:

```
@encryptPropertyValue("Virginia") == .myEncryptProperty
```

```
.myEncryptProperty != @encryptPropertyValue(.pyLabel)
```

The access when rule test and automatic encryption (or decryption) occur only as users interact with a form. In all other cases, your application must explicitly call the functions.

> ⚠ **CAUTION:** When you use comparisons other than equality or inequality, the results are unpredictable.

Type conversions

Pega Platform does not perform automatic type conversions for `TextEncrypted` properties during Property-Set operations. As a result, it is rarely useful to directly

PEGA ©2025 Pegasystems Inc

assign an encrypted value to another property, or assign another property value to a `TextEncrypted` property.

When a `TextEncrypted` property is set to a value, the system encrypts the value unless it is already encrypted. Consider the following example:

In an activity, a Property-Set method operates on two properties MyEncrypted of type `TextEncrypted`, and MyText of type `Text`.

After the first of these three lines executes, the value in property MyEncrypted is encrypted. After the second line, the value in property MyText matches the value of MyEncrypted. No encryption or decryption takes place. After the third line, MyEncrypted holds the encrypted value from "Rosebud." Encryption is implicit.

PropertiesName	PropertiesValue
.MyEncrypted	=@encryptedPropertyValue("Hello World")
.MyText	.MyEncrypted
.MyEncrypted	"Rosebud"

Reports

To use a `TextEncrypted` property as a selection criteria on the Content tab of a report definition rule:

1. Expose the `TextEncrypted` property a database column.
2. Enter the clear text value (or a property reference for the comparison) in the Value field.
3. Select either `Is Equal To` or `Is Not Equal To` for the comparisons.
4. Enter *EncryptTextPropertyValue* in the Edit Input field.

> ⚠ **CAUTION:** If you use comparisons other than `Is Equal To` or `Is Not Equal To`, the results are unpredictable.

©2025 Pegasystems Inc

Services and connectors

If an external system calls a service and sends to Pega Platform a (clear text) value for a field that is mapped to a `TextEncrypted` property, the value becomes encrypted as soon as it reaches the clipboard.

Responses to service calls and connector requests normally send only the encrypted value. Your application can call the standard function *@decryptPropertyValue( )* to send the decrypted, clear text value, but only in a context when the access when rule is true.

> **NOTE:**
>
> Do not send an encrypted property value to an external system that expects the clear-text value.
>
> On the Clipboard tool display, the value of a `TextEncrypted` property is blank.
>
> Except as described here, Pega Platform treats the value of a `TextEncrypted` property similar to a `Text` property. Your application can place dates or numbers in the value, but validation does not occur.

JavaServer Pages tags

JavaServer Pages tags can appear in all rule types derived from the *Rule-Stream* class.

Correspondence	Control
Correspondence Fragment	List View (deprecated)
Flow action	Paragraph
Harness	Section
HTML	Summary View (deprecated)

PEGA ©2025 Pegasystems Inc

HTML Fragment	XML

The following tags govern how the stream processing facility converts and assembles a text stream for a user's browser, for outgoing correspondence, or for outgoing XML text.

Tag	Use
acDataSource	Defines the source of text values for an autoComplete JSP tag.
assert	Use to make processing more efficient when working with one or more concrete classes and the Obj-List method.
autoComplete	Supports a user interface drop-down facility that matches candidates from the first few characters a user types.
binaryfile	Used within the `static` JSP tag to identify a binary file rule.
bundle	Used within the `static` JSP tag to identify a static content bundle rule.
choose	Use with the when tag to select one section of source code to process. The otherwise JSP tag may appear within the choose tag.
chart	Display a chart derived from a summary view rule.
contentURL	Include a non-rule-resolved image in a stream.
dataGrid	Reserved.
dsField	Supports the acDataSource tag.
dsFields	Supports the acDataSource tag.

©2025 Pegasystems Inc

Tag	Use
file	Used within the `static` JSP tag to identify a text file rule.
forEach	Iterate through the properties contained in a `Page` or `Page List` structure.
include	Insert another stream rule into the current stream.
listView	Support run-time display of list view reports within sections. The param JSP tag may appear within the listView tag.
lookup	Retrieve and include the value of a property from an instance that is not present on the clipboard.
onlyonce	Cause a named section of text to be evaluated during stream processing only the first time it appears.
r	Include the value of a property on the clipboard. Short form of the reference tag.
reference p:r	Include the value of a property on the clipboard.
save	Save a value with an associated name for reuse later in the current stream.
static	Include the contents of one or more static file bundle rule (*Rule-File-Bundle* rule type) or text file rules (*Rule-File-Text* rule type) in the stream.
url	Encode a URL string on an HTML form with the Pega Platform keywords.

©2025 Pegasystems Inc

Tag	Use
when	Test a Boolean condition or when rule, and include or exclude source text based on the results.
withEmbedded	Change the current embedded page.
withPage	Change the current base page.
withReference	Make a new property the currently active property.
withTarget	Override the control rule that applies to a property.

Additional tags can appear within the scope of one of these tags. These additional tags are documented with the enclosing tag. For example:

- `elseassert` tag, within the `assert` tag
- `key`, within the `lookup` tag
- `option`, within the `url` tag
- `otherwise`, within the `choose` tag
- `param`, within the `listview` tag

- **About the static JavaServer Page tag**
- **acDataSource JavaServer Page tag**
- **assert JavaServer Page tag**
- **autoComplete JavaServer Page tag**
- **binaryfile JavaServer Page tag**
- **choose JavaServer Page tag**
- **JSP tag — chart**

©2025 Pegasystems Inc

- contentURL JavaServer Page tag
- dataGrid JavaServer Page tag
- dsField JavaServer Page tag
- dsFields JavaServer Page tag
- file JavaServer Page tag
- forEach JavaServer Page tag
- include JavaServer Page tag
- listView JavaServer Pages tag
- lookup JavaServer Page tag — lookup
- onlyonce JavaServer Page tag
- p:r JavaServer Page tag
- Reference properties indirectly in JSP tags
- reference JavaServer Page tag
- save JavaServer Page tag
- URL JavaServer Page tag
- when JavaServer Page tag
- withEmbedded JavaServer Page tag
- withPage JavaServer Page tag
- withReference JavaServer Page tag
- withTarget JavaServer Page tag

PEGA ©2025 Pegasystems Inc

About the static JavaServer Page tag

Use the `static` JSP tag to cause one or more static file bundles, individual text files, or individual binary files to be included in the current stream. A static file bundle rule identifies a list of JavaScript, image, and Cascading Style Sheet files to be sent to a browser session together in one HTTP response, to reduce HTTP traffic and improve response time to the browser user.

For example:

```
<pega:static type="script" app="webwb" >
    <pega:bundle name="pega_tools" />
    <pega:bundle name="acmestyles" />
    <pega:file name="markov.js" />
    <pega:binaryfile name="johnson.gif" app="webwb" />
</pega:static>
```

This tag references two static bundle rules *pega_tools.script* and *Acmestyles.script*, which in turn may identify multiple text file rules (*Rule-File-Text* rule type) containing JavaScript scripts.

The <pega:file > tag identifies a single text file rule (a JavaScript file, webwb.markov.js) to be included in the stream. Similarly, the <pega:binary tag> identifies a single binary file rule (an image) to be included in the stream.

When to use

For example, when entering HTML source code directly into the **HTML** tab of a control rule (or similar form), you can minimize the size of the resulting HTML document with the <pega:onlyonce >, <pega:bundle> and <pega:static > tags.

Complete syntax

In the syntax presentation below:

©2025 Pegasystems Inc

- JSP delimiters <pega:...> and </pega:...> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:static app="applicationname" type="type">
<!-- one or more pega:bundle JSP tags -->
<!-- one or more pega:file JSP tags -->
</pega:static>
```

The `type` attribute is required.

You can include a pega:when tag within the static tag scope, to conditionalize which bundles or files are included.

References to CSS files (in the form of the <link rel="stylesheet" ...> tag) must appear within the <head> element of an HTML document. In contrast, a <script ..> tag can appear anywhere in the document.

Attribute	Value
`app`	The first key part (**Application Name Directory**) of the text file rule included in the <pega:static..> tag. All files included within one <pega:static ..> tag must have a common Application Name
`type`	One of two values to identify the type of content in the bundle rules: • `script` — all listed bundle rules have `script` as the second key part • `style` — all listed bundle rules have `style` as the second key part
`prefixURL`	Reserved. Do not use.

©2025 Pegasystems Inc

Example

If your application includes JavaScript files, CSS files, or other static content saved in multiple text file rules that are needed together at runtime on the user workstation, group them into a bundle.

acDataSource JavaServer Page tag

Use the `acDataSource` tag within an `autoComplete` tag to compute the list of values presented to the user.

The standard control rule *AutoComplete* incorporates an `autoComplete` JSP tag, and allows you to set most attributes of the `autoComplete` and `acDataSource` JSP tags as parameters. Enter an `autoComplete` JSP tag into hand-crafted HTML code only in special situations where the standard control does not meet your needs.

Example

```
<pega:autoComplete name="AC" highlight="true" size="50" minChars="2">
    <pega:acDataSource name="ACDS" type="ClipboardPage"
    sourceName="ACDatasource" clientCache="false"
    searchPropertyName="Employee" displayField="EmpName"
    partialSearch="true" ignoreCase="true"
    maxResults="20" allFields="true" />
</pega:autoComplete>
```

This acDataSource tag (within an autoComplete tag) identifies a clipboard page named ACDataSource as the source of values. That page contains a `Page List` property named Employee; each embedded page of the Employee contains a `Single Value` property named EmpName.

If the user types the two characters "AL", this tag instructs the system to search the EmpName values for names that start with "AL", such as Alan, Allen, or Alto. The search is not case sensitive, and halts after 20 matches are found. When the user selects a

©2025 Pegasystems Inc

match from the drop-down display, all fields from that page of the Employee list are selected.

Complete syntax

In the syntax presentations below:

- Square bracket characters `[` and `]` define optional parts of the tag. Do not type the brackets.
- JSP tag delimiters <pega and `/` > mark the start and end of the tag.
- A vertical stroke character | separates alternatives.
- Replace any text in italics with your choice of value of that type.

```
<pega:acDataSource
      name = "widgetname"
      sourceName ="sourcename"
      [type = "Activity|ClipboardPage" ]
      [activityClassName ="class" ]
      [clientCache ="true|false"]
      [searchPropertyName ="pagelistprop"]
      displayField = "propertyname"
      [partialSearch = "true|false" ]
      [ignoreCase = "true|false"]
      [maxResults = "nnn"]
      [dsFields = "tag" ]
</pega:ACDataSource>
```

The `name`, `SourceName`, and `displayField` attributes are required.

Attribute	Value
`name`	Name of this acDataSource.

©2025 Pegasystems Inc

Attribute	Value
`type`	`Activity` if the system is to execute an activity to construct a page containing the list of values, or `ClipboardPage` if the values are already present in a page with the appropriate structure. If omitted, the default is `Activity`.
`sourceName`	Clipboard page name, if type is `ClipboardPage`, or the second key part of the activity name, if type is `Activity`.
activityClassName	First key part of the activity identified in the `sourceName` field, if the type is `Activity`. If omitted, the system uses the class of the primary page to locate the activity.
`clientCache`	If true, HTML code causes the first-use results of this control to be cached in the client browser. Default is false.
`searchPropertyName`	Name of a `Page List` property that contains pages that contain a matched text value. If omitted, the property name pxResults is assumed.
`displayField`	Reference to a scalar property embedded on each page of the `Page List` property identified in the `searchPropertyName` attribute.
`partialSearch`	If true, text string matches are based on comparisons of user input with the initial portion of text values, so user input of "AL"

©2025 Pegasystems Inc

Attribute	Value
	matches ALPHA and ALABAMA but not PALINDROME. If false, text string matches can occur anywhere, so user input of "AL" matches ALPHA, PALINDROME, and PRINCIPAL. If omitted, the default is false.
`ignoreCase`	If true, matching of user input with text values to display is not case sensitive, so "ABC" matches "aBC" and "Abc". If false, matching is case sensitive. If omitted, the default is true.
`maxResults`	A positive integer setting a maximum limit on the number of matches to be displayed. If omitted, the default is 10.
`allFields`	If true, all fields are selected from the property identified in the `searchPropertyName` attribute. If false, only those properties specified in the `dsFields` attribute.
`dsFields`	This optional JSP tag identifies a list of dsField attributes.
`dsField`	This optional JSP tag, with the scope of the dsField tag, identifies the property to be displayed as a drop-down list element.

assert JavaServer Page tag

Use the `assert` JavaServer Page tag to improve processing efficiency when displaying the results of the Obj-List method.

©2025 Pegasystems Inc

Summary

In an activity that uses the Obj-List method, you identify the concrete classes for the `Page List` property that holds the results from the method. However, when you display the `Page List` in an HTML display, the system has no way — other than the Assert tag — to indicate in the HTML which classes are involved.

Accordingly, at runtime the system rechecks the class of each page in the list. The `pega:assert` tag provides a way to indicate in the HTML which classes to use for each page.

If you're working with a class group, use the `elseassert` option for each class in the class group. In addition, use a final `elseassert` option to display something when the system doesn't find anything in any of the classes listed. Someone later might add a class to a class group and neglect to update the Assert tag in HTML forms. In such cases, the `elseassert` option can be important for debugging.

Complete syntax

In the syntax presentations below:

- Square bracket characters `[` and `]` define optional parts of the tag. Do not type the brackets.
- JSP delimiters <pega: and /> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:assert name="class-name1" >
    insert content
    <pega:elseassert name="class-name2" />
    insert content
    <pega:elseassert />
    insert default content
</pega:assert>
```

©2025 Pegasystems Inc

Example

For example, in a worklist, the data belongs to a page of class *Code-Pega-List*. Each `pxResults` page is limited to one or a few classes. The assert tag reduces the number of rules the system reads to generate the display.

```
<pega:forEach name=".pxResults" >
    <pega:withEmbedded name="$this" >
        <pega:assert name="Assign-Worklist" >
            insert row definition
        </pega:assert>
    </pega:withEmbedded >
</pega:forEach>
```

You can assess the benefits of the assert tag by using the Performance tool to compare the Rule Count number when the assert tag is used with the number when it is not used.

JavaServer Pages tags

autoComplete JavaServer Page tag

Use the `autoComplete` tag to present a drop-down list of candidate text values that a user can select from for an input text box. The text values can be produced by an activity or can be on a clipboard page.

At run time, the **Autocomplete** icon indicates that the field is supported by an autocomplete tag.

