# Decision Management

Reference Guide

© 2015 by Pegasystems Inc. All rights reserved

1

## Technical Overview

Decision Management Enabled Applications

Dependencies

Application

Organization

Work Pool

Access Group & Operators

Packaging

Rulesets

Rule Types

Predictive Models

Scorecards

Adaptive Models

Decision Data

Implementation

Validation

Strategies

Interactions

Data Sets

Data Flows

Event Strategies

Agents

UpdateAdaptiveModels

ADMSnapshot

ProcessBatchJob Agent

About Large Simulations

Configuring Large Scale Simulations

Methods & Functions

Predictive Model

Scorecard

Adaptive Model

Obtain Predictor Information

Upload Interaction History

Decision Data

Strategy

Return List of Propositions

Return List of Properties

Compute Segment Logic

5

5

6

6

8

8

8

9

9

9

10

10

10

10

11

11

12

13

13

13

14

14

14

14

15

15

15

15

16

16

16

17

17

17
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limitations</td>
<td>50</td>
</tr>
<tr>
<td>Decision Manager</td>
<td>51</td>
</tr>
<tr>
<td>User Activity Boundaries</td>
<td>51</td>
</tr>
<tr>
<td>Business Sandbox Cycle</td>
<td>52</td>
</tr>
<tr>
<td>Revision Activation Cycle</td>
<td>54</td>
</tr>
<tr>
<td>Production Cycle</td>
<td>54</td>
</tr>
<tr>
<td>Strategies</td>
<td>56</td>
</tr>
<tr>
<td>Strategy Design Patterns</td>
<td>56</td>
</tr>
<tr>
<td>Next Best Action</td>
<td>57</td>
</tr>
<tr>
<td>Capture Results</td>
<td>57</td>
</tr>
<tr>
<td>Segmentation</td>
<td>58</td>
</tr>
<tr>
<td>Reusable Logic</td>
<td>59</td>
</tr>
<tr>
<td>Adaptive Components</td>
<td>59</td>
</tr>
<tr>
<td>Trend Detection</td>
<td>60</td>
</tr>
<tr>
<td>Strategy Design</td>
<td>62</td>
</tr>
<tr>
<td>Sub Strategies</td>
<td>62</td>
</tr>
<tr>
<td>Data Import</td>
<td>63</td>
</tr>
<tr>
<td>Segmentation</td>
<td>63</td>
</tr>
<tr>
<td>Data Enrichment</td>
<td>64</td>
</tr>
<tr>
<td>Aggregation</td>
<td>64</td>
</tr>
<tr>
<td>Arbitration</td>
<td>65</td>
</tr>
<tr>
<td>Decision Path Selection</td>
<td>65</td>
</tr>
<tr>
<td>Pages &amp; Alternative Pages</td>
<td>66</td>
</tr>
<tr>
<td>Expressions</td>
<td>66</td>
</tr>
<tr>
<td>Understanding the Expression Context</td>
<td>66</td>
</tr>
<tr>
<td>Using Component Properties in Expressions</td>
<td>66</td>
</tr>
<tr>
<td>External Input</td>
<td>67</td>
</tr>
<tr>
<td>Strategy Results</td>
<td>67</td>
</tr>
<tr>
<td>Strategy Properties</td>
<td>68</td>
</tr>
<tr>
<td>Strategy Execution</td>
<td>68</td>
</tr>
<tr>
<td>Interactions</td>
<td>69</td>
</tr>
<tr>
<td>Decision Execution</td>
<td>69</td>
</tr>
<tr>
<td>Multilevel Decisioning</td>
<td>70</td>
</tr>
<tr>
<td>Simulations</td>
<td>72</td>
</tr>
<tr>
<td>Simulation Status</td>
<td>72</td>
</tr>
<tr>
<td>Dynamic System Settings</td>
<td>73</td>
</tr>
<tr>
<td>Glossary</td>
<td>75</td>
</tr>
</tbody>
</table>
Decision Management functionality provides sophisticated mechanisms to create applications that determine which processes to run, and which products to offer to customers through the Next Best Action principle. This principle increases customer loyalty by addressing multiple issues in the decision making process. Decision Management functionality includes:

- Proposition management
- Data flow, strategy and interaction design
- Driving process flows using interaction, scorecard and predictive model rules
- Using third party models
- Event processing
- Multilevel decisioning
- Single and multinode simulation execution
- Business visualization and monitoring using Visual Business Director (VBD)
- Forecasting using VBD and simulations
- Advanced adaptive analytics using Adaptive Decision Manager (ADM)
- Predictive analytics using Predictive Analytics Director (PAD)
- Reporting
- Enabling business users to directly manage decisions within the boundaries defined by IT
- Revision management and activation cycle

Decision Management functionality is delivered through the combination of Decision Management rules sets and the Decision Management service layer for adaptive decisioning and business monitoring.

### Related Information

- [Reference Application](#) (page 5)
- [About this Release](#) (page 6)
- [What's New](#) (page 6)

### Reference Application

Process Commander includes DMSample, a Decision Management sample application that illustrates typical use cases of Decision Management functionality without the external Visual Business Director and Adaptive Decision Manager services. DMSample is designed to show the principles of using predictive analytics, building strategies, configuring interactions and wiring processes to use the power of Decision Management functionality. Although not using the external Adaptive Decision Manager service, DMSample demonstrates adaptive learning by using the corresponding embedded process.

You have two ways to access this application: add the DMSample:Administrators access group to your operator record, or login as administrator@pega.com and switch to DMSample. In the DMSample application, you can also launch the Predictive Analytics Director portal to access the project used to generate the churn scoring model used in DMSample strategy design. In a new Pega 7.1.8, this project is part of the default Predictive Analytics Director examples. In an environment that has been upgraded from a release previous to Pega 7.1.8, you have to perform the following steps to add this project to the default Predictive Analytics Director examples:

1. In Designer Studio, search for the DMSampleScoringModel binary file.
2. Use the download file button to save this archive onto disk as DMSampleScoringModel.zip.
3. Launch the Predictive Analytics Director portal.
4. In the Predictive Analytics Director portal's navigation, go to Manage Resources.
5. In the row containing Examples, use the import action to import the ZIP file you saved in the second step.
About this Release

The main themes of Decision Management in Pega 7.1.8 are Big Data, Complex Event Processing (CEP) and propositions under revision management.

The table below lists the contents of Decision Management in Additional_Products/DSM (full PRPC distribution).

<table>
<thead>
<tr>
<th>Folder</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Products</td>
<td>• Resources required to set up the Decision Management services.</td>
</tr>
<tr>
<td></td>
<td>• Adaptive Decision Manager</td>
</tr>
<tr>
<td></td>
<td>• Visual Business Director</td>
</tr>
<tr>
<td></td>
<td>• Scripts required to set up the Interaction History tables in a</td>
</tr>
<tr>
<td></td>
<td>dedicated schema.</td>
</tr>
<tr>
<td>Documentation</td>
<td>User documentation.</td>
</tr>
<tr>
<td>ThirdParty</td>
<td>Third party software (licenses and source libraries).</td>
</tr>
</tbody>
</table>

The table below lists the contents of Decision Management in Additional_Products/DSM (7.1.8 maintenance level update distribution).

<table>
<thead>
<tr>
<th>Folder</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Products/Adaptive Decision Manager</td>
<td>Resources required to set up the Adaptive Decision Manager server.</td>
</tr>
<tr>
<td>Products/Visual Business Director</td>
<td>Resources required to set up the Visual Business Director server.</td>
</tr>
<tr>
<td>Products/Interaction History</td>
<td>Scripts required to set up the Interaction History tables in a dedicated schema.</td>
</tr>
</tbody>
</table>

What's New

The topics listed below provide the overview of changes in Decision Management features and functionality since Pega 7.1.7:

- Propositions can now be placed under revision management.
  - The new proposition management implementation through decision data records makes propositions part of rule data, hence allowing for change management to be applied to propositions, something that was not possible with the old implementation where propositions were data instances of the group class.
  - A migration wizard provides the mechanism to convert propositions from data instances to decision data. You can decide to migrate all propositions, or take a phased approach and migrate groups at different points in time.
- Decision data implementation has been improved.
  - Previous known as decision parameters, decision data rules use auto-generated forms that simplify the process of defining which properties to set.
  - Custom forms allow you to extend the decision data instances beyond the capabilities offered by auto-generated forms.
- Identity matching API allows you to associate data from anonymous interactions with data from interactions with an identified customer, improving interaction data quality, and providing the means to have a single view of customer interactions. Interaction History supports identity matching by allowing you to:
  - Add associations using the `Call Data-Decision-IH-Association.pySaveAssociation` method.
  - Remove associations using the `Call Data-Decision-IH-Association.pyDeleteAssociation` method.
  - In strategies, expressions become more meaningful with the capability of using the class label instead of the `Primary` keyword.
- Real-time event processing
  - A new rule type, event strategy, allows you to process large volumes of data to identify important events and respond to them as quickly as possible.
  - Data comes in high velocity streams and processing is done in real-time.
• Event strategies can trigger actions depending on event conditions occurring in the data streams.
• Event strategies are used in data flows.
• Data flows
  • Data flows extended with the capability of writing the output to activities and cases by using the activity and case destination types.
  • Event strategy integration in data flows.
• Data sets support more types of data stores. Besides defining data in database tables, decision data stores and Visual Business Director data sources, data sets can now define data in streams, HDFS and HBase stores.
• The connection between PRPC and the Decision Management services can be encrypted by using secure connections (HTTPS).
Technical Overview

- Decision Management enabled applications (page 8)
- Rulesets (page 10)
- Rule types (page 10)
- Agents (page 14)
- Methods and functions (page 15)
- Financial functions (page 21)
- Application overlays (page 23)
- Revisions and change sets (page 25)
- Revision management (page 27)
- Decision Manager roles and privileges (page 28)
- Proposition cache synchronization (page 30)

Decision Management Enabled Applications

The easiest way to create the necessary rules with a standard configuration is through the New Application action in the application menu, and build the new application on PegaDM. This topic describes the technical details and additional guidelines that apply to the configuration and packaging of Decision Management enabled applications.

- Dependencies (page 8)
- Packaging (page 9)

Dependencies

- Application (page 8)
- Organization (page 8)
- Work pool (page 9)
- Access groups and operators (page 9)

Application

The dependency at the application level defines the Decision Management runtime capability of an application.

- Name: PegaDM
- Version: 07.10

The dependency at the ruleset level defines the Decision Management design capability of an application. If you are using the application validation mode in the ruleset containing the Decision Management rules, you only need to define the built on application dependency. However, if you use the ruleset validation mode, the ruleset requires the correct dependency:

- Required rulesets: Pega-DecisionArchitect
- Ruleset version: the highest 07-10 version available in the system

If building on PegaDM is not possible, add the highest available 07-10 version of Pega-DecisionArchitect, Pega-DecisionManager, Pega-DecisionEngine, PredictiveAnalytics-IntSvcs and PredictiveAnalytics-UI to the application's rulesets.

Organization

By default, the organizational structure is required in Decision Management enabled applications. The organization record provides the dedicated class that, unless specified differently in the
The decisioning class field value, becomes the default class containing the application's proposition hierarchy, and provides the application dimension information. Make sure the application's organization hierarchy is fully defined (organization, division, and unit) and available to all operators using or working with your Decision Management enabled application.

If working with multiple applications that need to access the same propositions, make sure the same top level class is configured in the Proposition Management landing page, and that applications have access to the same ruleset containing the classes supporting the proposition hierarchy.

**Work Pool**

The work class is typically necessary for work items. From the Decision Management perspective, it is necessary for simulations. In the class inheritance settings section of the concrete work class, make sure you are directed inheritance is set to the appropriate work class (for example, Work-Cover).

**Access Group & Operators**

If you created the application using default options, the `<Application_Name>:Administrators` access group does not require any changes. For all other access groups:

- Check the minimum required roles in the access group (PegaRULES:SysAdm4 or `<Application>:Administrator`, PegaRULES:SysArch4, PegaRULES:ProArch4, PegaRULES:WorkMgr4 or PegaRULES:User4) and the necessary portal layouts (for example, Developer, WorkManager, and Manager). PegaRULES:SysAdm4 or `<Application>:Administrator` should be present when configuring the proposition hierarchy top level class.
- Check that the local customization points to the application's ruleset and ruleset version are configured.

**Packaging**

To import your application into another node, you can use the application packaging wizard or manually create a product rule when exporting an application. The following steps describe the minimum required steps to manually define a product rule that includes the resources for a typical application.

1. Include the application(s).
2. Include the ruleset(s).
3. Include the classes that support the proposition hierarchy (by default, `<OrgClass>-<ApplicationName>-SR`).

Starting with Pega 7.1.8, propositions are no longer data instances of the group class. If your application was created using a previous release, and you have not migrated the proposition data to propositions managed by decision data instances, make sure you enable the Include Descendants option.

4. Include the necessary individual by using the SmartPrompt and the Query button to insert every instance:
   - Access group (Data-Admin-Operator-AccessGroup)
   - Operator ID (Data-Admin-Operator-ID)
   - Work pool (Data-Admin-DB-ClassGroup)
   - Organization (Data-Admin-Organization)
   - Division (Data-Admin-OrgDivision)
   - Unit (Data-Admin-OrgUnit)
   - Work group (Data-Admin-WorkGroup)
   - Workbasket (Data-Admin-WorkBasket)

5. If applicable, include your application's KPI definitions as individual instances:
   - Use the SmartPrompt to select the Data-Decision-Outcome-KPIDefinition class.
   - Press the Query button to view the instances.
• Include the instances corresponding to the KPIs used in your application.

## Rulesets

Decision Management functionality is delivered by the *Pega-DecisionEngine*, *Pega-DecisionArchitect*, *Pega-DecisionManager*, *PredictiveAnalytics-IntSvcs* and *PredictiveAnalytics-UI* rulesets.

<table>
<thead>
<tr>
<th>RuleSet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Pega-DecisionEngine</em></td>
<td>Provides the execution data model and runtime implementations supporting Decision Management rule types and landing pages.</td>
</tr>
<tr>
<td><em>Pega-DecisionArchitect</em></td>
<td>Provides the user interface, data model and forms.</td>
</tr>
<tr>
<td><em>Pega-DecisionManager</em></td>
<td>Provides the Decision Manager portal implementation, and supporting administration functionality (revision management, privileges and default roles).</td>
</tr>
<tr>
<td><em>PredictiveAnalytics-IntSvcs</em></td>
<td>Provides the Predictive Analytics Director runtime implementation.</td>
</tr>
<tr>
<td><em>PredictiveAnalytics-UI</em></td>
<td>Provides the Predictive Analytics Director portal implementation.</td>
</tr>
</tbody>
</table>

## Rule Types

Applications that depend on the functionality implemented by the Decision Management rulesets have access to the following rule types:

- Predictive models  (page 10)
- Scorecards  (page 10)
- Adaptive models  (page 11)
- Decision data  (page 11)
- Strategies  (page 13)
- Interactions  (page 13)
- Data sets  (page 13)
- Data flows  (page 14)
- Event strategies  (page 14)

## Predictive Models

Predictive models predict behavior for one or more segments (classes) based on customer data. Predictive models instances use a PAD model or a third party model in PMML (page 77) format (3.0, 3.1, 3.2, 4.0 or 4.1). Predictive models are used in strategies through predictive model components. In flows, predictive models are referenced through the decision shape by selecting the predictive model type. PAD models are constructed to generate the largest possible number of classes (segments) that exhibit predicted behavior, steadily increasing as the class number increases. However, business strategies translate to the two or three alternatives typically associated with the probability of predicted behavior (high, medium and low). Remapping the classification defined in the predictive model to the smaller number of business strategies allows you to increase the quality of business. For example, if a lower propensity (page 77) class is reassigned to the medium propensity class where fewer customers are presented with a product offer but a greater proportion responds, although the volume of business decreases, the quality increases. Starting with Pega 7.1.7, the Predictive Analytics Director portal allows you to generate a predictive model instance that already contains the OXL, or generate the OXL than you can then upload in predictive models instances.

## Scorecards

Scorecards create segmentation based on one or more conditions, and a combining method. The score based segmentation can be mapped to results by defining cutoff values used to map a given score range to a result. For example, your application can use a scorecard to calculate customer segmentation based on age and income, and then map particular score ranges to defined results. The output of a scorecard is a score and segments defined by the results. Scorecards are used in strategies through scorecard components. In flows, scorecards are referenced through the decision shape by selecting the scorecard model type.
Predictors can be defined by single value properties (for example, `.Salary`) or by expressions (for example, `@if(.EmploymentStatus == "Employed", .Salary, .CreditAmount)`). By default, every predictor is assigned the same weight (1). Changing the default value results in calculating the final score as weight multiplied by score (for example, `0.5*30`). Maintaining the default value implies that, effectively, only score is considered because the coefficient is 1 (for example, `1*30`).

Scores can also be defined by single value properties (for example, `.Score`), a computation expressed by an expression (for example, `.Score*.PenaltyMargin or @divide(.Score,100)`), as well as literal values (for example, 20).

Adaptive Models

Adaptive models configure the adaptive scoring models (page 75) in the ADM system. The output of an adaptive model is a partial list of adaptive statistics (evidence, propensity and performance).

Decision Data

Decision data records offer a flexible mechanism for the type of input values that require frequent changes. Checking in changes to decision makes the changes available to all users but, typically, the changes to decision data instances are made available when system architects activate the revision that contains the changes, or when revision managers activate a direct deployment revision.

Decision data records can provide a simple list of values (typically, this is the case with global control parameters), or a set of values that are available in a specific context (for example, proposition parameters and channel centric parameters). Decision data records are used in strategies through the decision data component. Additional to its direct use through strategies as a means of supplying flexible inputs, decision data records also provide the underlying implementation to define and manage propositions. This new feature (propositions part of rule data) is introduced in Pega 7.1.8, and allows for proposition data to be placed under versioning in the same way as any other rule data.

The values of decision data records are typically defined by business users through the Decision Manager portal, but this functionality is not tied to the facilities in the portal and can be used in Designer Studio as well. The content of decision data records is driven by custom forms that define which properties can be set through the record, or custom forms that allow system architects to provide a specific user interface.

