Enterprise Architect

User Guide Series

Decision Model and Notation (DMN)

How do I model decisions? Use Sparx Systems Enterprise Architect to apply the Decision Model and Notation (DMN) standard constructs to model and document decisions and business rules for business and technical users, in Decision Tables created in FEEL.

Author: Sparx Systems
Date: 13/11/2023
Version: 16.1

CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Decision Model and Notation (DMN) 5
- Getting Started 11
- Example Diagram 15
- Creating a Decision Model 19
- Decision Requirements Diagrams 32
- Decision Expression Editor 35
  - Decision Table 39
    - Toolbar for Decision Table Editor 52
    - Decision Table Hit Policy 56
    - Decision Table Validation 60
  - Literal Expression 64
    - Toolbar for Literal Expression Editor 69
    - Example - Loan Repayment 71
  - Boxed Context 73
    - Toolbar for Boxed Context Editor 78
    - Example - Loan Installment Calculation 80
  - Boxed List 86
  - Relation 91
  - Invocation 96
    - Toolbar for Invocation Editor 100
    - Example 1 - Bind Input Data to Business Knowledge Model 103
    - Example 2 - Bind Context Entry variables to Business Knowledge Model 106

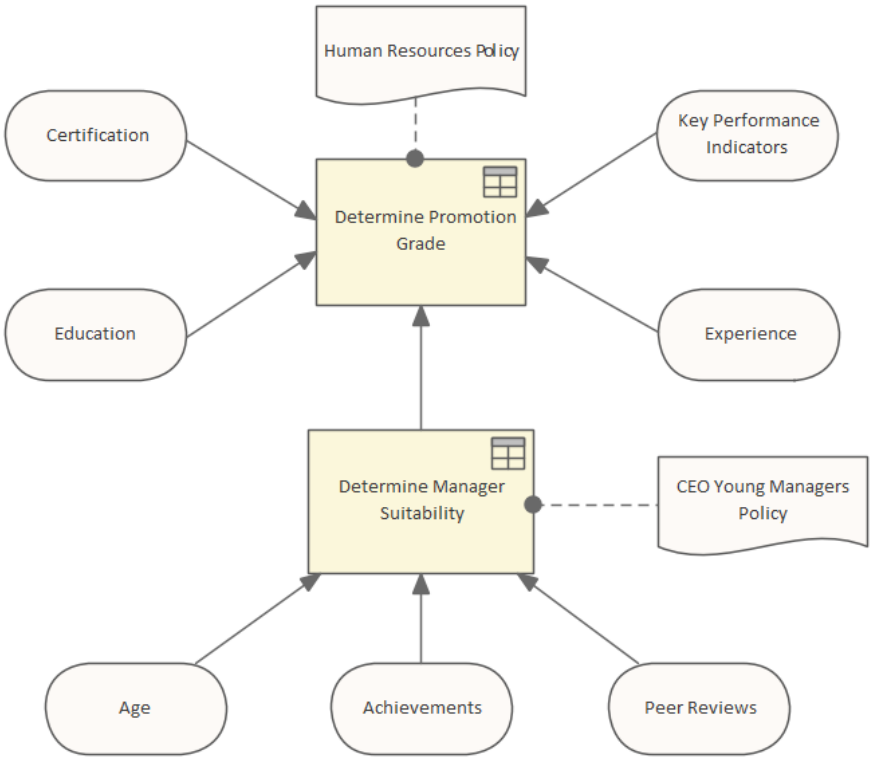
Edit DMN Expression Dialog 108
DMN Expression Validation 112
DMN Expression Auto Completion 116
Modeling with DMN 122
Decision 124
Business Knowledge Model 127
BKM Parameters 131
Input Parameter Values for Simulation 134
Decision Table Simulation Example 136
Literal Expression Simulation Example 139
InputData 141
InputData DMN Expression 143
ItemDefinition 146
Item Definition Toolbar 149
ItemDefinitions and Data Sets 151
Types of Component 155
Allowed Value Enumerations 158
Data Sets 160
Exchange Data Sets using DataObjects 164
Decision Service 168
Simulating a Decision Service 173
Code Generation and Test Module 177
Integrate Into BPSim for Simulation 182
Example: Integrate DMN Decision Service into BPSim Data Object and Property Parameter 191
Example: Integrate DMN Business Knowledge Model into BPSim Property Parameter 193

Integrate Into UML Class Element 196
Importing DMN XML 205
More Information 209

Decision Model and Notation (DMN)

Create and Simulate Detailed Models of Enterprise Decisions

Organizations face increasingly difficult operating environments with fierce and often unpredictable competition from existing and new market players, changes in government and industry regulations and upheavals in the social fabric of their customer base. The decisions that an organization makes in this context are critical to the success of the organization and its ability to steer a safe path through these uncharted corporate waters. Using Enterprise Architects Decision Model and Notation (DMN) features you can not only model the decisions that your organization makes but you can also run simulations from these models to predict outcomes based on example data sets. The power of the language is that business people can readily understand and work with simple but expressive Decision Requirements diagrams that detail what the decisions are and what the inputs to the decisions are and what the expected outputs are. The rules can be documented in a number of ways including easy to define decision tables. Once completed these diagrams with accompanying input data examples can be simulated to show the results of the decisions.



Decision Requirements diagram showing a Decision with a Business Knowledge Model and a number of inputs including another Decision.

Once these models have been defined, simulated and tested by the business, technologists and engineers can then refine these models and automatically generate software artifacts including programming code directly from the models reducing the possibility of errors of interpretation and reducing the time to implementation.

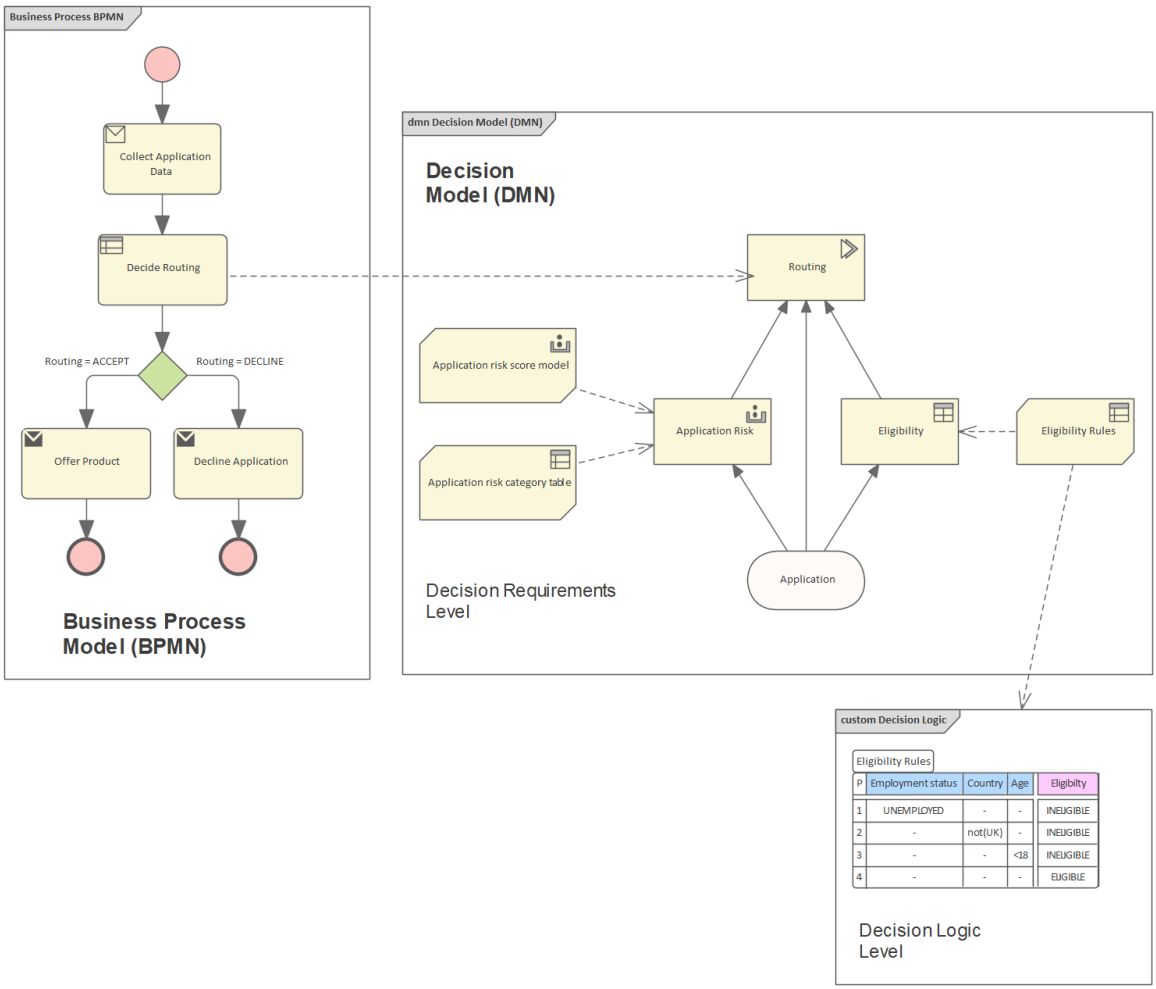
What is DMN?

DMN is intended to provide a bridge between business process models and decision logic models:

- Business process models will define tasks within business

processes where decision-making is required to occur

- Decision Requirements Diagrams will define the decisions to be made in those tasks, their interrelationships, and their requirements for decision logic
- Decision logic will define the required decisions in sufficient detail to allow validation and/or automation



Taken together, Decision Requirements diagrams and decision logic allow you to build a complete Decision Model that complements a business process model by specifying - in detail - the decision-making carried out in process tasks.

DMN provides constructs spanning both decision

requirements and decision logic modeling.

- For decision requirements modeling, it defines the concept of a Decision Requirements Graph (DRG) comprising a set of elements and their connection rules, and a corresponding notation: the Decision Requirements Diagram (DRD).
- For decision logic modeling it provides a language called FEEL for defining and assembling Decision Tables, calculations, if/then/else logic, simple data structures, and externally defined logic from Java and PMML into executable expressions with formally defined semantics.

Benefits of Using DMN in Enterprise Architect

Modeling decision-making processes using DMN allows you to record, specify and analyze complex decision processes as a system of interrelated decisions, business rules, data sets and knowledge sources. By doing so, you can decompose a highly complex decision making process into a network of supporting decisions and input data. This facilitates easier understanding of the overall process, supports refactoring of processes and simplifies the task of validating the process, by allowing you to easily validate the individual steps that make up the overall process.

When you build a Decision Model in Enterprise Architect using DMN, you can run simulations of the model to verify the correctness of the model. After you have verified your

model, you can generate a DMN Module in Java, JavaScript, C++ or C#. The generated DMN Module can be used with the Enterprise Architect BPSim Execution Engine, Executable StateMachine, or within a separate software system that you are implementing.

Enterprise Architect also provides a 'Test Module' facility, which is a preprocess for integrating DMN with BPMN. The aim is to produce BPMN2.0::DataObject elements, then use these to verify that a specified target decision is evaluated correctly with the DMN Module. You then configure BPSim by loading DataObjects and assigning DMN Module decisions to BPSim Properties.

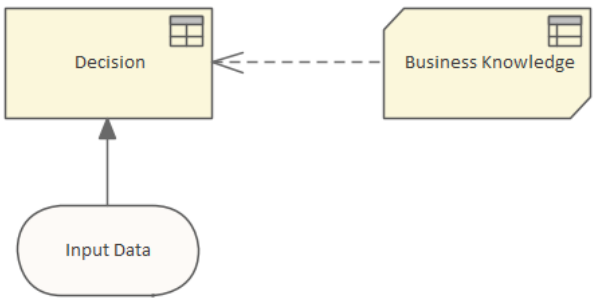
This feature is available in the Unified and Ultimate Editions of Enterprise Architect, from Release 15.0.

Decision Requirements Graphs

The DMN decision requirement model consists of a Decision Requirements Graph (DRG) depicted in one or more Decision Requirements Diagrams (DRDs). The elements modeled are decisions, areas of business knowledge, sources of business knowledge, input data and decision services.

A DRG is a graph composed of elements connected by requirements, and is self-contained in the sense that all the modeled requirements for any Decision in the DRG (its immediate sources of information, knowledge and authority) are present in the same DRG. It is important to distinguish this complete definition of the DRG from a DRD presenting

any particular view of it, which might be a partial or filtered display.



Getting Started

Decision Model and Notation (DMN) is a standard published and managed by the Object Management Group (OMG).

Portions of this topic have been used verbatim or are freely adapted from the DMN Specification, which is available on the OMG DMN web page (https://www.omg.org/spec/DMN). A full description of the DMN and its capabilities can be found on the OMG website.

The purpose of DMN is to provide the constructs that are needed to model decisions, so that organizational decision-making can be readily depicted in diagrams, accurately defined by business analysts, and (optionally) automated. It is also intended to facilitate the sharing and interchange of Decision Models between organizations.

Selecting the Perspective

Enterprise Architect partitions the tool's extensive features into Perspectives, which ensures that you can focus on a specific task and work with the tools you need without the distraction of other features. To work with the Decision Model and Notation features you first need to select this Perspective:

<perspective name> > Requirements > Decision Modeling

Setting the Perspective ensures that the Decision Model and Notation diagrams, their tool boxes and other features of the Perspective will be available by default.

Example Diagram

An example diagram provides a visual introduction to the topic and allows you to see some of the important elements and connectors that are created in specifying or describing the way decisions are modeled. The Decision Requirements diagram will introduce elements such as Decision Tables, Knowledge Sources, Input Date and more. Much of the power of Enterprise Architect relies in the ability to simulate or 'run' the decision models and to predict outcomes based on different data sets. This functionality will be described in later topics but starts with the creation of a Decision Requirements diagram.

Modeling with DMN

This topic introduces you to the most important elements that you need to create Decision models. This includes the creation of a Decision Requirements diagram which describes how decision are related and what inputs each decision has potentially including other decisions. You will learn about the mst important elements including: Decisions. Business Knowledge Models, Input Data Items Definitions, Data Set and Decision Services.

Code Generation and Test Module

This topic introduces you to the main concepts of the

language including its structure, architecture and the elements and connectors that are used to create Decision Model and Notation (DMN) models. Understanding the intent and structure of the language will help analysts create meaningful and productive decision models.

Integrate Into BPSim for Simulation

Enterprise Architect allows Decision models to be run (simulated) which allows you to visualize the outcomes of decisions. In addition to this core facility you can also integrate the Decision models with BPSim which is a simulation engine for the simulation of BPMN diagrams. In this topic you will a number of different ways to integrate DMN with BPSIm.

Integrate Into UML Class Element

In this topic you will learn the process of integrating a DMN Model with a UML Class element. The DMN Module can be integrated with a UML Class element, so the code generated from that Class element can reuse the DMN Module and be well-structured

Importing DMN XML

This topic describes how to import a DMN XML file from a different Enterprise Architect repository or another DMN compliant tool. One of the promises of open standards is the

ability to share models between different tools. Enterprise Architect often becomes the tool of choice for modeling because of the breadth of features and standards it supports. This allows Decision model to be related to Strategy, Requirements, Business Processes, Software Implementation elements and more.

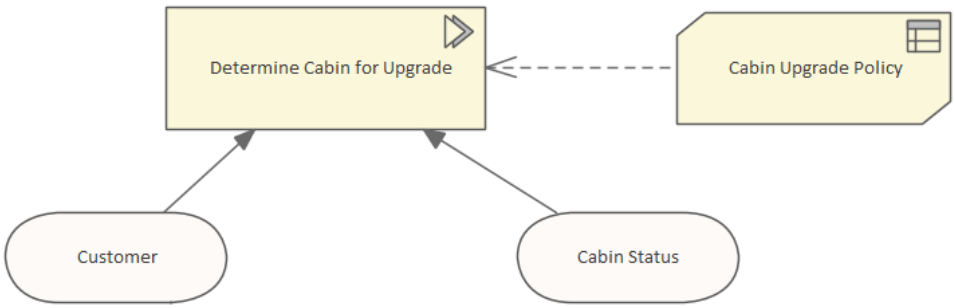
More Information

This section provides useful links to other topics and resources that you might find useful when working with the Decision Model and Notation tool features.

Example Diagram

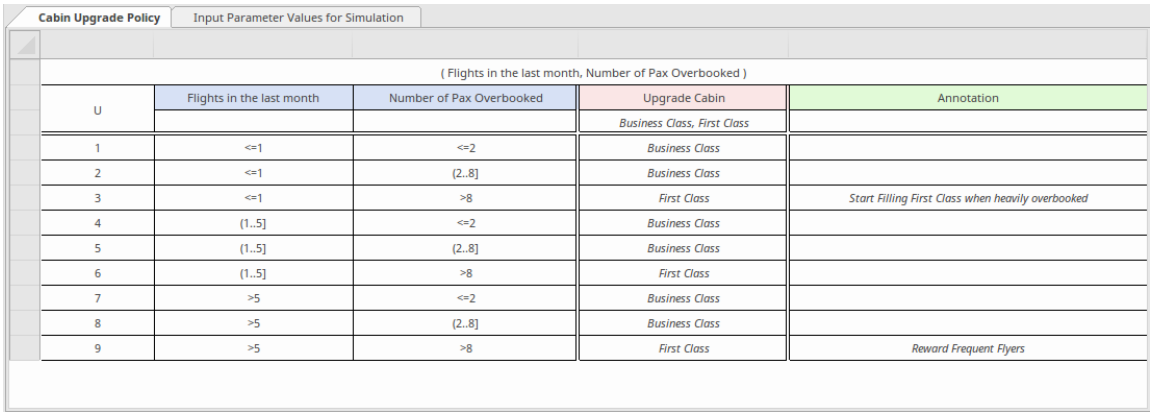
Imagine you are an Airline reservation officer working at the check-in counter for a busy domestic airline. Getting the aircraft off on-time is critical as delays can result in fees applied by the airport controllers, needing to fly at a lower altitude increasing the cost of fuel, and other penalties.

A message from the supervisor appears on your screen saying that the economy cabin is overbooked; you will need to upgrade some passengers to Business or First Class — but which passengers should be chosen and which cabin should they be upgraded to? A decision needs to be made but what factors should be considered? This can be recorded in a Decision Model using a Decision Requirements diagram.



This is helpful but the busy check-in officer would still need to weigh up all the factors and make an unbiased decision. Should a disgruntled passenger be given priority over a Gold level frequent flyer, or should the fact that a particular passenger is connecting to an international flight take precedence. These 'rules' can all be recorded in a Decision Table, making it clear which passengers should get an upgrade and to which cabin: Business or First Class. This

will make it much easier to make the decision and the rules can be formulated, agreed upon and checked for consistency back at head office. In this example we have kept it simple and used two factors: firstly the number of flights the passenger has made in the last month and secondly how overbooked the cabin is.



Cabin Upgrade Policy | Input Parameter Values for Simulation

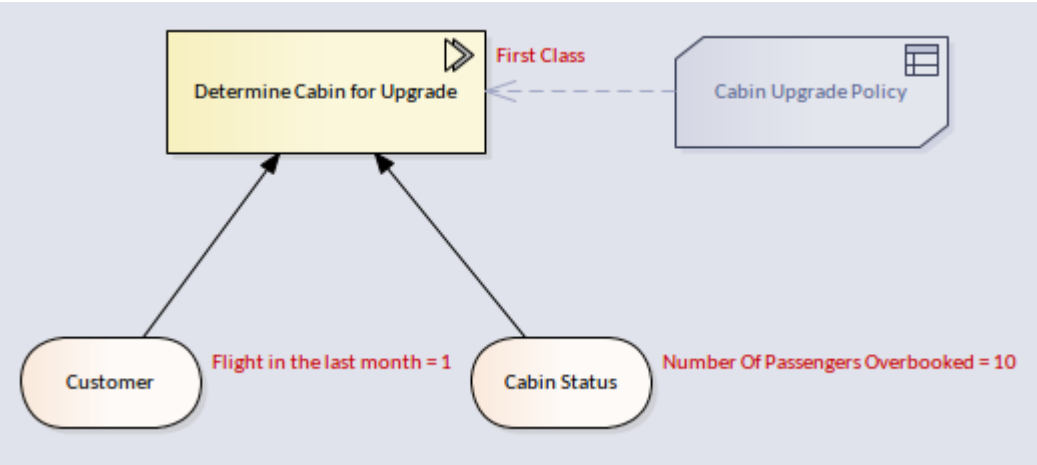
	(Flights in the last month, Number of Pax Overbooked)			
U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin	Annotation
			Business Class, First Class	
1	<=1	<=2	*Business Class*	
2	<=1	(2..8]	*Business Class*	
3	<=1	>8	*First Class*	*Start Filling First Class when heavily overbooked*
4	(1..5]	<=2	*Business Class*	
5	(1..5]	(2..8]	*Business Class*	
6	(1..5]	>8	*First Class*	
7	>5	<=2	*Business Class*	
8	>5	(2..8]	*Business Class*	
9	>5	>8	*First Class*	*Reward Frequent Flyers*

The table is divided into columns and rows. There are three types of column: inputs that are required to make the decision, outputs that are the result of applying the rules, and annotations.