- Unlike a Local List (defined as a **Table Type** on the **General** tab of the Property form), the list presented at run time by an autocomplete tag is not static and can vary from time to time or from user to user.

©2025 Pegasystems Inc

- Unlike the Dynamic Select control, the list presented at run time need not be obtained from a *Code-Pega-List* structure (typically produced by querying the PegaRULES database or another database).

The standard control *AutoComplete* incorporates an `autoComplete` JSP tag, and allows you to set most attributes of the `autoComplete` and `acDataSource` JSP tags as parameters. Enter an autoComplete JSP tag into hand-crafted HTML code only in special situations where the standard control does not meet your needs.

Example

This JSP tag begins operation after the user types two input characters. It presents a drop-down list for a text box that is 50 characters wide, highlighting the characters that match user input typed in each list element. The data is retrieved by the `acDataSource` JSP tag.

```
<pega:autoComplete
    name="AC"
    highlight="true"
    size="50"
    minChars="2">
        <pega:acDataSource name="ACDS" type="ClipboardPage"
        sourceName="ACDatasource" clientCache="false"
        searchPropertyName="Employee" displayField="EmpName"
        partialSearch="true" ignoreCase="true"
        maxResults="true" allFields="true" />
</pega:autoComplete>
```

Complete syntax

In the syntax presentations below:

- Square bracket characters `[` and `]` define optional parts of the tag. Do not type the brackets.

©2025 Pegasystems Inc

- JSP tag delimiters <pega and `/` > mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:autoComplete name="name"
      [highlight="true|false" ]
      [size = "nn" ]
      [sendingTimeout = "nn" ]
      [delimiter = "zzz"]
      [minChars ="nn" ]
      [listingWidth = "nn"]
      [value = "value"] >
<pega:acDataSource .... />
</pega:autocomplete>
```

The `name` attribute is required.

Attribute	Value
`name`	Name of this autocomplete control. You can use the keywords `$THIS-NAME` or `$SAVE` (saved variable name) as the value of this attribute.
`delimiter`	Optional. Character or characters to use as a separator, for optional fields. If omitted, three hyphens "---" are the delimiter.
`highlight`	Optional. "true" if the text that matches user input is to be highlighted, "false" otherwise. Defaults to "false" if omitted.
`sendingTimeout`	Optional. An integer specifying the number of milliseconds delay before the drop-down list appears. If omitted, the default is 10 milliseconds.

©2025 Pegasystems Inc

Attribute	Value
`size`	Optional. An integer specifying the width in characters of the input text box. Defaults to "50" if omitted.
`typingTimeout`	Optional. An integer specifying the number of milliseconds delay after the user types the minimum number of characters before the autocomplete processing starts. Defaults to 10 milliseconds if omitted.
`listingWidth`	Optional. Width of the element in pixels. If omitted, zero.
`minChars`	Optional. Number of characters the user must type or paste into the field before autocomplete processing starts. If omitted, the default is 1 character.
`value`	Optional. Initial value that appears in the text box. You can use the keywords `$THIS-VALUE`, `$SAVE` (saved variable name) or reference a parameter with `param.name` syntax in this field.

binaryfile JavaServer Page tag

Use the `binaryfile` JSP tag to identify a single binary file rule (*Rule-File-Binary* rule type) to be included in the current stream.

Use of this tag when sending an image or other binary object to the workstation can eliminate HTTP 304 (Not modified) error responses and improve static content performance.

For example:

©2025 Pegasystems Inc

```
<pega:binaryfile name="mycologo.jpg" app="webwb" />
```

This tag references the binary file rule named `webwb.mycolog.jpg`, which contains an image.

Complete syntax

In the syntax presentation below:

- JSP delimiters <pega:...> and </pega:...> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:binaryfile name="myname.type" app="application" / >
```

The `name` attribute is required.

Attribute	Value
`name`	Second and third key part of a binary file rule to be sent as static content.
`app`	First key part of a binary file rule to be sent as static content.

JavaServer Page tags

choose JavaServer Page tag

Use the `choose` JSP tag to select at most one enclosed branch of when tags.

Include one or more `when` tags and a final `otherwise` tag within the scope of the choose tag.

- If none of the `when` tags evaluate to true, HTML or XML text within the `otherwise` tag is processed.

©2025 Pegasystems Inc

- For the first of the `when` tags that evaluates to true, HTML or XML text within that when tag is processed. Subsequent stream processing continues after the close of the `choose` tag.

The choose tag has no attributes. For example:

```
<pega:choose>
    <pega:when zzzz1 >
        <!-- HTML or XML code to execute for first when -->
    </pega:when>
    <pega:when zzzz2 >
        <!-- HTML or XML code to execute for second when -->
    </pega:when>
    <pega:otherwise>
        <!-- selected if no previous when tag evaluates to true -->
    </pega:otherwise>
</pega:choose>
```

where zzzz1 and zzzz2 are a name=, test=, or other valid set of when JSP tag attributes.

You can nest when tags within when tags. You can't use JSP tags other than `when` and `otherwise` within the scope of a choose tag.

JavaServer Page tags

- when JavaServer Page tag

JSP tag — chart

Use the `chart` tag to present an interactive chart defined by a summary view rule, that has `Interactive` as the **Chart Output Type** value on the **Chart** tab of the rule form.

©2025 Pegasystems Inc

Use the chart tag only in advanced situations where hand-crafted HTML code and advanced options are necessary. In most cases, you can add a chart display to a Harness, Section, or Flow action directly using the **Chart** control in the Advanced Group of controls.

Syntax

If you do need to use the JSP tag, use this syntax:

```
<pega:chart
    name="Purpose"
    className="AppliesTo"
    owner="ALL"
    action="refresh"
    width="NNN"
    height="NNN"
    timeout="NNNNN"
</pega:chart>
```

The `name` attribute is required. All attributes are lowercase. Enter attribute values in exact case.

Attribute	Value
`name`	Second key part — **View Purpose** — of a summary view rule.
`className`	Optional. **Applies To** key part of a summary view rule. If omitted, the class of the primary page in the runtime context is used.
`owner`	Optional. Final key part — **Owner** — of a summary view rule. If omitted, the default value is `ALL`.

©2025 Pegasystems Inc

Attribute	Value
`action`	Optional. Choose: • `refresh` — To force recomputation and regeneration of the interactive chart. This is the default value if omitted. • `reuse` — to recompute and regenerate the interactive chart only if a previous file has expired.
`height`	Optional. Enter a number for the height of the interactive chart in pixels, or a percentage of the window size followed by the % character. If present, this overrides any value in the summary view rule.
`width`	Optional. Enter a number for the width of the interactive chart in pixels, or a percentage of the window size followed by the % character. If present, this overrides any value in the summary view rule.
`timeout`	Optional. Enter a number of seconds to cache the chart.

If the summary view depends on parameters, you can add parameters to the current parameter page using notation similar to:

```
<pega:param Name="NAME" value="VALUE" />
```

where the VALUE can be a literal constant or an expression.

©2025 Pegasystems Inc

Examples

The standard HTML rule *Data-Gadget.IChartsDashboard* presents the *WorkManager* portal using the chart JSP tag.

The HTML code below displays the interactive chart named *Work-.CountResolvedbyResolverTop5IC* executed in the class context of a work pool determined at runtime. The image occupies the full width of the window, but only half the height. The SinceDate parameter, part of the selection criteria, is set to 30 days ago by calling a function rule.

The generated image is stored as a static content item, but expires after an hour (3,600 seconds) and must be regenerated if needed after that timeout period.

```
<pega:chart
    name="CountResolvedByResolverTop5IC"
    className="$SAVE(workpool)"
    width="100%"
    height="50%"
    timeout="3200">
        <pega:param name="SinceDate"
        value="<%=pega_rules_businesscalendar.addDays("",-30, false,"")%>"/>
</pega:chart>
```

JavaServer Page tags

contentURL JavaServer Page tag

Use the `contentURL` JSP tag to include a system-generated cleartext URL in the current stream that requests an image saved in an image content data instance (*Data-Content-Image* class). This is a reusable, unhashed URL that can be used more than once to refer to the image.

For example:

©2025 Pegasystems Inc

```
<img src='<pega:contentURL name="filename.fileType" path="/filePath/" classType="Image" />' >
```

Generates the following cleartext URL:

```
<img src='../datacontent/Image/filePath/fileName.fileType'>
```

An image retrieved from an Image Content instance is the same for all requestors.

Example

For an example, review the standard HTML rule *@baseclass.Operator-Profile-Full*. This rule supports the display of operator images in the **View** > **My Profile** display. If an image content data instance for an operator is associated with an operator ID, that file is retrieved and included in the My Profile display. If no data content data instance is associated with the operator ID, the default image from the binary file rule *image.dialogprofileimage.gif* appears.

Complete syntax

In the syntax presentation below:

- JSP delimiters <pega:...> and </pega:...> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.
- Square brackets [ ] surround optional attributes.

```
<pega:contentURL name="filename.filetype"
    [path="filepath" ]
        classType = "Image"
    [ enforceAbsoluteURL = "true|false" ]
/>
```

The `name` and `classType` attributes are required.

©2025 Pegasystems Inc

Attribute	Value
`name`	Second key part (**File Name**) and third key part (**File Type**) of an image content data instance.
`path`	First key part (**File Path**) of an image content instance. Note that each path value starts and ends with a forward slash character. If the path is blank, this attribute can be omitted.
`classType`	`"Image"` is the only allowed value. .

`enforceAbsoluteURL`	Optional (only necessary for unusual situations). An absolute URL may be needed in unusual situations where the base URL of the HTML page containing the tag differs from typical structure. Set to " `true` " to generate an absolute HTTP URL. For example:
	`<img src='<pega:contentURL name="filename.fileType" path="/filePath/" classType="Image" />' >`
	Will result in:
	`<img src='http://myserver:8080/prweb/datacontent/Image/filePath/fileName.fileType'>`
	If false, the system generates a relative URL. The default if this attribute omitted is "false".

©2025 Pegasystems Inc

> **NOTE:** The contentURL tag generates a URL, not an <img ..> tag. To present an image, include the contentURL within an <img > tag, as shown above.

JavaServer Page tags

- About Image Content data instances

dataGrid JavaServer Page tag

The DataGrid control is deprecated. Update an existing DataGrid to a Grid control. For guidelines on upgrading, see *Upgrading deprecated and outdated controls* on Pega Community.

JavaServer Pages tags

dsField JavaServer Page tag

Use the `dsField` JSP tag within a `dsFields` tag as part of a `dsDataSource` tag. Example:

```
<pega:dsFields><pega:dsField name="pyStreamName" /><pega:dsField name="pyLabel" /></pega:dsFields>
```

The name attribute is required. Identify the name of a property that is to be included when the user selects one value from the drop-down list presented by an `autoComplete` tag.

The standard control *AutoComplete* incorporates an `autoComplete` JSP tag, and allows you to set most attributes of the `autoComplete` and `acDataSource` JSP tags as parameters. Enter an `autoComplete` JSP tag into hand-crafted HTML code only in special situations where the standard control does not meet your needs.

JavaServer Page tags

©2025 Pegasystems Inc

- acDataSource JavaServer Page tag
- dsFields JavaServer Page tag
- Specifying a data source in a control

dsFields JavaServer Page tag

Use the `dsFields` JSP tag within a dsDataSource tag, part of an autoComplete JSP tag configuration. This tag has no attributes. It can contain only dsField tags.

Example:

```
<pega:dsFields><pega:dsField name="pyStreamName" /><pega:dsField name="pyLa
bel" /></pega:dsFields>
```

The standard control *AutoComplete* incorporates an `autoComplete` JSP tag, and allows you to set most attributes of the `autoComplete` and `acDataSource` JSP tags as parameters. Enter an `autoComplete` JSP tag into hand-crafted HTML code only in special situations where the standard control does not meet your needs.

JavaServer Page tags

- acDataSource JavaServer Page tag
- dsField JavaServer Page tag
- Specifying a data source in a control

file JavaServer Page tag

Use the `file` JSP tag within a `static` JSP tag to identify a single text file rule (*Rule-File-Text* rule type) to be included in the current stream.

For example:

```
<pega:static type="script" app="webwb" ><pega:file name="Markov.js" /><pega:bund
le name="Acmestyles" /></pega:static>
```

©2025 Pegasystems Inc

This tag references the text file rule named webwb.Markov.js.

Complete syntax

In the syntax presentation below:

- JSP delimiters <pega:...> and </pega:...> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:file name="myname.type" / >
```

The `name` attribute is required.

Attribute	Value
`name`	Second and third key part of a text file rule to be sent as static content. All rules within one static tag must contain JavaScript, or all must contain CSS styles. All files included within one <pega:static ..> tag must have the same first key part (**Application Name Directory**).

forEach JavaServer Page tag

Use the `forEach` tag to repeat an action for each property in a page, for each page in a `Page List`, or for each property in any list or group.

In runtime stream processing that includes a section or control that includes complex parameters on the **Parameters** tab, you can use the `forEach` tag to iterate over arrays or groups of parameter values.

Basics

Properties with a mode other than `Single Value` are known as aggregates. The forEach tag iterates through the parts of an aggregate property.

©2025 Pegasystems Inc

For example, assume a clipboard page named Operators contains embedded `pxResults()` pages. The `pxResults()` pages each hold a property *pyUserIdentifier* identifying one customer. Use the forEach tag to create an HTML table that contains a row for each customer:

```
<TABLE>
<pega:forEach name="Operators.pxResults">
    <TR><TD>
        <pega:reference name="$THIS.pyUserIdentifier" />
    </TR></TD>
</pega:forEach >
</TABLE>
```

Within the scope of the forEach tag, special keywords are available:

- The keyword $this identifies the current (embedded) page.
- The syntax $this.propertyname represents a property on the current (embedded) page.
- The keyword $this-value represents the (scalar) current active property on the current embedded page.

Complete syntax

In the syntax presentations below:

- Square bracket characters `[` and `]` define optional parts of the tag. Do not type the brackets.
- JSP delimiters `<pega..>` and `</pega...>` mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

To do something for each property in a `Value List`, `Value Group`, `Page`, `Page List`, or `Page Group`, use this syntax. In place of the word aggregate-reference, enter the name of the aggregate property.

©2025 Pegasystems Inc

```
<pega:forEach name="aggregate-reference" >
    actionToRepeat or
</pega:forEach >
```

Attribute	Description
`name`	The name of the aggregate property, `$this`, or `$this`.propertyname. To refer to a complex parameter, use the keyword "param" followed by the complex parameter name.

To do some processing for each property in a top-level page, use this syntax:

```
<pega:forEach name="page-name" >
    actionToRepeat
</pega:forEach >
```

If the array contains only `Single Value` properties, use the keyword `$this` :