- Implementation (page 11)
- Validation (page 12)

Implementation

Technically speaking, the `pyEditElements` and `pyEditElement` sections in the decision data definition class drive the content of the decision data instance. Independent of the decision data instance’s applies to class, the decision data definition class contains the circumstanced `pyEditElement` section overridden through each decision data rule. Every decision data instance is initially set to use generated forms whose content is defined by the set of properties listed in the decision data instance’s `Form` tab. Additionally, decision data instances can also use custom forms, in which case you need to override the `pyEditElement` sections and, if necessary, the `pyEditElements` section. The following list provides the technical overview of the decision data extension points.

- The starting point is an initial decision data instance that is configured with the appropriate decision data definition class.
- Using generated forms, all you have to do is to define the set of properties decision data instances can set.
- You can press the use custom form button to override the `pyEditElement` section in the right decision data definition class. The natural place for the specialized sections is the class that contains the data model required to support the decision data instance.
- The `pyEditElement` section defines the parameters for each element (for example, marketing weight, start date and active/inactive proposition), and the controls that allow business users to change parameters.
- If you are setting properties that can have different values depending on the context (for example, channel centric parameters), the customization ends at this point. However, you can also can set properties that function as a simple list applicable at the application level. In this case, you need to
override the `pyEditElements` section @baseclass and save it under the same class as `pyEditElement`.

This is the section that defines the items themselves (for example, channel and description), the standard add and delete actions, operating on the basis of the `pyEditElement` flow action to register new property values. It is not always the case that this section requires controls to add and delete new elements.

- **There are two ways to customize `pyEditElements`:**
  - Using the grid repeat layout, the data source used in the grid repeat layout of the `pyEditElements` section is the `pxResults` property. In `Pages & Classes`, you always need to define the `pxResults` page.
    - Page Name: `.pxResults`
    - Class: the class containing the sections
  - You can discard the grid repeat layout, and use a section include instead. In `Pages & Classes`, define the `pxResults` page by index, and then configure the section include to select the section by name using the corresponding clipboard page.
    - Page name: `.pxResults(1)`
    - Class: the class containing the sections

**Validation**

Validation of the parameters set by decision data instances can be put in place using standard edit rules.

The following example shows how validation is put in place for a control parameter that allows users to change the start date of a proposition:

- **The Start Date SR property is used as a control parameters that applies to phones and tablets propositions. It is set up as such by being included in the properties listed in the corresponding decision data Form tab.**
- **To define the validation, you add an edit validate rule that defines the criteria for saving new parameter values. The following snipped shows the Java source defined in the edit validate rule named `IsValidStartDate`:**

```java
if (theValue.trim().length() == 0) {
    return false;
}
java.util.Date theDate =
tools.getDateTimeUtils().parseDateTimeString(theValue);
if(theDate == null){
    theProperty.addMessage("Please enter a valid date");
    return false;
}
java.util.Calendar now = java.util.Calendar.getInstance();
now.add(java.util.Calendar.DAY_OF_YEAR, 7);
if(theDate.after(now.getTime())){
    theProperty.addMessage("The date cannot be more than 7 days in the future. Please enter a valid date");
    return false;
}
now.add(java.util.Calendar.DAY_OF_YEAR, -14);
if(theDate.before(now.getTime())){
    theProperty.addMessage("The date cannot be more than 7 days in the past. Please enter a valid date");
    return false;
}
theProperty.addMessage("Enter a valid date. Start date cannot be in the past and no later than one week in future");
return true;
```

- **To introduce the validation, you configure the Start Date property to use the validate rule by referencing the `isValidStartDate` rule in the Advanced tab.**
Strategies define the decision that is delivered to an application. The decision is personalized and managed by the strategy to reflect the interest, risk and eligibility of an individual customer in the context of the current business priorities and objectives. The result of a strategy is a page (clipboard or virtual list) containing at least the results of the components that make up its output definition.

Interactions

Interaction rules orchestrate the process of issuing a decision and capturing interaction results. The way you configure the interaction to run the strategy determines the operation of this rule.

Data Sets

Data sets define collections of records, allowing you to set up instances that make use of data abstraction to represent information stored in different sources and formats. Depending on the type selected when creating a new instance, data sets represent VBD data sources, data in database tables, data in decision data stores, streams, data in Apache HBase data stores or data in Hadoop clusters. Through the data management operations for each data set type, you can read, insert and remove records. Data sets are used on their own through data management operations, and also in the form of combined data streams in data flows. VBD data sources can also be used in interaction rules when writing results to VBD.

Besides data sets you define in your application, there are two default data sets provided by the Pega-DecisionEngine ruleset under the Data-pxStrategyResult class:
Data Set Name | Description
-------------|---------------------------------------------------
pxInteractionHistory | This data set represents Interaction History results. It is used to write the captured response information to the Interaction History data store through activities or data flows.
pxAdaptiveAnalytics | This data set represents adaptive inputs. It is used to update the adaptive data store through activities or data flows.

Data Flows

Data flows allow you to sequence and combine data based on various sources, and write the results to a destination. The sequence itself is established through a set of instructions and execution points from source to destination. Source and destination points can be abstract or driven by data sets and other decision data flows. Between source and destination, you can apply compose, convert, merge, strategy and event strategy execution instructions. The execution of data flows is done through work items which are instances of Pega-DM-DDF-Work, and stored in the pc_work_dsm_batch table. The processing of data flows can be divided in two categories: batch data flows using a database table as main input, and real-time data flows that, in active state, continue processing incoming stream data and requests made through the available service interface.

Event Strategies

Event strategies provide the mechanism to simplify the complex event processing operations. They allow you to specify patterns of events, query for them across a data stream and react to the emerging patterns. The sequencing in event strategies is established through a set of instructions and execution points from real-time data to the final emit instruction. Between real-time data and emit, you can apply filter, window, aggregate and static data instructions. Event strategies are used in data flows.

Agents

Agents in the Pega-DecisionEngine ruleset:
- UpdateAdaptiveModels (page 14)
- ADMSnapshot (page 14)
- ProcessBatchJob (page 14)

UpdateAdaptiveModels

PRPC keeps a local cache of scoring models. The model update frequency is implemented by periodically triggering the UpdateAdaptiveModels agent (Pega-DecisionEngine ruleset, PegaDM:Administrators access group). The agent runs the pxUpdateModels activity to retrieve model updates. By default, the agent is scheduled to run every 30 seconds, retrieving the scoring models required for executing the strategy and the models that are different from those in the local cache.

ADMSnapshot

Adaptive Decision Manager can capture historical data for reporting purposes. The ADM data mart is implemented by periodically triggering the ADMSnapshot agent (Pega-DecisionEngine ruleset, PegaDM:Administrators access group). The agent runs the pzGetAllModelDetails activity. This activity captures the state of models, predictors and predictor binning in the ADM system at a particular point in time, and writes that information to a table using the Data-Decision-ADM-ModelSnapshot and Data-Decision-ADM-PredictiveBinningSnapshot classes. By default, the time interval for running the agent is 120 seconds. The admmart/batchSize dynamic system setting in the Pega-ProcessCommander ruleset determines the batch size which, by default, is 50, a value considered suitable even in cases of large size models.

ProcessBatchJob Agent

Large scale simulations are enabled by performing simulations across system nodes. This functionality requires extra configuration.
About Large Simulations

The assignment, queuing and management of large scale simulations is governed by the ProcessBatchJob agent configuration. The agent is scheduled to run with a given regularity (in seconds) to trigger checking assignments in the DSMBatchJobs@pega.com workbasket. If there are assignments, they will be queued to create threads based on the thread configuration for each node. The status of the work item is updated as it progresses in this process and you can monitor the assignment by viewing the instances in the workbasket.

Configuring Large Scale Simulations

So that you can enable large scale simulations, configure the ProcessBatchJob agent in your ruleset.

1. If the agents rule instance is not present in your application's ruleset, create one.
2. In the Schedule tab, add the ProcessBatchJob agent with the following settings:
   - Agent Name: ProcessBatchJob
   - Pattern: Periodic
   - Interval: for example, 5
   - Enabled: check this option
   - Class: Data-Decision-StrategyExecution-BatchAgent
   - Activity Name: pzInitializeProcessEngine
3. In the Security tab, enable the Bypass activity authentication option and configure the application specific access group.
4. Use SMA to make sure the agent is running.
5. Configure the number of threads that can run in each node in the batch topology landing page.

Methods & Functions

The sections below provide the overview of Decision Management methods and functions categorized by area. Additionally, Decision Management functionality delivers the financial functions library that can be used in expressions.

- Predictive model (page 15)
- Scorecard (page 15)
- Adaptive model (page 15)
- Decision data (page 16)
- Strategy (page 17)
- Interaction (page 17)
- Data set (page 18)
- Interaction History (page 20)
- Simulation (page 20)

Predictive Model

In expressions, you can obtain the segments calculated by the predictive model by using the Lib(Pega-DecisionEngine:PredictiveModel).ObtainValue(this, myStepPage, "predictivemodelrulename") syntax.

Scorecard

In expressions, you can obtain the segments calculated by the scorecard by using the Lib(Pega-DecisionEngine:Scorecard).ObtainValue(this, myStepPage, "scorecardrulename") syntax.

Adaptive Model

The methods listed below support the use of adaptive models in activities.

- Obtain predictor information (page 16)
• Train models using report definitions (page 16)

Obtain Predictor Information

You can obtain the predictor information of an adaptive model through the Call DSMPublicAPI-ADM.pxLoadPredictorInfo method with the following parameters:

• Option to include active, or active and inactive predictors.
  • Set it to true if you want to retrieve only active predictors.
  • Set it to false if you want to retrieve active and inactive predictors.
• Result page to store predictor information.
• Adaptive model key: the page (Embed-Decision-AdaptiveModel-Key) containing the adaptive model parameters. The Embed-Decision-AdaptiveModel-Key class is used to uniquely identify an adaptive model. The properties of data type text in this class provide the action dimension (pyIssue, pyGroup, and pyName), channel dimension (pyDirection, and pyChannel), the Applies To class of the adaptive model (pyConfigurationAppliesTo) and the name of the adaptive model (pyConfigurationName).

Upload Interaction History

You can train models by using the adaptive models management landing page to upload existing customer data representing previous behavior or sample data. The use of previous results allows for Adaptive Decision Manager to create models that are able to predict behavior. ADM only considers positive and negative cases that correspond to the possible outcomes taken into account by the settings defined in the adaptive model instance. Typically, this action is not performed in production, but only in systems that connect to the offline analytics data store. You can also train models using a report definition on Interaction History fact records through the Call DSMPublicAPI-ADM.pxUploadResponsesFromReport (recommended) or the Call Rule-Decision-AdaptiveModel.pyUploadResponsesFromReport methods with the following parameters:

• pyReportName: the name of the report definition rule.
• pyReportClass: the Applies To class of the report definition rule.
• outcomeColumnInfo: page of class Embed-Decision-OutcomeColumnInfo. This page needs to provide pyName as the outcome column in the report definition that defines the behavior and map these values to the possible outcomes the adaptive model rule learns from.
• adaptiveModelKey: page of class Embed-Decision-AdaptiveModel-Key. This page needs to provide the adaptive model parameters. Adaptive model parameters are values that point to the model in the channel (pyChannel and pyDirection), action dimension (pyIssue, pyGroup and pyName) and class context (pyConfigurationAppliesTo and pyConfigurationName). See also usage of the adaptive model key to obtain predictor information (page 16).

The report definition rule that gathers the sample data. Only properties that are optimized for reporting when they have been created should be used in the report definition. The following example corresponds to a report definition that gathers work data. If the data is in an external data source, use the Connector & Metadata Wizard to create the necessary classes and rules.

<table>
<thead>
<tr>
<th>Column Source</th>
<th>Column Name</th>
<th>Sort Type</th>
<th>Sort Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>.Outcome</td>
<td>Outcome</td>
<td>Highest to Lowest</td>
<td>3</td>
</tr>
<tr>
<td>.Age</td>
<td>Age</td>
<td>Highest to Lowest</td>
<td>2</td>
</tr>
<tr>
<td>.CreditHistory</td>
<td>Credit History</td>
<td>Highest to Lowest</td>
<td>1</td>
</tr>
</tbody>
</table>

Decision Data

In activities, you can run decision data instances through the Call pxRunDecisionParameters method with the following parameters:

• DecisionParameters: the name of the decision data instance.
• AppliesTo: applies to class of the decision data instance.
• RunOnPage: page to copy the decision data results. If you omit this option, the results are stored in the step page.
• Key: the parameter key for filtering purposes.
• Value: the parameter value for filtering purposes.
Strategy

The methods listed below support the use of strategies in activities.

• Return list of propositions (page 17)
• Return list of properties (page 17)
• Compute segmentation logic (page 17)

Return List of Propositions

The Call Rule-Decision-Strategy.pyGetStrategyPropositions method allows you to obtain the list of propositions returned by the strategy. This method takes the following parameters:

• Name of the strategy
• Name of the results component (publicComponent parameter)
• Name of the page to hold the list of propositions
• Applies to class of the strategy

Return List of Properties

The Call Rule-Decision-Strategy.pyGetStrategyProperties method allows you to obtain the list of properties used by components in the strategy. Duplicate values are ignored. This method takes the following parameters:

• Name of the strategy
• Name of the results component (publicComponent parameter)

  If you provide the name of this component, the method returns its properties and the properties of other components that are required in its execution path. If not defined, the method returns all properties used in strategy components.
• Name of the page to hold the list of properties
• Applies to class of the strategy
• Option to exclude referenced strategies

  By default, all strategies in the decision path are considered.

Compute Segment Logic

The Call Rule-Decision-Strategy.pyComputeSegmentLogic method obtains the list of segments that can be returned by the strategy. The segment logic computation goes through the chain of component connections, gathering information about segment components and logical connections between them. If there is a sub strategy component involved, also segments of the sub strategy are gathered. The result is represented in a tree structure that contains the resulting classes: Embed-AST (base class), Embed-AST-Operator-Boolean (logical operator and operands), Embed-AST-Constant-String (segment rule name).

The method generates the following:

• AND-nodes for segment components in a sequence (for example, SegmentA component connects to SegmentB component).
• OR-nodes for segment components that do not connect to each other, but connect instead to the same component (for example, SegmentA and SegmentB components connect to a set property component) generated.

You can run the activity in the strategy results page, or you can provide the name of the strategy and the Applies To class. This method takes the following parameters:

• Name of the strategy
• Name of the components
• Name of the page for the result of computing the segmentation logic
• Applies to class of the strategy

Interaction

In activities, you can run interaction rules by using the Call pxExecuteInteraction method with the following parameters:
• **Interaction**: the name of the interaction rule.
• **IncludePredictorData**: whether to include adaptive modeling predictor information (true) or not (false).
• Adaptive modeling predictor information:

<table>
<thead>
<tr>
<th>Interaction Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue decision (write results to clipboard)</td>
<td><strong>PredictorData</strong>: reference to the property where to store serialized predictor information when running the interaction.</td>
</tr>
<tr>
<td>Capture results (write results to Interaction History)</td>
<td><strong>SerializedPredictorData</strong>: reference to the property containing serialized predictor information. Typically, this is the property used in issue decision mode.</td>
</tr>
</tbody>
</table>

**Data Set**

The *DataSet-Execute* API allows you to use data sets in activities, and the same settings are available when you run the instance through the *Action* menu. The possible data management operations depend on the type of data set you select when defining the method parameters:

• **Database table type** (page 18)
• **Decision data store type** (page 18)
• **Visual Business Director type** (page 19)
• **Stream** (page 19)
• **HBase type** (page 19)
• **HDFS type** (page 19)
• **Pega-DecisionEngine data sets** (page 20)

**Database Table**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Save</strong></td>
<td>Save records passed by a page or data transform in the database table.</td>
</tr>
<tr>
<td><strong>Browse</strong></td>
<td>Read records.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Maximum number of records to read</strong>: enter a value to define the threshold for stopping the browse operation. You can also define this value through an expression.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Store results in</strong>: use this setting to define the result page. The result page consists of an existing <em>Code-Pega-List</em> page.</td>
</tr>
<tr>
<td><strong>Browse by keys</strong></td>
<td>Read database table records by key.</td>
</tr>
<tr>
<td></td>
<td>• Define the key value.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Store results in</strong>: define the clipboard page containing the results of this operation.</td>
</tr>
<tr>
<td><strong>Delete by keys</strong></td>
<td>Define the key value to delete records by key.</td>
</tr>
<tr>
<td><strong>Truncate</strong></td>
<td>Remove all records.</td>
</tr>
<tr>
<td><strong>Browse partitions</strong></td>
<td>Read data partitions.</td>
</tr>
</tbody>
</table>

**Decision Data Store**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Insert records</strong></td>
<td>Write records to the decision data store. You can refine this operation with the following parameters:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Specify time to live (in seconds)</strong>: use this setting to specify the longevity of the records in the decision data store. This parameter accepts constant values (for example, 3600), property references of values calculated through expressions.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Write single track</strong>: use this setting to write a single track represented by an embedded property. All other properties are ignored if you specify the single track.</td>
</tr>
<tr>
<td><strong>Read by keys</strong></td>
<td>Read records. You can refine this operation with the following parameters:</td>
</tr>
</tbody>
</table>
• **Result page**: use this setting to define the result page. The result page consists of an existing Code-Pega-List page. You can use keywords (such as, Primary or StepPage), in which case the operation results in updating the page with the first records from the data set, something specially suited when working with a single value.
• **Key values**: use this setting to specify filtering through constant values or expressions.

| **Remove track** | Remove a single track from the decision data store by specifying the embedded property that identifies the track to be removed by this operation. This operation can take a considerable amount of time to complete in environments with many decision nodes as it removes the values from every single decision node. |
| **Truncate** | Remove all records. |
| **Browse partitions** | Read data partitions. |

### Visual Business Director

<table>
<thead>
<tr>
<th><strong>Operation</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Save</strong></td>
<td>Save records passed by a page or data transform in the VBD data source. The data source is now visible in the data sources landing page, and you can use it when writing to VBD in interaction rules and decision data flows.</td>
</tr>
<tr>
<td><strong>Truncate</strong></td>
<td>Remove all records from the VBD data source.</td>
</tr>
<tr>
<td><strong>Get statistics</strong></td>
<td>Get the VBD data source statistics.</td>
</tr>
</tbody>
</table>

### Stream

Stream data sets data operations follow the publish-subscribe pattern.