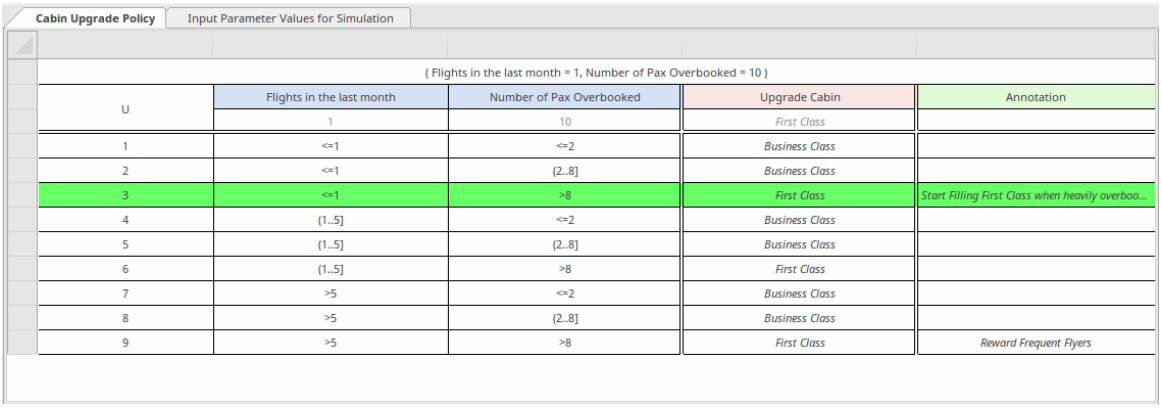
This is again very helpful but still requires the busy check-in officer to be able to source all the required information required to find the right row in the Decision Table. Even if all this information were available, a wrong decision could still result from human error in selecting the wrong row in the table.

Fortunately the Decision Models can be automated and generated to programming code that can be executed by an application. So our busy check-in officer would not need to do anything or make any decisions; as he or she was checking in the passengers, if a particular passenger was entitled to an upgrade it would be visible on the computer

screen. In the next diagram the model has been simulated so that the business and technical staff can agree that the model has been defined correctly. Any number of user-defined data sets can be used to test the model before generating the programming code that will run in the check-in system and display the result to the end user.



When developing the models a business or technical user can step through the simulation and the system will show that user which row in the Decision Table was fired to determine the output. This is very useful in models that are made up of multiple decisions.



Cabin Upgrade Policy | Input Parameter Values for Simulation

(Flights in the last month = 1, Number of Pax Overbooked = 10)

U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin	Annotation
	1	10	First Class	
1	<=1	<=2	Business Class	
2	<=1	(2..8]	Business Class	
3	<=1	>8	First Class	Start Filling First Class when heavily overboo...
4	(1..5]	<=2	Business Class	
5	(1..5]	(2..8]	Business Class	
6	(1..5]	>8	First Class	
7	>5	<=2	Business Class	
8	>5	(2..8]	Business Class	
9	>5	>8	First Class	Reward Frequent Flyers

It is common for the rules that govern the upgrade decision to change. For example, the Marketing Department might decide they want to reward passengers that travel on long-haul flights. The Decision Requirements diagram can

be altered to include the new input, the Decision Table modified, and the programming code regenerated. Once the changes have been pushed through to the airport systems, the right passengers will be automatically upgraded. The check-in officer could still view the Decision Tables during a training and briefing session to understand the rules.

Creating a Decision Model

In the model we described in *An Example of Decision Modeling*, we showed how a decision can be modeled using a Decision Table, in which a decision result is determined by finding a row in the table where the input values in the table match the input values under consideration, giving a particular output result.

We will now look at how such a model can be created in Enterprise Architect, by stepping through the process of creating the decision model for the Airline Cabin Upgrade example.

There are a number of model elements involved in this example, such as Input Data elements, Item Definitions that are used to describe the Input Data (defining the data types), a Decision element and also a Business Knowledge Model element that holds the Decision Table definition.

Create a Decision Requirements Diagram

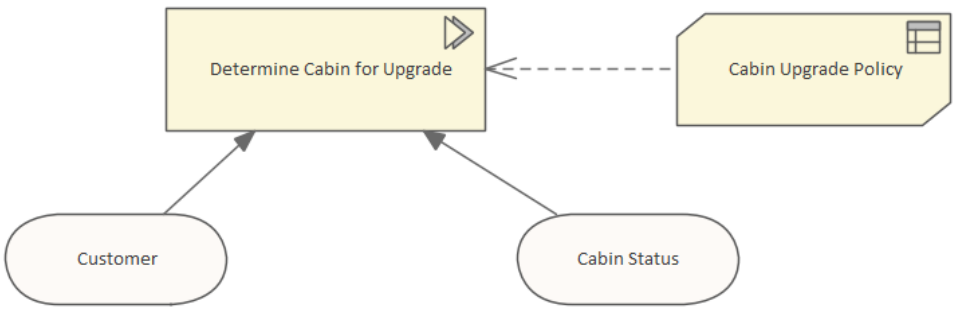
These steps will guide you through the creation of a simple Decision Requirements Diagram (DRD). In this example, we will create the model from scratch, rather than using a pattern from the Model Wizard.

Step	Description
1	Select the perspective 'Requirements | Decision Modeling'.

	(The Start Page 'Create from Pattern' tab (Model Wizard) is displayed, but we will not use it for this example.)
2	Create a new DMN diagram. Name it 'Airline Cabin Upgrade'.
3	Using the diagram toolbox, place a Decision element on the diagram. Choose 'Invocation' as the type - we will use this element to 'invoke' a decision from a Business Knowledge Model element. Name the element 'Determine Cabin for Upgrade'.
4	Place an InputData element on the diagram. Name this element 'Customer'.
5	Place another InputData element on the diagram. Name this element 'Cabin Status'.
6	Place a Business Knowledge Model element on the diagram. Choose the type 'Decision Table'. Name this element 'Cabin Upgrade Policy'.
7	Draw an 'Information Requirement' connector **from** the decision 'Determine

	Cabin for Upgrade' **to** the input data 'Customer'.
8	Draw an 'Information Requirement' connector **from** the decision 'Determine Cabin for Upgrade' **to** the input data 'Cabin Status'.
9	Draw a 'Knowledge Requirement' connector **from** the decision 'Determine Cabin for Upgrade' **to** the BKM 'Cabin Upgrade Policy'.

At this stage, we should have a simple DRD, that resembles this:

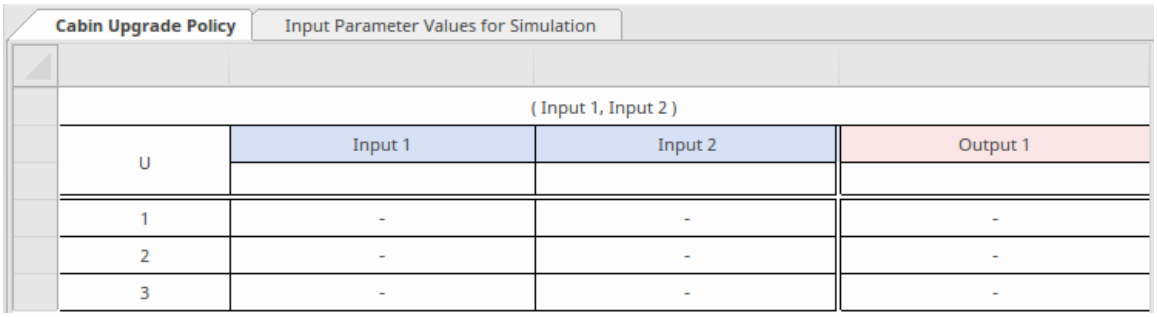


We can now specify the details for each of the elements making up this model.

Define the Decision Table

By double-clicking on the Business Knowledge Model

element 'Cabin Upgrade Policy', the 'DMN Expression' window is displayed, showing an empty Decision Table. This is where we will define the rules of our cabin upgrade policy.



By default, new Decision Tables are created with two input columns and one output column, a header row and three empty rules rows.

The left-most column in the table displays the Hit Policy and also numbers the rules. By default, the Hit Policy is 'U' for 'Unique'. This is the policy that we will use for our example, so you do not need to change this column heading.

For more information on Hit Policies, refer to the *Decision Table Hit Policy* Help topic.

Name and Define Types for Decision Table Inputs and Outputs

Step	Description
1	On the toolbar of the 'DMN Expression' window, click on the 'Edit Parameters' button, .

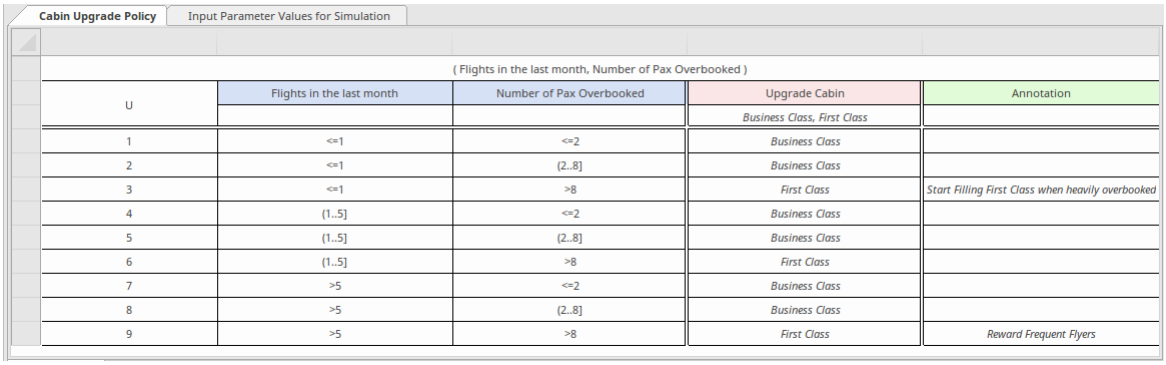
	The 'Edit Parameters' dialog displays.
2	Replace the parameter name 'Input 1' with 'Num of Pax Overbooked'. If necessary, click on the 'Type' drop-down arrow and set the type of this parameter to 'number'.
3	Replace the parameter name 'Input 2' with 'Num of Flights in Last Month by Pass'. Set the type of this parameter to 'number' as well. Close the 'Edit Parameters' dialog.
4	Edit the input expression that will be evaluated for column 1. Select the header cell (containing the text 'Input 1') then click again or press F2 to enter 'Edit' mode. Select all of the cell text, then press the Spacebar. The list of input parameters is displayed. Click on 'Num of Pax Overbooked', then press 'Enter'. The *expression* for column 1 is set to 'Num of Pax Overbooked'. **Note**: The input expressions evaluated for each column typically just use the corresponding input parameter; however,

	you *can* use a complex expression.
5	Right-click on the column 1 expression and check that its data type is set to 'number'.
6	Edit the input expression that will be evaluated for column 2. Select all of the text, then press the Spacebar. The list of input parameters is displayed. Choose 'Num of Flights in Last Month for Pass', then press 'Enter'. The *expression* for column 2 is set to 'Num of Flights in Last Month for Pass'.
7	Right-click on the column 2 expression and set its data type to 'number'.
8	Edit the name of the decision table output. Replace 'Output 1' with 'Upgrade Cabin', then press 'Enter'.
9	Set the data type of the decision output. Right-click on the output column header and choose 'string'.
10	Set the allowable values for the decision

	output. In the cell directly beneath the output column header (but above row 1), define the allowable values for output. Enter 'Business Class, First Class'. **Note**: There is no need for quote marks around the values, as the data type has been specified as 'string'.

Define the Rules of the Decision Table

Enter values into the table cells to match this image.



Cabin Upgrade Policy | Input Parameter Values for Simulation

(Flights in the last month, Number of Pax Overbooked)

U	Flights in the last month	Number of Pax Overbooked	Upgrade Cabin	Annotation
			Business Class, First Class	
1	<=1	<=2	Business Class	
2	<=1	(2..8]	Business Class	
3	<=1	>8	First Class	Start Filling First Class when heavily overbooked
4	(1..5]	<=2	Business Class	
5	(1..5]	(2..8]	Business Class	
6	(1..5]	>8	First Class	
7	>5	<=2	Business Class	
8	>5	(2..8]	Business Class	
9	>5	>8	First Class	Reward Frequent Flyers

Click on a cell to select it, and click again to edit it.

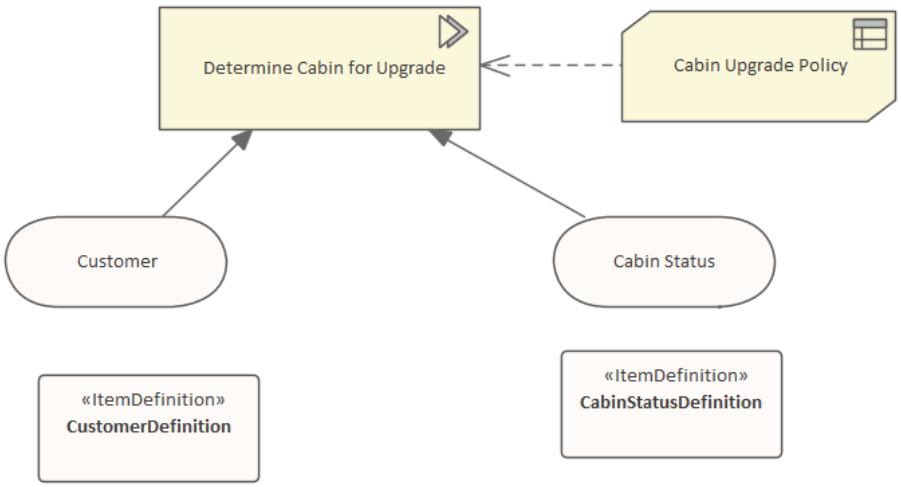
You can copy and paste existing rules by selecting the rows to copy (Shift+click adds to the selection), right-click and choose 'Copy', then right-click and choose 'Append'.

Once you have finished editing the rules, click on the Save button .

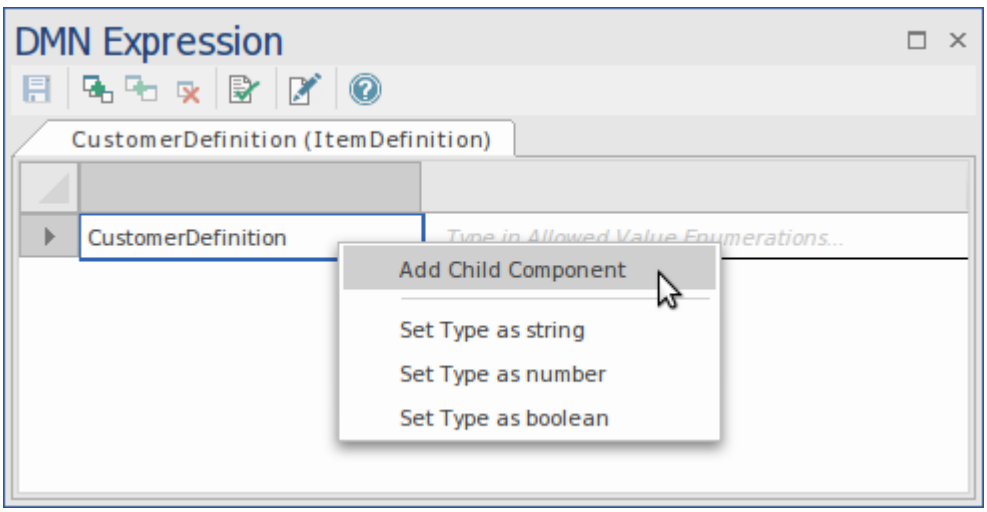
Finally, click the Validate button , to check for errors in the table of rules.

Create ItemDefinition Elements

Add two ItemDefinition elements to the diagram, one for each of the InputData elements. Name one element 'CustomerDefinition' and the other 'CabinStatusDefinition'.



Double-click the ItemDefinition named 'CustomerDefinition' to edit the definition. The DMN Expression window is displayed.



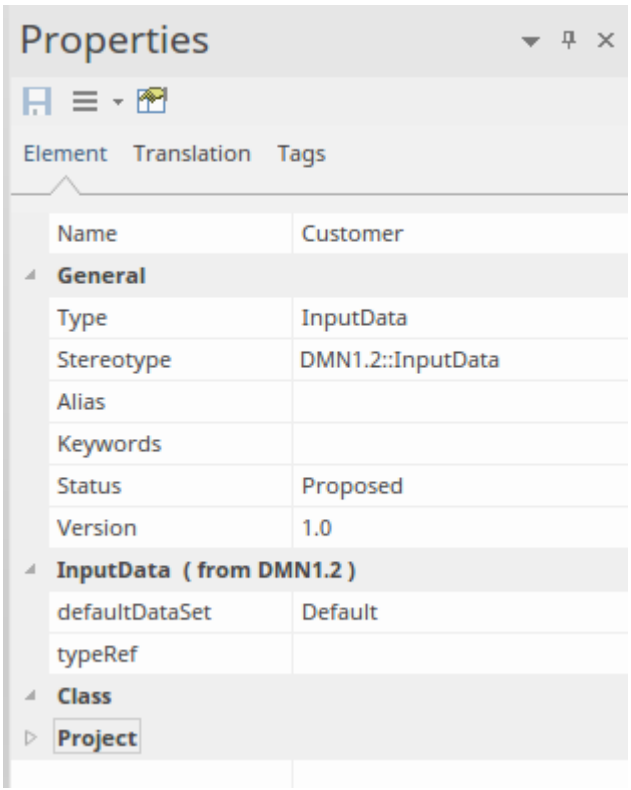
Right-click on the cell 'CustomerDefinition' and choose 'Add Child Component'. Overtype the name of the child component with 'Num of Flights in Last Month' and overtype its datatype with 'number'. Click the 'Save' button

to save the changes, and close the window.

Similarly, double-click on the ItemDefinition named 'CabinStatusDefinition', add a child component named 'Num of Pax Overbooked' and set its data type to 'number'. Save the changes and close the window.

Specify the Data Type For Each InputData Element

Select the InputData element 'Customer'. In the Properties window, select the property 'typeRef' and click on the ... button.



Select the ItemDefinition 'Customer Definition' as the type. Click on 'OK'.

Similarly, specify 'Cabin Status Definition' as the type for

'Cabin Status'.

Specify the Inputs to the Decision Element

Double-click on the decision element 'Determine Cabin for Upgrade'

In the DMN Expression window, locate the table row containing the text 'Num of Pax Overbooked' in the first column. Click in the cell in the second column of this row, and press the Spacebar. A list of possible input values is displayed. Choose 'Cabin Status . Num of Pax Overbooked' and press 'Enter'. The selection is written into the cell.

Repeat this process for the second table row 'Num of Flights in Last Month', choosing 'Customer . Num of Flights in Last Month'.

Click on the Save button.

Click on the Validate button.

Define Data Sets

The 'correctness' of your decision model can be tested, by running simulations using a range of representative data sets to verify that the model produces the correct result in all situations.

You can create numerous Data Sets with various names, using a range of data values. You can set one of the data sets as the *default value*.

We will now create a Data Set for each of our InputData

elements.

Step	Description
1	Double-click on the InputData element 'Customer'. The DMN Expression window displays.
2	In the DMN Expression window, click on the 'Edit Data Set' button . The 'Edit Data Set' window is displayed.
3	Click on the button. A new data set is created.
4	Overwrite the name of the data set if you wish. Leave the Type as 'number'. Enter a value of, for example, 3. Click on the Save icon and the OK button.
5	Repeat for the InputData 'Cabin Status'. Enter a value of, for example, 4.

Add a DMNSimConfiguration Artifact

Locate the DMN 'Simulation Configuration' Artifact in the Diagram Toolbox. Drop one of these onto the diagram as well.

Double-click on it to open the DMN Simulation window at the 'Simulate' tab.