```
<pega:forEach name="$this">
    actionToRepeat
</pega:forEach >
```

No conditional "break" or "leave" from the scope of a forEach tag is available: all iterations always occur. To perform only some, rather than all, iterations of a forEach tag, include a <pega:when > element:

```
<pega:forEach name="page-name" >
    <pega:when ...>
        actionToRepeat if true
```

©2025 Pegasystems Inc

```
    </pega:when>
</pega:forEach >
```

Nesting a forEach tag within the scope of another forEach tag is supported.

To iterate over complex parameters that are similar in structure to `Value List`, `Value Group`, `Page List`, and `Page Group` properties defined on the **Parameters** tab of a section or control rule:

```
<pega:forEach name="Param.MyPageList" >
    <pega:reference name="$this.Innervalue"/>
</pega:forEach>
```

Examples

These examples use the forEach tag to display values from clipboard pages.

Arrays within arrays

If the array contains lists of pages, then you can embed a forEach tag and iterate over that list as well:

```
<pega:forEach name=".pxResults">
    <pega:forEach name="$this.addresses">
        <pega:reference name="$this.streetName"/>
    </pega:forEach>
</pega:forEach>
```

Obtain values for each assignment in a flow

To obtain values from an embedded group of pages:

©2025 Pegasystems Inc

```
<pega:forEach name=".pxFlow" >
      value="<pega:reference ref="$this.pxAssignmentKey" /> "
</pega:forEach >
```

This example uses the *Work-.pxFlow* property, a `Page Group`. It identifies, in sequence, the value of the *pxAssignmentKey* property for each page in the `Page Group`.

Using the forEach tag in control rules

To create a selection box, use syntax similar to the following:

```
<select name="<p:r n="$this-name" /> ">
<pega:forEach name=".pyKeyDefList" >
      <option value="<pega:reference ref="$this.pyKeyName" />">
<pega:reference ref="$this.pyKeyCaption" >
</pega:forEach >
</select>
```

To create option (radio) buttons, use syntax similar to the following:

```
<pega:withTarget name="$this" >
      <pega:forEach name=" .pyKeyDefList" >
            <input type="radio" value="<pega:reference
                  ref="$this.pyKeyName" / >">
            <pega:reference ref="$this.pyKeyCaption" / >
      </pega:forEach>
</pega:withTarget>
```

An example of iteration over complex parameter values rather than property values:

```
<pega:forEach name="Param.MyPageList">
      <pega:withEmbedded name="$this" >
```

©2025 Pegasystems Inc

```
            <pega:reference name=".MyInnerValue" />
        </pega:withEmbedded>
</pega:forEach>
```

JavaServer Page tags

include JavaServer Page tag

Use the `include` JSP tag to instruct stream processing to insert one stream within the current stream. This tag lets you define a commonly used segment of HTML or XML text in one rule instance and then incorporate the text where needed.

This tag is useful in source HTML text for user interface forms, in source HTML text for correspondence, and in XML text for interfaces.

Use the include tag to include any of nine rule types:

- Fragments (*Rule-HTML-Fragment* rule type)
- Flow actions (*Rule-Obj-FlowAction* rule type)
- Paragraphs (*Rule-HTML-Paragraph* rule type)
- HTML rules (*Rule-Obj-HTML* rule type)
- Correspondence rules (*Rule-Obj-Corr* rule type)
- Correspondence fragments (*Rule-Corr-Fragment* rule type)
- Control rules (*Rule-HTML-Property* rule type)
- Sections (*Rule-HTML-Section* rule type)
- XML rules (*Rule-Obj-XML* rule type)

For example, this code inserts a fragment of HTML text into the currently processed stream:

```
<pega:include name="NewHeader" type="Rule-HTML-Fragment" />
```

The `name` attribute identifies only the final key part or key parts of the target rule to be included. For rule types that contain an **Applies To** key part, the system uses the

©2025 Pegasystems Inc

Applies To key part of the current rule at runtime and uses rule resolution to find the target rule to include.

<pega:include > JSP tags can also appear in rules of other types derived from the *Rule-Stream* class, such as harness, section, flow action, list view and summary view rules. In most cases, the HTML code and JSP tags for these rules are automatically generated.

Don't use this tag to include a rule within itself (recursion), as that often causes an infinite loop.

Complete syntax

In the following syntax presentation:

- Square bracket characters `[` and `]` define optional parts of the tag. Do not type the brackets.
- Replace any text in italics with your choice of value of that type.

```
<pega:include
    [name="name"]
    [ref="indirect reference "]
    [type="rule type"]
    [inline="true"|"false" ]
/>
```

Either `name` or `ref` is required as an attribute.

Attribute	Description
`name`	The name -- second key part -- of the rule to included. Use this attribute when the name is known during design.
`ref`	Used for indirect references, which allow the rule name to be determined dynamically at runtime. See How to reference properties indirectly in JSP tags.

©2025 Pegasystems Inc

Attribute	Description
`type`	Identify the class of the rule to be included, for example `Rule-Obj-HTML`, `Rule-HTML-Property`, `Rule-HTML-Paragraph`, or `Rule-HTML-Section`. If the type attribute is omitted, then the type that is included depends on the including type, as follows: • A *Rule-Obj-HTML* or *Rule-HTML-Fragment* includes a *Rule-HTML-Fragment*. • A *Rule-HTML-Property* includes a *Rule-HTML-Property*. • A *Rule-Obj-XML* includes a *Rule-Obj-XML*. • A *Rule-Obj-Corr* or a *Rule-Corr-Fragment* includes a *Rule-Corr-Fragment*. You can use any of these keywords: • `standard` — Indicates that the type of the rule to be included is the same as the type of the rule with the <pega:include > tag, with two exceptions: a *Rule-Corr-Fragment* rule includes a *Rule-Obj-Corr* rule, and a *Rule-HTML-Fragment* or *Rule-HTML-Property* rule includes a *Rule-Obj-HTML*. • `property` — Short for *Rule-HTML-Property*. • `fragment` — Short for *Rule-HTML-Fragment*. • `noinput` — Display the included HTML text as read-only. Like `standard`, noinput indicates that the type of the rule to be included is the same as the type of the rule with the <pega:include > tag, with two exceptions: a *Rule-Corr-Fragment* rule includes a *Rule-Obj-Corr* rule, and a *Rule-HTML-Fragment* or *Rule-HTML-Property* rule includes a *Rule-Obj-HTML* . You can't include a *Rule-HTML-Fragment* rule or *Rule-Obj-Corr* rule within an XML Stream rule. You can include a *Rule-HTML-Property* rule only within a *Rule-Obj-Corr* rule.

©2025 Pegasystems Inc

Attribute	Description
`inline`	Set to "false" in unusual situations to disable the default Java generation approach, which inlines Java code for this tag. The "false" setting may reduce the size of the Java class generated for the stream rule, possibly allowing it to fit within the 64KB limit. The default value is "true"; inlining is preferred as it normally provides better performance.
`mode`	Determines whether input is enabled when the included stream is displayed. Set to: • `input` — Properties in the included stream are displayed in input mode where indicated. (This is the default value if the mode attribute is omitted.) • `noinput` — Properties in the included stream are never displayed in input mode (even if their pega:reference tags specify mode="input").

Three special cases — XML, Correspondence, and Correspondence Fragments

Three stream rule types have three key parts. If the stream rule class is *Rule-Obj-Corr* or *Rule-Corr-Fragment*, identify the second and third key parts in the value of the name attribute, separated by a period. The third key part is the **Correspondence Type**.

For example, the standard correspondence type Email can appear:

```
<pega:include name="SuccessCorr.Email" type="Rule-Obj-Corr"/>
```

Similarly, you can include a correspondence fragment for phone text:

©2025 Pegasystems Inc

```
<pega:include name="WorkDetail.PhoneText" type="Rule-Corr-Fragment"/>
```

Likewise, if the stream rule class is *Rule-Obj-XML*, use a period to separate the second and third key parts of the target XML Stream rule. For example:

```
<pega:include name="Contact.MapFrom" type="Rule-Obj-XML" />
```

Including sections that accept parameters

A section may accept parameters, identified on the **Params** tab of the Section form. If the <pega:include > tag is for a section, you can use the <pega:params > JSP tag to supply parameter values.

For example:

```
<pega:include name=".GameResults" type="Rule-HTML-Section" >
    <pega:param name="HomeTeam" value="Boston"/>
    <pega:param name="VisitorTeam" value="Yankees" />
    <pega:param name="HomeTeamRuns" value="3" />
    <pega:param name="VisitorTeamRuns" value="7" />
</pega:include>
```

JavaServer Pages tags

listView JavaServer Pages tag

Use the `listView` JSP tag in a section to include the results of a list view rule in the run-time display of a section. The section can be part of a flow action.

Use the listView tag only in advanced situations where hand-crafted HTML code and advanced options are necessary.

©2025 Pegasystems Inc

Syntax

```
<pega:listView
    attribute= "value" ...
    <pega:param name="value" value="value" />
</pega:listView>
```

The `name` attribute is required. All attributes are lowercase; attribute values are in exact case.

Attribute	Value
`name`	Second key part — **View Purpose** — of a list view rule.
`className`	Optional. **Applies To** key part of a list view rule. If omitted, the class of the primary page in the runtime context is used.
`owner`	Optional. Final key part — **Owner** — of a list view rule. If omitted, the default value is `ALL`.
`action`	Optional. Choose: • `refresh` to execute the list view rule, extracting and formatting data. • `reuse` to bypass report extraction when this same list view rule has run (in this user context with appropriate parameters) recently and the *Code-Pega-List* page containing results. If omitted, `refresh` is used.

©2025 Pegasystems Inc

Attribute	Value
`header`	Optional. Set to `true` or `false` to indicate whether the header is to appear in the output display at runtime. If you omit this attribute, the header appears.
`maxRecords`	Optional. Positive integer of 9,999 or less to indicate the maximum records to display. If supplied, overrides the **Maximum Value** field on the **Content** tab of the list view rule. If omitted, the **Maximum Value** field applies. The system ignores this attribute when the **Enable paging?** box on the **Organize** tab is selected.
`removePages`	Optional. Set this attribute to delete the clipboard pages created when this list view rule executes. Choose: • `all` — Remove the pages that contain the list view rule and the pages that contain the results. • `listview` — Remove the pages that contain the list view rule. • `none` — Retain all pages. If this attribute is omitted, all pages are removed. If the user later attempts to sort and the pages are removed, the report data is re-extracted. If you choose a value other than `all`, design the processing to ensure that stale

©2025 Pegasystems Inc

Attribute	Value
	pages are eventually removed. This reduces the clipboard size and virtual memory demand for the requestor. **CAUTION:** Do not choose `all` if the **Selectable** tab is not blank and the **Copy to** field is set to `Content Page`. This combination removes the page that contains user selection.
`includeStyles`	Optional. Set a value to determine which CSS style sheets are processed to present the display: • `all` — Include all style sheets, both those defined by the skins rule and those defined on the List View form. (This is the default if the `includeStyles` attribute is omitted.) • `listview` — Include only the styles defined on the List View form. • `none` — Include no style sheets; primarily for list view displays embedded in a section or to aid in debugging.

©2025 Pegasystems Inc

Example

The following tag presents up to 25 rows from a list view rule named Delta-Mortgage-Details.Zebra.ALL. It sets two parameter values for the list view.

```
<pega:listView name="Zebra"
      className="Delta-Mortgage-Details"
      owner="ALL"
      header="true"
      action="refresh"
      maxRecords="250"
      removePages="all" >
      <pega:param name="State" value="VA" / >
      <pega:param name="Limit" value="400000.00" />
</pega:listView>
```

A single section, flow action, or HTML rule cannot contain two listView JSP tags that target the same list view report.

When presenting a list view display using this JSP tag in a handcrafted HTML form, place that the listView tag within the FORM element, as in the following:

```
<form action="" method="Post"
      <pega:listView name="zzzz " ... >
      </pega:listView>
</form>
```

In system-generated HTML code, this occurs automatically.

lookup JavaServer Page tag — lookup

Use the `lookup` tag to retrieve and display:

©2025 Pegasystems Inc

- Case 1: A property value of an instance that is stored in the database but is not present on the clipboard. (When the instance is already on the clipboard, use the reference tag.)
- Case 2: A localized version of a property value using field value rules.

The syntax allowed for these two options differs, so they are explained separately below.

Case 1: Displaying a property value not on the clipboard

The lookup tag does not require the object that contains the value to be present on the clipboard.

To display a property value for an object without opening the object, first determine the following:

- The name of the property for which you seek the value
- The class that contains the instance that contains the property
- The property or properties that identify the key parts for that class
- The key parts of the specific instance

Here is an example, explained in detail below.