<table>
<thead>
<tr>
<th><strong>Operation</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Save</strong></td>
<td>Publish records to the data set.</td>
</tr>
<tr>
<td><strong>Browse</strong></td>
<td>Subscribe to the data set and receive records when they become available.</td>
</tr>
<tr>
<td>• <strong>Stop browsing after</strong>: define the duration threshold for stopping the browse operation duration. The duration can be defined in seconds, minutes or hours.</td>
<td></td>
</tr>
<tr>
<td>• <strong>Maximum number of records to read</strong>: define the number of records threshold for stopping the browse operation. You can also define this value through an expression.</td>
<td></td>
</tr>
<tr>
<td>• <strong>Store results in</strong>: use this setting to define the result page.</td>
<td></td>
</tr>
</tbody>
</table>

### HDFS

<table>
<thead>
<tr>
<th><strong>Operation</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Save</strong></td>
<td>Save records passed by a page or data transform in the HDFS data storage.</td>
</tr>
<tr>
<td><strong>Browse</strong></td>
<td>Read records.</td>
</tr>
<tr>
<td>• <strong>Maximum number of records to read</strong>: enter a value to define the threshold for stopping the browse operation. You can also define this value through an expression.</td>
<td></td>
</tr>
<tr>
<td>• <strong>Store results in</strong>: use this setting to define the result page. The result page consists of an existing Code-Pega-List page.</td>
<td></td>
</tr>
<tr>
<td><strong>Browse partitions</strong></td>
<td>Read data partitions.</td>
</tr>
</tbody>
</table>

### HBase

<table>
<thead>
<tr>
<th><strong>Operation</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Save</strong></td>
<td>Save records passed by a page or data transform in the HBase data store.</td>
</tr>
</tbody>
</table>
Browse
Read records.

- **Maximum number of records to read**: enter a value to define the threshold for stopping the browse operation. You can also define this value through an expression.
- **Store results in**: use this setting to define the result page. The result page consists of an existing Code-Pega-List page.

Browse by keys
Read records by key.

- Define the key value.
- **Store results in**: define the clipboard page containing the results of this operation. The result page consists of an existing Code-Pega-List page.

Browse by step page
Read records by step page.

Browse partitions
Read data partitions.

Pega-DecisionEngine

The ADM and IH data sets allow you to save records passed by a page or data transform in the corresponding data store (ADM, or IH).

Interaction History

The pxAddIdentityAssociation function associates IDs generated by interactions through different channels and devices with a known customer ID, and pxRemoveIdentityAssociation removes these associations. Decision Management provides two default activities that you can run to associate IDs generated by external interactions with a given subject ID. You can combine identity matching in your activities by adding the Call Data-Decision-IH-Association.pySaveAssociation and Call Data-Decision-IH-Association.pyDeleteAssociation methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Call Data-Decision-IH-Association.pySaveAssociation** | Adds an association based on ID, subject ID and association strength. Parameters:  
  - **SubjectID**: identifier representing the customer ID.  
  - **AssociatedID**: identifier representing the ID captured in anonymous interactions.  
  - **AssociationStrength**: a numeric value that can be used to establish the weight, match confidence or relevance for filtering purposes. In strategies, you can implement association strength based filtering by adding filter components to define some filtering logic that applies to the input data passed by interaction history or proposition components.  
  
  This method creates two records: one record where the subject ID is determined by the SubjectID parameter and the associated ID determined by the AssociatedID parameter, and a second record where the subject ID is determined by the AssociatedID parameter and the associated ID determined by the SubjectID parameter. The same association strength value is applied to both records. |

| **Call Data-Decision-IH-Association.pyDeleteAssociation** | Removes the two association records based on AssociatedID and SubjectID. |

Simulation

The methods listed below support simulations in activities.

- **Create simulation**: create a simulation run by using the Call Pega-DM-Batch-Work.pxCreateSimulationRun method with the parameters described below.
  - **Input definition name**
• Interaction rule name
• Work ID (out parameter to override default generated IDs)

Two additional parameters are provided (apply constraints and constraint data). These parameters are only used in Next-Best-Action Marketing implementations.

• Run simulation: once defined, simulations can be triggered by using the Call Pega-DM-Batch-Work.pxInvokeDecisionExecution method and providing the work item's ID.

Financial Functions

You can use the financial functions available in the Financial library to perform financial calculations. The following functions are provided in the Financial library:

- cumipmt (page 21)
- cumprinc (page 21)
- db (page 21)
- dbb (page 22)
- fv (page 22)
- ipmt (page 22)
- nper (page 22)
- pmt (page 22)
- ppmt (page 22)
- pv (page 23)
- rate (page 23)
- sln (page 23)
- syd (page 23)
- vdb (page 23)

General remarks when using providing the arguments:

- Rate and number of periods are calculated using the same period unit. For example, if the rate is calculated in months, the number of periods should also be expressed in months.
- Payments should be expressed as an array of negative numeric values.
- Incomes/loans should be expressed as an array of positive numeric values.

Cumulative Interest

Calculates the cumulative interest paid on a loan for a given period of time taking the following arguments:

- Interest rate: the interest rate (page 23) applied to the loan.
- Total number of periods: the total number of periods (page 22) for the loan.
- Present value: the present value (page 23) of the loan.
- Starting period: the starting period for measuring the cumulative interest paid. Periods are index one based.
- Ending period: the ending period for measuring the cumulative interest paid.

Example: @(Pega-DecisionEngine:Financial).cumipmt(0.005, 360, 200000, 1, 12)

Cumulative Principal

Calculates the cumulative principal paid on a loan for a given period of time taking the same arguments as described in the cumulative interest paid function (page 21).

Example: @(Pega-DecisionEngine:Financial).cumprinc(0.005, 360, 200000, 1, 12)

Depreciation Using Fixed-Declining Balance

Calculates the depreciation of an asset using the fixed-declining balance method, a method that computes the depreciation at a fixed rate. This function takes the following arguments:

- Cost: the original cost of the asset.
- Salvage: the salvage value at the end of the depreciation.
- Number of periods: the number of periods (page 22) over which the asset is being depreciated, also known as the useful life of the asset.
- Period: using the same unit measure as provided for the number of periods, the period to calculate asset depreciation.
• Number of months in the first year: optional argument used to provide a value other than 12 for the first year of asset depreciation.

Example: @(Pega-DecisionEngine:Financial).db(30000, 7000, 10, 1)

Depreciation using Double Declining Balance

Calculates the depreciation of an asset using the double-declining balance method, or some user specified method. The four initial arguments are similar to the ones used with the fixed-declining balance function (page 21). The fifth factor argument is applied to provide the rate at which the balance declines (default is assumed to be 2).

Example: @(Pega-DecisionEngine:Financial).ddb(30000, 7000, 10, 1, 1.75)

Future Value

Calculates the future value of an investment taking the following arguments:

• Interest rate: the constant interest rate (page 23).
• Number of periods: number of periods (page 22) for the payments.
• Payments: the payment (page 22) (negative value) to be paid each period.
• Present value: the present value (page 23) of the investment.
• True/false: condition indicating if the payments are due at the end of each period (false, which is also the default value) or beginning of each period (true).


Interest Payment

Calculates the interest payment for a given period for an investment taking the interest rate (page 23), period, number of periods (page 22), present value (page 23) and future value (page 22) arguments.

Example: @(Pega-DecisionEngine:Financial).ipmt(2.66, 1, 36, 8000)

Number of Periods

Calculates the number of periods for an investment, using the future value (page 22), the present value (page 23), rate and payment amount. The function assumes periodic and constant payments a constant interest rate.


Payment

Calculates the payment of a loan based on constant interest rate and constant payments taking the same arguments as described in the interest payment function (page 22). Typically, the payment contains principal and interest and no other fees or taxes.

Example: @(Pega-DecisionEngine:Financial).pmt(2.66, 10, 10000, 9500)

Principal Payment

Calculates the payment on the principal for a given period of an investment based on periodic, constant payments and constant interest rate. This function takes the same arguments as described in the interest payment function (page 22). This calculation can also be expressed by payment (page 22) minus interest payment (page 22).

Example: @(Pega-DecisionEngine:Financial).ppmt(0.006, 10, 10000, 9500)
Present Value

Calculates the net present value. The function assumes periodic and constant payments a constant interest rate.

Example: @/(Pega-DecisionEngine:Financial).pv(2.66, 10, -200, 4000)

Rate

Calculates the interest rate per period of an annuity. This function takes the number of periods (page 22), payment (page 22), present value (page 23) and future value (page 22) arguments.

Example: @/(Pega-DecisionEngine:Financial).rate(0.16, -200, 4000)

Straight-Line Depreciation

Calculates the straight-line depreciation of an asset for one period in the life of an asset. Cost, salvage and life arguments are explained in the depreciation function (page 21).

Example: @/(Pega-DecisionEngine:Financial).sln(30000, 7000, 10)

Sum-of-Years' Depreciation

Calculates the sum-of-years’ digits depreciation of an asset after a specified period taking the same arguments as described in the depreciation function (page 21).

Example: @/(Pega-DecisionEngine:Financial).syd(30000, 7000, 10, 1)

Variable Depreciation

Calculates the depreciation of an asset for any specified period. The depreciation calculation is variable and uses the double-declining balance method or a user-specified method. The arguments are quite similar to the ones used in the double-declining depreciation function (page 22). Three extra arguments apply:

- Start period: the start period for which you want to calculate the depreciation.
- End period: the end period for which you want to calculate the depreciation.
- True/false: a condition specifying to switch to straight-line depreciation if depreciation is greater than the declining balance calculation (true) or not (false).

Example: @/(Pega-DecisionEngine:Financial).vdb(30000, 7000, 10, 0, 1, 1.75)

Application Overlays

An application overlay defines the boundaries for business users to make changes through the Decision Manager portal. The application overlay also defines the instances Decision Manager portal users can access. There are two two types of application overlays:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct deployment</td>
<td>Direct deployment overlays are generated by users accessing the application overlays facilities for the first time. Application overlays that can only contain decision data rules. The system generates one direct deployment application overlay per application. You define the rules included in the direct deployment revision ruleset, but you cannot remove or influence the name of the application overlay. This type of overlay cannot be generated in production environments; if present, it is carried over from development or pre-production.</td>
</tr>
<tr>
<td>Revision management</td>
<td>Revision management overlays are generated by running the application overlay wizard. You can create, change and delete revision management application overlays. Revision management overlays can include the following rule types: decision data, decision table, decision tree, event strategy, map value, predictive model, scorecard and strategy.</td>
</tr>
</tbody>
</table>
System operations underpinning the actions performed by system architects working with application overlays:

- Create revision management application overlay (page 25)
- Create direct deployment application overlay (page 24)
- Update application overlay (page 25)
- Delete application overlay (page 25)

Application overlays have an important relationship with the access groups that define what users can do in the Decision Manager portal. Refer to the Decision Manager roles and privileges (page 28) section for an overview of the technical details concerning the implementation of these roles and privileges.

In the process of creating application overlays, the wizard does not associate operators with the access groups. System architects have to configure the operator records accordingly.

Rule instances that are present in branches are not eligible for revision management. If an instance is in both the base revision and a branch, only the base revision is considered when creating the application overlay.

Create Revision Management Application Overlay

The system architect runs the new overlay wizard, resulting in the following operations:

- Create an initial revision ruleset version 01-01-01 with revision ruleset dependencies.
- Create a branch ruleset for the revision ruleset.
- Create the overlay application with revision ruleset 01-01-01 version in the application's ruleset list.
- Create the access groups with the selected role configuration. These access groups are configured to include `pyDecisionPortal` in each access group's portal settings.
- Create a copy of the `pxDecisioningClass` field value (location of the SR class) in the revision ruleset.
- Update the current application to contain the revision ruleset 01-01-01 in the first row of the application's ruleset list.
- Associate the branch ruleset with the application overlay.
- Create a data instance of Data-Decision-RevisionRecords to contain the list of the selected revision rules.
- Copy the instances selected for the application overlay into the 01-01-01 revision ruleset.

Create Direct Deployment Application Overlay

Application overlays for the purposes of direct deployment are automatically generated for each application in non-production systems. Generating these overlays is triggered by system architects accessing the application overlays landing page for the first time, resulting in the following operations:

- Check if there is an existing application overlay for direct deployment. If this application overlay does not exist, proceed with creating it.
- Create the overlay application with revision ruleset 01-01-01 version in the application's ruleset list. The name of the application overlay follows this pattern: `RTC-<first ten characters of application name>`.
- Create an initial revision ruleset version 01-01-01 with revision ruleset dependencies.
  - This ruleset uses application validation, not ruleset validation.
  - The name of the ruleset is based on the following pattern.
- Create a branch ruleset for the revision ruleset.
- Associate the branch ruleset with the application overlay.
- Create the `<application_name>`:DirectDeploy access group.
- Create a copy of the `pxDecisioningClass` field value (location of the SR class) in the revision ruleset.
• Create a data instance of Data-Decision-RevisionRecords to contain the list of the selected revision rules. Initially, this list is empty.

By default, the name of a direct deployment overlay is constructed by prefixing RTC- to the first ten characters of the application's name. You can override the default prefix by configuring the DirectDeploymentOverlayName dynamic system setting. Pega-DecisionManager is the owning ruleset of this dynamic system setting.

If any of the automatically generated names used in the process of creating a direct deployment application overlay exist, the names are appended with a count (for example, BranchRuleset1).

Update Application Overlay

The system architect updates a revision management application overlay, resulting in the following operations:

• Nothing happens if the update results in no changes.
• If the update results in removing rules, update the data instance of Data-Decision-RevisionRecords to remove these instances.
• If the list of rules changes:
  • Create a new minor revision ruleset version (for example, 01-02-01).
  • Copy the new instances to this new minor revision ruleset version, and lock the ruleset version.
  • Update the branch ruleset with the new minor revision ruleset version.
  • Update the current application's revision ruleset version with the new minor revision ruleset version.
  • Update the application overlay's revision ruleset version with the new minor revision ruleset version.

The system architect updates the list of rules (decision data) a direct deployment application overlay, resulting in the following operations:

• Every update consists of updating the revision records but the ruleset version remains the same (that is, version is not incremented).
• If the update results in removing rules, update the data instance of Data-Decision-RevisionRecords to remove these instances.
• If the update results in adding rules, update the data instance of Data-Decision-RevisionRecords to add these instances.

Delete Application Overlay

The system architect deletes an application overlay, resulting in the following operations:

• Where used by operators, remove the access groups from the operator's access group list.
• Delete the access groups associated with the application overlay.
• Delete the overlay application.
• Delete the data instance of Data-Decision-RevisionRecords containing the list of the selected revision rules.
• Delete the branch ruleset.

Deleting an application overlay can only be done for revision management application overlays, as direct deployment application overlays are system-defined, not user-defined.

Revisions & Change Requests

System operations underpinning the actions performed by Decision Manager portal users working with revisions and change sets:
• Assign change request (page 26)
• Send back change request (page 26)
• Submit change request for approval (page 26)
• Reject change request (page 26)
• Withdraw change request (page 26)
• Withdraw revision (page 26)
• Submit revision (page 27)
• Activate revision (page 27)
• Roll-back revision (page 27)

Assign Change Request
The revision manager defines and assigns a change request, resulting in the following operations:
• Copy the instances included in the change request to the branch rule set when the strategy designer edits them.
• Copy and check out the instances to the strategy designer’s personal ruleset.
• Change status to in progress.

Send Back Change Request
The strategy designer sends back the change request, resulting in the following operations:
• Verify checked out instances.
• Remove instances from the strategy designer’s personal ruleset.
• Change status to pending-update.

Submit Change Request
The strategy designer submits a change request for approval by the revision manager, resulting in the following operations:
• Check in instances into the branch ruleset.
• Remove instances from the strategy designer’s personal ruleset.
• Change status to pending-approval.

Reject Change Request
The revision manager rejects the change request requested for approval, resulting in the following operations:
• Remove from the branch the instances that are not included in an already approved change request. If the instances were included in another change request within the same revision, and that change set was already approved, rejecting the change request does not remove the instances from the branch, but instead replaces them with the versions of the approved change request.
• Change status to resolved-rejected.

Withdraw Change Request
The revision manager withdraws the change request assigned to the strategy designer, resulting in the following operations:
• Remove instances from the strategy designer’s personal ruleset.
• Change status to resolved-withdrawn.

Withdraw Revision
The revision manager withdraws revision, resulting in the following operations:
• Verify checked out instances.
• Remove instances from the strategy designer’s personal ruleset.
• Remove instances from the branch.
• Change status to resolved-withdrawn.

Submit Revision
The revision manager submits the revision, resulting in the following operations:
• Copy modified instances from the branch ruleset to the revision ruleset. The revision ruleset version is the next available version. After copying, lock the revision ruleset.
• Remove instances from the branch.
• Generate the revision package containing the last created and locked revision ruleset.
• If it is a generic revision (rule revision management purposes, not direct deployment), change status to resolved-completed.
• If it is a direct deployment revision, and there are changes in decision data instances, change status to pending-activated.

Activate Revision
The revision manager activates changes included in a direct deployment revision, resulting in the following operations:
• List access groups of the current version of the application the overlay is built on.
• For every access group, update the version of the production rulesets to match the new revision ruleset version.
• Change status to resolved-activated.

Roll-back Revision
The revision manager rolls back a resolved-activated revision, resulting in the following operations:
• List access groups of the current version of the application the overlay is built on.
• For every access group, update the version of the production rulesets with the previous available resolved-activated revision ruleset version. In the absence of such a previous version that can be used, remove the revision ruleset from each access group’s production rulesets.
• Change status to resolved-rolledback.