From the DMN Simulation window, you can run simulations of the completed Decision Model. You can also perform validation, generate code and generate test modules.

Step	Description
1	Locate the edit field in the toolbar of this window.
2	Click on the drop-down arrow in this field. A list displays, showing all of the Decision Services and Decision elements in the Package associated with the DMNSim Configuration Artifact. In this case, 'Determine Cabin for Upgrade' is the only item in the list.
3	Click on 'Determine Cabin for Upgrade'.
4	The body of the window now displays the InputData elements and the decision results that are available as inputs to the

	selected decision. Click on the Save button.
5	Use the 'Value' column to select one of the predefined DataSets for the InputValues, then you can click on the 'Run' button in the lower toolbar to run a simulation, using the selected data sets.

Decision Requirements Diagrams

The elements modeled in Decision Requirements graphs (DRGs) and Decision Requirements diagrams (DRDs) are Decision, Business Knowledge Model, Input Data, Knowledge Source and Decision Service. The dependencies between these elements express three kinds of requirement - Information, Knowledge and Authority.

Components of Decision Requirements Diagrams

This table summarizes the notation for all components of a Decision Requirements diagram.

Component	Description
Decision	A Decision element denotes the act of determining an output from a number of inputs (Input Data or Decision), using decision logic expressed as Literal Expressions, Decision Tables, Invocations or Boxed Context.
Business Knowledge Model	A Business Knowledge Model denotes a reusable module of decision logic represented by a function, which includes zero, one or more than one parameter.

Decision Service (expanded)	A Decision Service can enclose a set of reusable decisions that are invoked internally - for example, by another Decision or Business Knowledge Model - or externally - for example, by a BPMN Process. A good practice is to use a diagram to describe a single expanded Decision Service.
Decision Service (collapsed)	If a Decision Service element serves as an invocable element, connected with knowledge requirements to other elements with invocation logic, we can hide the details of the Decision Service to focus on the decision hierarchies in the big picture.
Input Data	An Input Data element denotes information used as an input to one or more Decisions.
Item Definition	An Item Definition is used to define the type and structure of data items used in the decision model. It is primarily referenced by Input Data elements as a basis for the type and structure of data expected to be input. It can also be referenced for setting the structure for an

	output. The Item Definition contains Data Sets that provide sets of values useful when performing varied simulations.
Knowledge Source	A Knowledge Source element denotes an authority for a Business Knowledge Model or Decision.
Information Requirement	An Information Requirement denotes Input Data or Decision output being used as input to a Decision.
Knowledge Requirement	A Knowledge Requirement denotes the invocation of a Business Knowledge Model or Decision Service.
Authority Requirement	An Authority Requirement denotes the dependence of a DRG element on another DRG element that acts as a source of guidance or knowledge.

Decision Expression Editor

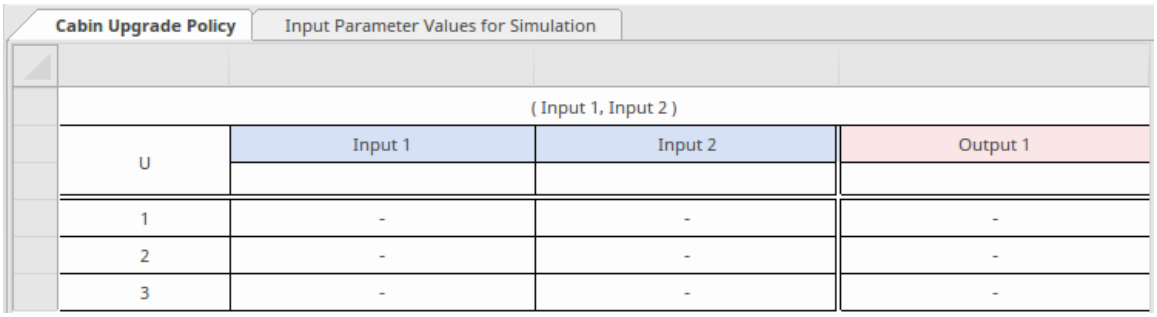
The DMN Expression Editor is the window in which you will define, review and update the details of most of the different types of DMN element within your model. Primarily, it is used for editing the Value Expressions of Decision elements and BusinessKnowledgeModel (BKM) elements.

A different version of the DMN Expression Editor is displayed for each of the four types of value expression used by Decision elements and BKM elements. For BKM elements a second window tab is also presented, for defining the input and output parameters used in calling the BKM.

Two additional versions of the DMN Expression Editor also exist to support editing of ItemDefinition and InputData elements.

The toolbar that is displayed, and the layout of the window content, are dependent upon the type of DMN element that is currently selected and, where applicable, the type of Value Expression being defined.

This image shows the version of the DMN Expression Editor used for defining a Decision Table. In this case, the underlying element is a BusinessKnowledgeModel, and so the decision logic is 'invoked' by other elements, with input and output passed via parameters.



Detailed explanations of the DMN Expression Editor's features for each element and expression type are provided in the child Help topics of this topic.

Access

Diagram	Double-click a DMN element on a diagram. The DMN Expression editor window corresponding to the element and its expression type is displayed.

Value Expressions

Summarized in this table are four distinct types of value expression with references to the Help topics detailing each of them.

Type and Icon	Description
Decision	A Decision Table is a tabular

Table	representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.
Literal Expression	A Literal Expression specifies the decision logic as a textual expression that describes how an output value is derived from its input values. To support simulation and execution, the Literal Expression can use JavaScript functions.
Boxed Context	A boxed context is a collection of context entries, consisting of (name, value) pairs, each with a result value. The context entries provide a means of decomposing a complex expression into a series of simple expressions, providing intermediate results that can be used in subsequent context entries.
Invocation	An invocation calls on another model element (a BusinessKnowledgeModel or a Decision Service) to provide a decision result. The invocation defines parameters that are passed into the 'invoked' element, providing context for evaluation of its decision logic. The decision result is then passed back to the 'invoking' element.

ItemDefinition and InputData Elements

Element	Description
ItemDefinition	ItemDefinition elements are used to define data structures and, optionally, to restrict the range of allowable values of the data. ItemDefinitions can range from a simple single type through to a complex structured type. ItemDefinitions are used to specify the type of InputData elements as well as input parameters.
InputData	InputData elements are used to provide input to Decision elements. The data type of an InputData element is defined using an ItemDefinition element. Data Sets can also be defined as part of an ItemDefinition and an InputData element can then specify a Data Set to be used when running a simulation.

Decision Table

A Decision Table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.

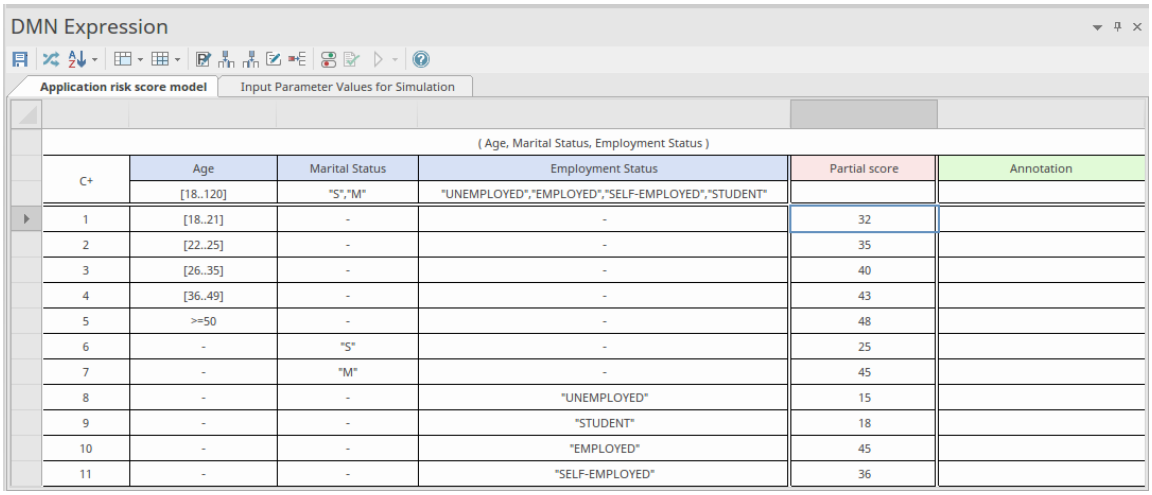
Decision Tables are supported by both the *Decision* and the *Business Knowledge Model* element types. They are denoted by the icon in the top right corner of the element on a diagram.

Access

Diagram	On a diagram, double-click on a Decision element or BusinessKnowledgeModel element. The DMN Expression window is displayed, showing details of the selected element.

Overview

This image shows the DMN Expression window as it appears for a Decision Table.


DMN Expression

Application risk score model | Input Parameter Values for Simulation

(Age, Marital Status, Employment Status)

	C+	Age	Marital Status	Employment Status	Partial score	Annotation
		[18..120]	"S","M"	"UNEMPLOYED","EMPLOYED","SELF-EMPLOYED","STUDENT"		
1		[18..21]	-	-	32	
2		[22..25]	-	-	35	
3		[26..35]	-	-	40	
4		[36..49]	-	-	43	
5		>=50	-	-	48	
6		-	"S"	-	25	
7		-	"M"	-	45	
8		-	-	"UNEMPLOYED"	15	
9		-	-	"STUDENT"	18	
10		-	-	"EMPLOYED"	45	
11		-	-	"SELF-EMPLOYED"	36	

A Decision Table consists of:

- The Table Hit Policy (C+, U, A, P and so on) that specifies how the rules are applied
- A list of rules (1, 2, 3, 4 and so on), where each rule row contains specific input entries and corresponding output entries
- A list of Input Clauses (under the blue headings), defined as expressions that generally involve one or more input values
- A list of Output Clauses (under the pink heading), defining the output corresponding to a specific set of inputs
- Optional annotations for each rule (under the green heading) which you can add through the window Toolbar

An Input Clause consists of an expression and an optional list of allowed values (the row just underneath the column headings). Very often, the expression is simply an unmodified input value; however, it could also be an expression involving more than one input value or perhaps a conditional statement such as 'Application Risk Score >

100'. The allowable values apply to the *expression result* rather than the input values used.

Each Output Clause consists of an identifier (a name) and again an optional list of allowed values for that clause.

The Decision Table should contain all the inputs - and only those inputs - required to determine an output.

In determining which rules are applied, the expressions defined in the Input Clauses are evaluated for the given inputs and the *expression results* are then used to find rules with matching input entries.

Where the DMN Expression window is not wide or deep enough to display all columns and rows, you can use scroll bars to access the hidden content, or drag the borders out to increase every column width. The 'Input' and 'Output' column widths are initially the same, but you can adjust each column width independently of the others, by dragging the column border either within the table or in the gray bar just below the tab names.

Toolbar for Decision Table Editor

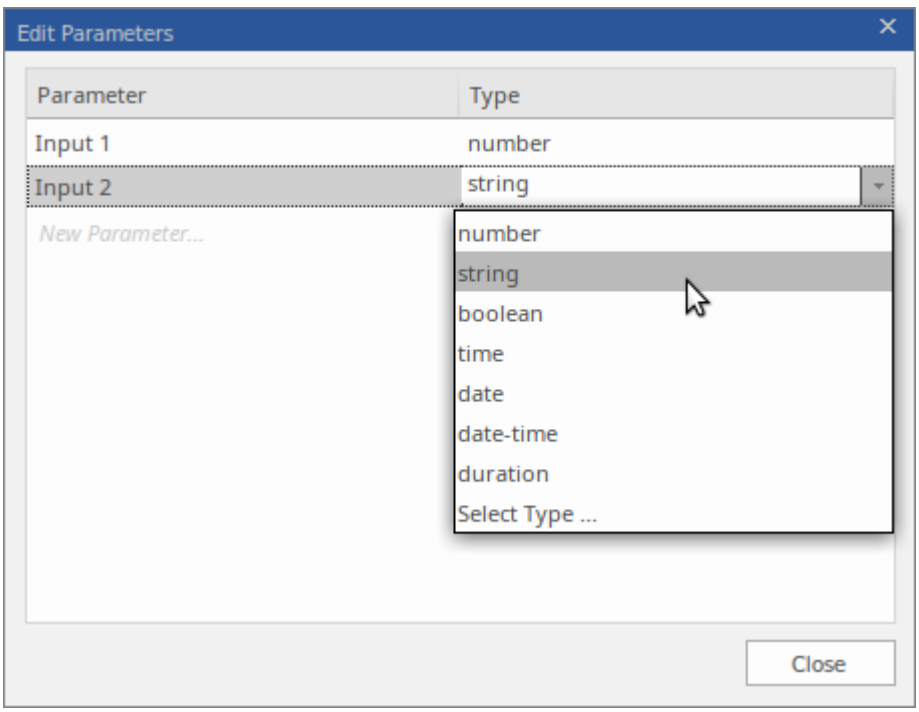
When a Decision Table is selected, the features available in the DMN Expression window are accessed via the Toolbar at the top of the window, as shown:

For more details refer to the *Toolbar for Decision Table Editor* Help topic.

Parameters

In the case of Business Knowledge Model (BKM) elements, parameters are used to pass input values supplied by the invoking element. The BKM's decision logic is evaluated using the input parameters and the result is returned to the invoking element. By default, a BKM element is created with two input parameters, 'Input 1' and 'Input 2'.

Click on the icon in the toolbar of the DMN Expression window to display the 'Edit Parameters' dialog.



Here you can change the parameter names, re-arrange the sequence, set their data types, create additional parameters or delete existing ones.

Hit Policy

Right-click on the 'Hit Policy Indicator', then choose the

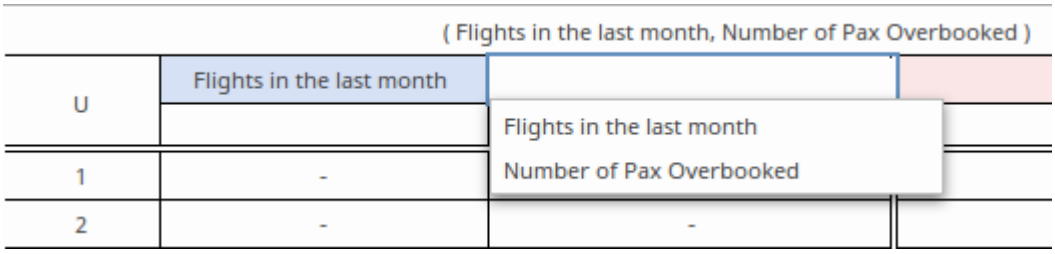
desired Hit Policy from the pop-up menu. The various Table Hit Policies are described in detail in the *Decision Table Hit Policy* Help topic.

Input Clauses

An Input Clause of a Decision Table is defined as an expression. Very often, the expression is simply an unmodified input value; however, it could also be an expression involving more than one input value or it could be defined as a conditional statement, such as 'Application Risk Score > 100'. The allowable values apply to the *expression result* rather than the input values used and, as such, the type of the values should match the type of the expression result.

Decision Tables are created with two default Input Clauses, 'Input 1' and 'Input 2'. The data type for both of these clauses is 'number'. In the DMN Expression window, the Input Clauses are displayed as column headings on the Decision Table. To modify an Input Clause, click on the column heading to select the cell, then click again or press F2 to edit.

Auto-completion is supported when editing Input Clauses. That means, for Decision elements, any inputs that are connected to the Decision element are made available for selection from a list. Similarly, for Business Knowledge Model elements, the invocation parameters are made available for selection from a list. See the *DMN Expression Auto-completion* Help topic for further information.



To add additional columns of input entries to the Decision Table, click on the icon on the toolbar of the DMN Expression window.

To remove input columns from the table, right-click within the unwanted input column, then select the option 'Delete Input Column' from the pop-up menu.

The order of the input columns in the table can be re-arranged by dragging and dropping columns to new positions. (Drag the unlabelled cell at the very top of the table column to the required position.)

Allowed Values

When defining an 'Input' or an 'Output' column, the second row of the column defines the Allowed Values. This is an optional cell in the column, but useful for clarifying the entries in the rows beneath it. When running a validation, each of the cells below the Allowed Values cell are checked to make sure they conform to the expression in this cell.

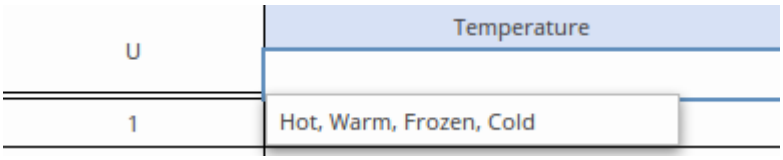
The expressions used in this cell depend on how the 'Input' or 'Output' column is typed. For example:

- Number - [18 ..35]
- String - 'High', 'Low', 'Medium'
- Boolean - true, false

- Undefined - '-'

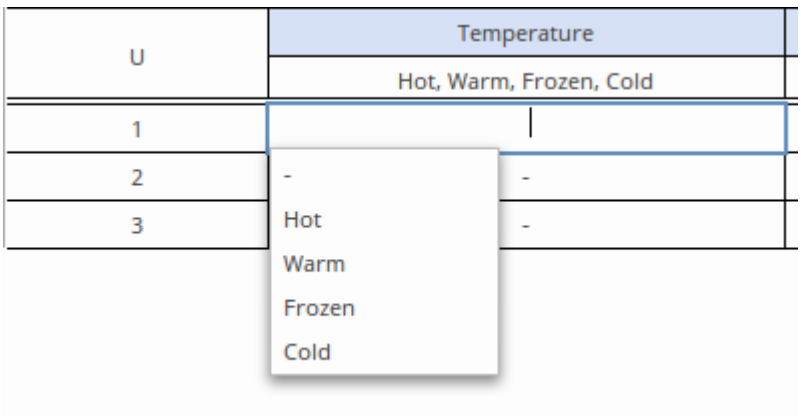
Fast Fill Allowed Values

The Input/Output Expression that this references can be a simple value or a complex FEEL expression; however, if it is directly related to an ItemDefinition's 'Allowed Values' field then pressing the Spacebar will enable a fast-fill option to set the 'Allowed Values' as defined in the ItemDefinition (usually referenced via an InputData element).



Fast Fill Rows

Once the 'Allowed Values' field is defined, as well as restricting the values that can be used when defining the rules in the table, the 'Allowed Values' field also provides the user with a fast fill option. This is invoked, in a rule cell, by pressing the Spacebar and selecting the required item:



For more details see the Help topic *DMN Expression Auto Completion*.

Output Clauses

An Output Clause consists of a name, a data type and an optional list of allowed values. To modify an Output Clause, click on the column heading cell to select the cell, then click again or press F2 to edit.

To add additional columns of output entries to the Decision Table, click on the icon on the toolbar of the Expression editor window.

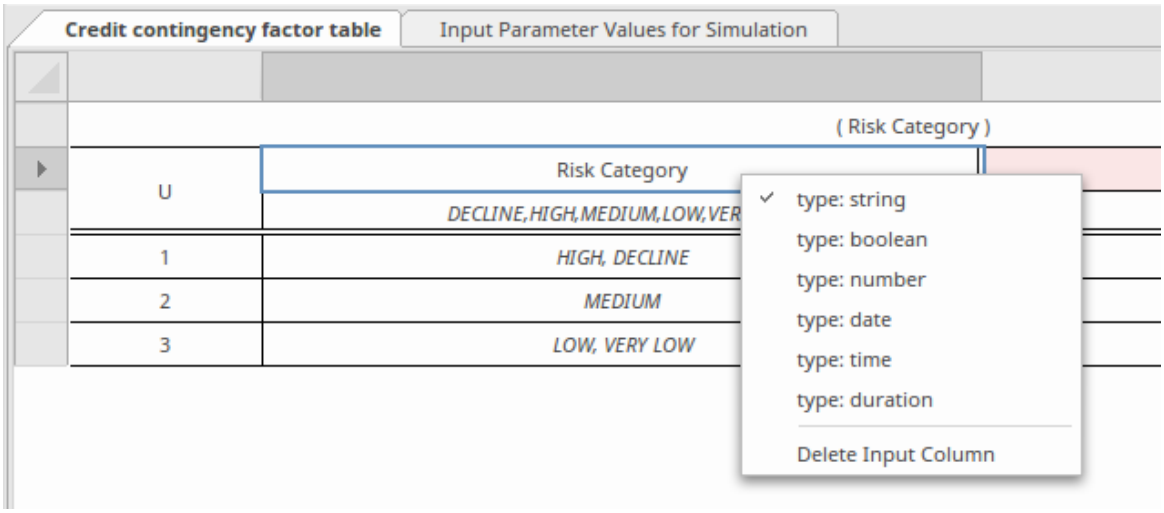
To remove Output columns from the table, right-click within the unwanted Output column, then select the option 'Delete Output Column' from the pop-up menu.