```
<pega:lookup className="Rule-Obj-HTML" property="pxCreateOperator">
    <pega:key name="pyClassName" value="Work-" />
    <pega:key name="pyStreamName" value="Newbook" />
</pega:lookup>
```

The `pega:key` tags are embedded in the body of the `pega:lookup` tag. The `pega:key` tag specifies the name for each key part of the instance.

To learn which properties form the key of any concrete class, review the **Basics** tab of a **Class** form. If the **KEYS** array is empty, review the **Basics** tab of the parent class rule or the associated class group data instance.

©2025 Pegasystems Inc

Attributes of the lookup tag

The `property` and `className` attributes are required.

Attribute	Value
`property`	Name of the property (second key part)
`className`	Class of the property (**Applies To** key part)
`formatOutput`	Optional. Set `formatOutput="true"` to apply the control rule referenced on the property form when presenting the value. Omit this attribute or set `formatOutput="false"` to present the property value as plain text. This attribute is not available when the lookup tag is used to retrieve a field value rule.
mode	Optional. Values are `normal`, `literal`, or `javascript`, or a JSP expression that evaluates to one of these three values. If no mode is specified, the default is normal **mode=normal** Use `mode=normal` to cause HTML encoding of specific characters when found in the input. The normal mode is the default when the mode attribute is omitted; this mode is useful in HTML-based correspondence and lookups of field value rules, which can

©2025 Pegasystems Inc

Attribute	Value
	contain these characters. Five characters are encoded: **Character** / **Encoded Output** & / & > / >amp; < / *lt; double quote / " single quote / ' mode=literal Use mode="literal" to prevent any HTML encoding. The special characters listed in the table above are passed through unchanged during stream processing. mode=javascript Setting `mode=javascript` is useful when the result of the <pega:reference > is to become part of a JavaScript script. Presents the property value in read-only mode, with certain characters within the property value escaped to a backslash equivalent. No control is applied. Six characters are escaped: **Character** / **Encoded Output** backspace / \b

©2025 Pegasystems Inc

Attribute	Value
	<table><tr><th>Character</th><th>Encoded Output</th></tr><tr><td>tab</td><td>\t</td></tr><tr><td>formfeed</td><td>\f</td></tr><tr><td>double quote</td><td>\"</td></tr><tr><td>single quote</td><td>\'</td></tr><tr><td>backslash</td><td>\\</td></tr></table>
useCache	Optional. For database lookups, the retrieved value is ordinarily retained in a temporary clipboard page, similar to the approach used for linked properties. (Such pages are deleted upon the next Commit operation in the Thread.) This caching improves performance, by eliminating a database fetch operation if the same value is needed a little later. However, it can result in the display of stale data, or the use of stale data in a calculation or decision. You can use either of two approaches to prevent this caching: • Per tag — Add the attribute and value " `useCase=false` " in the lookup tag System-wide — Create a dynamic system setting named `prconfig/cache/EnableForlookup` and set the value to "false". (Equivalently, you

©2025 Pegasystems Inc

Attribute	Value
	can use a `prconfig.xml` setting `cache/EnableForlookup`.)

Attributes of the key tag

Include a key tag that identifies a value for each key part. The name attributes is required. Either the `value` or `ref` attribute is required.

To learn which properties form the key of any concrete class, review the **Basics** tab of a **Class** form. If the **KEYS** array is empty, review the Basics tab of the parent class rule.

Attribute	Value
`name`	Property that forms part of the key.
`value`	Value of that key part.
ref	Expression that computes the value of that key part.

Examples:

- `<pega:key name="KeyProperty" ref=".SomeValue" ↗`
- `<pega:key name="KeyProperty" ref="Param.SomeParameter" ↗`
- `<pega:key name="KeyProperty" ref="$save(Something)" ↗`

Case 1 example: database lookup

To display the **Add Operator** field for an HTML rule instance named Work-.Newbook without opening the instance to the clipboard:

1. Determine the property that corresponds to the **Add Operator** field. The property that holds the Add Operator field is *pxCreateOperator*. To discover this, review any

©2025 Pegasystems Inc

HTML rule, and then click the Rule Data toolbar button on the toolbar. Scan the XML display to locate the property.
2. Determine the rule type of the HTML rule. The rule type is *Rule-Obj-HTML*.
3. Determine the properties that hold the key parts for any HTML rule. You can find the key parts of any class by reviewing the *Rule-Obj-Class* rule instance that defines that class. The key parts are *pyClassName* and *pyStreamName*.
4. Determine the values for the key parts for this specific instance. The value of the first key part is Work-; the value for the second key part is Newbook.

The result:

```
<pega:lookup className="Rule-Obj-HTML" property="pxCreateOperator" >
<pega:key name="pyClassName" value="Work-" />
<pega:key name="pyStreamName" value="Newbook" />
</pega:lookup>
```

The lookup tag can be used within other tags, to supply values used as HTML attribute values and in XML tags.

If the property **Table Type** is not None and the **Use for display (not validation)?** check box is selected, the output is translated according to the table contents. This can provide one approach to localization of the value.

Case 2: Displaying a localized property value

Using information in the **General** tab and field value rules for a property, you can use the lookup tag to convert a `Single Value` property text value from English (or another base language) to a locale-specific value.

To construct the localized values:

1. Select `Field Value` as the **Table Type** on the **General** tab of the property.
2. Enter the **Applies To** key part of the property as the **Class** value.

©2025 Pegasystems Inc

3. Enter the **Property Name** key part of the property as the **Field Name** value.
4. Create localization rulesets for each supported language and a version for each. For example, if the application ruleset is named ALPHA, create a ruleset named ALPHA_fr_FR to hold the French version of selected rules. Create a version 01-01-01 for ALPHA_fr_FR and add it to your access group or application rule.
5. Sign on after you have acquired the ability to add rules to the localization rulesets.
6. Create a single field value rule in each localization ruleset. Enter the language-specific value in the **Localized Label** field.
7. Select the **Use only for display (not validation)?** check box. Save the Property form.
8. In stream rules that display the property value, use the second form of the lookup JSP tag:

Attribute	Value
`property`	Reference to a property, in the context of the current stream rule.
`value`	Optional. English value of the field, if a known constant value.
`formatOutput`	Optional. Set `formatOutput` ="true" to apply the control rule referenced on the Property form when presenting the value. Omit this attribute or set `formatOutput="false"` to present the property value as plain text.
mode	Optional. Values are `normal`, `literal`, or `javascript`, or a JSP expression that evaluates to one of these three values. If no mode is specified, the default is normal **mode=normal** Use `mode=normal` to cause HTML encoding of specific characters when found in the input. The normal mode is the default when the mode attribute is omitted; this mode is useful in HTML-based correspondence and lookups of

©2025 Pegasystems Inc

Attribute	Value
	field value rules, which can contain these characters. Five characters are encoded: **Character** / **Encoded Output** & / & > / >amp; < / *lt; double quote / " single quote / ' mode=literal Use mode="literal" to prevent any HTML encoding. The special characters listed in the table above are passed through unchanged during stream processing. mode=javascript **NOTE:** Setting `mode=javascript` is useful when the result of the <pega:reference > is to become part of a JavaScript script. Presents the property value in read-only mode, with certain characters within the property value escaped to a backslash equivalent. No control is applied. Six characters are escaped: **Character** / **Encoded Output** backspace / \b tab / \t formfeed / \f double quote / \"

©2025 Pegasystems Inc

Attribute	Value
	<table><tr><th>Character</th><th>Encoded Output</th></tr><tr><td>single quote</td><td>\'</td></tr><tr><td>backslash</td><td>\\</td></tr></table>

During stream processing, the system uses the user's locale to access the localization ruleset. The output is the contents of the Localized Label field.

Errors

The text "No data available" appears when:

- The value of the `value` attribute is not valid.
- The `value` attribute is omitted but the property value is null.
- The value attribute is omitted but **Table Type** area of the **General** tab of the property does not support field value configuration.

Case 2 examples: localized values

Example 1

The **General** tab of the standard property *@baseclass.pyCaption* indicates that values are defined by field value rules:

©2025 Pegasystems Inc

Hundreds of standard field value rule define text values for the property *pyCaption*. The field value rule *@baseclass.pyCaption.City* in the Pega-ProCom ruleset contains an English text value "City"; the same value in the language-specific ruleset ALPHA_FR (for French-speaking locales) can have a **Localized Label** of "Ville". The JSP tag is:

```
<pega:lookup property=".pyCaption" value="City" />
```

Example 2

An application contains a property Order.BoxColor in ruleset ALPHA. In the English base application, values of this property are "red", "yellow", "magenta", and so on.

To support Italian-speaking application users, the developers add a localization ruleset named ALPHA_it_IT. To support French-speaking users, another localization ruleset named ALPHA_it_FR is created.

Two sets of field value rules are created, one for each color value in each ruleset. For example, each ruleset can contain rules with these key parts:

```
Order.BoxColor.red
```

©2025 Pegasystems Inc

```
Order.BoxColor.yellow
```

and so on. The **Localized Label** of the first field value rule contains `rosso` for the ALPHA_it_IT ruleset and `rouge` for the ALPHA_fr_FR ruleset.

The JSP tag is:

```
<pega:lookup property=".BoxColor" formatOutput="false" />
```

For users working with a locale setting of it_IT, stream processing produces "rosso" when Order.BoxColor is red. Users with a locale setting of fr_FR see "rouge."

Dynamic system setting for performance and behavior

The optional dynamic system setting `prconfig/useUdfToRetrievePropertyValue/database` can improve performance of the lookup JSP tag when the target property is not an exposed database column in the PegaRULES database.

- If true, database User Defined Functions are called to extract the property value from the BLOB or Storage Stream column. Only the property value is returned from the database.
- If false, the BLOB column is returned from the database and analyzed by the server software to extract the needed property value, a more expensive approach.

> **NOTE:** By default, the value of this dynamic system setting is false. As a best practice, define this setting and set it to "true".

When this setting is true, property values retrieved by the lookup JSP tag are saved temporarily on a special clipboard page, visible in the Linked Properties page grouping of the Clipboard tool. Such pages are deleted upon the next Commit operation by the requestor Thread; this reduces the likelihood of stale values.

PEGA ©2025 Pegasystems Inc

JavaServer Page tags

- Configuring dynamic system settings

onlyonce JavaServer Page tag

Use the `onlyonce` JSP tag to indicate that text within the body of the tag is to be included in the stream processing output only the first time that an onlyonce tag of that name is found.

This tag is primarily useful in complex HTML streams to prevent multiple definitions of JavaScript functions.

When to use

When entering HTML source code directly into the **HTML** tab of a control rule (or similar form), you can minimize the size of the resulting HTML document with the `<pega:onlyonce >` and `<pega:static >` tags.

Example

For example, this tag is named Alpha.

```
<!-- define scripts -->
<pega:onlyonce name="Alpha" >
The work item id is <pega:reference name=".pyID">
</pega:onlyonce >
```

During stream processing, if another onlyonce tag named Alpha appears, the body of the second (and later) tags are skipped over and not processed.

Complete syntax

In the syntax presentations below:

©2025 Pegasystems Inc

- JSP delimiters <pega: and </pega: mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:onlyonce name="name" >
HTML, XML or JavaScript code to process only once
</pega:onlyonce >
```

The `name` attribute is required.

Attribute	Description
`name`	Set the name attribute to an arbitrary non-blank text value. Case is significant. During stream processing, second and later appearances of an `onlyonce` tag with this name cause the body of the tag to be treated as a comment, contributing nothing to the stream results.

JavaServer Page Tags

p:r JavaServer Page tag

Use the `p:r` tag to display the values of properties, to allow users to enter values for properties, and to insert parameter values. (The p:r tag provides identical capabilities as the reference tag, but requires less typing.)

For a full explanation of the reference tag, see JavaServer Page tags — Reference.

Syntax

Use `p:r` rather than `pega:reference` as the tag. As attributes, substitute `n` for name, `f` for `format`, and `m` for `mode`. In the syntax presentations below:

©2025 Pegasystems Inc

- Square bracket characters `[` and `]` define optional parts of the tag. Do not type the brackets.
- JSP delimiters <p:r and /> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

The `n` attribute is required. The `f` and `m` options are optional.

```
<p:r n="[propertyref]" [f="option"] [m="mode"]/>
```

Property name or symbol — n attribute

Enter a property reference or a dynamic reference to a variable in the current stream, such as `$this-value` or `$page-message`. For a list of these keywords, see How to reference properties indirectly.

Format — f attribute

Include the optional `f` attribute to present the property using a control rule other than the one referenced in the property definition. Identify another control rule name within double quotes.

You can use the `f` attribute only when the `mode` attribute is omitted or is set to `display`. Additionally, the `n` attribute must explicitly identify a property, not a parameter or symbolic reference.

Presentation — m attribute

The optional `m` attribute controls additional facets of presentation and processing. If you omit the `m` attribute, the default value is `display`.

Option	Results
block	The same as `normal`, except that line breaks are replaced by the string `<br↗`. If the string appears, it is not altered.

©2025 Pegasystems Inc

Option	Results
	You can't use the format attribute with m="block".
`display`	Instructs the system to use a control rule as read only, detectable as ! `$mode-input`.) Stream processing adds a single space before the value and a single space after the value. In releases before PRPC Version 5.4, the keyword `$mode-display` indicated read-only output. The `$mode-display` keyword is deprecated for new development; use JSP tags and the display option.
`input`	Presents the property in update mode so that a user can enter, or select, a value for the property.
`javascript`	Useful when the result of the <pega:reference > is to become part of a JavaScript script. Presents the property value in read-only mode, with certain characters within the property value escaped to a backslash equivalent. No control is applied. For example, assume the value of property MyProperty is `This is a "demo"`. If a stream contains the fragment:

©2025 Pegasystems Inc

Option	Results

	`var result="<pega:reference name=".MyProperty" mode="javascript" />";`
	
	The output of stream processing is:
	
	`var result = "This is a \"demo\"";`
	
	which is a valid JavaScript statement. Six characters are escaped:
	
	Character / **Output**
	backspace: \b
	tab: \t
	formfeed: \f
	double quote: \"
	single quote: \'
	backslash: \\

`literal`	Causes the value of the property to be masked from HTML processing. Use this if the value may contain angle bracket characters or other HTML elements that are not to be interpreted. When you save a stream rule that includes a reference tag with this mode, you may receive a security warning message:

©2025 Pegasystems Inc

Option	Results
	>>Warning>> Using mode=literal can expose the system to cross-site scripting attacks - use with caution. For maximum security, do not use `mode="literal"` anywhere in the HTML code that formats a property value for a property that is an input value. For example, assume that a <textarea >on a non-autogenerated flow action form allows input of arbitrary text (including angle bracket characters) that is submitted to Pega Platform and returned to the browser in literal mode (not encoded). A malicious user could "inject" harmful JavaScript source code into the browser, a tactic loosely termed "cross-site scripting" or XSS. In all modes other than literal, the angle brackets are converted to HTML entities (<, >), invalidating the JavaScript code. In rare situations, your application may require mode="literal" in stream rules. For example, the JavaScript target of a button, executed with the `onClick=` attribute, may be dynamic. Design and test such cases with extreme care.

©2025 Pegasystems Inc

Option	Results
`text`	Causes the tag received in an input textarea to be converted to a new line character.
`stream`	On output, instructs the system to process the property to display HTML tags without interpreting them. In addition, the system replaces each newline character and with a space. It also performs the conversions listed for the `normal` keyword, so that the true value of the property appears, even if it includes characters that are usually interpreted as HTML elements.
`normal`	On output, converts the <,>, & and * characters to the corresponding HTML entities (`<` `>` , `&` and `*` respectively) so that they appear correctly in the resulting display or output.

m attribute examples

Property INFO has the value "Not italic or bold". The table shows the display that results from various settings for the m attribute, using an underscore to show added space characters.

Tag	User display (read-only)
<p:r n="INFO" />	_Not italic or bold_

©2025 Pegasystems Inc

Tag	User display (read-only)
<p:r n="INFO" m="display"/>	_Not italic or bold_
<p:r n="INFO" m="text"/>	Not italic or bold
<p:r n="INFO" m="stream"/>	Not italic or bold
<p:r n="INFO" m="literal"/>	Not *italic* or bold

Reference properties indirectly in JSP tags

In the reference, when, withReference, and forEach tags, you can reference properties indirectly rather than by name in JavaServer Page tags.

When you reference a property indirectly, you're referring to the current active property. Use any of these to set up a property as the active property.