Revision Management
System operations underpinning the actions performed by system architects managing the packaged revisions generated by revision managers in the Decision Manager portal:
• Import revision (page 27)
• Discard revision (page 27)
• Activate revision (page 28)
• Roll-back revision (page 28)

Import Revision
The system architect imports a revision, resulting in the following operations:
• Increment the application patch version.
• Create the roll-out access group based on the current access group.
• Update the roll-out access group to point to the new application version.
• Append the roll-out access group to the list of access groups for the selected test users.
• Change status to testing.

Discard Revision
The system architect discards a revision under testing, resulting in the following operations:
• Remove the incremented application patch version.
• Remove the roll-out access group.
• Clear the roll-out access group from any operator selected to test the revision.
• Change status to **discarded**.

### Activate Revision

The system architect activates revision under testing, resulting in the following operations:

• Update the current access group with the roll-out access group's configuration.
• Update the built on version of the application overlay to point to the new version (that is, the version of the application that was under testing).
• If this revision overrides decision data instances coming from direct deployments in non-production systems:
  • Remove the versions of decision data instances from the direct deployment overlay.
  • Remove the production ruleset versions pointing to the direct deployment revision ruleset.
  • Update all revisions to resolve the roll-back of direct deployment decision data instances.
  • Change the *resolved-activated* direct deployment revision in the Decision Manager portal to *resolved-rolledback*.
• If this revision does not override decision data instances coming from direct deployments in non-production systems:
  • Changes to decision data instances that are defined in both this revision and the direct deployment revision are not taken into account.
  • The *resolved-activated* direct deployment revision in the Decision Manager portal remains in *resolved-activated* state.
• Change the status of the currently active revision to **inactive**.
• Change the status of this new revision to **active**.

By default, a revision package containing decision data instances overrides the values of the parameters set by these instances in the direct deployment revision package in *resolved-activated* state (management of direct deployment revisions is done by revision managers in the Decision Manager portal). This behavior can be disabled by selecting the setting that allows you to preserve the changes made through direct deployment by revision managers. If you select this option, you discard changes in decision data instances that are also present in the direct deployment.

### Roll-back Revision

The system architect rolls-back a revision in production, resulting in the following operations:

• Create a new application version.
• Copy the contents of the previously active revision to this new application version.
• Update the current access group with the new application version.
• Update the built on version of the application overlay to point to the new application version.
• Change the status of the currently active revision to **rolled-back**.
• Change the status of this new revision to **active**.

You cannot issue the roll-back action in the same session as the one used to activate the revision. Logout and login again to be able to roll-back a newly activated revision.

### Decision Manager Roles & Privileges

Access to the Decision Manager portal is governed by access roles, and each access role is associated with privileges. The access to the Decision Strategy Manager portal requires the applicable access groups to list `pyDecisionPortal` in their portal settings.
Default Roles

Decision Manager defines default access roles that can be used to configure what users can do in the Decision Manager portal. Typically, roles apply to the business sandbox or the production environment.

Business sandbox roles:

<table>
<thead>
<tr>
<th>Access Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DecisionManager:StrategyDesigner</strong></td>
<td>Strategy designers work on change requests, test applications, run simulations, analyze results through reports and Visual Business Director, manage the assets used in the VBD planner and submit the change request for approval by revision managers. They also have access to the calendar.</td>
</tr>
<tr>
<td><strong>DecisionManager:RevisionManager</strong></td>
<td>Revision managers create revisions and change requests, assign change requests to workbaskets or strategy designers, manage the assets used in the VBD planner, run applications or simulations to validate changes, approve changes and package revisions. Revision managers can also act as strategy designers when change requests are assigned to them.</td>
</tr>
<tr>
<td><strong>DecisionManager:DecisionArchitect</strong></td>
<td>This role is reserved to Decision Management experts because it combines the privileges given to all other roles. Although typically applicable to the business sandbox, it can also be present in the production environment.</td>
</tr>
<tr>
<td><strong>DecisionManager:DirectDeploy</strong></td>
<td>The direct deployment access role provides users with the ability to directly deploy revisions originated from direct deployment overlays. Users with this role can manage revisions, work on change requests, run applications and view reports.</td>
</tr>
</tbody>
</table>

Production environment roles:

<table>
<thead>
<tr>
<th>Access Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DecisionManager:Supervisor</strong></td>
<td>Production supervisors keep interactions under observation through reports and Visual Business Director. They can also update adaptive reporting data.</td>
</tr>
<tr>
<td><strong>DecisionManager:Administrator</strong></td>
<td>Production administrators have the same access as supervisors. Additionally, they typically update propositions, manage the assets used in the VBD planner, manage adaptive models and run applications.</td>
</tr>
</tbody>
</table>

Privileges

Decision Manager defines the privileges used to set up access roles according to the level of access to the Decision Manager portal. These privileges are designed to be granular so that you can set up custom roles, and use them instead of the default Decision Manager roles.

<table>
<thead>
<tr>
<th>Access</th>
<th>Users with this privilege can...</th>
<th>Privilege</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Models</td>
<td>Monitor adaptive models, monitor predictors and access ADM reports. Update adaptive reporting data.</td>
<td>pyMonitorADMMModels</td>
</tr>
<tr>
<td>Monitoring</td>
<td></td>
<td>pyUpdateADMRreportingData</td>
</tr>
<tr>
<td>Adaptive Models</td>
<td>Manage adaptive models through the actions that allow for controlling the adaptive data store (clear, delete and train models).</td>
<td>pyManageADMMModels</td>
</tr>
<tr>
<td>Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reports</td>
<td>Access reports in the Interaction History category. Access the report browser.</td>
<td>pyMonitorInteractionHistory</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pyMonitorReports</td>
</tr>
<tr>
<td>Test Applications</td>
<td>Run applications.</td>
<td>pyTestFlows</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Visual Business Director Use the VBD planner to monitor actuals. pyMonitorVBDActuals</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use the VBD planner to monitor simulation data sources. pyMonitorVBDSimulation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Manage KPIs, data sources and views. pyManageVBD</td>
<td></td>
</tr>
<tr>
<td>Simulations</td>
<td>Run simulations. pyRunSimulations</td>
<td></td>
</tr>
<tr>
<td>Revisions</td>
<td>View business rules, revisions and work list. Access the revisions facilities. pyManageRevisions</td>
<td></td>
</tr>
<tr>
<td>Change Requests</td>
<td>View business rules, work on change requests, open rules, change rules and submit changes. pyWorkOnChangeSets</td>
<td></td>
</tr>
<tr>
<td>Plan</td>
<td>Access the Plan facilities. pyViewCalendar</td>
<td></td>
</tr>
</tbody>
</table>

Custom Roles

If the default roles do not meet the requirements for user activity in the Decision Manager portal, configure your own custom roles by using the granular privileges that, combined, define the group of actions for a given role. The following steps also apply if you need to add a new access group to an existing application overlay:

1. Define access of role to object instances using the Decision Manager privileges.
2. Associate each access role with the applicable access of role to object depending on the type of user exposure to Decision Manager portal functionality.
3. Associate each access group with the applicable access role.

Proposition Cache Synchronization

Proposition cache works on a single PRPC node. Consistent handling of the proposition cache in a multinode PRPC environment requires extra configuration.

- About proposition cache (page 30)
- Configuring proposition cache synchronization (page 30)

About Proposition Cache

When PRPC is running on multiple system nodes connected to the same database, Decision Management uses the system pulse to ensure the consistency of propositions across all nodes. The proposition cache is invalidated when a proposition is saved (triggered by adding or changing a proposition) or deleted. Adding records that result in the proposition cache to become invalid is done through two declare trigger rules that run the pyRefreshPropositions activity (pyPropositionSaved and pyPropositionRemoved in Data-pxStrategyResult).

Configuring Proposition Cache Synchronization

If your installation consists of different PRPC nodes connecting to the same database, you need to configure your PRPC installation to ensure the consistency of propositions across all nodes. System architects can follow these steps to enable PRPC to work with the proposition cache synchronization mechanism:

1. In the Records Explorer, go to SysAdmin > Agent Schedule.
2. Open the Pega-RULES: Core Engine Processing Agent data instances for every active node and add the PegaDM:Administrators access group.
3. Restart PRPC or use SMA to restart the agents.
Interaction between Process Commander and the Decision Management service layer (Adaptive Decision Manager and Visual Business Director) is triggered by:

- Executing an interaction rule or a data flow configured for capturing interaction results or writing results to VBD.
- Executing a strategy that creates or updates adaptive models.
- Adaptive model configuration through adaptive model rules.
- Actions performed in the adaptive models landing page.
- Scoring model updates by running the UpdateAdaptiveModels agent in the Pega-DecisionEngine ruleset.

This interaction consists of gathering the required information for scoring and capturing data resulting from interactions. If adaptive models are used in the decision execution process, models are executed and model data updated. Sending the necessary information to ADM is triggered by changes in adaptive model rules, managing models through the adaptive models landing page and running the UpdateAdaptiveModels agent (page 14). In the process of saving interaction results, the VBD data source known as actuals is automatically updated.

Communication with the ADM server in the service layer is not triggered when using the in-memory ADM process.

The diagram below provides an overview of the communication between PRPC, ADM and VBD. In the process of passing and retrieving information can be performed, PRPC needs to be aware of the necessary Decision Management service layer end points.
Related Information

- Adaptive Decision Manager Tables (page 32)
- Visual Business Director Tables (page 33)

Adaptive Decision Manager Tables

ADM is a stateful system (state is in memory). The frequency of writing its state is determined by the BACKUP INTERVAL settings. The state is also updated when you stop the ADM application. The table below explains the ADM database tables.

<table>
<thead>
<tr>
<th>Column Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETTINGS</td>
<td>Table containing the data associated with the behavior of the ADM system.</td>
</tr>
<tr>
<td>CONFIGURATIONS</td>
<td>Table containing the serialized ADM configurations. An ADM configuration</td>
</tr>
<tr>
<td></td>
<td>corresponds to the settings in the adaptive model rule that defines how</td>
</tr>
<tr>
<td></td>
<td>to create models. There is one configuration object per adaptive model</td>
</tr>
<tr>
<td>FACTORIES</td>
<td>rule and, typically, each object is shared by multiple factories.</td>
</tr>
<tr>
<td></td>
<td>Table containing the serialized ADM factories.</td>
</tr>
</tbody>
</table>
ADM_EVENTQUEUE table:
Default staging table containing new factories, as well as new interaction result records. Used by the fail safe mechanism.

SETTINGS table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td>Settings are stored using key-value pairs. This column represents the Primary Key that identifies the key-value pair.</td>
</tr>
</tbody>
</table>
| VALUE       | This column contains the values:  
  - **BackupInterval**: interval (in milliseconds) between regular backup of models in the ADM system. Default setting is 3600000.  
  - **LastResponseTime**: time stamp corresponding to the last interaction results handled by ADM. Used by the fail safe mechanism.  
  - **FailSafeEnabled**: indicates if ADM is operating with the fail safe mechanism enabled (**true**) or not (**false**). |

CONFIGURATIONS table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONFIG_PARTITION_ID</td>
<td>Primary Key. Hash code (version 5 UUID) of CONFIG_PARTITION.</td>
</tr>
<tr>
<td>CONFIG_PARTITION</td>
<td>The partition that identifies this configuration in canonical JSON format. The partition is a combination of a key-value, where pyClassName and pyPurpose are the keys.</td>
</tr>
<tr>
<td>FORMAT</td>
<td>The data serialization format (YAML) used to serialize the configuration.</td>
</tr>
<tr>
<td>CONFIGURATION</td>
<td>The serialized representation of the configuration. The serialization format used is specified in FORMAT.</td>
</tr>
</tbody>
</table>

FACTORIES table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL_PARTITION_ID</td>
<td>Primary Key. Hash code (version 5 UUID) of MODEL_PARTITION.</td>
</tr>
<tr>
<td>CONFIG_PARTITION_ID</td>
<td>Primary Key. Hash code (version 5 UUID) of CONFIG_PARTITION. Foreign Key: CONFIGURATIONS.CONFIG_PARTITION_ID</td>
</tr>
<tr>
<td>MODEL_PARTITION</td>
<td>The partition for which the factory should produce a model in canonical JSON format. Models are created for every unique combination of proposition and channel internally represented by key-value pairs, where pyIssue, pyGroup, pyName, pyDirection and pyChannel are the keys.</td>
</tr>
<tr>
<td>CONFIG_PARTITION</td>
<td>The partition identifying the configuration in canonical JSON format.</td>
</tr>
<tr>
<td>FORMAT</td>
<td>The data serialization format (YAML) used to serialize the factory.</td>
</tr>
<tr>
<td>FACTORY</td>
<td>The serialized representation of the factory. The serialization format used is specified in the FORMAT column.</td>
</tr>
</tbody>
</table>

ADM_EVENTQUEUE table:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT_TIME</td>
<td>Time of the event.</td>
</tr>
<tr>
<td>EVENT_TYPE</td>
<td>Type of the event (model creation, interaction results, etc.).</td>
</tr>
<tr>
<td>ENCODING_TYPE</td>
<td>Encoding used for OBJECT.</td>
</tr>
<tr>
<td>OBJECT</td>
<td>Encoded data of the event so that it can be replayed.</td>
</tr>
</tbody>
</table>

Visual Business Director Tables

The tables supporting the operation of VBD are included in the PRPC schema. The table below explains the VBD tables.

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR_DATA_VBD_CONFIG</td>
<td>Table containing configuration parameters. By default, this table is empty.</td>
</tr>
</tbody>
</table>
| PR_DATA_VBD_COLLECTION_CONFIG | Table containing the data sources corresponding to interaction results created through running an interaction rule. There are two types of data sources:  
|                             | • Actuals: JDBC based data source that is stored in memory. This data source is always updated in a VBD enabled system when writing results to Interaction History.  
|                             | • Simulation data sources: stored in memory and in the node’s file system for backup purposes. These data sources are updated through simulations designed for forecasting. |
Proposition data is supported by the proposition hierarchy class model that is defined in the SR (Strategy Result) class. The $pxDecisioningClass$ field value determines the location of the SR class in the class hierarchy. The ruleset is determined by finding the first unlocked application ruleset.

**Related Information**
- Classes (page 35)
- Properties (page 35)
- Propositions (page 36)

**Classes**
The $Data-pxStrategyResult$ class is the base class for Decision Management data. Classes supporting the proposition hierarchy are concrete classes. Propositions are organized by business issue and group. A business issue can have one or more groups, each group basically providing a label for a series of related propositions (for example, bundles, credit cards, loans and mortgages groups under the sales issue). If not present, the SR class is automatically created under the top level abstract class of the first unlocked ruleset in the application's ruleset list when you define the proposition hierarchy and, even if this hierarchy is not present, when you create strategies.

- The SR class uses pattern and directed inheritance from the $Data-pxStrategyResult$ class.
- Classes defining business issues do not have a key. They use pattern and directed inheritance from the SR class and have the $pyDecisioningItem$ custom field set to Issue.
- Classes defining groups have the $pyName$ key. They use pattern and directed inheritance from the class defining the business issue and have the $pyDecisioningItem$ custom field set to Group.
- The directed inheritance of strategies in the proposition hierarchy is established as follows:
  - Issue class if the strategy applies to a given business issue
  - Group class if the strategy also applies to a given group
  - SR class if the strategy is not associated with any given business issue or group

The $pyDecisioningItem$ custom field of classes supporting the issue and group definition is set to MarkedForDeletion if you remove the corresponding business issue or group.

Changing the decision hierarchy requires the ruleset the SR class belongs to be unlocked, and the same applies to properties.

**Properties**
The $Data-pxStrategyResult$ class contains properties that define the basic output of a decision.

**Note**
Decision Management does not support declare expressions targeting properties in the SR classes.

Properties in the classes representing the proposition hierarchy are defined at the applicable level depending on its scope in the proposition hierarchy.
**Scope**

**Top level class**
The top level class is defined in the application’s `pxDecisioningClass` field value rule and determines the proposition hierarchy your application can access. Characteristics of the top level class:
- Directed inheritance from `Data-pxStrategyResult`
- Supports properties for which issue has not been defined
- By default, the pattern `<OrgClass>-<ApplicationName>-SR` is assumed.

**Issue class**
Characteristics of the issue class:
- Directed inheritance from the top level class
- Supports properties whose scope is business issue but not group.
- By default, the pattern `<OrgClass>-<ApplicationName>-SR-<Issue>` is assumed.

**Group class**
Characteristics of the group class:
- Directed inheritance from the issue class
- Supports properties whose scope is group
- By default, the pattern `<OrgClass>-Data-<ApplicationName>-SR-<Issue>-<Group>` is assumed.

The data model of the SR classes can contain properties defined to support propositions, properties defined as strategy properties, and generic properties.

**Purpose**

**Proposition**
Proposition properties configured with the `pyDecisioningItem` custom field set to `PropositionProperty`. The custom field is automatically set when adding properties through the `Proposition Management` landing page.

**Generic**
Standard or generic properties that belong to the data model of the classes that represent the proposition hierarchy. These properties are available as strategy properties.

In terms of managing these properties, only properties that are specific to your application can be deleted. Unlike business issues or groups, deleting a proposition property results in actually deleting the property rule.

**Propositions**

Previous to Pega 7.1.8, propositions were data instances of the group data class. Starting with Pega 7.1.8, propositions are part of the decision data rule instance managing propositions for a given group. In terms of data model, propositions also inherit the properties of the top level and issue classes. Each proposition can set values for properties defined as proposition attributes through the corresponding decision data rule (or, previous to Pega 7.1.8, the `Proposition Management` landing page).

For propositions that have been not migrated to decision data, you can continue doing proposition management as you did previous to Pega 7.1.8. For propositions that have been migrated to the new implementation, or applications created in Pega 7.1.8, proposition management is done through decision data rules. Proposition management can operate exclusively in this new mode by setting the `PropositionManagement/isOnlyVersionedProposition` dynamic system setting to `true`; if this setting is set to false, proposition management can operate in both modes, versioned, and non-versioned.
Interaction History

About Interaction History

Intelligent decisioning is not a static exercise. Customer behavior is constantly shifting, actions by both the enterprise and competitors impact customer behavior, changing business objectives and priorities. Feedback on decisions made by consumers in response to propositions is vital if the enterprise is to learn what works and what does not. Interaction management consists of retrieving the interaction data and history that can be used in the process of issuing a decision, or capturing the results of the interaction based on the combination of pySubjectID and strategy components configured for interaction management (proposition data, interaction history and data representing customer behavior).