The order of the columns in the table can be re-arranged by dragging and dropping columns to new positions. (Drag the unlabelled cell at the very top of the table column to the desired position.)

Data Type for Input/Output Clauses

For the simulation to work it is critical to set the data type for all Input and Output Clauses. Range, gap and overlap validations are supported for clauses of type 'number', but validation cannot be performed if the type has not been specified. Code Generation for typed languages such as C++, C# and Java requires that the data types are specified. When the data type is specified as 'string', there is no need to enclose each string literal within quotes. String values are displayed using italic font if the type has been declared.

To set the data type, right-click on the Input or Output column header and select the required type from the list.



Defining Decision Table Rules

Decision Table rules are defined by specifying input entries and corresponding output entries within the cells of a table row. For 'number' data types, input entries can be specified as a single value, or as a number range, such as '<10', '>100' or '[2..8)'. (When defining number ranges, the use of round brackets indicates that the bounding number is NOT included; use of square brackets indicates the bounding number is included.) Output entries should specify a single value per cell.

Additional rules can be appended to the list of rules by clicking on the icon in the toolbar. Unwanted rules can be deleted from the table by right-clicking on the rule and selecting the option 'Delete Rule Row' from the pop-up menu.

Existing rules can be copied and pasted within the table by first selecting the rules, (use 'Ctrl+Click' to add/remove from selection), then using the menu options 'Copy Rules to Clipboard' and 'Paste Rules from Clipboard' to perform the

copy and paste. The copied rules can then be modified by selecting and editing individual cell entries.

If the 'Allowed Values' field is set for a string or Boolean expression, the Spacebar can be used to display a list of values to select from, as shown in the earlier *Allowed Values - Fast Fill Rows* section.

Rules can also be sorted within the table, either by:

- Clicking the icon on the toolbar, then choosing to either 'Sort By Input' or 'Sort By Output', or
- Right-clicking on individual rules within the table and selecting the 'Move Rule Up' or 'Move Rule Down' option from the pop-up menu

To determine which table rows are selected for output, the *expressions* that are defined by the Input Clauses are evaluated for the given inputs and the *results* of the expressions are then compared against the input entries of the table rows. Where the expression results match the input entries of a table row, that row is selected for output.

The Decision Table's 'Hit Policy' determines how the table's matching rows are then used to produce its output.

Rule Formats

You can select - using a Toolbar icon - to display the Decision Table in one of three formats, as shown here.

Rule-as-Row format, where the rule is developed along rows with the inputs, outputs and annotations set in columns:

(Existing Customer, Application Risk Score)				
U	Existing Customer	Application Risk Score	Pre-Bureau Risk Category	Annotations
	true,false		HIGH, MEDIUM, LOW, VERY LOW, DECLINE	
1	true	<80	DECLINE	Use standard letter DEC0004
2	true	[80..90)	HIGH	
3	true	[90..110]	MEDIUM	
4	true	>110	LOW	
5	false	<100	HIGH	
6	false	[100..120)	MEDIUM	
7	false	[120..130]	LOW	
8	false	>130	VERY LOW	Refer to Loans Officer

Rule-as-Column format, where the rules are developed down columns with the inputs, outputs and annotations set along the rows:

(Existing Customer, Application Risk Score)									
Existing Customer	true,false	true	true	true	true	false	false	false	false
Application Risk Score		<80	[80..90)	[90..110]	>110	<100	[100..120)	[120..130]	>130
Pre-Bureau Risk Category	HIGH, MEDIUM, LOW, VERY LOW, DECLINE	DECLINE	HIGH	MEDIUM	LOW	HIGH	MEDIUM	LOW	VERY LOW
Annotations		Use standard letter DEC0004							Refer to Loans Officer
U		1	2	3	4	5	6	7	8

Rule-as-Crosstab format, where the rules are formed from inputs defined as a set of rows AND a combination of columns, with outputs set in the intersecting cells. (Note that this format hides the 'Annotation' fields):

(Existing Customer, Application Risk Score)								
Pre-Bureau Risk Category		Application Risk Score						
		<80	[80..90)	[90..110]	>110	<100	[100..120)	[120..130]	>130
Existing Customer	false					HIGH	MEDIUM	LOW	VERY LOW
	true	DECLINE	HIGH	MEDIUM	LOW				

At the end of a simulation, in a Crosstab Decision Table, related input entries and output entries are highlighted. For example, in this simulation processing resulted in the output of a 0.10 discount for a Business Customer where Order Size was less than or equal to 10 and delivery was not applicable.

DMN Expression

Discount

Discount		Customer, Delivery			
		Business	Private	Private	Government
		-	sameday	slow	-
OrderSize	<10	0.05	0	0.05	0.15
	>=10	0.10	0	0.05	0.15

Crosstab Settings

In Rule-as-Crosstab format, as the inputs form both rows and columns and the outputs are at the intersections, the steps for setting values are slightly different from those for the other two formats.

1. To add another type of input, right-click on the input column header and select the 'Add Input' option. You are prompted to enter the input name; the input is added as a set of fields under the current column fields.
   To delete an input type from the columns, right-click on its set of fields and select the 'Delete Input' option. The name of the input and its set of fields are removed from the column headings.
2. To add another type of output, right-click on the Output block in the top left of the window and select the 'Add Output' option. You are prompted to enter the Output name; the name is added to the Output block and a new row is added to each cell in the body of the window.
   To delete an Output type, right-click on the type name in the Output block in the top left of the window and select the 'Delete Output' option. The Output name and its fields in the grid are removed.
3. You can rotate between the input types to select one for the row headers. Right-click on the row and click on the 'Select Input as Row Header' option. This displays a list of the input types; click on the type to use as the row header; the other types are combined in the columns.

4. To add a value entry row or column to the inputs, right-click on a current row or column and select the 'Add Input Entry Row' or 'Add Input Entry Column' option, as appropriate. A prompt displays for the name of the input entry; when you enter this, the appropriate row or column is added to the Decision Table.
   To delete a value entry row or column, right-click on it and select the 'Delete Input Entry Row' or 'Delete Input Entry Column' option. The selected row or column is deleted from the table.
5. In the Input columns, each row matches a type of input. If you want to move the row for one type of input above or below another, right-click on it and select the 'Move Input Up' or 'Move Input Down' option. The context menu only provides the options that can be actioned, so as it is not possible to move, say, the last row downwards, the 'Move Input Down' option is not listed.

Toolbar for Decision Table Editor

This table provides descriptions of the features accessible in the DMN Expression window when a Decision Table is selected.

Toolbar Options

Icon	Description
	Save changes to the currently selected Decision or BusinessKnowledgeModel element.
	Toggle the view for the Decision Table between Rule-as-Row, Rule-as-Column and Rule-as-Crosstab. Alternatively, click on the drop-down arrow and select the format you require.
	Click on 'Sort By Input' to sort the rules by input columns; click on 'Sort By Output' to sort the rules by output columns. The columns can be dragged and dropped to organize the sorting order.
	Merge cells of adjacent rules, where the content of the input entries is the same.

	You can edit the content of the merged cells. During simulation, the merged items are highlighted.
	Split input entry cells that have previously been merged.
	Display the 'Edit Parameters' window, where you can specify the names and data types of the parameters that are passed when invoking the decision logic of a BusinessKnowledgeModel element.
	Append an input column to the Decision Table.
	Append an output column to the Decision Table.
	Append an annotation column (with a green heading cell) to the table, in which you can record short notes or comments on the rule. (See the illustrations in the Rule-as-Row/Rule-as-Column row earlier.) You can add more than one annotation column if required, typing in an appropriate column title in each heading cell. To remove an annotation column,

	right-click on it and select the 'Delete Annotation Column' option.
	Append a rule to the Decision Table.
	Show or hide the allowed values fields for the 'Input' and 'Output' columns. The allowed values defined for an input or output will be used for validation and auto completion editing.
	Perform validation of the Decision Table. Enterprise Architect will perform a series of validations to help you discover any errors in the Decision Table.
	This button is enabled when a Decision Table is defined for a BusinessKnowledgeModel element. Select the 'Input Parameter Values for Simulation' tab, complete the fields, then click on this button. The test result will be presented on the Decision Table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted. You can use this functionality to unit test a BusinessKnowledgeModel element, without specifying its context. A number of menu options are available

	for this tool bar button. For more information, see the Help topic *Simulate DMN Model*.

Decision Table Hit Policy

The Hit Policy specifies the result of the Decision Table in cases of overlapping rules. The single character in a particular Decision Table cell indicates the table type and unambiguously reflects the decision logic.

Single Hit Policies:

- **U**nique: no overlap is possible and all rules are disjoint; only a single rule can be matched (this is the default)
- **A**ny: there might be overlap, but all the matching rules show equal output entries for each output, so any match can be used
- **P**riority: multiple rules can match, with different output entries; this policy returns the matching rule with the highest output priority
- **F**irst: multiple (overlapping) rules can match, with different output entries; the first hit by rule order is returned

Multiple Hit Policies:

- **O**utput order: returns all hits in decreasing output priority order
- **R**ule order: returns all hits in rule order
- **C**ollect: returns all hits in arbitrary order; an operator (‘+’, ‘<’, ‘>’, ‘#’) can be added to apply a simple function to the outputs

Collect operators are:

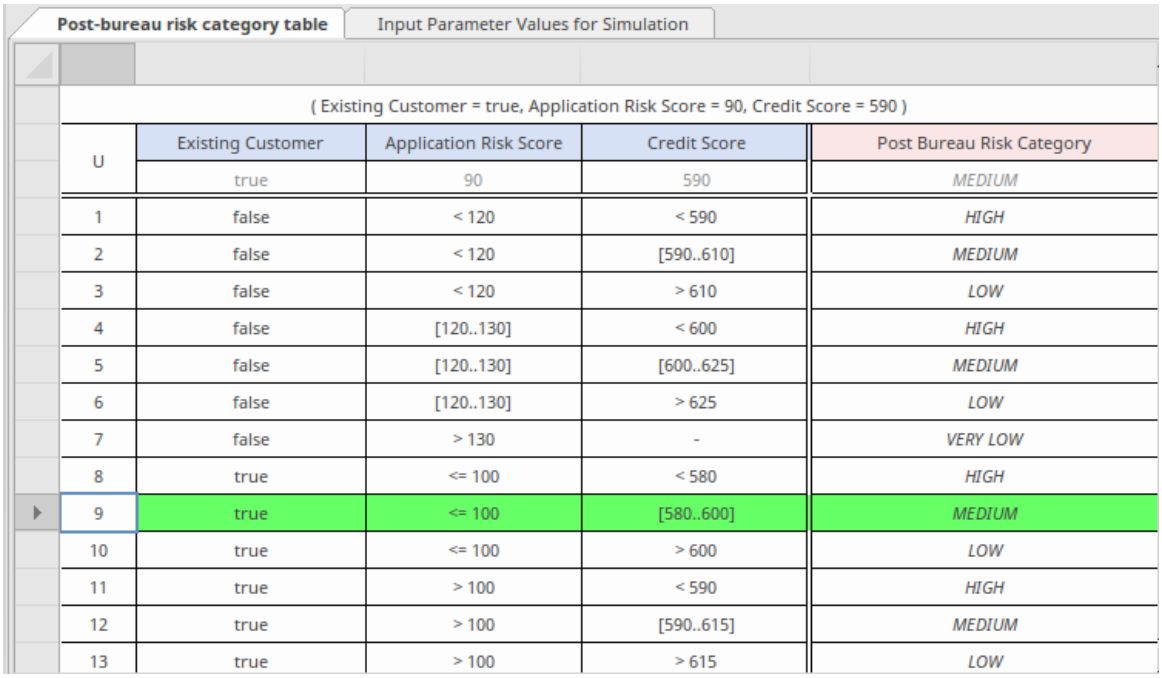
- + (sum): the result of the Decision Table is the sum of all

the distinct outputs

- < (min): the result of the Decision Table is the smallest value of all the outputs
- > (max): the result of the Decision Table is the largest value of all the outputs
- # (count): the result of the Decision Table is the number of distinct outputs

Example of Unique Hit Policy

The 'Unique' Hit Policy is the most popular type for a Decision Table and all rules are disjoint.



Post-bureau risk category table | Input Parameter Values for Simulation

(Existing Customer = true, Application Risk Score = 90, Credit Score = 590)

U	Existing Customer	Application Risk Score	Credit Score	Post Bureau Risk Category
	true	90	590	*MEDIUM*
1	false	< 120	< 590	*HIGH*
2	false	< 120	[590..610]	*MEDIUM*
3	false	< 120	> 610	*LOW*
4	false	[120..130]	< 600	*HIGH*
5	false	[120..130]	[600..625]	*MEDIUM*
6	false	[120..130]	> 625	*LOW*
7	false	> 130	-	*VERY LOW*
8	true	<= 100	< 580	*HIGH*
9	true	<= 100	[580..600]	*MEDIUM*
10	true	<= 100	> 600	*LOW*
11	true	> 100	< 590	*HIGH*
12	true	> 100	[590..615]	*MEDIUM*
13	true	> 100	> 615	*LOW*

Example of Priority Hit Policy

In a table with the 'Priority' Hit Policy, multiple rules can match, with different output entries. This policy returns the matching rule with the highest output priority.

Eligibility rules | Input Parameter Values for Simulation

(Pre-Bureau Risk Category, Pre-Bureau Affordability, Age)

P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
				INELIGIBLE,ELIGIBLE
1	DECLINE	-	-	INELIGIBLE
2	-	false	-	INELIGIBLE
3	-	-	< 18	INELIGIBLE
4	-	-	-	ELIGIBLE

Note: The list of allowable values is used to define the output priority. Here, the allowable values are listed as INELIGIBLE and ELIGIBLE; INELIGIBLE is defined as having a higher priority than ELIGIBLE.

One possible simulation result might resemble this:

Eligibility rules | Input Parameter Values for Simulation

(Pre-Bureau Risk Category = "HIGH", Pre-Bureau Affordability = false, Age = 25)

P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
	HIGH	false	25	INELIGIBLE
1	DECLINE	-	-	INELIGIBLE
2	-	false	-	INELIGIBLE
3	-	-	< 18	INELIGIBLE
4	-	-	-	ELIGIBLE

The matching rules are highlighted, but the output from rule 2 is chosen because INELIGIBLE has higher priority than ELIGIBLE.

Example of Collection-Sum Hit Policy

For a Decision Table with the 'Collect-Sum' (C+) Hit Policy, the result of the Decision Table is the sum of all the distinct outputs.

Application risk score model | Input Parameter Values for Simulation

(Age = 40, Marital Status = "M", Employment Status = "EMPLOYED")

C+	Age	Marital Status	Employment Status	Partial score
	40	"M"	"EMPLOYED"	133
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	"S"	-	25
7	-	"M"	-	45
8	-	-	"UNEMPLOYED"	15
9	-	-	"STUDENT"	18
10	-	-	"EMPLOYED"	45
11	-	-	"SELF-EMPLOYED"	36

In this example, the output Partial Score is calculated as 43 + 45 + 45 = 133

Decision Table Validation

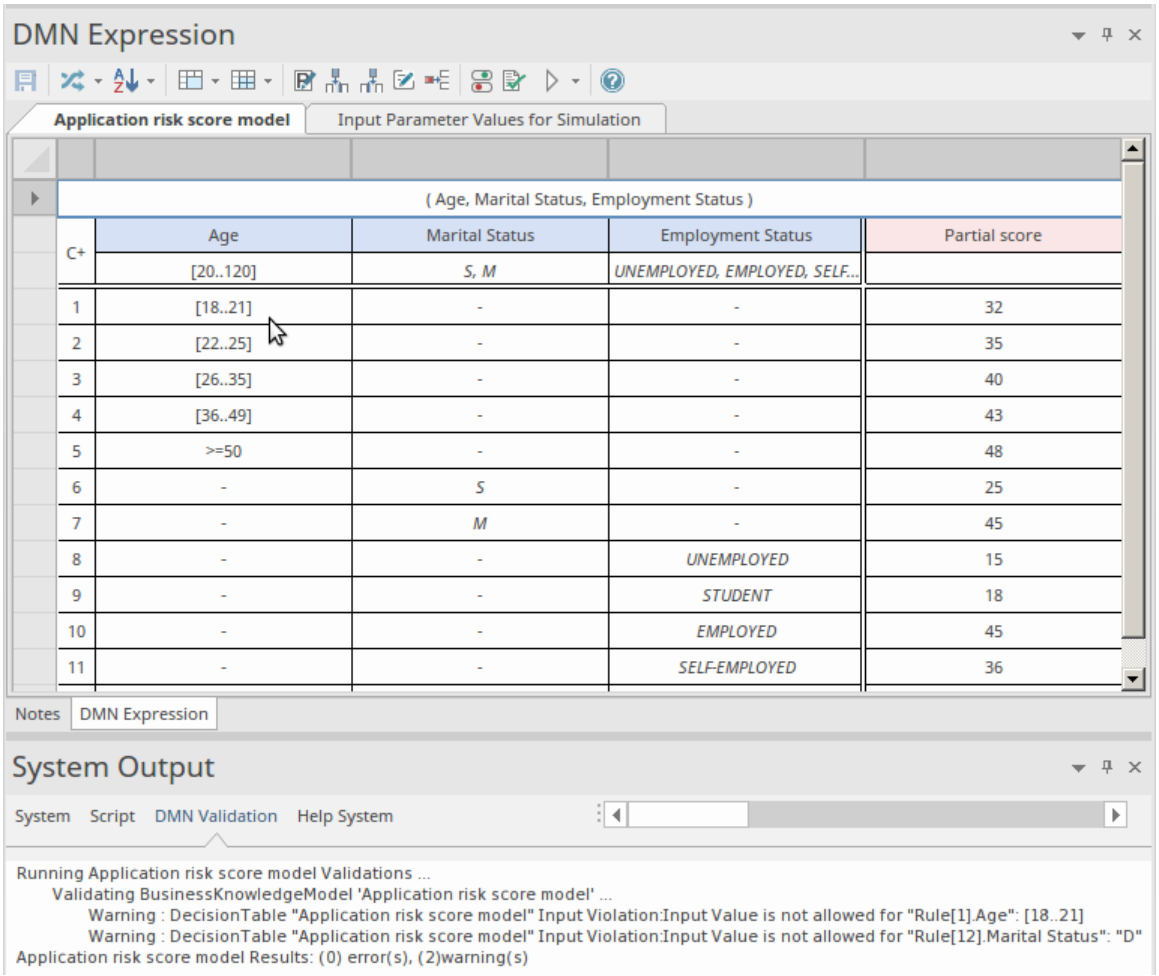
A Decision Table is one of the most common and useful DMN Expressions used to express decision logic. However, modeling a Decision Table can also be complicated, especially if multiple Input Clauses are used in combination for many Decision Table rules. Enterprise Architect provides the facility to validate Decision Tables, as explained in this topic.

Access

DMN Expression Window	Simulate > Decision Analysis > DMN > DMN Expression : Validate button
DMN Simulation Window	Simulate > Decision Analysis > DMN > Open DMN Simulation > Configure : Validate button

Entries out of range detection

It is good practice to define 'allowed values' for the Input Clauses and Output Clauses of a Decision Table. The 'allowed values' list is used to perform range-checking of the input and output entry values for the table rules.