- The withReference tag
- The forEach tag
- The reference tag, to reference a property associated with a control rule. This is useful and meaningful only when writing source HTML for a control rule.

©2025 Pegasystems Inc

Using symbolic values

After your HTML code has established the active property, use symbolic values (keywords) to reference the property indirectly. Use these symbolic values in the `reference`, `when`, `forEach`, and `with` JSP tags.

Symbolic value	Description
`$this`	Use `$this` in the `withReference`, `when`, and `forEach` tags to identify the current page.
`$this` -value	Use `$this-value` in the `withReference`, `when`, and `forEach` tags to identify the value of a (scalar) current active property.
`param`.parameter-name	Use this keyword to refer to the current value of a parameter on the current parameter page of the activity. Example: <input type=hidden name=<pega:reference name="param.UserID"/> value="VANDJ" >
`$this-name`	Use `$this-name` in an HTML tag in the reference tags, to display the `name` of the active property. For example, if you have a selection box that lets a user choose an account type, and Account_Type is the active property, use the active property as the name of the selection

©2025 Pegasystems Inc

Symbolic value	Description
	box. That way, the user's choice becomes the value for the property named Account_Type: `<SELECT NAME="<p:r n="$this-name" />" > <OPTION VALUE="Checking">Checking Account <OPTION VALUE="Savings">Savings Account</SELECT>` This keyword can't be used within JSP tags other than pega:reference or p:r.
`$this-definition` (myProperty)	Use `$this-definition` in the reference tag to display the value of a property in the Rule-Obj-Property definition of the active property. For example, to display the value of the *pyLabel* (Short Description) for the active property, use: `<pega:reference name="$this-definition(.pyLabel)" />` This symbolic value is frequently used to specify the maximum length or expected size.
`$this-message`	Use `$this-message` in the reference tag to display messages associated with the active property.

©2025 Pegasystems Inc

Symbolic value	Description
`$this-qualifier`	Use `$THIS-QUALIFIER` in the reference tag to display a property qualifier (*Rule-Obj-Property-Qualifier* rule type) associated with the active property.
`$PAGE-NAME`	Use `$PAGE-NAME` in the reference tag to display the name of the base page. The base page is the same as the primary page, if it has not been changed by the with tag. If the with tag has changed the primary page, the base page is the page established by the with tag.
`$PAGE-CLASS`	Use `$PAGE-CLASS` in the reference tag to display the name of the class of the base page. The base page is the same as the primary page if it has not been changed by the with tag. If the with tag has changed the primary page, the base page is the page established by the with tag.
`$PAGE-MESSAGE`	Use `$PAGE-MESSAGE` in the reference tag to display page messages. Page messages are associated with the current base page, not with one property on that page.
`$PAGE-ALLMESSAGES`	Use `$PAGE-ALLMESSAGES` in the reference tag to display all messages associated with the (current) base page.

PEGA ©2025 Pegasystems Inc

Symbolic value	Description
`$PAGE-DEFINITION(propertyname)`	Use this keyword to supply the named property from the Rule-Obj-Class instance that corresponds to the class of the current page.
`$SAVE` (name)	Use `$SAVE(name)` in the reference tag to retrieve and display the scratchpad value associated with name.
`$STREAM-DEFINITION (property)`	Identifies the named property of the current stream definition.
`$STREAM-HANDLE`	Identifies the handle of the stream definition.

Qualifying symbolic values

Use this syntax to qualify the value of the `$THIS` symbol when the current property is an aggregate:

Symbolic Value	Description
`$THIS.property-reference`	If the mode of the property is `Page`, `<pega: reference name= "$THIS.myProperty" />` identifies the value of myProperty within that page.

reference JavaServer Page tag

Use the `reference` tag to display the values of properties or activity parameters and to allow users to enter values for `Single Value` properties or single elements of aggregate properties.

©2025 Pegasystems Inc

The output of the reference tag normally reflects the control rule associated with the property. For example, if the property's control rule contains HTML code for a selection box, the property appears on a form (in read-write mode) as a selection box. In unusual situations, you can override the output format using the `format` attribute of the reference tag.

After you become familiar with the reference tag options, you can save typing by using the equivalent, but briefer, p:r tag. See JavaServer Page tags — p:r tag.

Simple formats

Display the value of a property

To display the value of a `Single Value` property on the primary page:

```
<pega:reference name=".myProperty" />
```

To display the value of a property on another page:

```
<pega:reference name="myPage.myProperty" />
```

If the property has a mode of `Value List` or `Value Group`, use an index (subscript) to identify one element in the list or group. The index may be a constant value or a property reference:

```
<pega:reference name="myPage.myProperty(83)" />
    <pega:reference name="myPage.myProperty(California)" />
    <pega:reference name="myPage.myProperty(.pyState)" />
```

Let a user enter a value

To allow a user to update or enter the value for a `Single Value` property, use the mode attribute and `input` keyword.

©2025 Pegasystems Inc

```
<pega:reference name="myProperty" mode="input" />
```

Insert the value of an activity parameter

Use the `Param` keyword to identify an activity parameter value.

```
<pega:reference name="Param.myParameter" />
```

Set the value of the $this-value keyword

Each use of the reference tag updates the activity property — the value of the `$this-value` keyword.

Complete syntax

In the syntax presentations below:

- Square bracket characters `[` and `]` define optional parts of the tag. Do not type the brackets.
- JSP delimiters <pega: and /> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

The `name` attribute is required. The `format` and `mode` options are optional. The `param` element is used only in special cases, described below.

```
<pega:reference name="[propertyref]" [format="option"]
    [mode="mode"] >
    [ <pega:param name="paramname" value="paramvalue" ref="propref" >
</pega:reference >
```

©2025 Pegasystems Inc

Attribute	Value
`name`	Reference to a property (or in certain cases to a parameter value) in the context of the current stream rule. For a property on a page other than the current page, identify the page name and its class on the **Pages & Classes** tab of the form. You can use an indirect property reference here, such as `$this-value` or `$page-message`. If you use the `$this` keyword, the reference tag must appear within the body of a `forEach`, `withEmbedded`, or `withReference` JSP tag. See How to reference properties indirectly within tags. If the run time context of a control rule, you can reference a parameter using the notation `name="param.zzzzzz"` where zzzzz is a scalar parameter defined on the **Parameters** tab of the Section form or control form. See Parameters.
`format`	Optional. Include the optional `format` attribute to present the property using a control rule other than the one referenced in the property definition. Identify another control rule within double quotes.

©2025 Pegasystems Inc

Attribute	Value
	You can use the `format` attribute only when the `mode` attribute is omitted or is set to `display`. Additionally, the `name` attribute must explicitly identify a property, not a parameter or symbolic reference. If the control rule uses parameters, you can set parameter values using the parameter syntax. See Parameters. `<pega:param name="name" value="value >` For example, assume the property named IsSmarterThanAverage normally is presented as a check box, because the control rule referenced in the property is *Checkbox*. Include format=" `TFDropBox` " to present this property as a drop-box containing True and False as choices.
`mode`	Optional. The optional `mode` attribute controls additional facets of presentation and processing described below.

Even when the mode is `display`, you cannot present an entire `Value List` or `Value Group` property, only a single element of an aggregate property.

Values for the mode attribute

The optional mode attribute can have one of six keyword values. If you omit the `mode` attribute, the `display` value is defaulted.

©2025 Pegasystems Inc

Value	Results
block	The same as `normal`, except that line breaks are replaced by the string `<br↗`. If the string appears, it is not altered. You can't use the format attribute with mode="block".
`display`	Instructs the system to use a control rule as read-only, detectable as ! `$mode-input`.) Stream processing adds a single space before the value and a single space after the value. The `$mode-display` keyword is deprecated for new development; use JSP tags and the display option.
`editable`	Presents the property in update mode so that a user can enter or select a value for the property. Editable overrides the input state of stream rule including the control.
`input`	Presents the property in update mode so that a user can enter, or select, a value for the property.
`javascript`	Useful when the result of the <pega:reference > is to become part of a JavaScript script. Presents the property value in read-only mode, with certain characters within the property value escaped to a backslash equivalent. No control is applied.

©2025 Pegasystems Inc

<table>
<tr><th>Value</th><th>Results</th></tr>
<tr><td></td><td>For example, assume the value of property <code>MyProperty</code> is <code>This is a "demo"</code>. If a stream contains the fragment:

var result="<pega:reference name=".MyProperty" mode="javascript" />";

The output of stream processing is:

var result = "This is a \"demo\"";

which is a valid JavaScript statement. Six characters are escaped:

<table><tr><th>Character</th><th>Encoded Output</th></tr><tr><td>backspace</td><td>\b</td></tr><tr><td>tab</td><td>\t</td></tr><tr><td>formfeed</td><td>\f</td></tr><tr><td>double quote</td><td>\"</td></tr><tr><td>single quote</td><td>\'</td></tr><tr><td>backslash</td><td>\\</td></tr></table></td></tr>
<tr><td><code>literal</code></td><td>Causes the value of the property to be masked from HTML processing. Use this if the value may contain angle bracket characters or other HTML elements that are not to be interpreted.</td></tr>
</table>

©2025 Pegasystems Inc

Value	Results
	When you save a stream rule that includes a reference tag with this mode, you may receive a security warning message: >>Warning>> Using mode=literal can expose the system to cross-site scripting attacks - use with caution. **CAUTION:** For maximum security, do not use `mode="literal"` anywhere in the HTML code that formats a property value for a property that is an input value. For example, assume that a <textarea >on a non-autogenerated flow action form allows input of arbitrary text (including angle bracket characters) that is submitted to Pega Platform and returned to the browser in literal mode (not encoded). A malicious user could "inject" harmful JavaScript source code into the browser, a tactic loosely termed "cross-site scripting" or XSS. Consider `richtext` mode as a more secure alternative.

©2025 Pegasystems Inc

Value	Results
	In all modes other than literal, the angle brackets are converted to HTML entities (<, >), invalidating the JavaScript code. In rare situations, your application may require mode="literal" in stream rules. For example, the JavaScript target of a button, executed with the `onClick=` attribute, may be dynamic. Design and test such cases with extreme care.
`richtext`	Causes filtering of user-supplied HTML such as created by the *RichTextEditor* control. This is a more secure choice than `literal`. The filtering preserves the browser display appearance but removes inappropriate markup that might introduce or allow cross-site scripting.
`text`	Causes the tag received in an input textarea to be converted to a new line character. Additionally, performs the conversions listed for the `normal` keyword.
`stream`	On output, instructs the system to process the property to display HTML tags without interpreting them. In addition, the system replaces each newline character and with a space. It also performs the conversions listed for the `normal` keyword, so that the exact value of the property appears, even if it includes

©2025 Pegasystems Inc

Value	Results
	characters that are usually interpreted as HTML elements.
`normal`	On output, converts the <, >, & and * characters to the corresponding HTML entities (`<` `>` , `&` and `*` respectively) so that they appear correctly in the resulting display or output. The control rule identified in the property is ignored. When you specify `normal` , stream processing does not place any space characters before and after the value (unlike the processing for the `display` keyword).

Mode examples

Property INFO has the value "Not italic or bold". The table shows the display that results from various modes, using an underscore to show added space characters.

Tag	User display (read-only)
`<pega:reference name="INFO" ↗`	_Not italic or bold_
`<pega:reference name="INFO" mode="display"↗`	_Not italic or bold_
`<pega:reference name="INFO" mode="text"↗`	Not italic or bold

©2025 Pegasystems Inc

`<pega:reference name="INFO" mode="stream"↗`	Not italic or bold
`<pega:reference name="INFO" mode="literal"↗`	Not *italic* or bold

Parameters for non-auto-generated control rules

The optional embedded <pega:param... > elements typically appear in system-generated HTML code for harnesses and flow actions. However, they can be handcrafted. These elements supplies parameter values stored on a parameter page for the property. (This page is similar to but distinct from the parameter pages of activities.)

The <pega:param... > element is used only when the reference JSP tag references a property that uses a non-auto-generated control rule that accepts parameters.

Typically, the name and data type of each parameter is recorded on the **Parameters** tab of the rule.

The <pega:param...> element requires the `name` attribute and either the `value` or `ref` attribute (not both).

Attribute	Description
`name`	Name of a parameter that appears on the **Parameters** tab of a control rule. This may be a scalar parameter or a complex parameter. Complex parameters include

©2025 Pegasystems Inc

Attribute	Description
	parentheses or a period character in the name. Use the following syntax to identify complex parameters: • .MyValueList() — for a `Value List` or `Value Group` structure • .MyPageList().Prop1, MyPageList().Prop2 — for parameters Prop1 and Prop2 on the embedded pages of the `Page List` structure MyPageList() As with scalar parameters, these complex parameters are not true properties — no *Rule-Obj-Property* rule exists for them — but they mimic the structure of the property types. Complex parameter definitions are accepted on the **Parameters** tab of the control rule form.
`value`	Optional. Enter a literal constant consistent with the data type of the parameter `String`, `Boolean`, or `Integer` as the source of the value for the parameter. If the `name` attribute identifies a complex parameter, the `value` attribute is required.
`ref`	Optional. Enter a property reference (consistent with the data type of the parameter) as the source of the value. This can use the scratchpad and `$save()`

©2025 Pegasystems Inc

Attribute	Description
	function to retrieve a value saved to the scratchpad earlier with the Save JSP tag. You can't use the ref attribute to set the value of a complex parameter.

For example, the standard control rule named *Decimal* accepts three parameters named `ThousandsSeparator`, `Precision`, and `Locale`. A reference JSP tag can supply constant parameters using this syntax:

```
<pega:reference name=".LoanAmount" mode="display" format="Decimal">
    <pega:param name="ThousandsSeparator" value="-1" />
    <pega:param name="Precision" value="3" />
    <pega:param name="Locale" value="en-us" />
</pega:reference>
```

The following examples illustrate setting the value of complex parameters:

```
<pega:reference name=".GameResults" >
    <pega:param name=".ListElement(4)" value="Boston"/>
    <pega:param name=".GroupElement(NYC)" value="Yankees" />
    <pega:param name=".Embedded(4).Embedded(MA).Size" value="3.14" />
    <pega:param name=".MyList(<append>)" value="Maine" />
</pega:reference>
```

save JavaServer Page tag

Use the `save` tag to set up named values at runtime, while the source HTML (or XML) is processed, onto an internal symbol table known as the scratchpad. This in-memory scratchpad is available throughout the generation of the HTML (or XML) stream.

PEGA ©2025 Pegasystems Inc

Use the `save` JSP tag to add a name-value pair to the scratchpad. Use the `$save( name )` JavaScript function to retrieve the value previously associated with the name. Retrieving the value does not remove it from the scratchpad.