Interaction History provides the persistence layer for storing interaction results. Interaction results are materialized in the form of fact records and dimension information. It is also based on this information that PRPC updates ADM and VBD, including updating the state of adaptive models and saving interaction results for monitoring and reporting purposes. Interaction History tables can be queried and analyzed, thus allowing for identifying where changes should be made and where new opportunities arise. The Interaction History reports are an example of reporting on interaction results.

Database Tables

To minimize duplication of data, Interaction History uses a star schema that consists of one fact table and eight dimension tables. By default, Interaction History tables are part of the PegaDATA schema.

<table>
<thead>
<tr>
<th>Database Table</th>
<th>Description</th>
</tr>
</thead>
</table>
| PR_DATA_IH_FACT             | • Class: Data-Decision-IH-Fact  
• Fact table  
• Contains FKs to the dimension tables, measurements and fact properties |
| PR_DATA_IH_DIM_ACTION       | • Class: Data-Decision-IH-Action  
• Action dimension table |
| PR_DATA_IH_DIM_APPLICATION  | • Class: Data-Decision-IH-Application  
• Application dimension table |
| PR_DATA_IH_DIM_CHANNEL      | • Class: Data-Decision-IH-Channel  
• Channel dimension table |
| PR_DATA_IH_DIM_CONTEXT      | • Class: Data-Decision-IH-Context  
• Context dimension table |
| PR_DATA_IH_DIM_CUSTOMER     | • Class: Data-Decision-IH-Customer  
• Customer dimension table |
| PR_DATA_IH_DIM_LOCATION     | • Class: Data-Decision-IH-Location  
• Location dimension table |
| PR_DATA_IH_DIM_OPERATOR     | • Class: Data-Decision-IH-Operator  
• Operator dimension table |
| PR_DATA_IH_DIM_OUTCOME      | • Class: Data-Decision-IH-Outcome |
The primary key of the dimension tables is a hash code of the property names and values of that dimension expressed as a 64-bit long value. The primary key of each dimension table is named `pzID`. The foreign keys are also 64-bit long values. The foreign keys for each dimension table in the fact table manage the relation between fact and dimension tables. In PRPC, this relation is captured in association rules under `Data-Decision-IH-Fact` to facilitate building reports on Interaction History (for example, when building a report based on the fact class, you can add `pxActionDimension.pyIssue` to join the `pyIssue` property of the action dimension).

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Foreign Key</th>
<th>Association Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>pzActionID</td>
<td>pxActionDimension</td>
</tr>
<tr>
<td>Application</td>
<td>pzApplicationID</td>
<td>pxApplicationDimension</td>
</tr>
<tr>
<td>Channel</td>
<td>pzChannelID</td>
<td>pxChannelDimension</td>
</tr>
<tr>
<td>Context</td>
<td>pzContextID</td>
<td>pxContextDimension</td>
</tr>
<tr>
<td>Customer</td>
<td>pzCustomerID</td>
<td>pxCustomerDimension</td>
</tr>
<tr>
<td>Location</td>
<td>pzLocationID</td>
<td>pxLocationDimension</td>
</tr>
<tr>
<td>Operator</td>
<td>pzOperatorID</td>
<td>pxOperatorDimension</td>
</tr>
<tr>
<td>Outcome</td>
<td>pzOutcomeID</td>
<td>pxOutcomeDimension</td>
</tr>
</tbody>
</table>

Currently, the schema does not define any Foreign Key constraints.

**Properties**

The Strategy Result class provides the basis for building the fact and dimension records. When writing strategy results to Interaction History, properties are split into fact and dimension properties, and saved into the appropriate table. Properties used to build fact and dimension records are mapped to database columns, defined in the SR class data, and also in Interaction History. The name of an Interaction History property is used to identify the property in the SR class. Properties defined in the SR class that are not in Interaction History are not considered when writing results to the fact and dimension tables; similarly, properties defined in Interaction History that are not in the SR class are also not considered when retrieving data from Interaction History. The name of Interaction History properties needs to be the same as the column names they are mapped to, something that applies to any of the default properties, as well as any property that is the result of extending Interaction History.

- Default fact properties (page 38)
- Default dimension properties (page 39)
- Default identity matching properties (page 40)

**Default Fact Properties**

Properties that have a wide range of unique values should be defined as fact properties. Numeric fact properties can be used as key performance indicators and visualized as such in Visual Business Director. Default fact properties defined on the `Data-Decision-IH-Fact` class:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pxDecisionTime</td>
<td>Time stamp corresponding to when the decision was issued.</td>
</tr>
<tr>
<td>pxFactID</td>
<td>Unique fact record ID.</td>
</tr>
<tr>
<td>pxInteractionID</td>
<td>The unique 64 bit identifier for all records that belong to the same interaction.</td>
</tr>
<tr>
<td>pxOutcomeTime</td>
<td>Time stamp corresponding to when the record was stored in the Interaction History tables.</td>
</tr>
<tr>
<td>pxPriority</td>
<td>Action priority.</td>
</tr>
<tr>
<td>pxRank</td>
<td>Action rank.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>pyExternalID</code></td>
<td>Work item ID that was set through interaction rules previous to PRPC 7.1. This property is not automatically set.</td>
</tr>
<tr>
<td><code>pyGroupId</code></td>
<td>The ID of the group the subject ID belongs to.</td>
</tr>
<tr>
<td><code>pyLatitude</code></td>
<td>Geographical latitude.</td>
</tr>
<tr>
<td><code>pyLongitude</code></td>
<td>Geographical longitude.</td>
</tr>
<tr>
<td><code>pyPropensity</code></td>
<td>Calculated propensity for the action.</td>
</tr>
<tr>
<td><code>pySubjectID</code></td>
<td>The subject ID used when setting the results.</td>
</tr>
<tr>
<td><code>pzActionID</code></td>
<td>Action ID.</td>
</tr>
<tr>
<td><code>pzApplicationID</code></td>
<td>Application ID.</td>
</tr>
<tr>
<td><code>pzChannelID</code></td>
<td>Channel ID.</td>
</tr>
<tr>
<td><code>pzContextID</code></td>
<td>Decision context ID.</td>
</tr>
<tr>
<td><code>pzCustomerID</code></td>
<td>Customer ID.</td>
</tr>
<tr>
<td><code>pzLocationID</code></td>
<td>Location ID.</td>
</tr>
<tr>
<td><code>pzOperatorID</code></td>
<td>Operator ID.</td>
</tr>
<tr>
<td><code>pzOutcomeID</code></td>
<td>Outcome ID.</td>
</tr>
</tbody>
</table>

### Default Dimension Properties

The historical information captured in interaction results is based on dimensions. Dimension properties are designed to have a limited list of unique values. For this reason, a dimension like location should not contain geographical locations because that would generate an excessive amount of records to be inserted into the table.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Action</strong></td>
<td>The action dimension captures what was offered or presented to the customer. Default properties defined on <code>Data-Decision-IH-Dimension-Action</code>:</td>
</tr>
<tr>
<td></td>
<td>• <code>pyIssue</code>: the issue the proposition belongs to.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyGroup</code>: the group the proposition belongs to.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyName</code>: the name of the proposition.</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>The application dimension captures the decision path that issued the action. Default properties defined on <code>Data-Decision-IH-Dimension-Application</code>:</td>
</tr>
<tr>
<td></td>
<td>• <code>pyApplication</code>: the application that runs the interaction.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyInteraction</code>: the interaction rule.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyStrategy</code>: the strategy rule.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyComponent</code>: the strategy component.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyApplicationVersion</code>: the version of the application that runs the interaction.</td>
</tr>
<tr>
<td><strong>Channel</strong></td>
<td>The channel dimension captures the channel used in the interaction with the customer. Default properties defined on <code>Data-Decision-IH-Dimension-Channel</code>:</td>
</tr>
<tr>
<td></td>
<td>• <code>pyChannel</code>: the channel used in the interaction.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyDirection</code>: the direction used in the interaction.</td>
</tr>
<tr>
<td><strong>Context</strong></td>
<td>The context dimension captures the reason for the action. By default, no properties are defined on <code>Data-Decision-IH-Dimension-Context</code>.</td>
</tr>
<tr>
<td><strong>Operator</strong></td>
<td>The operator dimension captures information about who handled the interaction. Default properties defined on <code>Data-Decision-IH-Dimension-Operator</code>:</td>
</tr>
<tr>
<td></td>
<td>• <code>pyOrganization</code>: organization the operator belongs to.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyDivision</code>: the division.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyUnit</code>: the unit.</td>
</tr>
<tr>
<td></td>
<td>• <code>pyOperator</code>: the operator.</td>
</tr>
</tbody>
</table>
Outcome

The outcome dimension captures the result of the interaction. By default, there is one property defined on `Data-Decision-IH-Dimension-Outcome: pyOutcome`, which contains the result of the interaction.

Customer

The customer dimension captures the characteristics of the customer the proposition was offered to. By default, no properties are defined on `Data-Decision-IH-Dimension-Customer`.

Location

The location dimension captures the location of the customer when the interaction takes place. By default, no properties are defined on `Data-Decision-IH-Dimension-Location`.

Do not use this dimension to store geographic coordinates because this may result in inserting more rows than what the table is designed to contain. The location should be generic enough to contain a limited set of values. If you require geographic coordinates, or more detailed location information, use the `pyLongitude` and `pyLatitude` properties of the fact class, or extend the fact table to suit the need for capturing detailed location information.

Default Identity Matching Properties

The Interaction History association table provides the infrastructure to perform identity matching across channels and interactions. Default fact properties defined on the `Data-Decision-IH-Association` class:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyAssociatedID</td>
<td>Observed ID to associate with a primary ID represented by the pySubjectID property.</td>
</tr>
<tr>
<td>pyAssociationStrength</td>
<td>A numeric value that can be used to define the probability, relevance or certainty of identity matching records.</td>
</tr>
<tr>
<td>pxCreateDateTime</td>
<td>Time stamp corresponding to when the identity matching record was saved.</td>
</tr>
<tr>
<td>pySubjectID</td>
<td>Subject ID for this record.</td>
</tr>
</tbody>
</table>

Interaction History Extension

Interaction History’s (page 37) extension model is database driven. Case sensitive name matching is used between properties in the database table, SR properties and Interaction History properties. Typically, the same Interaction History is used by multiple applications using a shared implementation. You can extend Interaction History with new properties. Numeric properties in the Interaction History’s fact table can be used as key performance indicators (KPIs), in which case it is recommended that you use values that can be summed up so that the property is really usable in VBD as a KPI.

- Interaction History configuration (page 40)
- Extending Interaction History (page 41)
- Extra steps for KPIs (page 43)
- Excluding properties (page 43)

Interaction History Configuration

- Database schema (page 40)
- Properties (page 41)

Database Schema

By default, Interaction History tables are included in the PegaDATA schema. It is possible to use a dedicated schema for these tables. In this case, after setting up the database for Interaction History and creating the data source that allows VBD and PRPC to be able to operate with Interaction History data,
follow these steps to configure the database table data instances to use a database schema other than PegaDATA:

1. Create a database data instance pointing to the schema containing the Interaction History tables.
2. Use a JDBC connection pool to connect to the `jdbc/ihDataSource`.
3. In the Records Explorer, go to SysAdmin > Database Table.
4. To limit the results to the relevant database tables (page 37), filter by `Data-Decision-IH`.
5. Change the database setting of the database table data instances so that they use the connection configured in the first step.

Testing the connectivity to the database throws errors about the table containing internal properties. These errors can be safely ignored.

If using a different database vendor than the one supporting the PRPC database, make sure the correct JDBC drivers are available. For more details, refer to the configuration guidelines in Pega 7 Help, topic About Database data instances.

Properties

The `pyInteractionHistoryConfiguration` data transform defined on `Data-Decision-IH-Configuration` configures Interaction History by setting the properties in this class. In your application ruleset, you can override the data transform to customize Interaction History. The table below provides an overview the properties that you can set through the `pyInteractionHistoryConfiguration` data transform.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pyFactProperties</code></td>
<td>Value List (Text)</td>
<td>List of fact properties.</td>
</tr>
<tr>
<td><code>pyActionProperties</code></td>
<td>Value List (Text)</td>
<td>List of action dimension properties.</td>
</tr>
<tr>
<td><code>pyApplicationProperties</code></td>
<td>Value List (Text)</td>
<td>List of application dimension properties.</td>
</tr>
<tr>
<td><code>pyChannelProperties</code></td>
<td>Value List (Text)</td>
<td>List of channel dimension properties.</td>
</tr>
<tr>
<td><code>pyContextProperties</code></td>
<td>Value List (Text)</td>
<td>List of context dimension properties.</td>
</tr>
<tr>
<td><code>pyCustomerProperties</code></td>
<td>Value List (Text)</td>
<td>List of customer dimension properties.</td>
</tr>
<tr>
<td><code>pyLocationProperties</code></td>
<td>Value List (Text)</td>
<td>List of location dimension properties.</td>
</tr>
<tr>
<td><code>pyOperatorProperties</code></td>
<td>Value List (Text)</td>
<td>List of operator dimension properties.</td>
</tr>
<tr>
<td><code>pyOutcomeProperties</code></td>
<td>Value List (Text)</td>
<td>List of outcome dimension properties.</td>
</tr>
<tr>
<td><code>pyMeasurements</code></td>
<td>Value List (Text)</td>
<td>The subset of <code>pyFactProperties</code> that can be used as KPIs.</td>
</tr>
</tbody>
</table>

Setting outcome properties should be approached as a special case. Since this dimension drives interaction results, adaptive learning, and monitoring through the VBD planner, only `pyOutcome` is considered even if you explicitly set other properties in this dimension.

Extending Interaction History

The process of extending Interaction History consists of:

1. Adding columns to the appropriate database table.
2. Adding a property with the same name to the fact or dimension class in the application ruleset.
3. Adding a property with the same name to the application’s Strategy Result class.
4. Overriding the `pyInteractionHistoryConfiguration` data transform in the applicable application rule sets. When overriding the data transform, the newly added properties can be added to the corresponding value list properties.

The first step in extending Interaction History consists of deciding which table should be used. The recommendation is to add it to the fact table if the column is expected to have a large number of unique values. The fact record is also the table to add columns that represent properties to be used as KPIs. In any other case, you can consider adding it to one of the dimension tables. For example, add two columns so that you can extend Interaction History with the `ProductID` and `HandleTime` properties:

- **ProductID**: ID of the offered product.
  - Since the list of product IDs consists of a limited number of values, and it is part of a proposition, add the column to the action dimension table.
  - It is a text value, the column should be added as such. For example, `VARCHAR2(255)`.
- **HandleTime**: duration of the conversation.
  - A large number of distinct values is expected, and it is used in KPIs. Add the column to the fact table.
  - It is an integer value, the column should be added as such. For example, `NUMBER(18,0)`.

The next step is determining the applicability of the new extensions. Some properties are application specific, some shared in the same implementation and there may be security or privacy considerations that make certain properties only available in certain applications. In our example, the new `ProductID` column should be exposed in both applications, but `HandleTime` is applicable only to the call center application.

- To extend Interaction History with the `ProductID` property for all applications in the action dimension table:
  - Add a new property `ProductID` of type text to the `Data-Decision-IH-Dimension-Action` data model in the application rulesets of the call center application, as well as the outbound marketing application.
  - Add the `ProductID` property (same name and data type) to the Strategy Result class of both applications.
  - Override the `pyInteractionHistoryConfiguration` data transform in `Data-Decision-IH-Configuration` for both application rulesets.
    - Add a new set action, set the source to `Primary.pyActionProperties(<APPEND>)` and the target to "ProductID".
    - Enable the call superclass data transform setting to make sure the default configuration is included.
- To extend Interaction History with the `HandleTime` property for the call center application in the fact table:
  - Add a new property `HandleTime` of type integer to the `Data-Decision-IH-Fact` data model in the application ruleset of the call center application.
  - Add the `HandleTime` property (same name and data type) to the SR class of the call center application.
  - Override the `pyInteractionHistoryConfiguration` data transform in `Data-Decision-IH-Configuration` for the call center ruleset.
    - Add a new set action, set the source to `Primary.pyFactProperties(<APPEND>)` and the target to "HandleTime".
    - Add another set action, set the source to `Primary.pyMeasurements(<APPEND>)` and the target to "HandleTime". This step is necessary so that you can use it as a KPI.
    - Enable the call superclass data transform setting to make sure the default configuration is included.

If the new properties are not recognized, save the corresponding database table rules, log out, and log in again.
Extra Steps for KPIs

The Interaction History extension model allows you to make numeric properties ready to be used as KPIs. These extra steps apply so that you can use them as KPIs:

- Follow the KPI definition process.
- If you want the new KPI to be used in VBD's view of Interaction History data (actuals), run the `pzUpdateActualsConfiguration` activity under Data-Decision-VBD-Configuration to update the VBD server.

Excluding Properties

To prevent issues with loading or saving instances the class, if you add a property to the fact or dimension classes but this property is not to be considered part of Interaction History, configure the property's persistence settings so that it does not save property data.

Identity Matching

Customers interact with your business through multiple devices and multiple channels. For example, consider a single event where a customer reaches out to the call center to request a service upgrade. In the route to that single event, this customer could have visited the company's website using the work laptop, looked up more information using the personal mobile phone whilst commuting, and check reviews and service upgrade comparisons using the personal tablet at home. Identity matching consists of the ability to correlate data from anonymous interactions with data from interactions with an identified customer, improving interaction data quality, and providing the means to improve the single customer view.