In this example:

- The 'Age' Input Clause defines a range of [20..120]; however, the input entry for rule 1 specifies a range of [18..21]; this is outside the range of allowed values, so rule 1 is reported as invalid
- The 'Marital Status' clause defines its allowed values as an enumeration of 'S, M'; rule 12 specifies a value of 'D', hence that rule is also reported as invalid

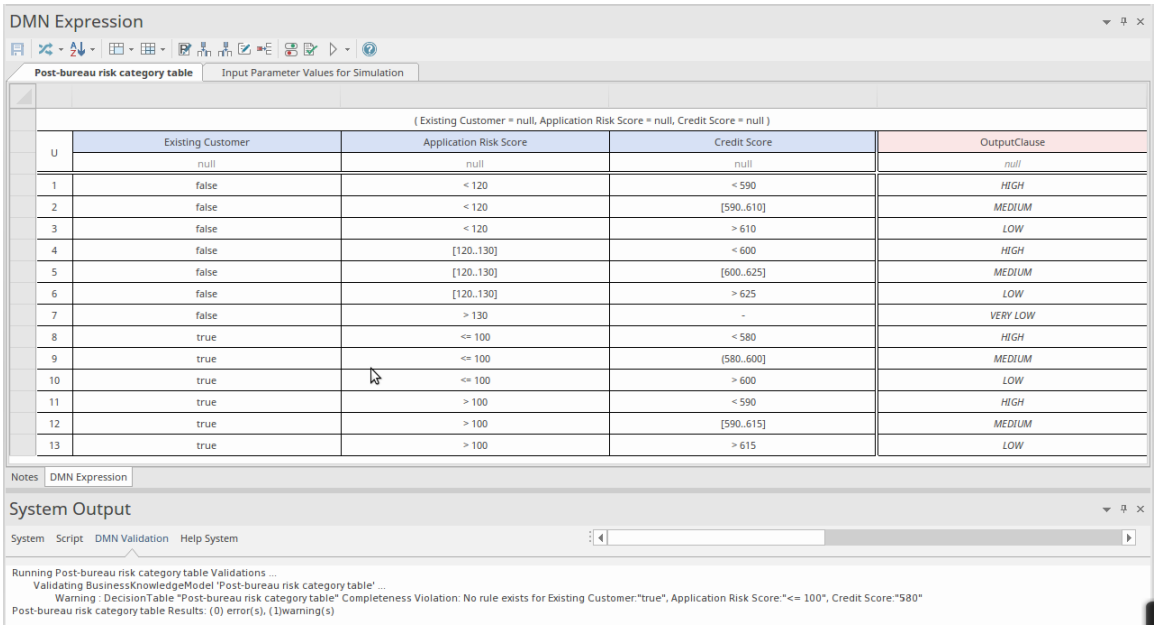
These issues can be corrected, either by updating the 'allowed values' or by modifying the input entries for the invalid rules, depending on the actual business rules.

Completeness detection - report gaps in the rules

The gaps in rules for a Decision Table mean that, given a combination of input values, no rule is matched. This indicates that some logic or rule might be missing (unless a default output is defined).

When the Decision Table contains many rules that specify number ranges, it becomes difficult to visually detect gaps and quite time-consuming to compose and run exhaustive test cases.

For example:



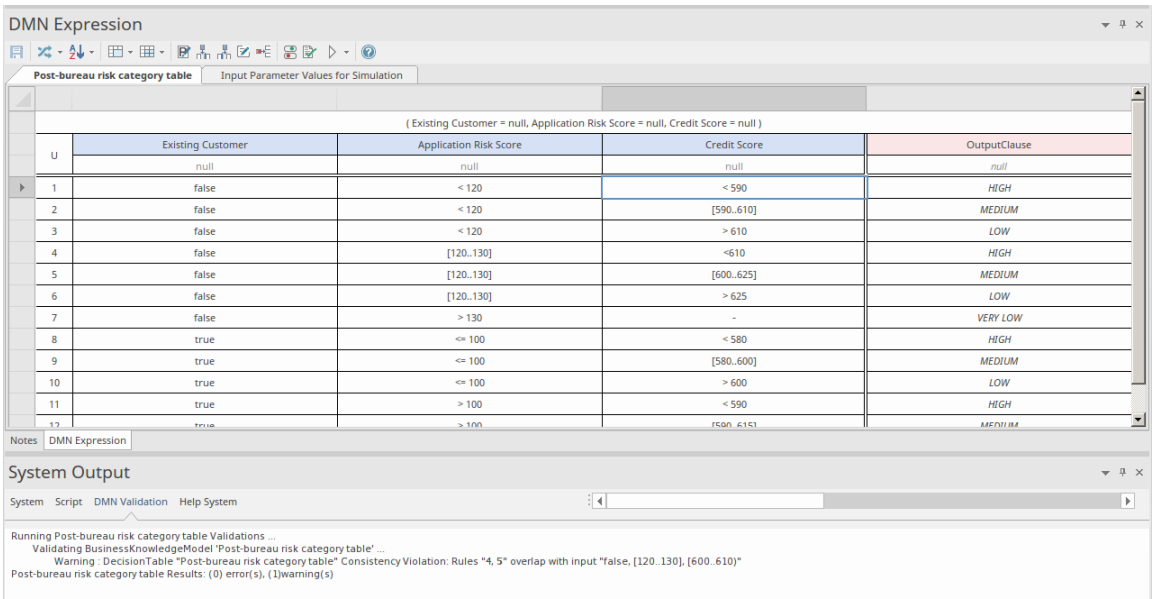
The validation reports a gap in the rules. Closer inspection reveals an error in rule 9. The input entry (580..600], should be [580..600].

Rule Overlap detection for Unique Hit Policy

When rules overlap, for a given combination of input values, multiple rules are matched. This is a violation if the Decision Table specifies its Hit Policy as 'Unique'.

When the Decision Table contains many rules that specify number ranges, it becomes difficult to visually detect gaps and quite time-consuming to compose and run exhaustive test cases.

For example:



The validation reports an overlap in the rules, involving rules 4 & 5. Closer inspection reveals the overlap exists in the third input 'Credit Score', where '<610' overlaps with '[600..625]'. You could correct this issue either by changing rule 4 to '<600' or by changing rule 5 to '[610..625]', to reflect the actual business rules.

Literal Expression

A Literal Expression is the simplest form of DMN expression; it is commonly defined as a single-line statement, or an if-else conditional block. The Literal Expression is a type of value expression used in both Decision elements and Business Knowledge Model (BKM) elements. As the expression becomes more complex, you might prefer a Boxed Context or, in order to improve the readability, you can encapsulate some of the logic as a function in the DMN Library.

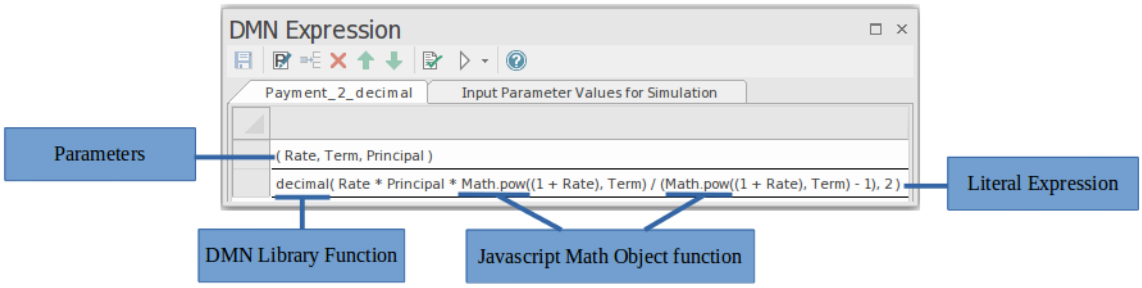
The icon on the top right corner of the Decision or BKM element indicates that it is implemented as a *Literal Expression*.

Access

Diagram	On a diagram, double-click on a Decision element or BusinessKnowledgeModel element. The DMN Expression editor window displays showing details of the selected element.

Overview

This image shows the DMN Expression editor window, as it appears for a Literal Expression.



The Literal Expression is a textual representation of the decision logic. It describes how an output value is derived from its input values, using mathematical and logical operations.

The expression editor window presents the Literal Expression as a table, with two key rows:

- Parameters: defines the input parameters used in the expression
- Literal Expression: where the formula for the expression is defined - this defines the output of the Decision

In order to support simulation and execution, the literal expression can use JavaScript global functions or JavaScript object functions. Users can also create DMN Library functions for use within the expressions.

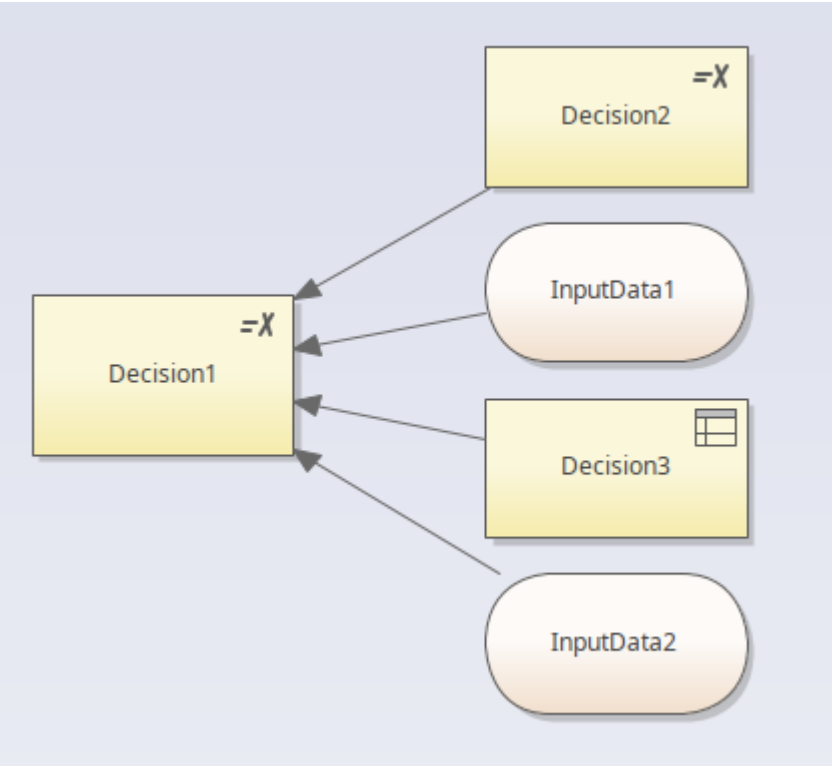
Toolbar for Literal Expression Editor

When a Literal Expression is selected, the layout of features accessible in the DMN Expression window is:

For more details refer to the Help topic *Toolbar for Literal Expression Editor*.

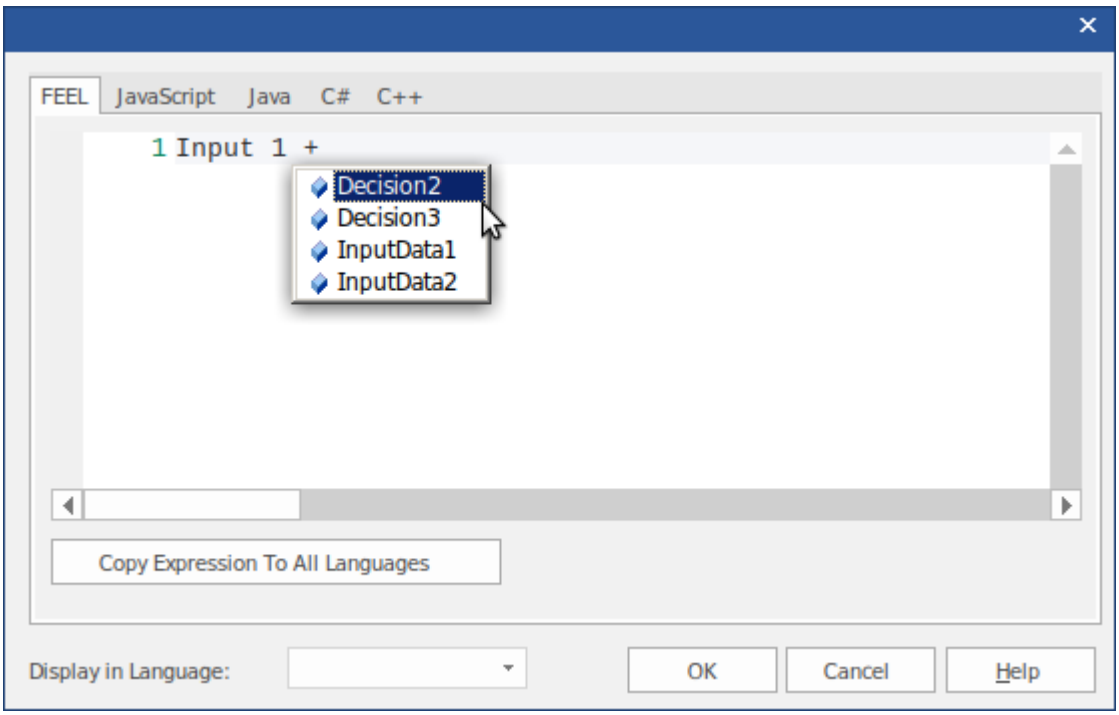
Expression Editor and Intelli-sense support

In accordance with the Friendly Enough Expression Language (FEEL) language specification, parameter names can contain spaces, which makes the expression easier to read. Enterprise Architect also provides Intelli-sense support for editing the expressions, allowing for minimal typing and fewer mistakes.



Given a decision hierarchy such as the one shown, when editing the expression for 'Decision1', the inputs to 'Decision1' - namely 'Decision2', 'Decision3', 'InputData1' and 'InputData2' - will be available through Intelli-sense in the editor.

By right-clicking on the 'Expression' row of the DMN Expression window, then choosing the menu option 'Edit Expressions...', the expression code editor dialog is displayed. Pressing Ctrl+Space displays the Intelli-sense menu:



- For 'Decision' elements, all of the inputs to the decision will be displayed
- For Business Knowledge Model (BKM) elements, all of the input parameters will be displayed

The DMN Model can be generated as source code in JavaScript, Java, C# or C++; since some languages might have different syntax for some expressions, Enterprise Architect provides language override pages for each language. If no override code is specified for a language, the expression defined for the FEEL language will be used.

In the generated code, the space inside a variable name will be replaced by an underscore.

Toolbar for Literal Expression Editor

When a Literal Expression is selected, the DMN Expression window displays a toolbar specific to that type of expression.

Toolbar Options

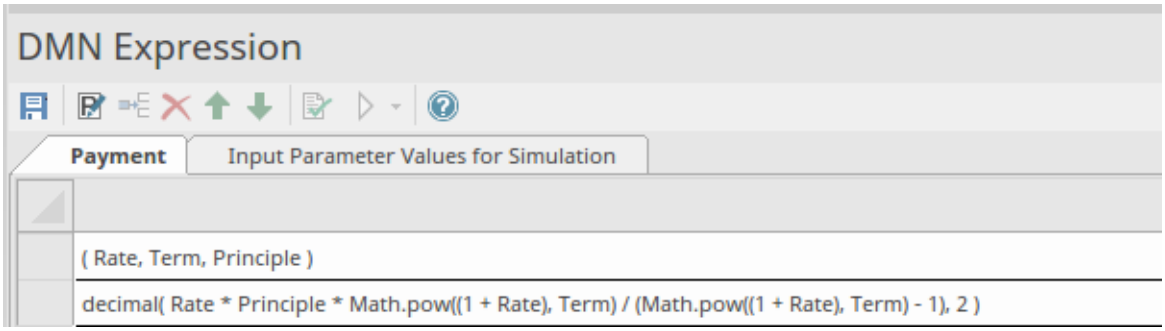
This table provides descriptions of the features accessible from the toolbar in the DMN Expression window when a Literal Expression is selected.

Options	Description
	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
	Click on this button to edit parameters for the Business Knowledge Model.
	This option is disabled for Literal Expressions.
	This option is disabled for Literal Expressions.
	This option is disabled for Literal Expressions.

	This option is disabled for Literal Expressions.
	Click on this button to perform validation of the Literal Expression. Enterprise Architect will perform a series of validations to help you locate any errors in the Expression.
	This button is enabled when the Literal Expression is defined for a BusinessKnowledgeModel element.

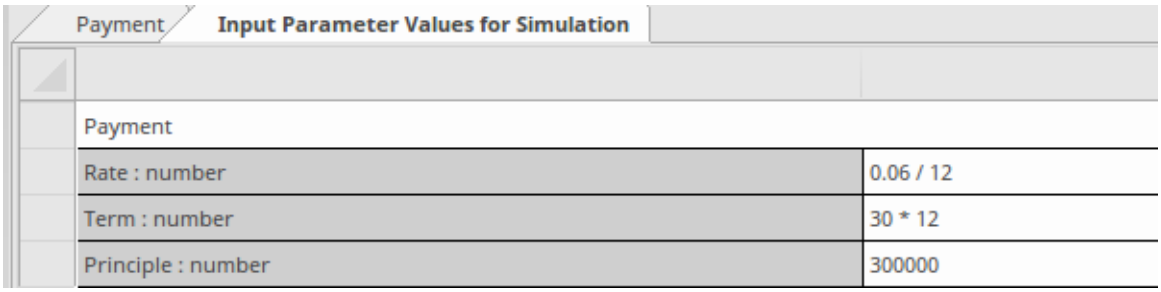
Example - Loan Repayment

This Business Knowledge Model (BKM) *Payment* is implemented as a Literal Expression.

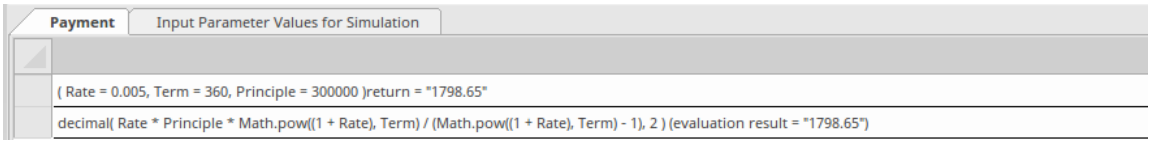


- The BKM defines three parameters: Rate, Term and Principle

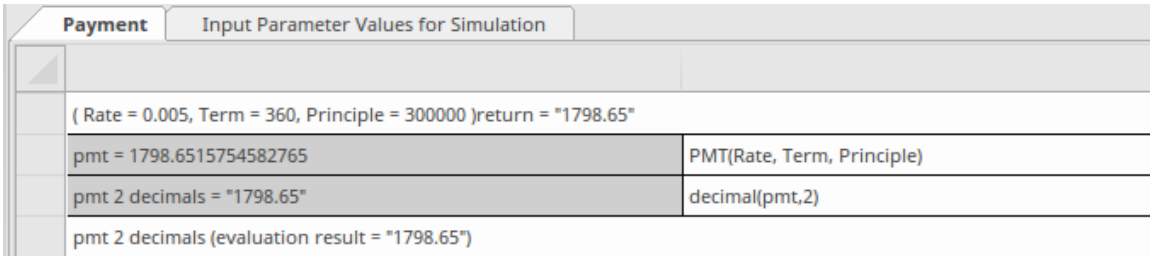
Set the values for the Input Parameters and evaluate the model:



- The runtime parameter value will be displayed; for example, Rate = 00.005
- The BKM's result will be evaluated by the literal expression and the value is displayed on the declaration line; for example, return = 1798.65

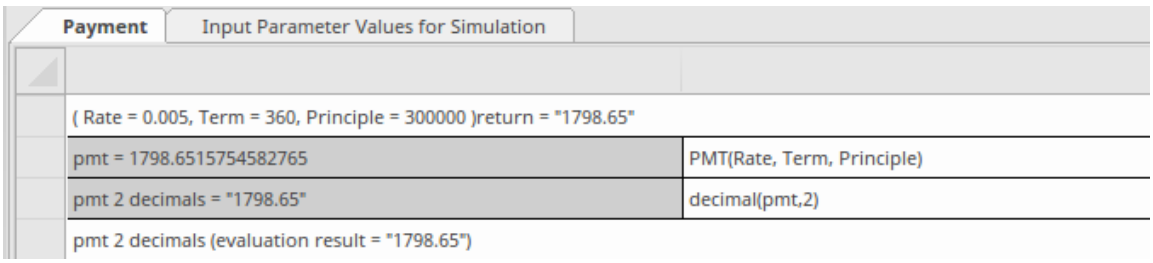


Although the formula for this can be written in one line, it is quite complicated. We can re-factor this model with Built-In function and Boxed Context to improve readability:



- The Boxed Context defines two variable-expression paired entries; these variables serve as 'local variables', which can be used in later expressions
- Return value: the expression can use the value of 'local variables'
- Any expressions in a Boxed Context can use built-in functions that are defined in the customizable Template — *DMN Library*; for example, functions PMT(...) and decimal(...) are used in this example

The simulation result is exactly the same as a Literal Expression:



Boxed Context

A Boxed Context is a collection of context entries, presented in the form of a table, followed by a final result expression.

These context entries consist of a variable paired with a value expression and can be thought of as intermediate results. This allows for complex expressions to be decomposed into a series of simple expressions, with the final result being evaluated in a much simpler form.