Example

Use the save tag to pass values among multiple HTML definitions that stream processing combines to generate a single output stream.

For example, use the save tag to save the value "Pegasystems Inc." and store it in a scratchpad variable named company.

```
<pega:save name="company" value="Pegasystems Inc." />
```

Later in the stream, you can use the `$save()` function to look up the saved value on the scratchpad and display the saved value.

```
<pega:reference name="$save(company)" />
```

The save tag can save values determined dynamically at runtime. For example:

```
<pega:forEach name=".pxResults">
    <pega:withEmbedded name="this.address" >
        <pega:save name="currentAddress" ref="$this-value" />
    </pega:withEmbedded>
</pega:forEach>
```

Complete syntax

In the syntax presentations below:

- Square bracket characters `[` and `]` define optional parts of the tag. Do not type the brackets.

©2025 Pegasystems Inc

- JSP tag delimiters <pega and `/` > mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:save name="anyname"
    value="value"
    append="text"
    appendRef="aref"
    prepend="text"
    prependRef="aref"
    ref="indirect reference"/>
```

The `name` attribute is required. Include either the `value` or `ref` attribute.

Attribute	Value
`name`	A text string of your choosing, used for later retrieval. Choose a name that is unique for each save operation. (If you save using the name FUM and later save FUM again, the more recent value overwrites the earlier value.)
`value`	Optional. A constant text string representing the value to be saved and associated with the name.
`append`	Optional. Enter literal text to be appended after the value of the `name` attribute.
`appendRef`	Optional. Enter a property reference that at runtime evaluates to text to be appended after the value of the `name` attribute.
`prepend`	Optional. Enter literal text to be prepended before the value of the `name` attribute.

©2025 Pegasystems Inc

Attribute	Value
`prependRef`	Optional. Enter a property reference that at runtime evaluates to text to be prepended before the value of the `name` attribute.
`ref`	Optional. Identifies a source of text for the value, determined at runtime.

URL JavaServer Page tag

The URL JavaServer Page (JSP) tag provides a means to include a Uniform Resource Locator string into the output HTML produced by stream processing.

The URL tag has options related to navigation, target frames, and the primary page. Options promote transaction integrity between the client browser session and the requestor state on the Pega Platform server. You can use the URL tag to avoid synchronization problems that can arise if a user clicks the browser's **Back** button rather than using navigation provided in HTML displays.

Wherever you insert an anchor or a submit button into your HTML, you can use the URL tag. If you use the URL tag with a submit button, work with the `FORM` tag as well.

Here is an example of the URL tag used with an anchor.

```
<A HREF="<pega:url value="pyStream=myStream" >
<option name="long">
</pega:url >" >
<B>My Anchor Text</B></A>
```

> ⚠ **CAUTION:** As a best practice for security reasons, confirm that URL obfuscation is enabled on systems that support applications that use the URL JSP tag. Additionally, if your application dynamically determines the parameters to the JSP tag through JavaScript code (and URL obfuscation is

©2025 Pegasystems Inc

> enabled), use the standard SafeURL JavaScript functions to obfuscate the URL. Click the **Show JS Doc** button to access the SafeURL documentation.

Complete syntax

In the syntax presentations below, replace any text in italics with your choice of value of that type.

```
<pega:url value=action-specification >
    <pega:option name="option1" />
    <pega:option name="option2" />
</pega:url>
```

The `value` attribute is required. The action-specification is either *pyStream* =myStream or *pyActivity* =myActivityClass.myActivityName.

Attribute	Value

`value`	Text to be appended to the end of the system-generated URL as a Common Gateway Interface (CGI) percent-encoded string. In most cases, the URL identifies a stream rule name or an activity, in the format:
	`pyStreamName= myStreamName`
	or
	`pyActivity= myAppliesTo.myActivityName`

©2025 Pegasystems Inc

Attribute	Value
	NOTE: Using the URL JSP tag, you can start only activities with the **May Start?** box selected on the **Security** tab.

When using the URL JSP tag in a FORM tag, you can enter the action specification in either the FORM tag, the Submit button tag, or the anchor tag. If you have more than one action specification, enter it in the button tag or the anchor tag, rather than in the form tag.

```
<pega:url value= "pyStream=stream">
<pega:option name="option" />
</pega:url>
```

Enter the action-specification in the anchor or submit button.

```
<A HREF="<pega:url value=pyStream=myStream">
<pega:option name="primary" />
</pega:url>"
<B>MyAnchorText</B></A>
```

Options

This table lists each option and its corresponding property. Each option is explained in detail below. You can abbreviate the option using its first letter.

Option	Abbreviate as	Adds this parameter
`transid`	`t`	*pzTransactionId*
`frame`	`f`	*pzFromFrame*

©2025 Pegasystems Inc

Option	Abbreviate as	Adds this parameter
`primary`	`p`	*pzPrimaryPageName*
`long`	`l`	Adds all three parameters

transid option — State synchronization

Used with a form and a **Submit** button, the `transid` option of the URL tag can be useful to maintain synchronization between the servers state and the user's browser state. Ideally, such synchronization works even when a user clicks the Back icon in the browser window.

The `transid` option causes the *pzTransactionID* parameter value to be included in the URL.

When the browser state changes as the result of a user submitting an HTML stream, the *pzTransactionID* parameter stores the server state as it corresponds to the current browser stream, tracking where the user has been and where the user is. In some situations, it is important that the browser state and the server state correspond accurately.

In situations where the server state data is not relevant, you can omit the `transid` option.

The `transid` option implies the `frame` option.

frame option — Target frame

Add the `frame` option to include the *pzFromFrame* parameter in the URL. Use this option if the target frame for the anchor is not the `Standard` frame, but is the current default frame. (The second parameter of the Show-HTML method determines the target frame for the HTML.)

The *pzFromFrame* property provides the default value for the *pyTargetFrame* parameter, the target frame used by the stream. If omitted, the system uses the `Standard` frame,

PEGA ©2025 Pegasystems Inc

To add an explicit target frame, see below.

primary option — Called activity's primary page

Add the `primary` option to the URL tag to instruct the system to include the name of the current primary page as the value of the *pzPrimaryPageName* parameter in the URL when the user clicks an anchor or submit button. This may be used only if the current primary page is a top-level named page.

Use this option to tell the system to use the current primary page as the primary page when a user clicks the anchor. If the activity the anchor calls expects a primary page, use the `primary` option, or enter an explicit primary page using the *pyPrimaryPageName* parameter in your URL.

For example, an anchor that calls an activity to delete a page must indicate what page to delete. By contrast, an anchor that calls an activity that sets up its own primary page does not require such information.

long option — Include all options

To specify all three options, use the `long` option:

```
<A HREF="<pega:url "pyStream=myStream" >
      <option name="long" / >
</pega:url >"
<B>MyAnchorText</B></A>
```

Adding parameters and property-value pairs explicitly

You can append property-value pairs by property name to an anchor. To add property-value pairs, use an ampersand (&) character and enter the property value pair. Encode any special characters with the % character followed by two percent-encoded hexadecimal digits. (As a security measure, the value of activity parameters cannot contain certain characters including < and >.)

©2025 Pegasystems Inc

For example, you can cause an anchor to explicitly assign a primary page, using the *pyPrimaryPageName* parameter.

```
<A HREF="<pega:url value="pyActivity=myActivityClass.myActivity&pyPrimaryPageNa
me=myPage" /> ">
<B>MyAnchorText</B></A>
```

For another example, you can add an explicit target frame using the *pyTargetFrame* parameter.

```
<A HREF="<pega:url
      value=MyStream=myStream&pyTargetFrame=myLeftFrame" />
<B>MyAnchorText</B></A>
```

The following query string parameters appear in many URL JSP tags:

Parameter or property	Description of value
pyActivity	Activity Name, second key part of an activity rule.
pzPrimaryPageName	Primary Page Name
pzFromFrame	HTML target frame
pzTransactionID	computed transaction ID, a hash code
pyStream	Second key part of an HTML rule.
Purpose	Second key part of a harness.

Setting property values

In an HTTP POST request, query string parameters with names that start with the two-character prefix $ are instructions to set property values on the clipboard.

©2025 Pegasystems Inc

```
...$PpyWorkPage$ppyLabel=Loan%20Underwriting...
```

This instructs Pega Platform to update the clipboard property *pyWorkPage.pyLabel* to the value "Loan Underwriting".

At runtime, Pega Platform checks each query string are against strict security criteria. To ensure the integrity of the system, clipboard updates are allowed only for properties that appear as an input field within the HTML FORM on the page. For example, a URL tag can includes the following legitimate property value setting:

```
...$PpyWorkPage$ppyWorkParty$gOriginator$ppyWorkPartyUri=smith%40myco.com
..
```

If the property *pyWorkPage.pyWorkParty(Originator).pyWorkPartyUri* appears as an input field on the form, this query string sets the property value to "smith@myco.com". However, if the property value does not appear as an input value within the <FORM..> element, a security alert is logged:

- The property value (if any) on the clipboard is not updated.
- A security alert of type `SECU001` (unexpected property) is added to the Alert log, with details in the Pega log.

The input field may be visible to the user, or in a hidden field within the <FORM tag>:

```
<input type="hidden" name="pyStatusWork" value="Resolved-Completed" />
```

An HTTP POST request string can also include a representation of an entire XML document that creates or updates an entire clipboard page. This feature is primarily used by internal processing facilities. This is not allowed from a web node. If a user on a web node attempts to send an entire XML document with HTTP POST, the clipboard updates do not occur and the system adds an `SECU0002` alert to the Alert log.

PEGA ©2025 Pegasystems Inc

Submit buttons

To use the URL tag with a **Submit** button, type the URL tag in the `ACTION` attribute of the Form tag like this:

```
<FORM action="<pega:url value="...">
      <pega:option name="long" />
</pega:url >
```

Reference the *pyActivity* or *pyStream* in the submit button.

```
<INPUT TYPE="Submit" NAME="pyActivity=Add">
```

When a user clicks the **Submit** button, the URL in the FORM tag is sent.

Always include the `long` option when the URL tag is used with the ACTION attribute of the FORM tag. Unlike stream processing with directives, stream processing with JSP tags does not automatically create hidden fields (<INPUT TYPE=HIDDEN..>) after a FORM tag.

Starting a new thread

Execution of the URL tag can start execution of a new Thread (clipboard user context) for the current requestor. Using the Property-Set method or Java, set the name of a new Thread as the value of the *pxThread.pyThreadUse* property before the URL tag executes. The default thread is named `STANDARD`; you can use any other alphabetic text for the new thread name.

JavaServer Page tags

- Alerts

©2025 Pegasystems Inc

when JavaServer Page tag

Use the `when` tag to conditionalize a segment of HTML or XML, whether used for display, user input or correspondence. Use it to control whether stream processing includes, or omits, parts of HTML text under conditions that you determine. You can base the conditions on Boolean values returned by properties, Java methods, or when condition rules.

Examples

Example A. Using the test attribute, you can check the value of a text property using the Java == operator.

```
<!-- does the property .Color have the exact value RED? -->
<pega:when test=".Color == 'RED' " >
    ... other HTML code here ...
</pega:when >
```

Example B: Assume an HTML rule contains the following HTML segment:

```
<!-- When this work item has resolved properties set -->
<pega:when name="HasBeenResolved" >
    <font class="dataLabelStyle">Resolved by</font>
    <pega:reference name=".pyResolvedUserID" /> 
</pega:when >
```

This when tag references a when condition rule named HasBeenResolved. At run time, if this rule evaluates to true, the output HTML contains the text "Resolved by" followed by the ID of the user who resolved the work item.

Complete syntax

In the syntax presentations below:

©2025 Pegasystems Inc

- Square bracket characters `[` and `]` surround optional parts of the tag. Do not type the brackets.
- JSP delimiters <pega: and </pega: mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:when
    [name="When rule" ]
    [java="<%=expression %>"]
    [test="keyword"]>
HTML segment to include if true
</pega:when >
```

Exactly one attribute — `test`, `name`, or `java` — must appear.

Attribute	Description
`name`	The name (second key part) of a when rule to be evaluated.
`java`	A Java expression that evaluates to true or false. You can use the Java operators any standard Java operators within the expression. For example: `<pega:when java="<%= !tools.getProperty("OperatorID.pyPwdVerify").getMessages().equals("") %>"></pega:when>`
`test`	Other test conditions, typically using the keywords `$mode-display`, `$mode-input`, and `$input-enabled`. You can use parentheses and Java operators (but not Java inline code) in the test condition.

©2025 Pegasystems Inc

Attribute	Description

	You can't use the <% and %> delimiters to insert inline Java code into a test condition. Additionally, complex conditions involving multiple || or && operators are not supported.
	
	For example, you can test the contents of the scratchpad:
	
	`<pega:when test='$save(mysymbol)!="foo" '>`
	`<!-- yes -->`
	`</pega:when>`

Referencing a when condition rule with the name attribute

Use the `name` attribute to identify a when condition rule. At run time, the system determines the **Applies To** key part of the when rule from the **Applies To** key part of the current rule, and uses rule resolution to locate the when rule.

When you save the current rule, the system confirms that the when condition rule you referenced is found.

The $mode-input and $input-enabled keywords

Two boolean keywords useful in control rules are available if an active property reference appears at the current position in the source HTML. These Booleans let you conditionalize stream processing based on the stream processing context.

- The `$MODE-INPUT` keyword is true when all of the following are true:
  - The current stream is an input-enabled stream.

©2025 Pegasystems Inc

    - The keyword is part of the processing of a <pega:reference > tag that specifies the INPUT option, with or without a stream name value.
    - The current property identified in the <pega:reference > tag is modifiable.
- The `$INPUT-ENABLED` keyword indicates that this HTML stream accepts input. This keyword is normally true, unless it falls within the scope of a <pega:include > tag that uses the `NOINPUT mode.`

For example:

```
<pega:when test="!$mode-input">this area is read-only </pega:when>
<pega:when test="$mode-input"> this area may allow input </pega:when>
<pega:when test="$input-enabled">this area may allow input</pega:when>
```

The `$mode-display` keyword is deprecated for new development; use `!$mode-input` instead.

Using when tags with choose

To implement more sophisticated testing and conditional operation in a stream rule, use a when tag within a choose tag. The otherwise tag, used only within a choose tag body, implements a none-of-the-above choice. See JavaServer Page Tags — choose.

Using a When rule

Use the name attribute to identify a when condition rule. At run time, system determines the **Applies To** key part of the when rule from the **Applies To** key part of the current rule, and uses rule resolution to locate the when rule.

When you save the current rule, the system confirms that the when condition rule you referenced is found.