Interaction History supports identity matching by allowing you to add or remove associations between records representing interaction data. You can combine identity matching in your activities by adding the Call Data-Decision-IH-Association.pySaveAssociation and Call Data-Decision-IH-Association.pyDeleteAssociation methods.
Adaptive Decision Manager (ADM) provides the capability to learn behavior in real time. Increasingly accurate decisions are made by automatically adapting models when behavior changes. For instance, if a customer accepts an offer, the likelihood for customers with a similar profile slightly increases. There are mathematical ways to express these probabilities and the way they adapt to change.

ADM is an integrated method establishing customer preferences without previously collected historical data. It extends predictive analytics with an adaptive mechanism for establishing customer preferences with customer responses in real time. In contrast with predictive analytics, which requires historic data and human resources to develop a reliable predictive model, ADM can calculate behavior without historical data. ADM captures and analyzes data to deliver predictions where no history is available to develop offline models and when behavior is volatile. Besides keeping count of the number of times a specific outcome is observed, ADM can take into account predictive data to predict behavior. If data and time are available for offline modeling, predictive models can be used as an alternative or in conjunction with adaptive models.

Adaptive models become more accurate with time, requiring monitoring not to become less sensitive after a sustained period of use. The advantage of using ADM is considerable in business areas where mistakes are not critical, such as marketing.

Related Information

- Adaptive Modeling (page 44)
- Predicting Behavior (page 45)
- Model Learning (page 45)
- Fail Safe (page 49)

Adaptive Modeling

Adaptive Decision Manager is part of the Decision Management service layer PRPC connects to. It is fully integrated to work together with predictive models that address more critical issues (for example, detecting complex fraud patterns) and other strategy components. As of PRPC 7.1.7, ADM can run as an internal process, allowing you to add adaptive analytics capabilities to your application without requiring the external ADM server. The internal process does not persist models if the system is restarted, and it is designed to facilitate the application development process in non production systems, not as a replacement for ADM in the service layer.

ADM generates adaptive models when a strategy containing adaptive model components is executed. Factories are the entities responsible for actually generating the models and keeping track of counts and statistics. There is a one to one relationship between factories and models (one factory per model). The way a model is generated depends on the configuration of the adaptive model rule referenced by the adaptive model strategy component. An adaptive model rule is typically shared by multiple models, and therefore also factories.

When models are generated, ADM starts capturing the data (counts, and statistics) relevant to the modeling process, maintaining statistics with high granularity. This data forms the backbone for the creation of adaptive MDAP models that are used to assess propensities. Without any data, the scoring models are empty and only track overall propensity. The prioritization scheme ensures all propositions are considered but focus on the observed best propensity proposition, thus ensuring early data collection for all propositions while maximizing interaction results. Interaction results are processed by the adaptive analytics engine and stored in a set of adaptive statistics from which the engine continuously creates new scoring models. Statistics and models are stored in the adaptive data store. Scoring models drive the decision process and statistics ensure persistence. Once a data set has been captured, new scoring models are created. In this second stage, the data is used to identify propositions with higher or lower average propensity.
The adaptive modeling cycle is very similar to the predictive analytics process in Predictive Analytics Director. However, due to ADM's analytical nature, no preset intervals or groups need to be identified beforehand and extensive selection of predictors does not need to take place. The full adaptive modeling cycle consists of:

1. Capturing historical data with fine granularity.
2. Regularly:
   • Using sophisticated auto-grouping to create coarse-grained, statistically reliable numeric intervals or sets of symbols.
   • Using predictor grouping (page 77) to assess inter-correlations in the data.
   • Selecting predictors to establish an uncorrelated view that contains all aspects relevant to the proposition.
3. Using the resulting statistically robust adaptive scoring model for scoring.
4. Whenever new data is available, updating the scoring model.

New models are published automatically the first time a strategy containing adaptive models is executed, when (for existing scoring models) the Memory setting of the corresponding adaptive model rule is changed, right after data analysis, or by recalculating the predictor binning. Any other change in adaptive model settings results in changing the scoring model and therefore overriding the previous settings.

Model changes are not tracked in the ADM server. At any point in time, there is only one version of a particular scoring model.

Predicting Behavior

ADM employs a Multidimensional Analytical Profiler (MDAP) as its main method of predicting behavior. The technique maintains a set of sufficient statistics in order to create models particularly intended to function in collaboration with predictive models and other strategy components. However it does not require either. To increase the scope and reliability of this basic technique, the following is applied:

• Sophisticated auto-grouping.
• Correlation detection and feature selection.
• Adaptive prioritization for selecting a proposition in the presence of increasing reliability.
• An integration and warning system to signal the opportunity to analyze and fix the data collection in a robust and non-linear model.

Model Learning

Adaptive models are executed in Process Commander. Adaptive Decision Manager performs data analysis depending on the run data analysis after setting of the adaptive model instance and the model update frequency set in the UpdateAdaptiveModels agent (page 14). The combination of these settings guard the speed at which newly learned information is seen in Process Commander. An alternative learning method (local learning in PRPC) can be used when learning based on the settings that trigger data analysis is not producing models that output useful predictions.

• About model learning (page 45)
• Model learning in the ADM system (page 47)
• Local learning (page 48)
• Adaptive model settings (page 48)

About Model Learning

Adaptive models learn based on predictors and behavior:

• You can define predictors (page 77) based on properties in the data model or adaptive model parameters. If an adaptive model does not have explicitly defined predictors, ADM dynamically adjusts to keep all information encountered within its internal data representation. Which data inputs you should use to define predictors depends on the potential predictive performance (page 77). Predictors should be monitored through the behavior profile that analyzes the most important predictive fields detected by the adaptive analytics engine (page 75) as it learns.
• Behavior is defined according to the possible outcome values to associate with positive or negative behavior. The values defined for positive and negative outcome should coincide with the outcome definition as configured in the interaction rule running the strategy containing the adaptive models configured by the adaptive model rule.

The capability of using predictors depends on the ADM configuration in the services landing page. If the corresponding setting is disabled, you can define and use adaptive models in your application, but these models do not operate based on predictive data.

Previous to Pega 7.1, adaptive model learning was based on the Interaction Services behavior dimension. Starting with Pega 7.1, this dimension is replaced by the outcome dimension in Interaction History. The behavior dimension could be defined by the behavior level (for example, Positive) or combination of behavior and response (for example, Positive-Accepted). Adaptive models upgraded to Pega 7.1 preserve the value corresponding to the response level in the behavior dimension (for example, Accepted), but not the value corresponding to the behavior level.

Without using parameters, your adaptive model can only learn from properties in the primary page. The calculation is typically made by the strategy and the result passed as parameters in the process of executing the adaptive model. Adaptive model parameters allow you to define predictors based on:

- Output of other components and strategy calculated values (page 46)
- Interaction History (page 47)

### Component and Strategy Results

Adaptive models can learn from the result of other strategy components. This allows you to combine adaptive and predictive analytics by defining adaptive models that learn based on other models (scorecards or predictive models).

This example illustrates modeling risk based on an adaptive component that, besides learning from the predictors defined in the Predict Risk adaptive model rule, it also uses the scorecard’s score result for learning.

• The adaptive model is configured to learn from credit history. So that it can learn from the scorecard result, the parameter RiskScore of decimal data type has been added in the parameters tab and included as a predictor.
• In the strategy, the Predict Risk adaptive model component allows you to include the output of the scorecard by using the parameters section where you can map the RiskScore parameter to the output of the Defection Risk scorecard component.

![Adaptive Model Properties](image)

The same principle can be applied to modeling proposition based values that are calculated by the strategy.

**Interaction History**

Using the same principle as described in the previous section, you can use historical data to model propositions. For example, enable learning based on the consecutive customer response to a sales proposition.

• In the adaptive model, add two parameters to represent the customer response to the proposition (text) and elapsed time between the last response and the current interaction (date time).
• In the strategy:
  • Add a proposition data component that imports the sales propositions and corresponding interaction history.
  • Add a set property component to map a property representing the last response to the pyOutcome Interaction History property and calculate the time elapsed since then using the pyOutcomeTime and current date time.
  • Connect the proposition data to the set property component.
  • Add an adaptive model component that references the instance you created in the first step.
  • Set the two parameters based on properties you defined in the set property component.

**Model Learning Explained**

The run data analysis after setting defines the number of new responses that, when reached, trigger data analysis. There is a general system setting for running data analysis, which is 50. Data analysis is a processing intensive operation. For this reason, an additional parameter can be configured to control model refresh, a light weight analysis process where predictor binning is recalculated without changing the predictor grouping. The setting that controls model refresh is the refresh after setting. If the values of both settings are the same, the light weight analysis process is never triggered.

When the model update agent runs, the current number of responses processed since running the last data analysis count or model refresh is considered in order to compare to data analysis and model refresh rates. ADM runs data analysis in the following circumstances:
• If no initial data analysis has been done and the number of responses is above the general system setting. Initial models are created in three stages:
  • If the number of responses is below the initial data analysis count, a model with a propensity of 0.5 is created.
  • If the number of responses is above the initial data analysis count for the first time, a model with a base propensity (number of positive responses divided by the sum of positive and negative responses) is created. Additionally, grouped predictors are created to allow gathering responses for behavior profile purposes.
  • If the number of responses is reached after the previous stage for the first time, the first model with grouped predictors and outcome profile is returned.
• If the difference between the number of responses at which the model was last created and the number of responses stored since then is more than the number of responses triggering data analysis. If the difference between the number of responses stored and the delta obtained in this context is more than the number of responses triggering model refresh, the model refresh mechanism is triggered.

Local Learning
Local learning can be enabled when the number of responses is not sufficient to evolve the model. Local learning is enabled through the enable local updates setting in the adaptive model instance. It consists of configuring models to adapt with every response. This feature allows learning to take place when model update takes too long for the model to be considered useful, but it is not designed as a replacement of learning in the ADM system (page 47) since models produced through learning in the ADM system are superior in predictive quality than models produced through local learning.

Local learning always remains local even in an environment running multiple system nodes.

Adaptive Model Settings
Adaptive model settings configure how ADM operates by controlling the runtime throughput of ADM and the creation and update of the individual scoring models. The settings should be configured to appropriate values to prevent high loads on the database. Settings used in adaptive model operations are grouped by category:
• Responsiveness (page 48)
• Data analysis (page 48)
• Advanced configuration (page 49)

Responsiveness
The responsiveness setting specifies the amount of interaction results, which are translated in number of cases (page 75), the scoring models maintain during predictions (page 77). By default, it is set to never discard information (0). The memory configuration allows you to discard the oldest cases and it is used to implement trend detection (page 78) by creating multiple adaptive models, all triggered by the same proposition (page 78), but with different memory settings. This setting influences the binning of predictors as the behavior changes when new cases are recorded.
• Low memory values allow for the identification of new trends.
• High memory values provide robust and long-term predictive power (page 77).

Data Analysis
The data analysis settings influence data analysis. When changing these settings, you should look at the reports for every model that is created based on these settings.
• Run data analysis after: determines the number of interaction results that trigger running data analysis. Data analysis is triggered after the number of interaction results configured in this setting is reached. This setting should be configured according to the resources available to the ADM system,
and taking into account the minimum set of responses required for models to evolve. Default setting is 500.

• The grouping settings work in conjunction to control how predictor grouping is established. The fact that a predictor has more groups typically increases the performance, but the model may become less robust.
  • *Grouping granularity*: a value between 0 and 1 that determines the granularity of predictor groups; higher values result in more groups, lower values in less groups. This setting establishes the threshold for a statistical measure that indicates the distinctive behavior between predictors groups. If the measure is above, the groups indicate significant distinctive behavior, otherwise they will be collapsed. Default setting is 0.25.
  • *Grouping minimum cases*: a value between 0 and 1 that determines the minimum percentage of cases per interval. Higher values result in decreasing the number of groups, which may be used to increase the robustness of the model. Lower values result in increasing the number of groups, which may be used to increase the performance of the model. Default setting is 0.05.

• *Performance threshold*: a value between 0 and 1 that determines the threshold for excluding poorly performing predictors. This setting allows you to control the reliability of predictors. Lower values can result in models using weak predictors since predictors with a CoC (page 75) lower than 51.00 are typically considered weak. Default setting is 0.52.
• *Correlation threshold*: a value between 0 and 1 that determines the threshold for excluding correlated predictors. Predictors are considered correlated when there is a predictive relationship in the behavior they predict. Lower values can result in models using duplicate predictors. Default setting is 0.8.

Advanced Configuration

The advanced configuration settings control other operations performed in the ADM database.

• *Performance memory*: determines the number of cases of moving window size per proposition. The number of cases of moving window size per proposition influences the calculation of the CoC (page 75) and it is implemented to perform equal comparison between models. Default setting is 0.
• *Refresh after*: determines the number of interaction results that trigger refreshing the scoring models in the ADM database. Model refresh is performed when the number of interaction results in this setting is reached. The value should be lower than the one for running data analysis (page 48), and realistic in the context of the minimum set of responses required for models to evolve. Default setting is 150.
• *Enable local updates*: enable local (PRPC) learning for the models that are created based on these settings. By default, local learning is enabled.
• Check the audit notes option if you want adaptive model execution details captured in the work item’s history. By default, audit notes are disabled.

Fail Safe

Adaptive Decision Manager can operate in fail safe mode.

• *Fail safe mode* (page 49)
• *Known limitations* (page 50)

Fail Safe Mode

The fail safe operational mode prevents data loss in case of ADM unavailability. This mode is configured by enabling the fail safe mechanism. The pxEnableADMFailSafe property reflects the configuration of this setting. Changes to the pxEnableADMFailSafe property are communicated every time the UpdateAdaptiveModels agent (page 14) runs. In fail safe mode:

• A staging table is used to store new models in the context of executing strategies, as well as new response records in the context of capturing responses.
• In order to prevent handling records in the staging table that were already processed but are still stored in the staging table, the time stamp corresponding to the last received response is updated every time a response record is sent to ADM.
• It is still possible to create new models, perform scoring and handle responses if the ADM service is unavailable.
• Records in the staging table older than the last received response are removed every time the ADM system is started. If records remain in the staging table, they are handled by time stamp (old to recent) and a new backup takes place. When the backup process is complete, the records before the last backup are deleted and the last received response is updated accordingly.

Limitations

• The solution prevents data loss in the sense that responses that were sent between the moment the ADM system goes down and the moment it starts again are restored from the staging table at the moment the system is active again. However, responses that were not processed by the ADM system for any other reason (for example, because of networking issues) are not recovered and they will be deleted from the staging table the next time ADM performs a backup.

  This limitation may not be applicable to all JMS providers. It applies when not using JMS (Tomcat).
• Data loss can still happen for responses sent immediately after reading the staging table but before the client was able to connect and actually set the response in ADM.
• The solution does not include any monitoring or notification mechanism. Except for errors or warnings in the log files, there is no indication in case the ADM system stops running.
• Staging table limitations:
  • There is no limitation on table size, and no warning or monitoring mechanism to prevent the table from encountering size issues.
  • It is assumed the staging table is always online and available.
Decision Manager allows business users to make controlled changes in the business sandbox and perform monitoring in production within the boundaries defined by IT. The access group configuration determines what users can do in the Decision Manager portal. Business users work in the Decision Manager portal and system architects in Designer Studio. System architects import revisions, manage the state of revisions, define user access to propositions in a proposition group and configure application overlays. The state of a revision determines which versions of the application’s resources and artifacts are active in the system.

Although Decision Manager functionality can be available in different systems, the Decision Manager portal itself is typically used in the business sandbox and in production. Typically, each system has a different view of the data stores that support adaptive learning, interactions and business monitoring (online data stores in the production cycle, offline data stores in the remaining cycles). For this reason, unless data is replicated, the actual monitoring activities take place in the production system, and offline analysis in the business sandbox.

Related Information
- User Activity Boundaries (page 51)
- Business Sandbox Cycle (page 52)
- Revision Activation Cycle (page 54)
- Production Cycle (page 54)

User Activity Boundaries

In Designer Studio, system architects define the boundaries for the business user activity in the business sandbox and in production through the Decision Manager portal. They are responsible for creating and managing application overlays, defining the exposure of propositions to Decision Manager portal users and maintaining access group user membership. Starting with Pega 7.1.8, the exposure of propositions to Decision Manager portal users is determined by the decision data instances included in an application overlay.
Business Sandbox Cycle

In the Decision Manager portal, revision managers and strategy designers work in the application overlay to achieve the revision objectives.

Revision managers address production issues reflected in the business sandbox, and define the revisions and change requests to resolve these issues. Change requests take the form of assignments strategy designers need to complete so that the revision can be processed by revision managers. To support the case in which the original set of rules is not sufficient to achieve the goals of the change request, strategy designers can send back change requests so that revision managers can review and alter the rules included in the change request.

Strategy designers perform the work necessary to achieve the objectives of the revision by addressing the goals of the change request assigned to them. They make the necessary changes, test applications, run simulations and analyze the impact of the changes, a cycle that is repeated until the objectives are met. They make the changes visible to revision managers by submitting the change request. If the strategy designer rejects the change request, the changes are rolled back to the version before the definition of the revision.
Revision managers process the changes submitted in the context of the change request. They can run applications and simulations to validate the change request, and then accept or reject the changes. Once all change requests are completed (accepted or rejected), revision managers process the revision. The flow followed by the revision manager is determined by the type of application overlay:

- In revision management overlays, revision managers submit the revision to package the changes in the form of a JAR file, which can then be sent to system architects for deployment in another system.
- In direct deployment overlays, revision managers activate changes to decision data instances. They can also roll-back direct deployment revisions.
Revision Activation Cycle

System architects import the revisions handed over to them by revision managers. The revision import process allows for reflecting in other systems the changes made and tested in the business sandbox. In this process, a step for validation is introduced so that test users can validate the impact of the changes in the end user application. After validation, system architects manage the state of the revision by making the revision available to all users, or by discarding the changes.

Production Cycle

Once system architects activate the revision for all users in the production environment, production supervisors monitor the performance and accuracy of interactions through the Decision Manager portal.