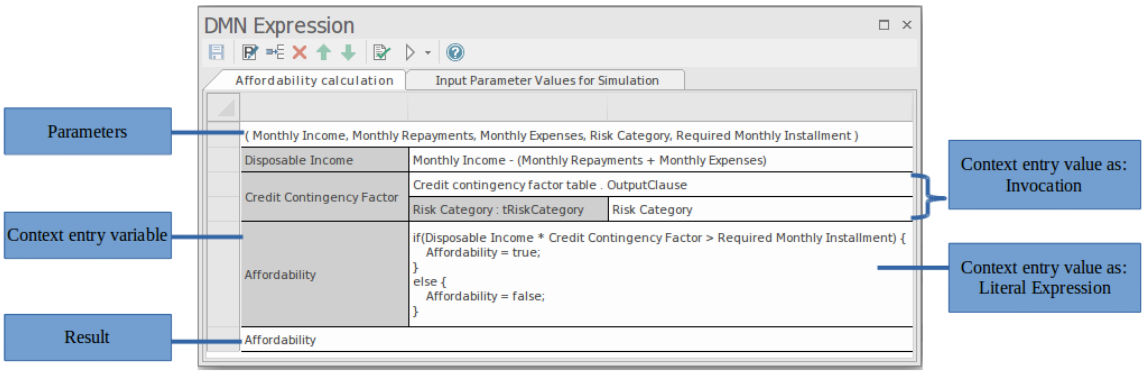
The Boxed Context type is supported in both the *Decision* and the *Business Knowledge Model* element types. It is denoted by the icon.

Access

Diagram	On a diagram, double-click on a Decision element or BKM element. The DMN Expression editor window is displayed, showing details for the selected element.

Overview

This image shows the DMN Expression editor window as it appears for a Boxed Context expression.



A Boxed Context is a collection of context entries, presented in the form of a table, followed by a final result expression. Each context entry consists of a variable and a value expression. The variable can be considered as an intermediate result, and it can be used within the value expression of any subsequent context entry. The value expression of a context entry can be either a Literal Expression or an Invocation, and can make use of any available inputs such as parameters (to a BKM element), InputData or decision results, as well as any previously defined context variables.

The final result of a Boxed Context expression is determined by working through each context entry in turn, evaluating the value expression and assigning its result to the variable, then finally evaluating the result expression. The result expression can also make use of any input or local variable, but must evaluate to provide a result.

Toolbar for Boxed Context Editor

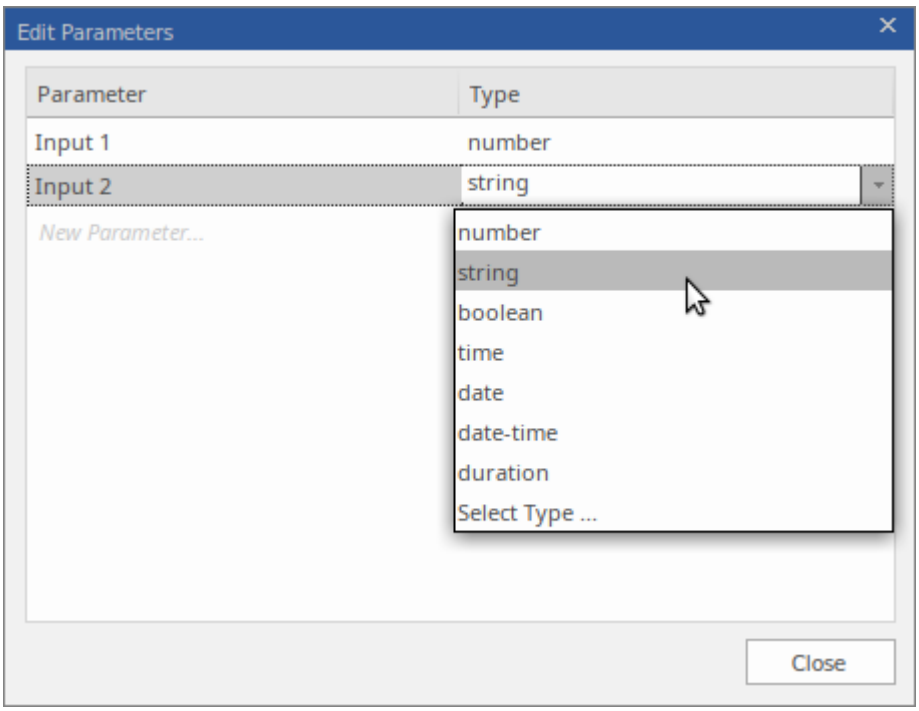
When a Boxed Context expression is selected, the layout of features accessible in the DMN Expression window is:

For more details refer to the Help topic '*Toolbar for Boxed Context Editor*'.

Specifying Parameters

In the case of BusinessKnowledgeModel elements, parameters are used to pass input values supplied by the invoking element. The BKM's decision logic is evaluated using the input parameters and the result is returned to the invoking element. By default, a BKM element is created with two input parameters, 'Input 1' and 'Input 2'.

Click on the icon in the toolbar of the DMN Expression window to display the 'Edit Parameters' window.



Here you can change the parameter names, set their data types, create additional parameters or delete existing ones.

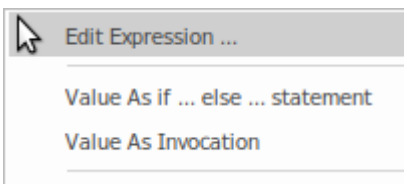
Specifying Context Entries

Each context entry consists of a variable-expression pair.

The variable name can be any text that you like and can even contain spaces. To edit the variable name, click on the cell to select it, then click again or press F2 to enter edit mode. To exit edit mode, click elsewhere or press the Enter key.

In general, it is not necessary to specify a data type for the expression or variables - the type will be inferred from the value. However, if you intend to generate code for compiled languages such as Java, C++ or C#, you will have to specify the type of all context entry variables.

The value expression of a context entry can be either a Literal Expression or an Invocation and can make use of any available inputs, such as parameters (to a Business Knowledge Model element), InputData or decision results, as well as any previously defined context variables. Right-clicking on the expression cell displays a pop-up menu that provides options for displaying an expression code editor, or for setting the value expression as an If-Else statement or an Invocation.



You can also edit the value expression by entering text directly into the expression cell.

For further information on how to specify Literal Expressions or Invocations, please see the Help topics covering those subjects.

Toolbar for Boxed Context Editor

This table provides descriptions of the features accessible in the DMN Expression window when a Boxed Context is selected.

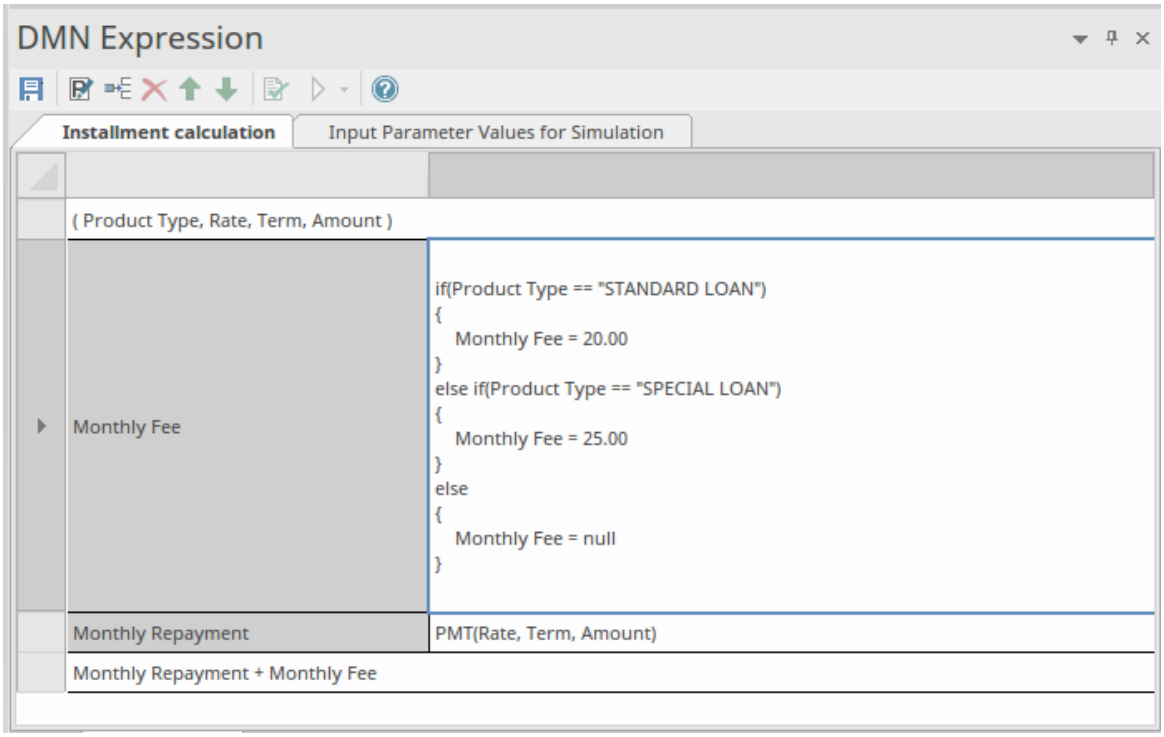
Toolbar Options

This toolbar is for Boxed Context.

Options	Description
	Save changes to the currently selected Decision or BusinessKnowledgeModel element.
	Display the 'Edit Parameters' window, where you can specify the name and data type of each parameter that is passed when invoking the decision logic of a BusinessKnowledgeModel element.
	Create a new context entry and append it to the list of context entries.
	Delete the currently selected context entry.
	Move the currently selected context entry

	up one position in the list.
	Move the currently selected context entry down one position in the list.
	Perform validation of the BoxedContext. Enterprise Architect will perform a series of validations to help you discover any errors in the BoxedContext definition.
	This button is enabled when a Decision Table is defined for a Business Knowledge Model element. Select the 'Input Parameter Values for Simulation' tab, complete the fields, then click on this button. The test result will be presented on the Decision Table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted. You can use this functionality to unit test a BusinessKnowledgeModel element, without specifying its context. A number of menu options are available for this tool bar button. For more information, see the Help topic *Simulate DMN Model.*

Example - Loan Installment Calculation



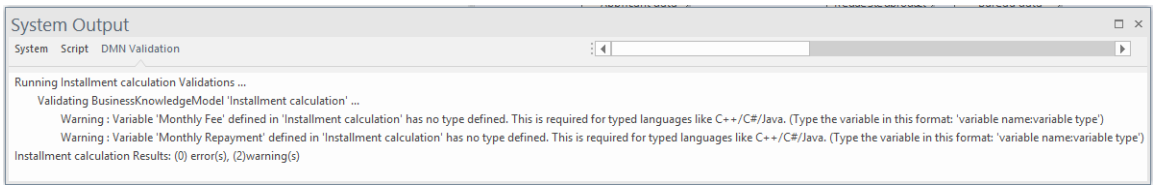
The Business Knowledge Model (BKM) *Installment calculation* is implemented as Boxed Context.

- The BKM defines four parameters: Product Type, Rate, Term and Amount
- The Boxed Context defines two variable-expression pair entries; these variables serve as 'local variables' that can be used in later expressions
- Return value: The expression can use the value of 'local variables'
- Any expressions in a Boxed Context can use built-in functions, which are defined in the customizable Template — *DMN Library*; the functions PMT(...) and decimal(...) are used in this example

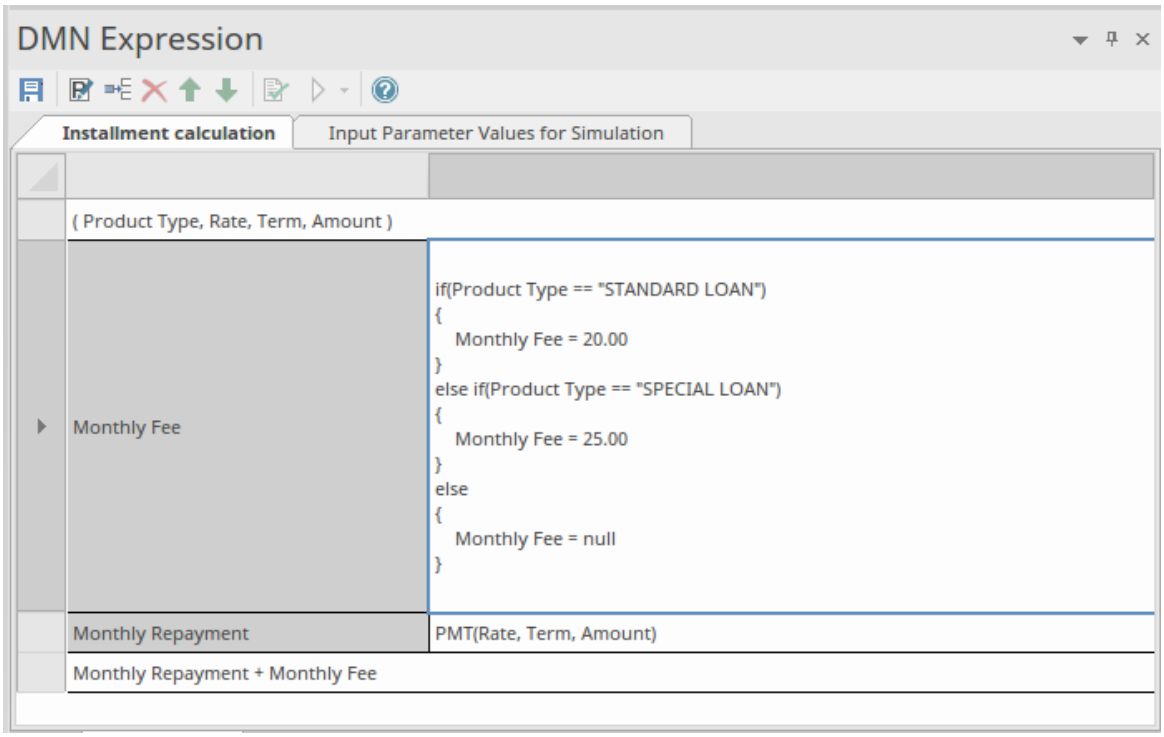
Specify Type to Context Entry Variable

In general, the expression and variables do not have to specify a type, which is inferred from the value provided. This feature is supported generically by JavaScript, which is used for Enterprise Architect's DMN Simulation.

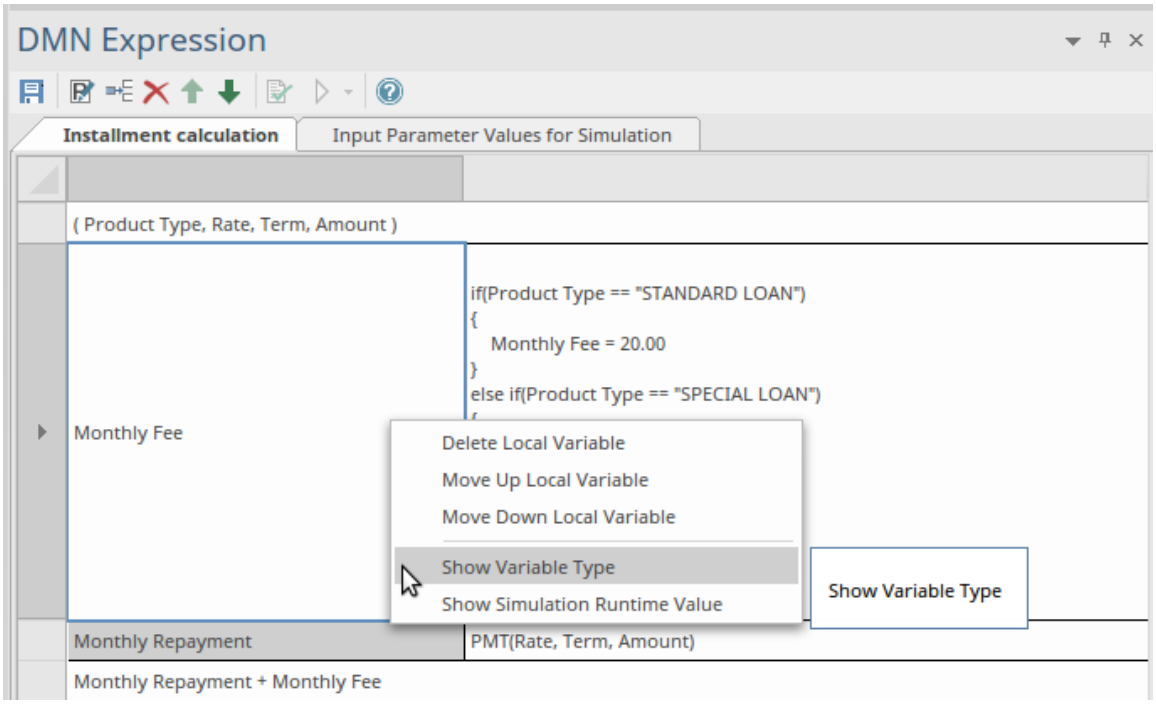
However, if you want to generate code from a DMN model to compiled languages such as Java, C++ or C#, you will have to specify the type for each Context Entry Variable. Otherwise, if you validate the model, you will see warnings such as:



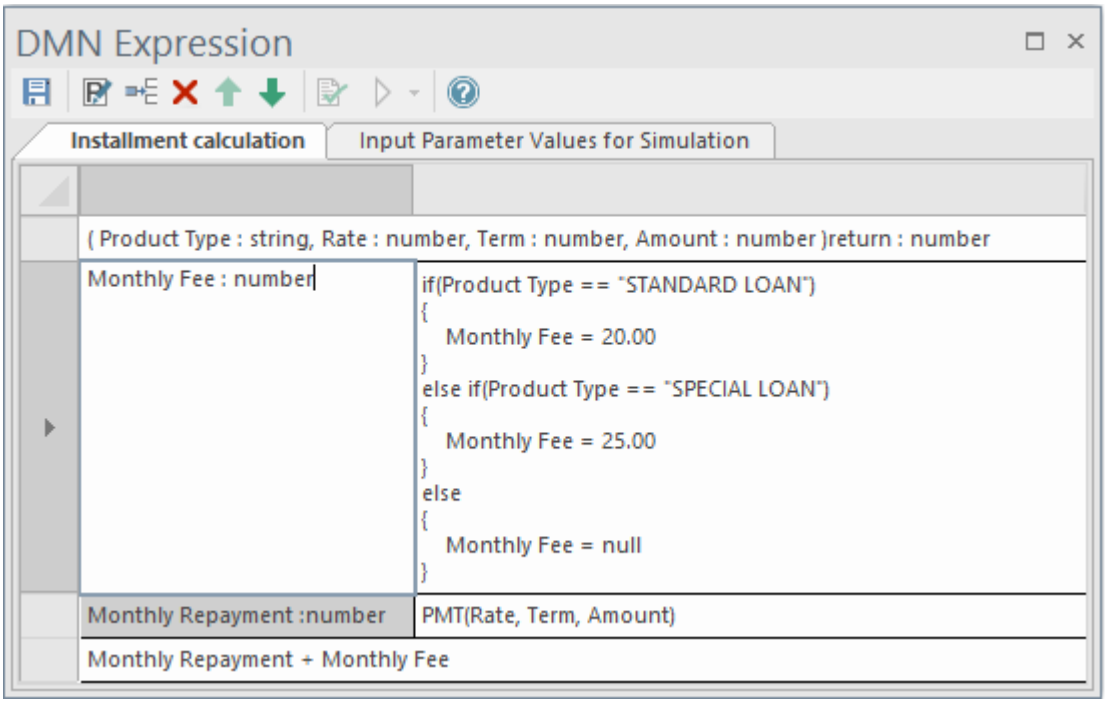
Right-click on the Context Entry Variable (Monthly Fee, Monthly Repayment) in this model.



Select the 'Show Variable Type' option.



Now type in the variable type, appending it to the variable name and separated by a colon, as shown here.