The When condition rule is evaluated at run time on its own primary page, which may not be the primary page of the HTML or other stream rule. Review the **Pages & Classes**

©2025 Pegasystems Inc

tab of the when condition to find its primary page. You can temporarily change the base page of the HTML rule during stream processing using the withPage tag.

Using PublicAPI methods

Use the `test` attribute to condition the output HTML on the results of these PublicAPI methods:

To determine whether a property value is presented as an input field, use the *isModifiable* PublicAPI method, which returns `True` if the property is modifiable. The underlying Java code determines whether a specific property is modifiable or not.

```
<pega:when test="$THIS:isModifiable" >
    <input type="text">
</pega:when>
```

To determine whether a property is scalar, use the *isScalar* PublicAPI method. Only scalar property references can be displayed or input in HTML.

```
<pega:when test="$THIS:isScalar" >
    <input type="text">
</pega:when>
```

To determine whether the current active property is a special property, use the *isSpecial* PublicAPI method:

```
<pega:when test="$THIS:isSpecial" >
    <input type="text">
</pega:when>
```

©2025 Pegasystems Inc

Evaluating a Java expression

Use the `java` attribute within the when JSP tag to evaluate a boolean Java statement. Enclose the java statement within the normal JSP delimiters <% and %>

For example:

```
<pega:when java=
"<%= tools.getParamValue("one").
      equalsIgnoreCase(tools.getParamValue("two")) %>">
      <H1>Update <p:r n=".pyID" /> <p:r n=".pyLabel" /> </H1>
</pega:when>
```

Working with more than one condition

If your JSP tag contains more than one expression, you can combine them using standard Java logical operators. Examples:

&& is the AND operator

```
<pega:when test="Monday && Morning" >
      do something
</pega:when>
```

|| is the inclusive OR operator

```
<pega:when test="VIP || NewCustomer" >
      do something
</pega:when>
```

! is the NOT operator

©2025 Pegasystems Inc

```
<pega:when test= ".customerID !=18" >
      do something
</pega:when>
```

Use only one of these operators in the Java statement. Expressions involving multiple operators or parentheses are not supported in the `test` attribute.

JavaServer Page Tags

withEmbedded JavaServer Page tag

Use the `withEmbedded` JSP tag to set the current embedded page during stream processing to a different embedded page. The new page applies only to HTML or XML code within the body of the withEmbedded tag.

Here is an example:

```
<pega:withEmbedded name="$this(7)" >                    Enter your HTML and/or JSP ta
gs here.        </pega:withEmbedded >
```

Complete syntax

In the syntax presentations below:

- JSP delimiters <pega:... > and </pega:...> mark the start and end of the tag.
- Replace text in italics with your choice of value of that type.

```
<pega:withEmbedded name="embeddedpagestring" > Enter <pega:reference name=
".Surname" /> here.                  </pega:withEmbedded >
```

The `name` attribute is required.

©2025 Pegasystems Inc

Attribute	Value
`name`	An embedded clipboard page name, which may include the keyword `$this`. Examples: • `$this` • `$this.myPage` • `$this(3)` • `$this(string that represents the group ID)`

About JavaServer Page tags

withPage JavaServer Page tag

Use the `withPage` tag to replace the primary page with a different page (the base page) throughout a segment of the HTML or XML text, as a coding convenience. The new page applies only to HTML or XML code within the body of the `withPage` tag.

Example

For example, the reference tag refers to property pyWorkPage. *pyStatusWork* :

```
<pega:withPage name="pyWorkPage">
    and your application status is:
    <pega:reference name=".pyStatusWork" />
</pega:withPage>
```

Complete syntax

In the syntax presentation below:

- JSP delimiters <pega:...> and </pega:...> mark the start and end of the tag.

©2025 Pegasystems Inc

- Replace any text in italics with your choice of value of that type.

```
<pega:withPage name="pagename">
<!-- HTML and/or tags here with pagename as the base page -->
</pega:withPage>
```

The `name` attribute is required. The page you identify must also display in the **Pages & Classes** tab of the current rule.

Attribute	Value
`name`	A clipboard page name, which may include the keyword `$this`. Examples: • Reference to a property of mode `Page` • `$this` — for the current page • `$this`.pagename — for an embedded page in the current page

About JavaServer Page tags

withReference JavaServer Page tag

Use the `withReference` tag to identify a new property as the active property for a limited scope. The new property applies only to HTML or XML code within the body of the `withReference` tag.

Here is an example:

```
<pega:withReference name=".pyStatusWork" > <!-- XML or HTML source here --></pega:withReference>
```

©2025 Pegasystems Inc

Within the scope of this tag, you can use the keyword `$this-value` to identify the active property set by this tag.

Complete syntax

In the syntax presentations below:

- JSP delimiters <pega:...> and </pega:...> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:withReference name="myReference" > Enter your HTML text and JSP tags here . </pega:withReference>
```

The `name` reference is required.

Attribute	Value
`name`	A reference to the new property that is to become the current active property. For example: • A single scalar destination property • `$this` — for the current page • `$this.propertyname`

About JavaServer Page tags

withTarget JavaServer Page tag

Use the `withTarget` tag to override the control rule that ordinarily applies to a property.

For example:

©2025 Pegasystems Inc

```
<pega:withTarget name="CheckBox" > <pega:reference name="IsSoupYet" / ></pega:
withTarget>
```

The `withTarget` tag is useful for defining radio buttons and selection boxes on forms. This tag lets the system examine the value of for the current selection. The target reference must be a single scalar destination.

Syntax

In the syntax presentation below:

- JSP delimiters <pega:...> and </pega:...> mark the start and end of the tag.
- Replace any text in italics with your choice of value of that type.