Production administrators can perform the same monitoring activities as supervisors but they go beyond a supervisor’s observer activities. They can monitor adaptive models, update the data used in adaptive models monitoring and reporting, perform on adaptive models, update propositions by updating the corresponding decision data instances, and run applications.
Strategies (page 78) define the decision that is delivered to an application. The decision is personalized and managed by the strategy to reflect the interest, risk and eligibility of an individual customer in the context of the current business priorities and objectives. There are two typical use patterns when defining a strategy:

- Strategies using propositions: the level at which the strategy is created (top level, business issue or group) determines the properties it can access. Strategies for which business issue is not defined apply to all business issues and groups in the context of the proposition hierarchy.
- Strategies without propositions: strategies are creating on a data class that indirectly derived from `Data-pxStrategyResult`. If left empty, the strategy result class is automatically considered to be the top level class of your application.

**Related Information**
- Strategy Design Patterns (page 56)
- Strategy Design (page 62)
- Strategy Properties (page 68)
- Strategy Execution (page 68)

**Strategy Design Patterns**

Typically, you design a strategy (page 78) to deliver a personalized recommendation for a single decision (page 76). For example, a strategy to recommend the most important issue to be dealt with for a particular customer, via a channel or system, and at a given point in time. Combined with the current objectives and priorities of the company, predicted risks and interests are part of the strategy.

A recommendation can be part of a sequence. After determining the most important issue to address, the decision chain may need to address which credit strategy to use, which retention strategy or which product to offer first. Every decision employs a combination of strategy components that define the underlying logic required to deliver a recommendation. Components allow you to create personalized customer interactions consistently across contact channels. The advantage of building decision strategies from these smaller components is that each one can be readily understood, developed, edited and tracked on an ongoing basis. You can use components to model sophisticated customer behavior and there are some common design patterns that you end up reusing frequently.

In the context of using strategies in combination with propositions, you create a strategy to deliver the decision for one business issue or group. The scope in the proposition hierarchy corresponds to the issue or group level of the action dimension. The level at which you create the strategy determines the properties it can access, and these are the properties that define the output structure of components in the strategy.

You can develop decision strategies as a self-contained single strategy, or multiple strategies combined using sub strategy components. Combining strategies allows for concurrent development of large scale strategies by creating smaller strategies that can be developed in a relatively independent manner. The other use case of multiple strategies is reusing a logical pattern.

**Related Information**
- Next Best Action (page 57)
- Capture Results (page 57)
- Segmentation (page 58)
- Reusable Logic (page 59)
- Adaptive Components (page 59)
- Trend Detection (page 60)
Next Best Action

The standard approach for finding the Next Best Action (page 76) for each customer consists of segmenting customers, assessing the propensity, selecting the action for each customer segment and, finally, selecting the best decision path. The following list describes a sequence that can be used as a starting point when planning your strategy.

1. Plan the final decision and work backwards.
   - Starting point that allows you to define the strategy plan(s), such as the most important issue to address, what drives the decision, the most appropriate proposition (and how to determine it), the probability factors, characteristics and preferences to take into account in the decision.
   - What do you want to deliver?
   - What action to take in order to achieve this?
   - What data is required?

2. Build from customer, product, environment and other required information to deliver the decision.
   - Define propositions.
   - Import propositions.
   - Prioritize between propositions.
   - Balance issues.
   - Finalize decision.

The visual orientation of the strategy is a logical translation of the output orientation working backwards from the Next Best Action end point. Structurally, this can be explained by using a top-down tree model. For example, assume that you need to build a strategy that addresses the following:

- A number of segmentation components are available that classify customers based on product and risk of customer attrition. Different issues need to be addressed, such as sales, recruitment and retention.
- Arbitration between the different propositions is done with a Next Best Offer (page 76) prioritization:
  - In the sales context, the offer that has the highest cross sell score.
  - In the risk of customer attrition context, the offer that addresses cases falling in segments with the highest customer attrition risk.
- Depending on the issue to be addressed, a final recommendation needs to be issued.

The diagram below visualizes the concepts used when planning the strategy that provides the decision. A strategy implementing the logical structure abstraction is the final result. The fundamental NBA pattern starts from the final decision point and has a right-to-left orientation, but the flow of the arrows starts with import components (page 63), then segmentation components (page 63) for which possible actions are defined, next the data enrichment components (page 64), proceeding with arbitration components (page 65) and, finally, the end selection component (page 65) that delivers the best action in the interaction.

Capture Results

Starting with Pega 7.1, the information necessary to capture the interaction results is delivered through the strategy, not the interaction rule.
The minimal design of a strategy for capturing interaction results consists of a data import component that provides the page containing the results of running the strategy that delivers the decision and provides the necessary information for writing the interaction results to the Interaction History tables.

### Data Import Properties

- **Name**: Import Strategy Results
- **Component ID**: ImportStrategyResults
- **Description**: Use generated or Use custom
- **Source**: Properties mapping or Auto-run

#### Define page

- **Page**: OfferedProposition
- **Class**: Data-pxStrategyResult

### Segmentation

Split components provide strategies with a mechanism to derive segments from results created by decision analytics and business rules components. Combined with the exclusion component, you can create powerful segmentation trees that assign results based on a percentage of cases, and then conditionally exclude results based on relevant conditions.

![Segmentation Diagram](image-url)
Reusable Logic

Through the **external inputs setting** (page 67), you design strategies as centralized pieces of logic that can be used by more strategies. The strategy referred to by the sub strategy component has the external input option switched on. This external input connects to the starting components that define the reusable chain of components.

![Diagram of external input and reusable logic](image)

In another strategy, the sub strategy component refers to the reusable strategy and it is driven by other components. When you run this strategy, the sub strategy component effectively results in replacing the component with the chain of components that are propagated by the sub strategy.

**Adaptive Components**

Strategies can introduce adaptive models to model customer responses for a set of propositions. The strategy can contain a mix of predictive models, adaptive models and prioritization components. Prioritization components can be employed to offer the customer the best action based on predicted propensity (page 77) and data used in the prediction (page 77). Predictive models could be used to predict customer attrition, fraud and customer lifetime value.

The example described in this topic illustrates the use of adaptive learning in a product offers strategy.

- The strategy imports the product offers, proceeds with defining eligibility and setting the channel dimension. Adaptive learning is introduced to learn from customer responses by adding an adaptive component. This model sets evidence and propensity through its output mapping configuration.
Eligible product offers are modeled by the adaptive model component, working on the basis of smoothing propensities to calculate the acceptance probability. Finally, the strategy prioritizes the top three offers based on multiplying marketing weight by the base propensity.

Trend Detection

To design a strategy for trend detection (page 78), you need to apply a specific design pattern. This pattern consists of adding a group by component that selects the adaptive models in the process of issuing the decision. For example, in a strategy containing three adaptive models, you can add a prioritization component to arbitrate which adaptive model selection should be selected based on performance. The performance output field is typically used to dynamically select between multiple adaptive models and/or predictive models. In the example below, adaptive model components in the strategy use adaptive model rules differentiated on the basis of performance window size. When the characteristics of customers change, the fast model (1000 window size) detects the change in outcome fastest and has the higher performance; this model is used to decide on the predicted propensity. When the other models (500 window size and 0 window size) start to capture this change in behavior and earlier behavior has been discarded, they are again selected because they can make more accurate predictions as they use more data. Positive and negative cases can be used to calculate the expected or base level propensity and, together with the propensity output field, calculate the lift (page 76) of individual predictions.
The models are selected by a group by component that groups the output by the applicable level(s) in the proposition hierarchy. The adaptive components model all propositions in the same business issue (Sales), so the group by component needs to group by the additional levels in the hierarchy (group and name). Additionally, it also needs to set how to propagate data; in the example below, propagation of data is handled by copying the first value.

### Group By Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Group Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component ID</td>
<td>GroupBy</td>
</tr>
<tr>
<td>Description</td>
<td>Use generated</td>
</tr>
</tbody>
</table>

**Source Components**

- **Properties**
  - Group output rows by
    - Add Item
    - Delete
    - .pyGroup
    - .pyName

- **Aggregators**
  - Add Item
  - Delete
  - Property: Performance to AVERAGE of .pyPropensity

**For remaining properties**

- Select: First

The selection between adaptive models is performed by a prioritization component that selects the highest performing model.
Strategy Design

A strategy is defined by the relationships of the components that are used in the interaction (page 76) that delivers the decision (page 76).

Related Information

- Sub Strategies (page 62)
- Data Import (page 63)
- Segmentation (page 63)
- Aggregation (page 64)
- Arbitration (page 65)
- Decision Path Selection (page 65)
- Pages & Alternative Pages (page 66)
- Expressions (page 66)
- External Input (page 67)
- Strategy Results (page 67)

Sub Strategies

A strategy (page 78) can use other strategies through sub strategy components. Including strategies allows for using specialized group or issue level strategies that address a specific business case, and combining them in a more generic strategy that is typically at the top level class in the proposition hierarchy. The strategy design pattern used when including sub strategies can be seen as always including more specialized cases to address all issues in an NBA strategy. Sub strategy define the way two strategies are related to each other, access the components in the strategy they refer to, and determine how to run the strategy if it is in another class. A sub strategy component can represent a reusable piece of logic provided the strategy it refers to has the external input (page 67) option enabled, and the sub strategy component itself is driven by other components. A sub strategy can run on the current page, in which case the imported strategy runs on the class the strategy belong to, or another page, in which case the imported strategy runs on a defined page; if the defined page is a page group.
or list, the decision making process is iterated over as many times as defined in the page group or list (for example, if a strategy runs through a sub strategy component over a list containing two customers, and assuming the strategy outputs three offers, the sub strategy component results in a list containing six offers).

**Data Import**

Components in this category bring data into the current strategy.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data import</td>
<td>Data import components import data from pages available to the strategy. Data import components that refer to named or embedded pages can map the page's single value properties to strategy properties. Data import components defined in releases previous to Pega 7.1 were subject to auto-mapping. That is still the case, but the mapping by matching name between target and source is implicitly done when the strategy is executed. You only have to explicitly map properties if exact name matching can not be applied or you want to override the implicit target/source mapping. If using named pages, these pages have to be included through the strategy's pages and classes.</td>
</tr>
<tr>
<td>Interaction history</td>
<td>Interaction history components import the results stored in Interaction History for a subject ID and associated IDs. If you do not define any conditions or specify properties, the component retrieves all results for the subject ID and associated IDs. Defining criteria reduces the amount of information brought to the strategy by this component. Some properties are always retrieved by the interaction history component (for example, subject ID, associated ID, fact ID and proposition identifier). With the introduction of the functionality supporting identity matching in Pega 7.1.8, you should be aware that this component can become ambiguous if used to build the interaction result because it will not only import the records for the subject ID, but also the ones for any associated ID. Database limitations related to data type changes apply if you are filtering on days. This setting is not suitable if you are working with dates earlier than January 1, 1970.</td>
</tr>
<tr>
<td>Proposition data</td>
<td>Proposition data components import propositions defined in the proposition hierarchy. The configuration of proposition data components is directly related to the level of the strategy in terms of the proposition hierarchy (business issue and group). Proposition data components allow you to also include results stored in Interaction History as specified in the conditions and properties settings; the settings defined for including recorded interaction results are similar to the interaction history component but, unlike the interaction history component, the component only retrieves results for the subject ID if you define which properties to use.</td>
</tr>
</tbody>
</table>

**Segmentation**

Components in the business rules and decision analytics categories use data to segment cases based on characteristics and predicted behavior and place each case in a segment (page 78) or score (page 78). In terms of component execution, components that you define to run on the strategy applies to class are evaluated one time on the primary page of the current strategy, whilst components that you define to run on the strategy result class are evaluated on every incoming step page. Adaptive models, decision tables, decision trees and map values allow for defining parameters. Through segment filtering connections, you can create segmentation trees (for example, considering a scorecard that outputs an accept and a reject result, you start by defining a strategy path for cases falling in the accept segment and another one for cases falling in the reject segment).

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predictive model</td>
<td>Predictive model components reference predictive model rules.</td>
</tr>
<tr>
<td>Adaptive model</td>
<td>Adaptive model components reference adaptive model instances. Adaptive models depend on the action, channel and outcome dimensions; the latter is configured in the adaptive model rule. Since the scope in the proposition hierarchy is propagated through</td>
</tr>
</tbody>
</table>
**proposition data components, if proposition data components connect to the adaptive model component, this information is implicitly configured.**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scorecard</td>
<td>Scorecard components reference scorecard rules.</td>
</tr>
<tr>
<td>Decision table</td>
<td>Decision table components reference decision tables used to implement characteristic based segmentation by referencing a decision table that takes into account customer data to segment on a given trait (for example, salary, age and mortgage).</td>
</tr>
<tr>
<td>Decision tree</td>
<td>Decision tree components reference decision trees. Decision trees can often be used for the same purpose as decision tables.</td>
</tr>
<tr>
<td>Map value</td>
<td>Map value components reference map values that use a multidimensional table to derive a result. For example, a matrix that allocates customers to a segment based on age and salary.</td>
</tr>
<tr>
<td>Split</td>
<td>Split components branch the decision results according to the percentage of cases the result should cover. These components are typically used to build traditional segmentation trees in strategies, allowing you to derive segments based on the standard segments defined by the results of other segmentation components. You define the result ( pxSegment ) and the percentage of cases to assign to that result.</td>
</tr>
</tbody>
</table>

### Data Enrichment

Components in this category add information and value to strategies.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set property</td>
<td>Set property components enrich data by adding information to other components, allowing you to define personalized data to be delivered when issuing a decision. Personalized data often depends on segmentation components (page 63) and includes definitions used in the process of creating and controlling a personalized interaction. Set property components created with releases previous to PRPC 6.2 SP2 DSM Edition supported overriding property values in the strategy through the <strong>Overrides</strong> tab, but this functionality has been removed since then. Strategy components defined in this way will show the configuration in the <strong>Overrides</strong> tab if there was a previous configuration, but no changes can be made.</td>
</tr>
<tr>
<td>Data join</td>
<td>Data join components import data in an embedded page, named page or strategy component, and map strategy properties to properties from the page or component. This type of component can be used to join lists of values; for example, a data join component that has one or more components as source and uses the results of another strategy component to define the join conditions. The criteria for matching data is defined as one or more value pairs between properties in the data join component and properties in the page or strategy component. You can determine the amount of data to include by excluding source components that do not match the conditions, in which case the data join is configured to perform an inner join operation. Data join components defined in releases previous to PRPC 7.1 were subject to auto-mapping. This is no longer the case: you have to explicitly map the properties required for strategy execution purposes.</td>
</tr>
<tr>
<td>Decision data</td>
<td>Decision data components import the data defined in decision data instances into the strategy. Conditions allow you to match properties brought by the decision data instance and properties defined by the component, conditions that can be provided by a property or an expression.</td>
</tr>
</tbody>
</table>

### Aggregation

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group by</td>
<td>Group by components set strategy properties using an aggregation method applied to properties from the source components. The properties that can be used to group the results are strategy properties; that is, properties of <strong>Data-pxStrategyResult</strong>, and properties available to the strategy depending on its applicability in the context of the proposition hierarchy. The aggregators themselves are defined based on the relationship between a property and its source, where the property is a strategy property and the...</td>
</tr>
</tbody>
</table>
source is selected among SR default properties or properties available to the strategy depending on its applicability in the context of the proposition hierarchy. Properties that are not mapped in the component are automatically copied but you can change this by defining how to handle the remaining properties.

**Iteration**

Iteration components perform cumulative calculations based on the settings defined in the parameters tab. Iteration components operate in two modes:

- Without source components, you can define the properties, number of iterations and early stop conditions. The order of the properties is taken into account when performing the calculation. Depending on the setting used to control how to return the results, the component returns only the final calculation, or final calculation and intermediate results.
- With source components, the number of iterations equals the number of results in the source component. The result of running the iteration component contains the final calculation and no intermediate results. If the value of the arguments is set through source components, the order of the components in the source tab is important because it determines the order of arguments considered to perform the calculation.

**Financial calculation**

Financial calculation components perform financial calculations using the following functions:

- Internal rate of return calculates the internal rate of return for a series of cash flows.
- Modified internal rate of return calculates the modified internal rate of return for a series of periodic cash flows.
- Net present value calculates the net present value of an investment.

The arguments can be provided by strategy properties of type decimal, double or integer. If the value of the arguments is set through source components, the order of the source components is important because it is directly related to the order of arguments considered by the function to perform the financial calculation. Typically, the `Payments` argument should be a list of values and not a single value. So that you can use a list of values to provide the `Payments` argument, use a data import component to set properties that can be used by this component.

**Arbitration**

Components in this category filter, rank or sort the information from the source components. Enriched data representing equivalent alternatives is typically selected by prioritization components.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>Filter components apply a filter condition to the outputs of the source components.</td>
</tr>
<tr>
<td>Prioritize</td>
<td>Prioritization components rank the components that connect to it based on the value of a strategy property or an expression. These components can be used to determine the service/product offer predicted to have the highest level of interest or profit. Two modes can be used to order the results (by priority or alphabetically) and each mode toggles its own specific settings.</td>
</tr>
</tbody>
</table>

**Decision Path Selection**

Strategies are balanced to determine the most important issue when interacting with a customer. The first step in applying this pattern is adding prioritization components (page 65) to filter the possible alternatives (for example, determining the most interesting proposition for a given customer). The second step is to balance company objectives by defining the conditions when one strategy should take precedence over another. This optimization can be accomplished by a champion challenger or a switch component that selects the decision path.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Champion challenger</td>
<td>Champion challenger components randomly allocate customers between two or more alternative components, thus allowing for testing the effectiveness of various alternatives. For example, you can specify that 70% of customers are offered product X and 30% are offered product Y. All alternative decision paths need to add up to 100%.</td>
</tr>
</tbody>
</table>
Switch components apply conditions to select between components. These components are typically used to select different issues (such as, interest or risk) or they select a component based on customer characteristics or the current situation.

Exclusion components conditionally stop the propagation of results by restricting the selection to results that do not meet the exclusion criteria. These components are typically used to build traditional segmentation trees in strategies. The criteria for excluding results is defined as one or more conditions expressed as value pairs between properties in the exclude component and, depending on the type of data you selected, properties in the page or strategy component. If you do not define any condition, the component stops the propagation of the results of its source components.