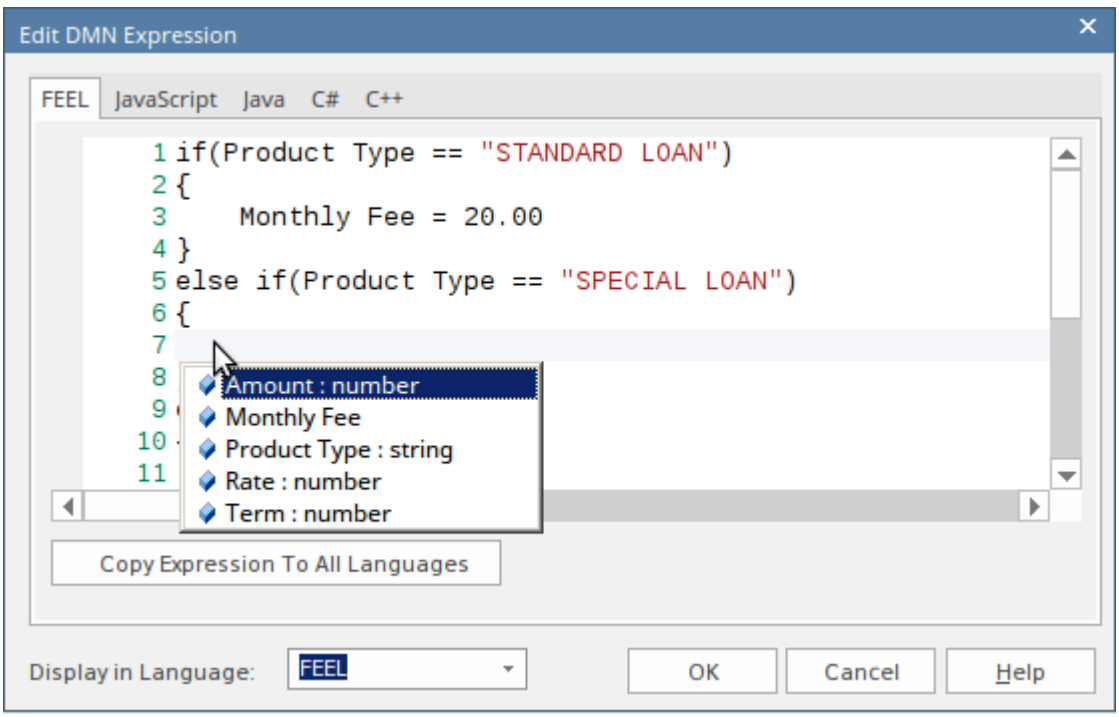


Then click on the Save button on the toolbar to save the expression, and click on the button to validate the model again.

Expression Editor and Intelli-sense Support

The parameter and Context Entry's variable name can contain spaces, according to the FEEL language specification. This feature makes the expression easy to read. In order to help you edit the expressions with less typing and making fewer mistakes, Enterprise Architect provides Intelli-sense support for editing expressions:

To edit an expression, right-click on the expression (in the right-hand field) and select the 'Edit Expressions' menu option. The 'Expression' dialog displays. Click on the required line and press Ctrl+Space to show the Intelli-sense menu:



- All the Context Entry Variables earlier than the current one will be included (the Context Entry Variables later than the current one are excluded)
- For a Business Knowledge Model (BKM), all the

parameters will be included

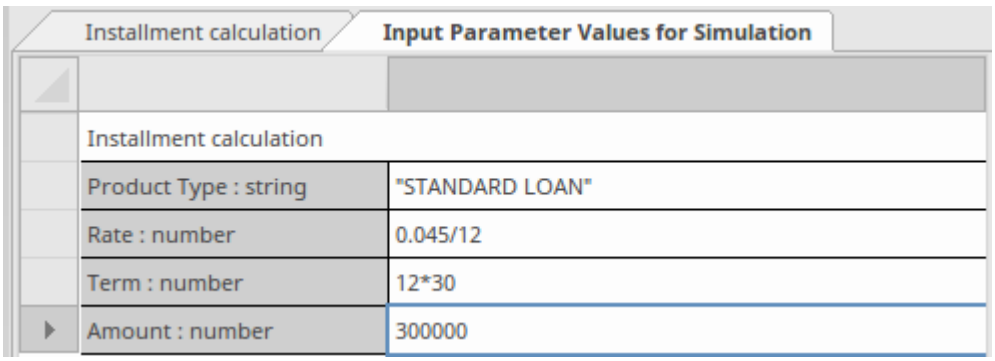
- For a Decision, all the required Decisions will be included

The DMN model can be generated as source code for JavaScript, Java, C# and C++. Since some languages might have different syntax for some expressions, Enterprise Architect provides language override pages for each language. If no override code is specified for a language, the expression defined for the FEEL language will be used.

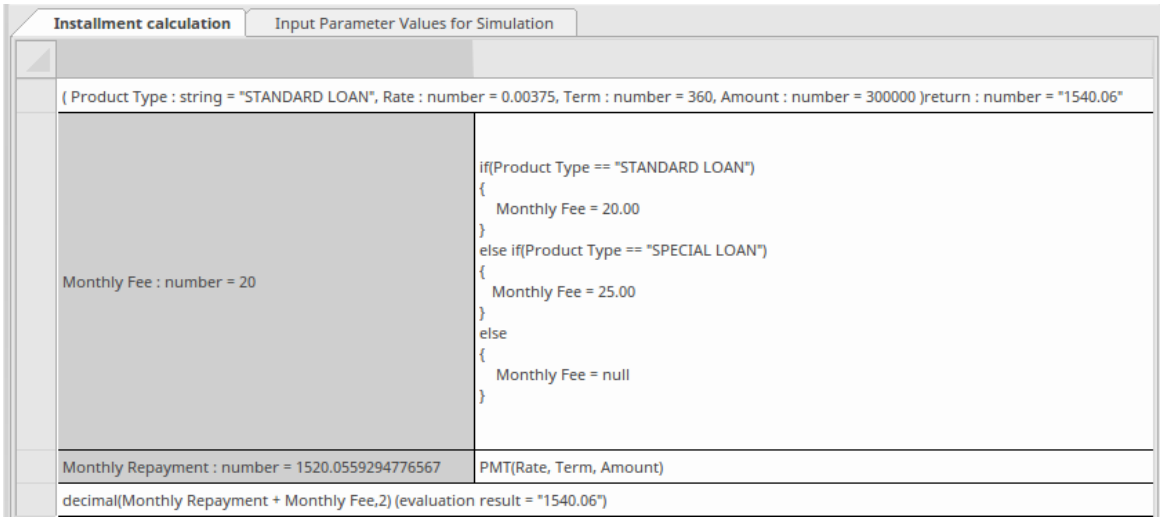
In the generated code, the space inside a variable name will be replaced by an underscore.

Simulation of Business Knowledge Model

Select the 'Input Parameter Values for Simulation' tab and complete each field.



Click on the Save button and then on the Simulation button on the toolbar; the test result will be presented in the Boxed Context expression.



- The runtime parameter value will be displayed; for example, 'Rate = 0.00375'
- The 'Context Entry' variable's runtime value will be displayed; for example, 'Monthly Repayment = 1520.06'
- The Business Knowledge Model (BKM)'s result will be evaluated by the last entry and the values displayed on the declaration line; for example, 'return = 1540.06'

You can use this functionality to unit test a BKM without knowing the context so that later on it can be invoked by a Decision or another BKM.

Boxed List

A DMN Boxed List is a Decision element that contains a list of boxed expressions. These items are arranged as a vertical list in the DMN Expression window.

A Boxed List is often used in conjunction with a *for loop* expression that is contained in a related Decision element. The *for loop* expression is used to iterate over each row in the Boxed List, binding the List's Element-field to the corresponding variable, and evaluating the expression in the scope. The output of the *for loop* is a list containing the evaluation of the expression for each individual iteration.

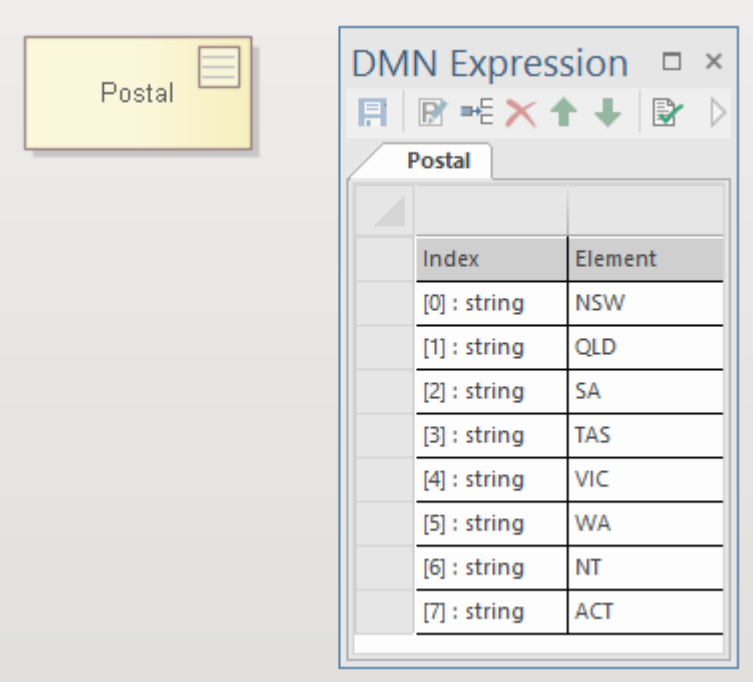
Access

Diagram Toolbox	Drag a Decision element or BKM element from the Toolbox onto a DMN diagram, and select 'List' from the pop-up expression menu. Double-click on the DMN element; the DMN Expression window is displayed, showing details of the selected element.
Properties	Right-click on a DMN Decision or BKM element on the diagram, and select the 'Properties | Properties' menu option. On the General page, select the 'Tags' tab,

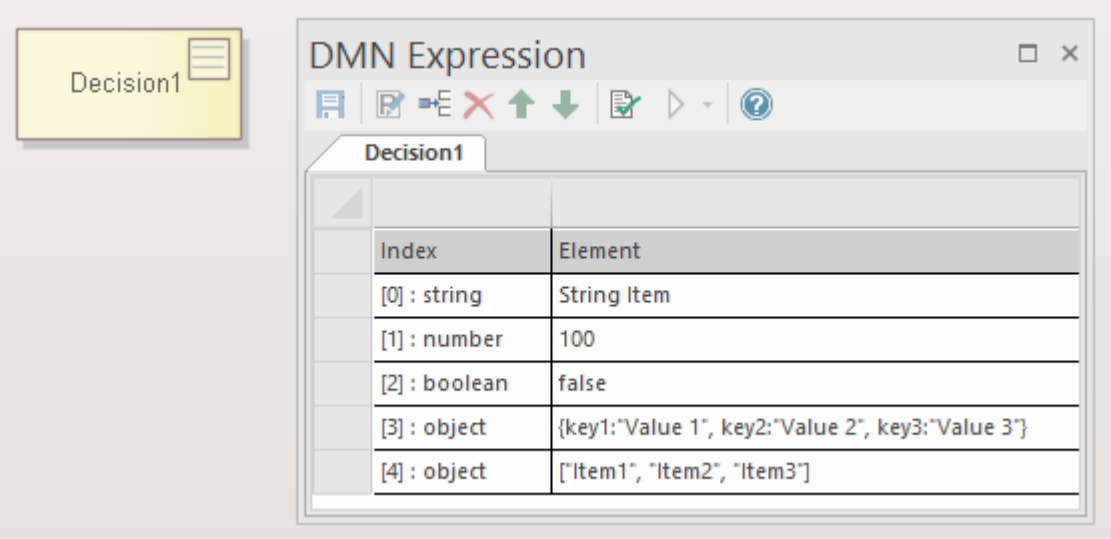
	and in the 'expressionType' value field click on the drop-down arrow and select 'List'. Click on the OK button. Double click on the DMN element; the DMN Expression window is displayed, showing details for the selected element.

Overview

It is common for Boxed Lists to be used as Enumerations, where all List items are of the same type.



It is also common for Boxed Lists to be used as a collection of data, where each item might have a different type.



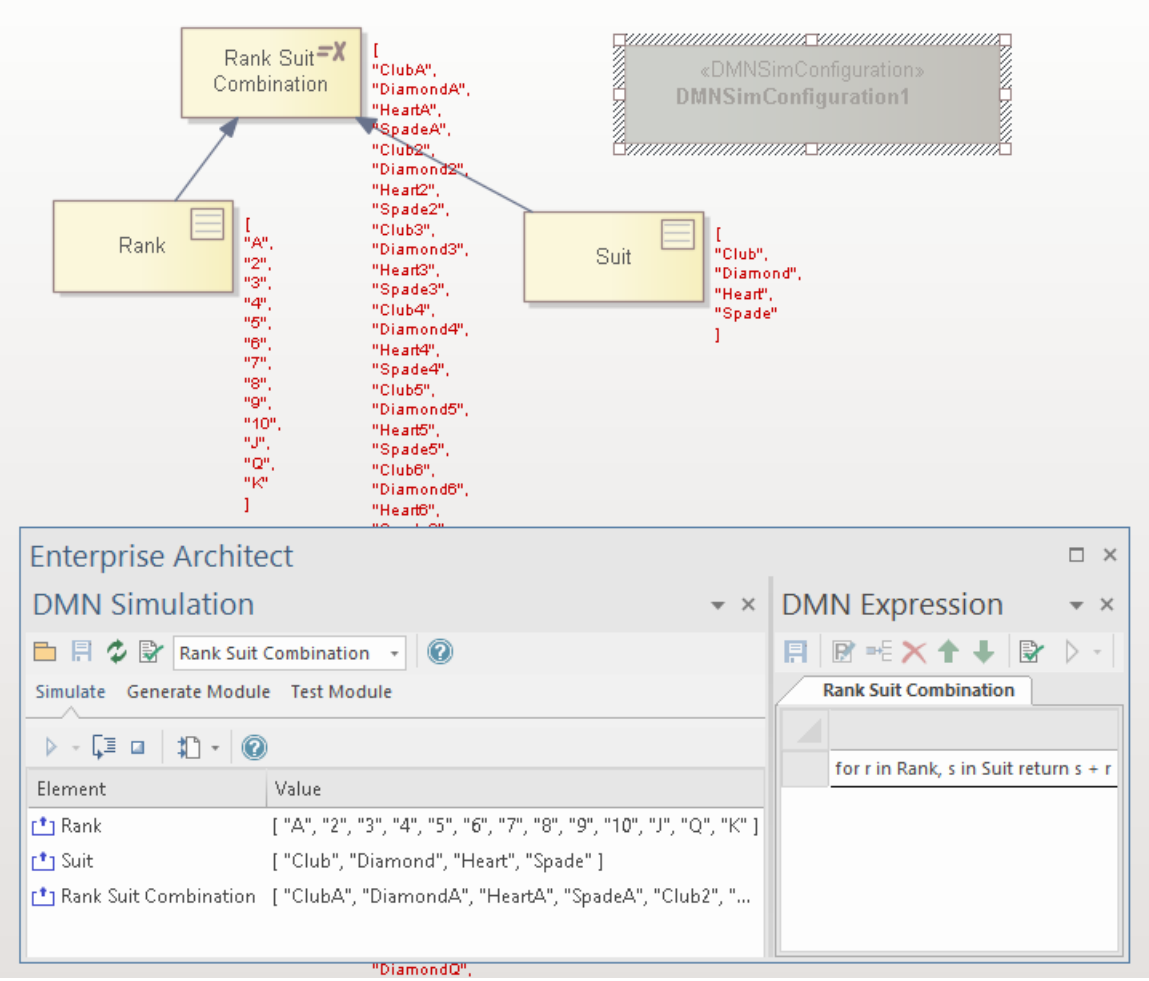
Editing Boxed Lists

The DMN Expression window has a toolbar providing the options 'Add New List Item', 'Delete Existing List Item' and 'Move Item Up or Down'.

Right-click on a List item to display context menu options for setting the List item's type - string, number, boolean or object.

Example - Poker Rank and Suit

In this example, we have three Decisions: Rank, Suit and Rank Suit Combination



- Decision Rank is represented as a Boxed List with 13 Items from 'A' through to 'K'
- Decision Suit is represented as a Boxed List with 4 Items: 'Club', 'Diamond', 'Heart' and 'Spade'
- Decision Rank Suit Combination is represented as a Literal Expression with a *for* loop: *for r in Rank, s in Suit return s + r*

When multiple iteration contexts are defined in the same *for* loop expression, the resulting iteration is a cross-product of the elements of the iteration contexts. The iteration order is from the inner iteration context to the outer iteration context.

In this example, the cross-product of Rank (13 items) and Suit (4 items) is a list of 13 * 4 = 52 elements.

Relation

A DMN Decision Relation element provides a convenient shorthand method for defining a list of related values in a DMN diagram. A Decision Relation is like a relation table with columns and rows. The header of the grid displays the name of each column. Each row displays the set of values for the corresponding columns.

Access

Toolbox	To create a Decision Relation: • Ensure the Perspective is set to: Requirements > Decision Modeling • From the DMN Components page of the Diagram Toolbox, drag a Decision element or BKM element onto a DMN diagram • Select 'Relation' from the pop-up expression menu • Double-click on the DMN element to display the DMN Expression window.

Alternatively, you can change an existing DMN Decision or BKM element to a Decision-Relation type. To do this:

- Right-click on a DMN Decision or BKM element on the diagram, and select the 'Properties | Properties' menu option
- On the General page, select the 'Tags' tab, and in the 'expressionType' value field click on the drop-down arrow and select 'Relation'; click on the OK button

Overview

The DMN *Relation* Decision-type is a vertical list containing rows of values. A key to using the Decision Relation is the means to iterate through the rows of values using a *For Loop*. The For Loop can be defined in, say, a related Literal Expression Decision as a formula to process the rows in the Decision-Relation element.

Editing DMN Relations

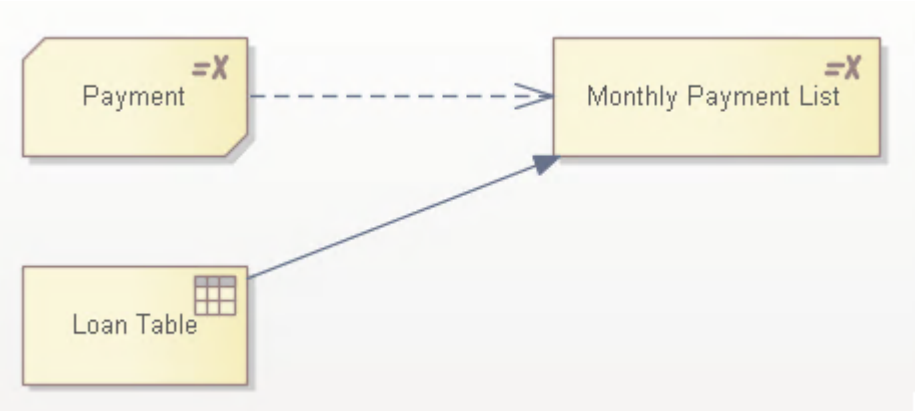
The DMN Expression window has a toolbar providing options to add a new row, delete an existing row and move the selected row up or down.

You can also:

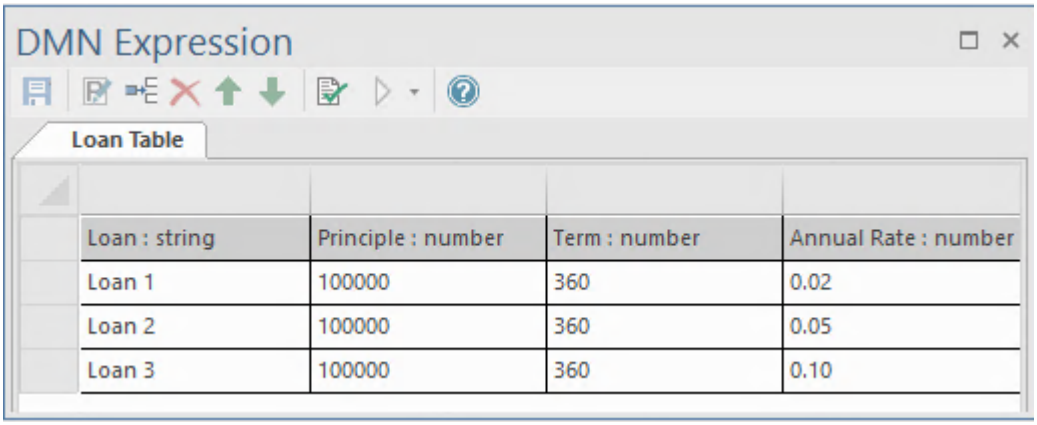
- Drag the grid header to reposition columns.
- Right-click on the header cells to display the context menu options for setting the column's Type to: 'string', 'number', 'boolean' or 'object'.

Example - Loan Table

In this example, we have two Decisions - 'Loan Table' and 'Monthly Payment List' - and one Business Knowledge Model - 'Payment'.