```
<pega:withTarget name="HTMLPropertyRuleName"> <!-- property to be presented --
></pega:withTarget >
```

The `name` attribute is required.

Attribute	Value
`name`	Name of a control rule. The rule must exist when the current rule is saved. Two special keywords are also supported: • `$this` — Use the control rule that applies to the current active property • `$none` — No output created

About JavaServer Page tags

©2025 Pegasystems Inc

Fields that support the Global Resource Settings syntax

Learn about the fields that support the Global Resource Settings (GRS) syntax for references to external systems and the default properties that are associated with those fields.Global Resource Settings facilitate migrating your application from one environment to another (for example, from Development to Production) because you can configure values for such settings as endpoint URLs, or JNDI names that vary depending on the environment, without the need to manually update data instances or integration rules.

TIP: Consider the following best practices when configuring GRS fields:

- The fields that support GRS also support application settings. To store and organize the environment-specific settings in your application, create application settings and reference them in your fields. For more information, see Creating an application setting and Categorizing application settings.
- Use a data page rule to hold global resource settings. These allow service rules and connector rules to be identical in multiple Pega Platform instances, such as a test system and a production system, while still using different ports, URLs, and servers.

Connector rules

Rule type	Field label
Rule-Connect-SOAP	• Service endpoint URL • Response timeout • Authentication profile

©2025 Pegasystems Inc

Rule type	Field label
Rule-Connect-JMS	• JNDI server name • Connection factory • Request destination name • Response destination name • Response timeout • Authentication profile
Rule-Connect-MQ	• MQ server name • Request queue manager • Request queue name • Response queue/model name • Response queue manager • Response message timeout
Rule-Connect-EJB	JNDI server name
Rule-Connect-HTTP	• Endpoint URL • Response timeout • Authentication profile
Rule-Connect-dotNet	• Service endpoint URL • Response timeout • Authentication profile
Rule-Connect-SAP	• Authentication profile • Response timeout
Rule-Connect-CMIS	Timeout

©2025 Pegasystems Inc

Rule type	Field label
Rule-Connect-REST	Response timeout

Service rules

Rule type	Field label
Rule-Service-File	Frequency

Mapping rules

Rule type	Field label
Rule-Obj-XML	Schema URL
Rule-Parse-XML	Schema URL

Data classes

Rule type	Field label
Data-Admin-Connect-EmailListener	• Requestor login user ID • Requestor login password
Data-EmailAccount	• Sender email address • Sender user ID • Sender password • Sender SMTP host • Sender port • Message signing keystore instance • Message signing alias • Receiver user ID • Receiver password • Receiver host • Receiver port

©2025 Pegasystems Inc

Rule type	Field label
	• Truststore • Display name • Reply to
Data-Admin-Connect-EmailServer	• Host Name • Port Number • User ID • Password • Send Email As • Administrator Email
Data-Admin-Connect-FileListener	• Source location • Requestor login user ID • Requestor login password
Data-Admin-Connect-JMSListener	• JNDI server name • Connection factory name • Request destination name • Response destination name • Requestor login user ID • Requestor login password
Data-Admin-Connect-JMSMDBListener	• JNDI Server • Topic or Queue JNDI Name (**MDB Descriptor Properties** section) • Connection Factory JNDI Name

©2025 Pegasystems Inc

Rule type	Field label
	• Topic or Queue JNDI Name (**REPLYTO Connection** section)
Data-Admin-Connect-JNDIServer	• Provider URL • Security principal • Security credentials
Data-Admin-Connect-MQListener	• Transaction backout queue • Request queue name • Request queue manager • Response queue name (**Reply To** section) • Response queue manager (**Reply To** section) • Requestor login user ID • Requestor login password
Data-Admin-Connect-MQServer	• Host name • Port number • Channel • User ID • Password • SSL/TLS truststore • SSL/TLS keystore • SSL/TLS secret key reset count • SSL/TLS peer name

©2025 Pegasystems Inc

Rule type	Field label
Data-Admin-Security-OAuth2-Client	• Single sign-on (SSO) identity provider (IdP) federation ID • Redirect URI endpoint
Data-Admin-Security-WSSecurityProfile	Out Flow > Username > Username configuration • User name • Password
Data-Admin-Connect-FTPServer	• Port number • Host name

SAML authentication service

Type	Field label
Identity provider information	• Entity Identification (Issuer) • Login location • Logout location • Artifact Resolution Service (ARS) location • TLS/SSL truststore
Service provider settings	• Entity Identification • Assertion Consumer Service (ACS) location • Redirect logout location • SOAP logout location

©2025 Pegasystems Inc

Type	Field label

	• Artifact Resolution Service (ARS) location
	• Signing certificate
	• Decryption certificate
	• Signing certificate:
	  ◦ *Keystore name*
	  ◦ *Sigining alias*
	  ◦ *Signing password*
	• Decryption certificate
	  ◦ *Keystore name*
	  ◦ *Sigining alias*
	  ◦ *Signing password*

Authentication profile

Authentication profile	Field label
Basic	• User name • Set password • Host name • Realm
OAUTH 2.0	• Redirect URI endpoint **NOTE:** When you use this GRS syntax is used, set the data page access group and the Oauth2 service package to the same value, and thes set data page to be read-only.

©2025 Pegasystems Inc

Authentication profile	Field label
	• Single sign-on (SSO) identity provider (IdP) federation ID
NTLM	• User name • Set password • Host name • Domain

- Configuring dynamic references to external systems by using the Global Resource Settings feature
- Integration-Connectors category

Integration-Connectors category

The rule types in the Integration Connectors category support connector interfaces between Pega Platform and external systems.

> **NOTE:** The rule type Rule-Connect-BPEL is deprecated. Use Service SOAP rules instead.

Wizards

This book contains reference topics for wizards in Pega Platform.

- **About Edit Input rules**
- **Connector and Metadata wizard – Completing the Choose Purpose form**
- **Service Wizard: Select Service Purpose**
- **Displaying all Functions in the Expression Builder**

PEGA ©2025 Pegasystems Inc

- About the eForm Rules accelerator
- Running other upgrade utilities

About Edit Input rules

An edit input rule provides a conversion facility. Use edit input rules to convert data entered by a user (or received from an external system) from a format that your application doesn't use into another format.

Edit input rules perform conversions, not validations. This rule type does not cause any response to a user about the validity of the input. Use validate rules and edit validate rules for validations.

Edit input rules referenced by a property execute automatically only during HTTP requests, not for properties set through data mapping, Property-Set methods, or other computations.

Edit input rules use Java code for the conversion.

Where referenced

You can reference an edit input rule in the Edit Input Value field on the Property form.

Access

Use the Records Explorer to list all edit input rules available to you.

Category

Edit input rules are instances of the *Rule-Edit-Input* class. They are part of the Data Model category.

- Edit Input rules - Completing the Create, Save As, or Specialization form

©2025 Pegasystems Inc

Edit Input rules - Completing the Create, Save As, or Specialization form

Records can be created in various ways. You can add a new record to your application or copy an existing one. You can specialize existing Rules by creating a copy in a specific Ruleset, against a different class or (in some cases) with a set of circumstance definitions. You can copy data instances but they do not support specialization because they are not versioned.

Create an edit input rule by selecting `Edit Input` from the `Data Model` category.

Key parts

An edit input rule has a single key part, the name:

Field	Description
Identifier	Enter a name that's a valid Java identifier. Begin with a letter and use only letters, numbers, and hyphens. See How to enter a Java identifier.

Rule resolution

When searching for edit input rules, the system:

- Filters candidate rules based on a requestor's RuleSet list of RuleSets and versions
- Finds circumstance-qualified rules that override base rules
- Finds time-qualified rules that override base rules

The class hierarchy is not relevant to rule resolution of edit input rules.

Connector and Metadata wizard – Completing the Choose Purpose form

Complete the fields in this form and then click **Next**.

©2025 Pegasystems Inc

Field	Description
Purpose	Select either `Generate Connector Rules` or `Generate Data Mapping Rules`.
Metadata Type	Select the kind of metadata the wizard is to import and examine. The options in the selection list vary depending on what you specified in the Purpose field. Generate Connector Rules • EJB Interface • Java class • DOT NET • SQL See About generating connector rules. Generate Data Mapping Rules • XML Schema Definition (XSD) • JavaBean
Property Page Mode	This field is displayed when you select Generate Data Mapping Rules in the Purpose field and Java Bean in the Metadata Type field. Select whether you want to generate Java wrapper classes. • **Java** – Select this option when you want to generate Java wrapper classes.

©2025 Pegasystems Inc

Field	Description
	• **Standard value and page** – Select this option when you do not want to generate Java wrapper classes.
RuleSet Name	Select an existing RuleSet to contain the generated rules.
RuleSet Version	Select the version of the RuleSet for the generated rules.
Base Class	Select the abstract class rule that you identified or created for the wizard to use as the container or base for the generated rules. The wizard creates one or more subclasses of this class rule and adds properties and other rules as appropriate.
Connector Activity Class	This field appears when generating connector rules only. Select the class rule to become the Applies To key part of the connector activity. By default, this field displays the name of your current work pool, a subclass of the *Work-* base class.

- **About the Connector and Metadata wizard**

About the Connector and Metadata wizard

The Connector and Metadata wizard imports information about an external application or system and generates rules and data objects that your Pega Platform™ applications can use to communicate with that external system. You use this wizard to generate connector rules or data mapping rules. The wizard uses a flow to guide you through the

©2025 Pegasystems Inc

process of creating a service. Because it uses a flow, the data you enter about the rules is stored as a work item.

Generate Connector Rules

Select this option when you want to generate the rules and data objects necessary for your application to send messages to or make requests of external systems and then process the results or response. The Connector and Metadata wizard can import Java class metadata through introspection, and connect to and then determine the tables and column names in an external database. It then converts the resulting information into the class, property, activity, and connector rules necessary to build a connector.

The wizard can generate rules for only those connector types that communicate with external systems through protocols or technologies that can expose information or metadata about the construction of that external application. You can use the wizard to generate connectors for EJB, Java, .NET, SAP, SOAP, and SQL.

Before you begin, see About generating connector rules.

See also the Pega Community document *Creating Connectors with the Connector and Metadata Wizard* for step-by-step instructions about using this wizard.

Generate Data Mapping Rules

Select this option when you want to perform one of the following tasks:

- Generate Parse XML, XML stream, class, data transform, property, and other rules that implement in Pega Platform the Data Model of an external system based on the schema described in an XSD file. Before you begin, see About importing XSD files.
- Generate classes, data transforms, and Java properties that reflect the Data Model described by a JavaBean. In this case, the wizard also generates Java wrapper classes that access the get and set methods from the source Java class through a `Clipboard` API interface.

©2025 Pegasystems Inc

Starting the wizard

To start the wizard, in Dev Studio, click **Configure** > **Integration** > **Connectors** > Connector and Metadata wizard.

Updating classes and properties after XSD changes

If you have produced a SOAP connector by using the Connector and Metadata wizard, and later obtain updated XSD files, you do not need to re-run the wizard. You can generate the needed class and property changes from the XML Stream rule form and XML Parse rule form:

1. Using Save As, copy the current XML Stream rule or XML Parse rule into an unlocked RuleSet version.
2. Identify the XSD file on the XML tab.
3. Click the **Update** button on the Mapping tab.
4. Save the rule form.

- **About generating connector rules**

About generating connector rules

When you use the Connector and Metadata wizard to generate connector rules, the wizard generates a specific set of items for connectors of each type, as appropriate for that protocol or technology.

For external SQL databases, the wizard creates an external class and properties, rather than a connector rule.

Results

The wizard generates the following items:

- Classes and properties to support the objects or data described in the WSDL, Java class, or EJB.
- One connector rule for each method or operation selected.

©2025 Pegasystems Inc

- One connector activity for each connector. For SQL, the generated activities interact directly with the external database through the Obj- methods and do not need to invoke a connector rule to connect to the external table.

For .NET, SOAP and SAP, the wizard also generates draft request and response data transforms that are called from the connector activity. Use the generated Request Data Transform to define the mapping of application (source) properties to connector request (target) properties. Use the generated Response Data Transform to define the mapping of connector response (source) properties to application (target) properties.

For SQL connections, the wizard generates the following:

- One database table data object (*Data-Admin-DB-Table*) and one class rule that together represent the external table. The Pega Platform class rule is known as an external class because the data it represents is not stored in the PegaRULES database — it is stored in the external table represented by the class rule.
- A property for each column you specify. The property is mapped to the column it represents on the Advanced tab of the Class form of the class that the property applies to.
- Activities that use the Obj- methods to retrieve, update, or delete records through the external class, which means directly from the external table. Because these activities can interact directly with the external table represented by the external class rule, there's no need for a connector rule. These activities are known by their purpose: browse (list), open, save, and delete.

For more information about which rules are generated, see the Pega Community document *Creating Connectors with the Connector and Metadata Wizard*.

Starting the wizard

To start the wizard, select **Configure** > **Integration** > **Connectors** > Connector and Metadata wizard.

©2025 Pegasystems Inc

Resuming the wizard

This wizard creates a work item with prefix `pxW-`. To find open wizard work items, open the wizard as described above, and select the desired work item from the list.

Prerequisites that apply to all connector types

Before you use the Connector and Metadata wizard to generate connector rules, follow these steps:

1. Identify the RuleSet and version to contain the generated rules — classes, properties, connector rules, activities, and so on.
2. Identify or create the following class rules to use for the generated rules.
   - An abstract class rule — typically one that inherits from the *Data-* base class — for the wizard to use as the container or base for the generated items with the exception of the connector activities. (If you do not create this class before you begin, the wizard can create one for you.)
   - A concrete class rule for the connector activities. If you plan to call the connector from a flow, choose a class that inherits from the *Work-* base class, so that the connector activities can be called directly from an Integrator shape.
3. Review the class rules you created or identified. Confirm that they allow subclassing and allow rules to be added to the RuleSet you identified

Prerequisites for specific connector types

Additional setup tasks vary according to the type of connector you want to create. Following are checklists for each connector type.

For EJB connectors (Rule-Connect-EJB):

1. Create a JNDI server data instance (*Data-Admin-Connect-JNDIServer* class) that identifies the EJB container of the bean you want to connect to. If this JNDI server requires authentication, obtain a user name and password.
2. Obtain copies of these Java classes:

PEGA ©2025 Pegasystems Inc

- The appropriate EJB interface classes. If your EJB connector is designed to communicate with the EJB through Remote Method Invocation (RMI), you need the remote and remote home interface classes. If your EJB connect is to communicate through the local interface, you need the local and local home interface classes.
- All encapsulated Java classes (Java classes that are referred to as a property by the interface classes).
- When you are using the remote interface and the EJB is deployed in WebSphere, the container-generated stub classes.

3. Package the Java classes into a .jar file.
4. Add the .jar file on the Pega Platform class paths. For instructions, see the Pega Community article *About the PRPC class paths.*

For Java connectors (Rule-Connect-Java):

1. Obtain a copy of the Java class file.
2. Add the Java class to the Pega Platform class paths. For instructions, For help with this step, see Pega Community article *About the Process Commander class paths*.

For SQL:

1. In the external database, create or identify a database user account for Pega Platform access.
2. Make the appropriate JDBC driver available to the application server running Pega Platform. Then create a JDBC data source for the database in the application server.
3. Create a database name instance (*Data-Admin-DB-Name*) that identifies the external database

NOTE:

©2025 Pegasystems Inc

> If errors occur during processing, the wizard may create a connect rule with availability set to `Draft/No`. In this case, review and save the generated rules to isolate the source of the error; change the availability to `Yes` before testing.
>
> If your WSDL file or XSD changes after you've generated a connector, you can regenerate only the changed classes, properties, and so on. For an example, see the Pega Community article *How to manage changes to WSDL and XSD metadata in SOAP connectors using RuleSet Versioning*.

Service Wizard: Select Service Purpose

In the first step of the wizard, complete each field, then click **Next**.

Field	Description
Service Purpose	Select a purpose for the service that you want to create: • Create and manage work • Invoke existing activity rules • Process input or output data
Service Type	Select the type of service that you want to create: • **SOAP** For more information, see Connect-SOAP method • **.Net** For more information, see Connect-dotNet method (deprecated) • **HTTP** For more information, see About Connect HTTP rules (deprecated) • **EJB** For more information, see About EJB connector rules (deprecated)

©2025 Pegasystems Inc

Field	Description
	• Java For more information, see Service Java rules (to be deprecated) • JSR94 For more information, see Service JSR94 rules (to be deprecated) • JMS For more information, see Connect-JMS method • MQ For more information, see Service MQ rules • File For more information, see Connect-File method (deprecated)
Use MDB Listener	This field displays if your Service Type is `JMS`. Select this check box if the Pega Platform is deployed as an enterprise application and the service is to use a message-driven bean (MDB) as the listener.

Displaying all Functions in the Expression Builder

Build advanced Expressions that automate calculations in your application by displaying internal Functions and libraries from your entire system in the Expression Builder. By default, the Expression Builder displays only non-internal libraries and Functions. You can display all Functions in the system by overriding the *pyShowInternalLibrary* When Rule setting.

Internal Functions are not displayed by default, because they are subject to change without notice. Use internal Functions with caution. A Function is internal if the *pyMethodStatus* property of the library that stores the Function has the value of `Internal`. To check the *pyMethodStatus* property, view the XML or the clipboard page

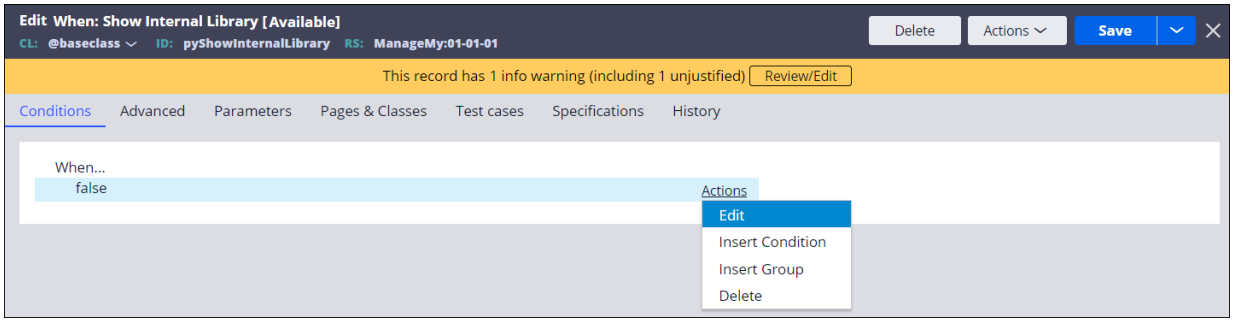
©2025 Pegasystems Inc

of the Rule. For more information, see Viewing the XML of a Rule and Using the Clipboard tool.

1. In the navigation pane of Dev Studio, click **Records**.
2. Expand the **Decision** category, and then click **When**.
3. In the list of instances of When Rules, click the **pyShowInternalLibrary** Rule.

> **TIP:** To find the Rule faster, you can filter the Rules by name.

4. In the Rule form header, click **Save as**.
5. On the **Save As When: Show Internal Library** form, in the **Context** section, in the **Add to Ruleset** list, select your application Ruleset and Ruleset version.
6. Click **Create and open**.
7. On the **Conditions** tab, in the **When...** section, click **false**.
8. Click **Actions** > **Edit**, as shown in the following figure:



Editing the When Rule

9. In the **Condition** dialog box, enter `true`, and then click **Submit**.
10. Click **Save**.

> **Result:**
>
> When you author Expressions in the Expression Builder, you can browse through all Functions and libraries in the system.

©2025 Pegasystems Inc

Related concepts

- When Condition Rules

About the eForm Rules accelerator

Description

The Pega Platform provides two rule types to support applications that use forms in the Portable Document Format (PDF):

- eForm File rules (*Rule-File-eForm* rule type)
- Map eForm rules (*Rule-Map-eForm* rule type)

The eForm accelerator uses a flow to guide you through the process of creating an eForm file rule and its corresponding eForm map rule.

The accelerator imports and parses a PDF form that you identify. Then it lists the names of the interactive form fields in the PDF form, making it easier for you to match the fields to Pega Platform properties. If you create the rules yourself, enter the name of each form field in an eForm map rule and verify that you made no data entry errors.

Because the wizard uses a flow rule, the data you enter is stored as a work item until you finish the flow. When you resolve the work item, the eForm accelerator creates an eForm file rule for the uploaded PDF form and an eForm map rule that specifies how the fields in a form like that one map to Pega Platform properties.

For information about using the PDF feature to process incoming PDF forms or to generate a PDF form, see the Pega Community document *Working with PDF Forms and Documents*.

Prerequisites

Before you begin, complete the following:

©2025 Pegasystems Inc

- Copy the PDF form (that has one or more interactive field definitions) to your workstation.
- Print a copy of the PDF form.
- Determine the names of all the interactive fields and write each field name code next to the text that represents the field directly on the printed copy of the form. Reference this as you run the accelerator.
- Determine the Applies To class for the new rules. Typically, this is a class that inherits from the *Work-* base class.
- Determine the names and classes of the Pega Platform properties that map to the form fields. The eForm accelerator enables you to select properties from the Applies To class only. If you need to map a form field to a property that does not belong to the Applies To class, edit the eForm map rule after the accelerator saves it.
- Determine the RuleSet and version for the new rules.

Starting the accelerator

To start the eForm Rules accelerator, select the Dev Studio > *Integration* > *Tools* > *PDF for Eform wizard*.

- **eForm accelerator Help: Completing the Select eForm Properties form**
- **eForm accelerator Help: Completing the Upload File form**
- **eForm accelerator Help: Completing the Select eForm Fields form**
- **eForm accelerator Help: Completing the Map eForm Fields form**
- **eForm accelerator Help: Completing the Review and Save form**

eForm accelerator Help: Completing the Select eForm Properties form

Begin defining the eForm rules by completing the fields in the Select eForm Properties form. Then click **Next**.

©2025 Pegasystems Inc

Field	Description
Applies To	Select the class that you want the generated rules to apply to. The properties that you map form fields to must belong to this class.
RuleSet Name	Select the RuleSet to hold the generated eForm rules.
RuleSet Version	Select a version of the RuleSet.

eForm accelerator Help: Completing the Upload File form

Click **Browse** and select the PDF form you want to create eForm rules for. Then click **Next**.

eForm accelerator Help: Completing the Select eForm Fields form

This form lists field codes for all the interactive form fields in the uploaded PDF form.

Select the fields that you want to map. Then click **Next**.

eForm accelerator Help: Completing the Map eForm Fields form

In each of the selection fields, select the name of the appropriate property name. The selection fields display the properties from the Applies To class you specified in the Select eForm Properties form.

Click **Next**.

If you need to map a form field to a property from another Applies To class, select any property to use as a placeholder. Then, after the accelerator creates the map eForm rule:

©2025 Pegasystems Inc

1. Open the rule.
2. Create an entry for the appropriate class on the Pages & Classes tab.
3. And update the mapping as needed.

eForm accelerator Help: Completing the Review and Save form

Expand the sections.

Examine the values set in the fields, and make any changes as necessary.

When you finish, click **Next**.

Running other upgrade utilities

To identify and modify elements of the application that need to be changed to be compatible with the upgraded release, run the utilities on the **Upgrade Tools** page. The specific utilities depend on the version from which you upgraded. Run all recommended utilities.

1. In the header of Dev Studio, click **Configure** > **System** > **Release** > **Upgrade**.
2. Follow the on-screen instructions to run the tools.

©2025 Pegasystems Inc