Pages & Alternative Pages

You can supply data to components that reference an external rule instance. This is particularly useful if you want to drive the component results using customer data. This capability requires a specific set up for the referenced instances and the strategy referencing them:

- In the referenced rule instance, the data is included in the rule instance's pages and classes. For example, a predictive model defines a page named *Purchase*, which is mapped to the *Purchase* data class.
- Pages from the referenced rule instance's pages and classes are listed under *Available pages & classes* in the strategy component. If you enable the supply with data setting, data passed by the page is used to evaluate and execute the component.
- It is also possible to provide an alternative page. If the alternative page data is not available, the component falls back to the originally set page (in the example above, *Purchase*).

Expressions

Working with strategies means working with the strategy result data classes and the class the strategy belongs to. These classes can be combined in expressions or by introducing segmentation components (page 63) that work on the strategy result data class, and not the class the strategy belongs to.

- Understanding the expression context (page 66)
- Using component properties in expressions (page 66)

Understanding the Expression Context

Using the dot notation in the SmartPrompt accesses the context of an expression, which is always the strategy result class (for example, `.pyPropensity`). To use properties of the Applies To context, declare the class label without any spaces (for example, `Customer.Price`) or use the primary page (for example, `Primary.Price`). If the properties used in expressions are single page properties, you can omit the class label or the *Primary* keyword (for example, instead of `Customer.SelectedPropositon.pyName`, use `SelectedPropositon.pyName`). Pages of the Applies To context and pages added through Pages & Classes are displayed together in the smart prompt.

Considerations to take into account:

- When using page properties without declaring the *Primary* keyword or the class label, there is no disambiguation mechanism to differentiate between referencing the embedded page in the Applies To class (for example, a `Customer.Name` embedded page) and the output of a component (for example, `Customer.Name`, where `Name` is the output of a component named `Customer`).
- If there is a conflict with already defined pages (such as component names, embedded pages from the Applies To context and pages defined through pages and classes), you have to use the *Primary* keyword.
- If the class label changes, the alias you used instead of the *Primary* keyword becomes invalid and it needs to be readjusted to reflect the new class label.

Using Component Properties in Expressions

To use properties of one strategy component in another, declare the name of the component. For example, `RetentionBudget.MaximumBudgetRatio < 80`. If the component used in the expression...
outputs a list (multiple results), only the first element in the result list is considered when computing the expression.

Two strategy properties allow you to define expressions that are evaluated in the context of the decision path (pyComponent and pyPreviousComponent). The strategy property that accesses the current component in the decision path is pyComponent, and pyPreviousComponent accesses the input component selected by the current component. For example, when you have two source components, you can define expressions that apply if a particular component is selected in the decision path:

```
@if(.pyPreviousComponent == "HomeLoans", "Web", "CallCenter")
```

**External Input**

A strategy can be a centralized piece of logic (page 59) that can be used by one or more strategies. You can also use the external input functionality to test strategies with predefined inputs. The external input strategy setting is disabled by default in new strategies, and you can toggle this functionality using the corresponding option in the context menu.

**Strategy Results**

Each strategy contains a standard component that defines its output. Through connecting components to the results component, you define what can be accessed by other instances using the strategy.
Strategy Properties

The properties available to the strategy are determined by its applicability in the decision hierarchy (top level, business issue and group). A newly created strategy lists the Data-pxStrategyResult properties. It also lists every property defined at the SR level (all business issues). If the business issue level applicability was selected in the process of creating the new strategy, properties in the business issue’s data model are also listed. If the group level applicability was selected in the process of creating the new strategy, the strategy can access the properties in the group’s data model. The deeper the scope of the strategy, the more properties it accesses.

With the exception of predictive model outputs, the output of segmentation rules is generally available to the strategy. If you need to use the output of a predictive model in expressions, add the property to the appropriate class in the proposition hierarchy.

Strategy Execution

Strategy execution is performed in the opposite direction of the dependency chain represented by the gray arrows, taking the last component, recursively executing the dependent components and calling out the components whose configuration is tied to other decision instances, data references reading data records and named pages or properties from a page depending on the Applies To class of the strategy. In general, components that reference a rule or a page are subject to auto-mapping, which means that properties with the same name in the referenced rule/page and in the data class defined for the strategy are automatically mapped even if not explicitly mapped through components. The data class can be the strategy result class defined for the strategy or the class corresponding to the scope of the strategy in the proposition hierarchy.

Typically, the last component is a selection component that, through the results component, is exposed to other rules using the strategy. Components whose configuration is tied to other rules are components in the prediction/segmentation category and data import components. Each component creates its own page list from which the embedded pages are of the class the strategy properties belong to. This mechanism allows you to acquire and enrich data. The result of executing a strategy can be a single result or a list. List processing can be implemented by importing a set of propositions by group or by combining data. The only components that do not combine data are champion challenger and switch components.
Interactions

Interaction rules define the parameters for running the strategy, how to save the interaction results and the possible outcomes. Interaction rules can have a fallback policy that allows you to define an alternative path to strategy execution in case the results of the current interaction contain errors. Each interaction can only have a single alternative path but the fallback policy applied in the referenced interaction is triggered if its results also contain errors. Fallback support is not available when you run the interaction through simulations.

Related Information

• Decision Execution (page 69)
• Multilevel Decisioning (page 70)

Decision Execution

Interaction rules can run a strategy based on four modes, or settings:

<table>
<thead>
<tr>
<th>Run Strategy Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clipboard</td>
<td>In this mode, you provide the page (one result) or page list (multiple results) property that holds the output of the corresponding strategy component. After executing the strategy, the interaction context is considered to perform the last steps, which consist of saving the information (clipboard pages or virtual lists) used when executing adaptive models and mapping results to properties. The latter is implemented by mapping pxResults from Code-Pega-List to page or page list properties. The interaction rule publishes strategy results as virtual lists if writing to the pxResults property of a page of class Code-Pega-List. In any other case, it publishes the results as normal clipboard pages.</td>
</tr>
<tr>
<td>Interaction History</td>
<td>If you configure the interaction rule to write results to Interaction History, the strategy provides all the necessary information to build and save the interaction record. The minimum construct the strategy designer needs to apply is adding a data import component that imports the data provided by the page containing the customer response information. Data resulting from an interaction consists of inputs used when issuing the decision, the recommendation (proposition) and the outcome (customer response). Some dimensions are always automatically set (outcome, operator and application), as well as the pySubjectID as defined in the strategy. In case the strategy does not define the pySubjectID, the interaction rule considers the first key defined in the strategy’s primary class.</td>
</tr>
<tr>
<td>Visual Business Director Database via class</td>
<td>Writing to VBD results in creating or updating simulation data sources. In this mode, the interaction also writes the results in a database table. If the class has an associated output definition, the simulation output definition determines how to save and record the results.</td>
</tr>
</tbody>
</table>

The following methods trigger the execution of a decision:

• Flow execution when the flow contains a run interaction shape.
• Activity execution when the activity contains a step that invokes an interaction rule.
• Running work items created in the context of a simulation.
Multilevel Decisioning

Multilevel decisioning allows for aggregating customers into a single group so that this group may be treated as an entity. Action is taken based on the information gathered about the group as a whole. In the decision making context, multilevel decisioning is commonly referred to as a business-to-business (B2B) scenario or household scenario.

Multilevel decisioning relies on a few key strategy and interaction rule design patterns:

- To issue the decision, there is one interaction rule that runs the account level strategy. Running the strategy at the next level is implemented by importing the strategy for that level. For example, account level strategy imports the member level strategy, where the account level strategy defines issuing a decision over a variable number of customers.
• Strategy and interaction rule automatically set the corresponding dimension and fact properties.
• The relationship between the levels is established by `pyGroupID` and `pySubjectID`.
• `pxDecisionReference` and `pxIdentifier` are used to link the process of running the strategies to issue the decision and running the response strategy that is used to write results to Interaction History.
• The key is set by each class, which means the class can not belong to a class group.
• By default, interaction results are not captured at account level. If you want to capture interaction results at this level, make sure the account strategy defines how to write these results.
• So that action is based on information about every member of the group and not just the member targeted in the interaction, use a group by component in front of sub strategy components. The subject ID of each member of the group account defines the ID for which the interaction results are retrieved. The results are combined through a group by component that provides the proposition for the entire group.
A simulation consists of executing an interaction based on data sets defined through input configurations. Running simulations is done through work items (Rule-Obj-Class: Pega-DM-Batch-Work). The work items are stored in the pc_work_dsm_batch table and run asynchronously via QueueForAgent. You can create these work items:

- Using the simulations management landing page.
- In activities, by using the Call Pega-DM-Batch-Work.pxCreateSimulationRun method.

Simulations assume the availability of customer data, data classes and report definition rules. The definition of input data, output settings and topology is performed by system architects, the definition and triggering of simulations is typically performed by business users.

Because batch execution uses DSM pages, Decision Management does not support declare expressions targeting properties that are included as input data.

To guarantee the simulation runs process the correct amount of records, the report definition instances used in input definitions need to be checked in.

Simulation Status

The diagram below shows the possible status of work items created in the context of simulations, which can be automatically triggered at the scheduled time or by explicit user request. A simulation can only run one strategy; if the interaction rule has multiple strategies, the simulation takes the first strategy.
Progress checks are applied to work items in open-running and pending-scheduled status. The system behavior can be configured with values other than default by defining the corresponding dynamic system settings.

- If the status is open-running without any progress (that is, number of processed records remains unchanged) for five minutes, the work item's status is set to resolved-failed.
- A similar action is applied to work items in pending-scheduled status. If the status has not changed to open-running for 30 minutes, the work item's status is also set to resolved-failed.

Dynamic System Settings

If the default system behavior that determines the transition of work items corresponding to simulations to resolved-failed status needs to be changed, add the corresponding system settings and set the value to the desired time interval in minutes.

- By default, work items in open-running status without any progress for five minutes are set to resolved-failed. Default system behavior can be modified by adding the `dsm/batch/autoRecoveryTimeForOpenRunningStatus` dynamic system setting owned by Pega-DecisionEngine and configuring it with the desired value (for example, 10).
- By default, work items in pending-scheduled status for 30 minutes are set to resolved-failed. Default system behavior can be modified by adding the `dsm/batch/autoRecoveryTimeForPendingStatus`
dynamic system setting owned by Pega-DecisionEngine and configuring it with the desired value (for example, 60).
Adaptive Analytics Engine

Adaptive Decision Manager’s main process. The engine is responsible for storing sufficient adaptive statistics (page 75), analyzing them and producing individual scoring models (page 78). These statistics keep the relevant values for adaptive models defined in strategies. From these statistics, the adaptive analytics engine creates scoring models that are published to the adaptive data store (page 75). PRPC retrieves the scoring models from the database and uses them to calculate the prediction.

Adaptive Data Store

The database scoring adaptive statistics (page 75) and adaptive models (page 75).

Adaptive Model

Adaptive models are scoring models (page 78) created through running strategies when adaptive model components are executed in the process of issuing the decision. These models output predictions (page 77) calculated and adapted in real time as interaction results are captured. Adaptive model instances configure the models in the ADM system by defining the settings that influence the behavior of the adaptive models. When adding adaptive model components to a strategy, you configure the propositions (page 78) the adaptive model is going to model and the interpretation of the outputs. Adaptive models belong to the self-learning aspect of Decision Management. They are typically used in the absence of historical records.

Adaptive Statistics

The persistent information resulting from running a strategy (page 78) containing adaptive models (page 75).

Behavioral Profile

A behavioral profile represents a model created on the basis of univariate performance (page 79). The probabilities of positive behavior for each interval/category are score bands (page 78) that can be used to predict in the same way as those of any other model.

Case

A case can be a person, company or event that exhibits some defined outcome.

Coefficient

A weight that is used for each predictor (page 77) in the logistic regression formula. The coefficient is an indication of the importance of a predictor. Negative coefficients imply the presence of predictors with a very similar behavioral profile (page 75). If present, they can lead to over fitting and unreliable models. Consider reanalyzing the predictor grouping to ensure predictors with highly correlated behavior are placed in the same predictor group.

Coefficient of Concordance

The Coefficient of Concordance (CoC) is a non-parametric coefficient (page 75) sensitive to the complete range of score bands (page 78) irrespective of their distribution. The CoC measures how well the scores generated by the model separate positive and negative behavior using the statistic known as coefficient of concordance. CoC can vary between 50% (a random distribution of positive and negative cases by score band) and 100% (a perfect separation). The minimum is 50% because the scores are simply used in reverse if a set of scores orders negative cases before positive cases. Its virtue as a
measure is that it encourages models to be predictive across the score range. If the desired operational circumstances (volume or quality of business) are unknown, CoC generates powerful and generalized models.

**Data Source**

Data about customers and their previous behavior. You can use data sources for modeling, strategy design, simulations and forecasting (page 76). A data source should contain one record per customer with the same structure for each record. Ideally, data should be present for all fields and customers, but some missing data can be tolerated.

**Decision**

The result of running a strategy in an interaction. Several decisions can be involved in a single interaction (page 76).

**Dimensions**

Dimensions provide the context for the facts and results associated with an interaction (page 76). Dimensions are defined in Interaction History.

**Forecasting**

VBD supports making future projections based on simulation (page 78) data generated by running interaction rules that write results to VBD. In order to use this functionality, the customer sample used in forecasting is based on customer data. This data also needs to provide a table/view with the case ID column.

**Interaction**

Some contact with the customer in real time or offline.

**Interaction Result**

The reaction of a customer to a proposition (page 78). Interaction results are recorded in the Interaction History database tables and propagated to ADM and VBD.

**Interval**

Typically, the values of numeric predictors (page 77) are grouped in intervals. Each interval provides a useful building block for understanding behavior.

**Lift**

A measure (multiplied by 100) of the improvement in behavior exhibited by cases (page 75) in one interval (page 76) or segment (page 78) over the average of all cases.

**Modeling**

The process of generating a model as a conceptual representation to identify behavior patterns.

**Next Best Action**

The Next Best Action strategy (page 78) allows applications to take the best decision (page 76) in a multidimensional context (retention, recruitment, risk, recommendation, etc.).

**Next Best Offer**

Next Best Offer decisions (page 76) deliver the facilities to take the best proposition (page 78) based on different product ratings, taking into account other factors, such as products already owned by the customer.
**OXL**

Omega XML Language. The XML file format of predictive models (page 77) generated by Predictive Analytics Director in the process of exporting a model.

**PMML**

Predictive Model Markup Language. An XML-based language that provides compatibility methods for applications to define statistical and data mining models and further sharing these models between PMML compliant applications.

**Population**

The group of cases (page 75) with known behavior, which is consistent with the group of cases whose behavior is to be predicted. In predictive analytics, it is from the population that samples (page 78) are extracted for modeling and validation.

**Prediction**

The behavior to be predicted. The behavior is specific to a form of outcome at a given point in time.

**Predictive Model**

An algorithm that delivers predicted behavior and values for one or more segments (page 78) based on known data. Predictive models are developed using the Predictive Analytics Director desktop application (CDM) or the Predictive Analytics Director functionality in Decision Management enabled applications (PRPC).

**Predictive Performance**

Some measure of the scores (page 78) or segments (page 78) generated by models. Performance can be measured in terms of predictive power (page 77), value or rate achieved under selected conditions.

**Predictive Power**

The predictive power of a scoring model is the measure of the ability of a model to separate cases (page 75) with positive and negative behavior.

**Predictor Grouping**

The grouping of predictors (page 77) whose relationship with behavior are correlated at (or above) a selected level of correlation.

**Predictors**

Predictors are properties considered to have a predictive relationship with the behavior. Predictors contain information available about the cases (page 75) whose values may potentially show some association with the behavior you are trying to predict. Examples include:

- Demographic
  - For example, age, gender and marital status.
- Geo-demographic
  - For example, home address and employment address.
- Financial
  - For example, income and expenses.
- Activity or transaction information
  - For example, the amount of loan taken out of the price of the product.

**Propensity**

The probability of positive behavior or membership.
Proposition
A tangible product offer (a handset or a subscription) or less tangible (benefits, compensations or services).

Sample
A subset of historical data extracted by applying a selection and/or sampling method on the data source (page 76). To be meaningful and reliable, it is essential to make sure that sufficient records are used and, considering the population (page 77), the distribution of values and behavior pattern is representative.

Score
The value calculated by the model that places a case (page 75) on a numerical scale. High scores are associated with good performance and low scores with bad performance. Typically, the range of scores is grouped in intervals (page 76) of increasing likelihood of one of the two types of behavior (positive or negative), based on the behavior of the cases in the development sample (page 78) that fall into each interval. Score intervals (page 76) are aggregated under a score band (page 78).

Score Band
Set of score intervals (page 76).

Score Distribution
The way a predictive model (page 77) segments the cases (page 75) in the population (page 77).

Scoring Model
Scoring models use behavior defined in terms of two opposite types (positive and negative), either a symbol indicating the type of behavior, or the probability of a case being positive or negative.

Segment
A group of customers defined by predicted outcome, score and characteristics. Segments are implemented through segmentation components in a strategy (page 78). They drive the decision by placing a customer in a given segment for which actions/results are defined.

Simulation
Simulations are executed based on changes in the strategy. The strategy decides the top propositions (page 78) to be offered to the customer.

Strategy
The reasoning built up by a set of components that define the business strategy. A strategy provides the decision (page 76) support to manage the interaction (page 76) in the context of the decision hierarchy. Each component has a well defined functionality.

Treatment of Predictors
Symbolic predictors (page 77) can be treated as categorical or ordinal data. Numeric predictors can be treated as categorical or continuous data. Categorical treatment captures data based on the probabilities of positive behavior for each interval/category, ordinal treatment on the sequence code of each category, and continuous treatment on the raw data of the predictor.

Trend Detection
Detecting trends is possible by comparing the performance of multiple models that are triggered by the same propositions (page 78) but are configured with different performance window size to determine
the time frame in number of cases (page 75) over which the performance is calculated. Implementing trend detection requires a combination of strategy design patterns and using compatible adaptive models.

*Univariate Performance*

Univariate performance represents the potential performance of a predictor on its own.