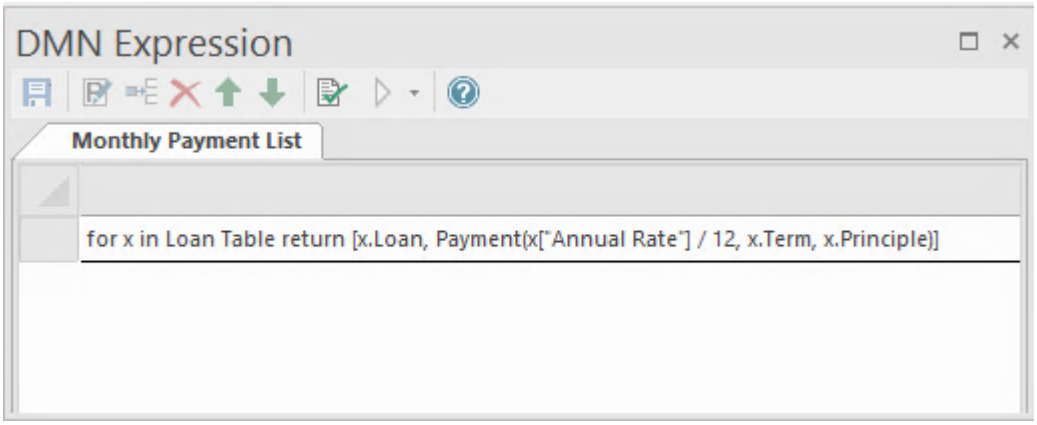


'Loan Table' is a Decision implemented as a Relation with four columns: 'Loan', 'Principle', 'Term' and 'Annual Rate'.



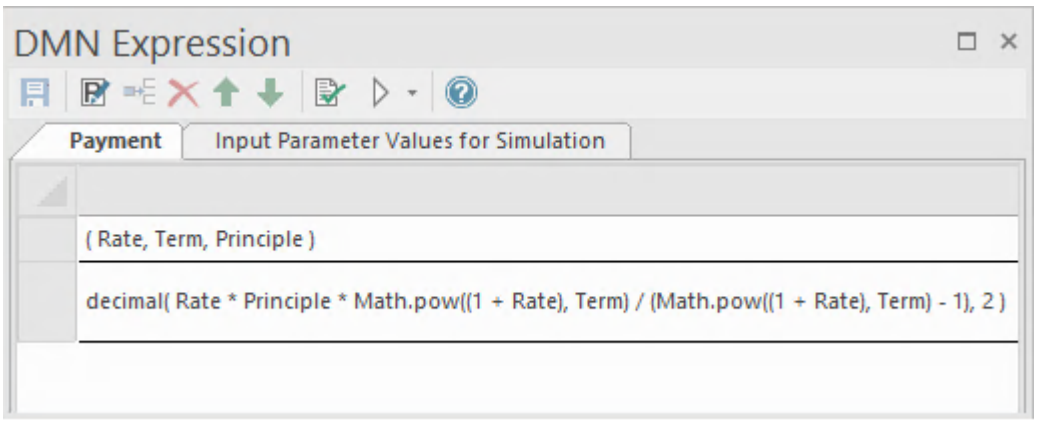
Loan : string	Principle : number	Term : number	Annual Rate : number
Loan 1	100000	360	0.02
Loan 2	100000	360	0.05
Loan 3	100000	360	0.10

'Monthly Payment List' is a Decision implemented as a Literal Expression with a *for* loop:

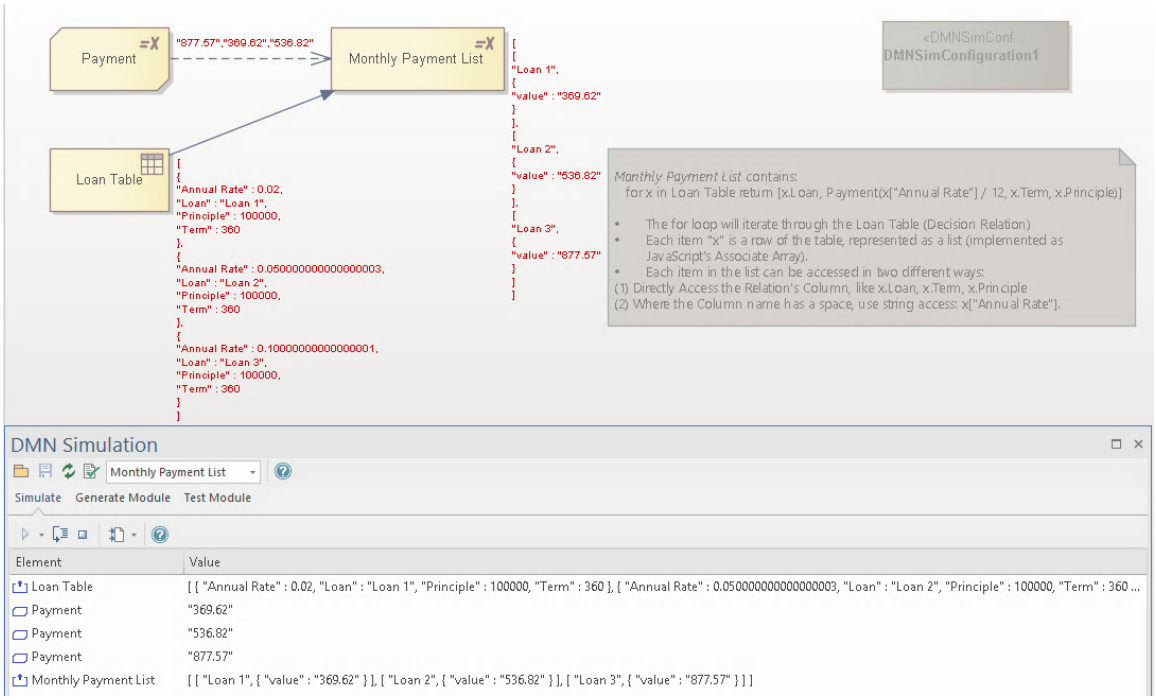


The *for* loop will iterate through 'Loan Table':

- Each item 'x' is a row of the table, represented as a list
- With each row item 'x', invoke Business Knowledge Model 'Payment' with the items in the row list
- Each item in the list can be accessed in two different ways:
  (1) Directly access the Relation's column, like x.Loan, x.Term, x.Principle
  (2) Where the column name has a space, use string access: x["Annual Rate"]



On simulation, the runtime values show in the Simulation window and on the diagram beside the element; you can click on a step to see the simulation process.


Payment
"877.57","369.62","536.82"
Monthly Payment List
Loan Table
«DMNSimConf...
DMNSimConfiguration1
Monthly Payment List contains:
for x in Loan Table return [x.Loan, Payment(x["Annual Rate"] / 12, x.Term, x.Principle)]
The for loop will iterate through the Loan Table (Decision Relation)
Each item "x" is a row of the table, represented as a list (implemented as JavaScript's Associate Array).
Each item in the list can be accessed in two different ways:
(1) Directly Access the Relation's Column, like x.Loan, x.Term, x.Principle
(2) Where the Column name has a space, use string access: x["Annual Rate"].
DMN Simulation
Monthly Payment List
Simulate
Generate Module
Test Module
Element
Value
Loan Table
[{ "Annual Rate" : 0.02, "Loan" : "Loan 1", "Principle" : 100000, "Term" : 360 }, { "Annual Rate" : 0.05000000000000003, "Loan" : "Loan 2", "Principle" : 100000, "Term" : 360 ...
Payment
"369.62"
Payment
"536.82"
Payment
"877.57"
Monthly Payment List
[["Loan 1", { "value" : "369.62" }], ["Loan 2", { "value" : "536.82" }], ["Loan 3", { "value" : "877.57" }]]

Invocation

An invocation is a container for the parameter bindings that provide the context for the evaluation of the body of a Business Knowledge Model. There are two common use cases for an Invocation:

- Bind an Input Data to the Business Knowledge Model
- Bind parameters or context entry variables to the Business Knowledge Model

An example of each is provided in the sub-topics of this Help topic.

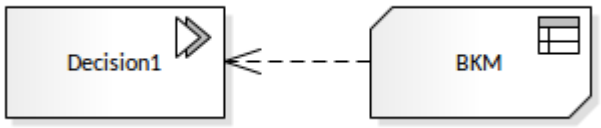
Access

Diagram	Double-click on the appropriate Decision element or BKM element. The DMN Expression window displays, showing details for the selected element.

Overview

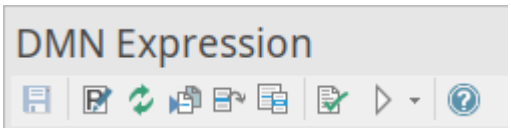
An Invocation is a type of value expression applicable to both Decision elements and Business Knowledge Model elements. It is a tabular representation of how decision logic defined within an invocable element (a Business Knowledge

Model or a Decision Service) is invoked by a Decision or by another Business Knowledge Model.



Toolbar for Invocation Editor

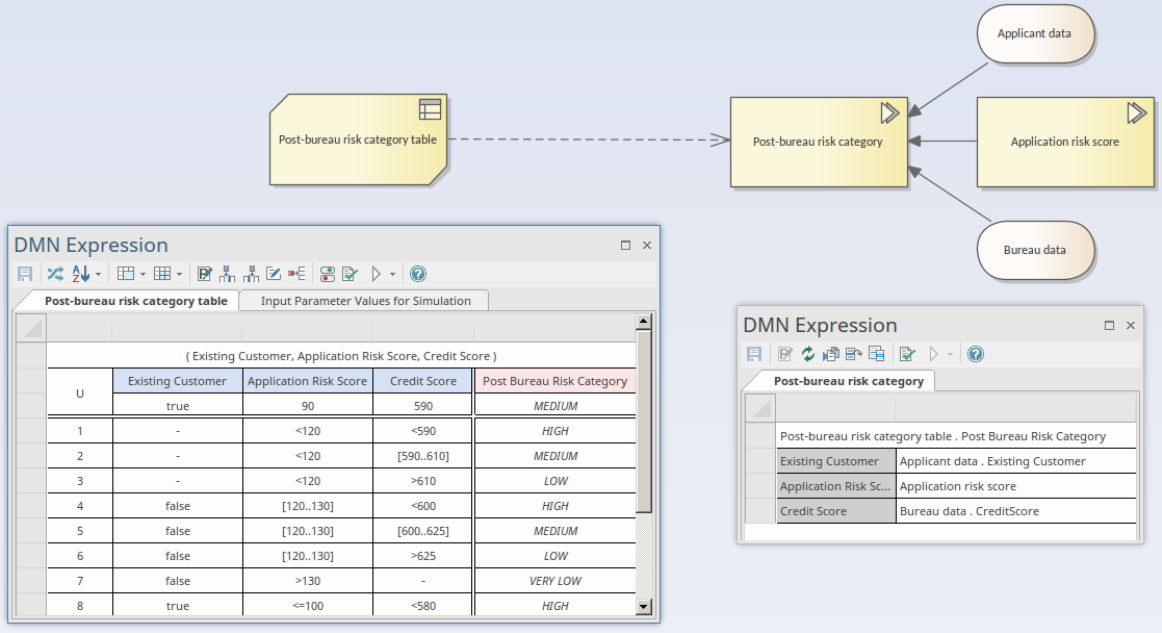
When an Invocation is selected, a number of facilities for working on it are accessible from the toolbar of the DMN Expression window:



For more details refer to the Help topic '*Toolbar for Invocation Editor*'.

Bindings

The parameter bindings of an Invocation provide the context for evaluation of the body of the invocable element.

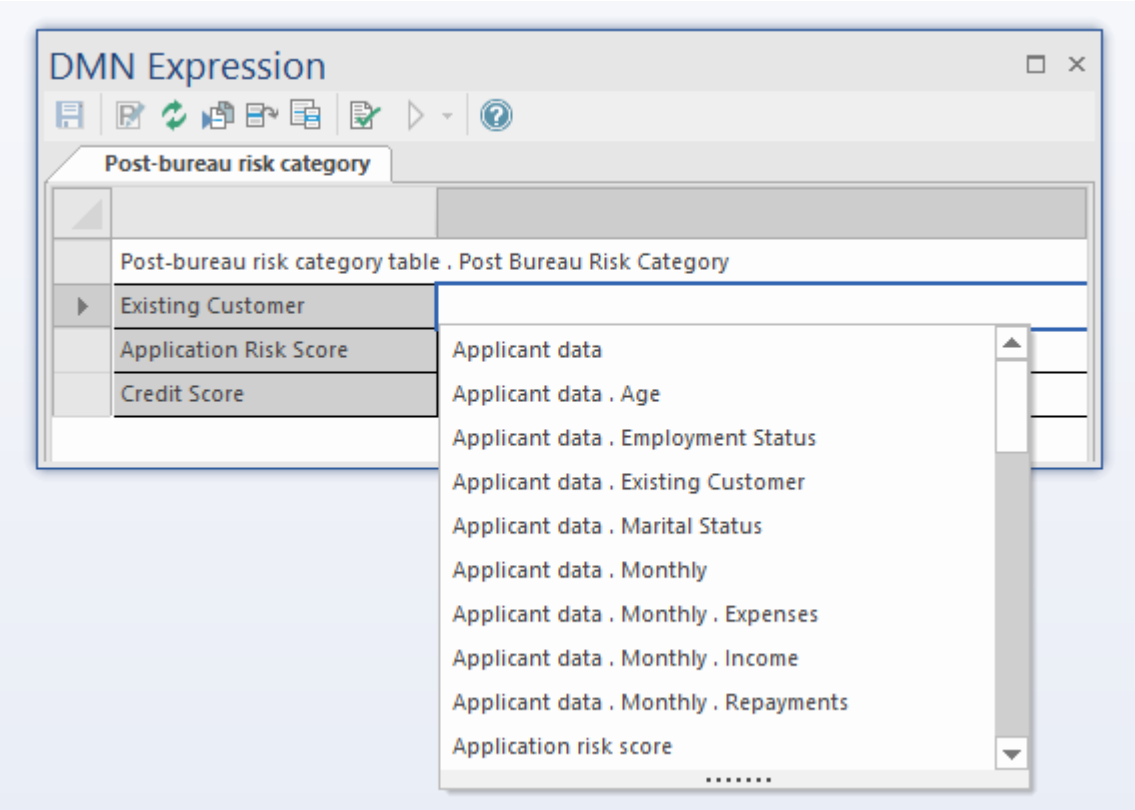


In this example:

- The Decision 'Post-bureau risk category' is represented as an Invocation connecting to the Business Knowledge Model 'Post-bureau risk category table', implemented as a Decision Table
- The Decision 'Post-bureau risk category' is the target of three Information Requirement connectors from two Input Data elements and one Decision element
- The binding list binds the input values to the Business Knowledge Model's parameters
- The Invocation also specifies the requested 'OutputClause'; in the case where a Decision Table has multiple Output Clauses defined, the Invocation must explicitly request an Output Clause as the result of the expression

Inputs

Inputs from other Decisions and InputData elements can be set by pressing the Spacebar in the field:



Output

As an Invocation can only invoke one Business Knowledge Model, the output is defined by the Business Knowledge Model output.

Toolbar for Invocation Editor

When an Invocation expression is selected, the DMN Expression window toolbar provides options specific to that expression type.

Toolbar Options

This table provides descriptions of the features accessible in the DMN Expression window when an Invocation is selected.

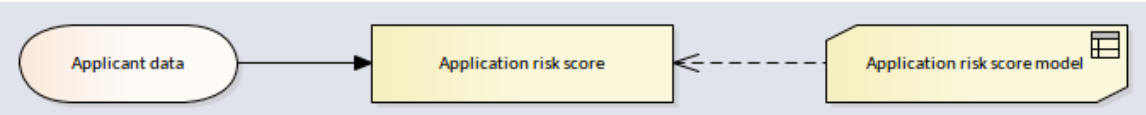
Options	Description
	Click on this button to save the configuration to the current Decision or BusinessKnowledgeModel.
	Click on this button to edit parameters for the Business Knowledge Model.
	Applicable to Invocation value expressions, for both Decision elements and Business Knowledge Model (BKM) elements. Click on this button to synchronize with the invoked BKM. For example, if the BKM changes name, parameters, outputs or types, click on this button to

	synchronize these changes.
	Applicable to Invocation value expressions, for both Decision elements and Business Knowledge Model (BKM) elements. Click on this button to set or change a BKM as an invocation.
	Applicable to Invocation value expressions, for both Decision elements and Business Knowledge Model (BKM) elements. Click on this button to open the invoked BKM in the DMN Expression window.
	Applicable to Invocation value expressions, for both Decision elements and Business Knowledge Model (BKM) elements. When a BKM is implemented as a Decision Table, it could define multiple Output Clauses; the invocation on this BKM might have to specify which output is requested. Click on this button to list all the available outputs in a context menu; the currently configured output is checked.

	Perform validation of the Invocation. Enterprise Architect will perform a series of validations to help you locate any errors in the Invocation definition.
	This button is enabled when the Invocation is defined for a Business Knowledge Model. Select the 'Input Parameter Values for Simulation' tab, complete the fields and click on this button. The test result will be presented on the Decision Table, with the runtime values of inputs and outputs displayed and valid rule(s) highlighted. You can use this functionality to unit test a Business Knowledge Model without knowing the context and later on invoked by a Decision or other Business Knowledge Model. Menu options are available for this toolbar button. For more information, see the *Simulate DMN Model* Help topic.

Example 1 - Bind Input Data to Business Knowledge Model

A full example can be created with a Model Pattern (in the ribbon, select 'Simulate > Decision Analysis > DMN > Apply Perspective > DMN Decision > Decision With BKM : Create Model(s)').

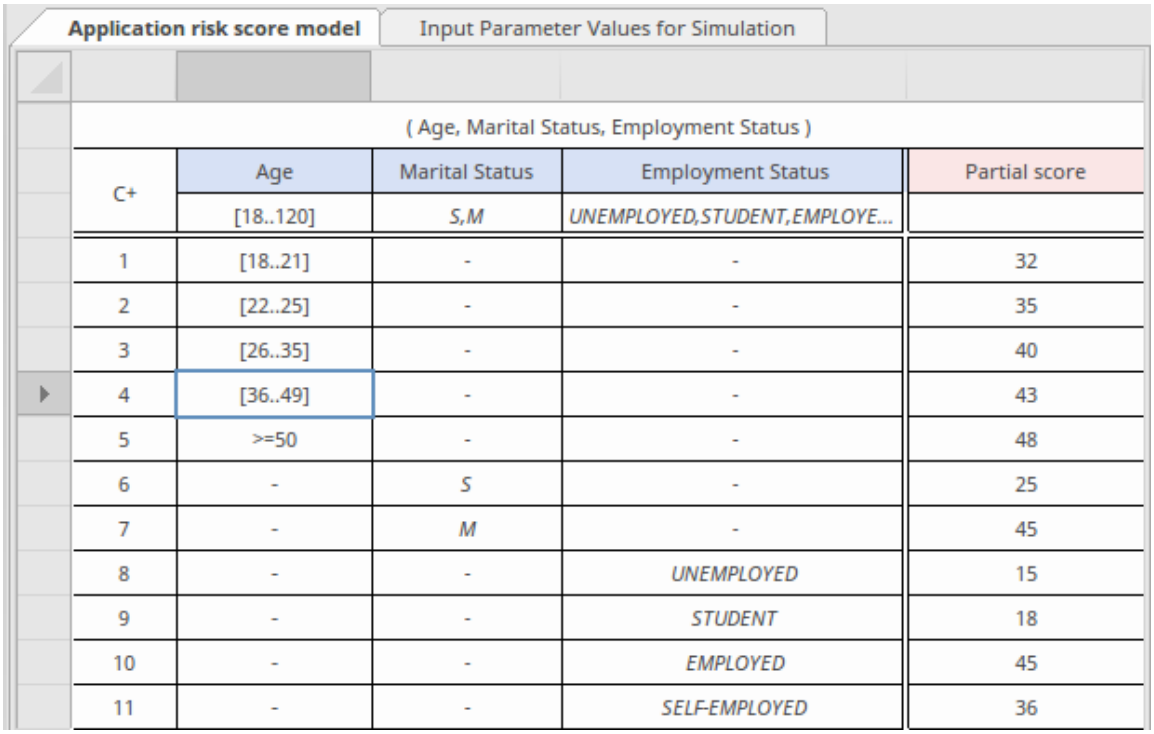


In this example, Input Data *Applicant Data* is typed to *Applicant data Definition,* which has three components.

Applicant data : Applicant data Definition

Applicant data Definition	Age : number	40
	Employment Status : string	"EMPLOYED"
	Marital Status : string	"M"

The Business Knowledge Model *Application risk score model* is implemented as a Decision Table with three inputs and one output.

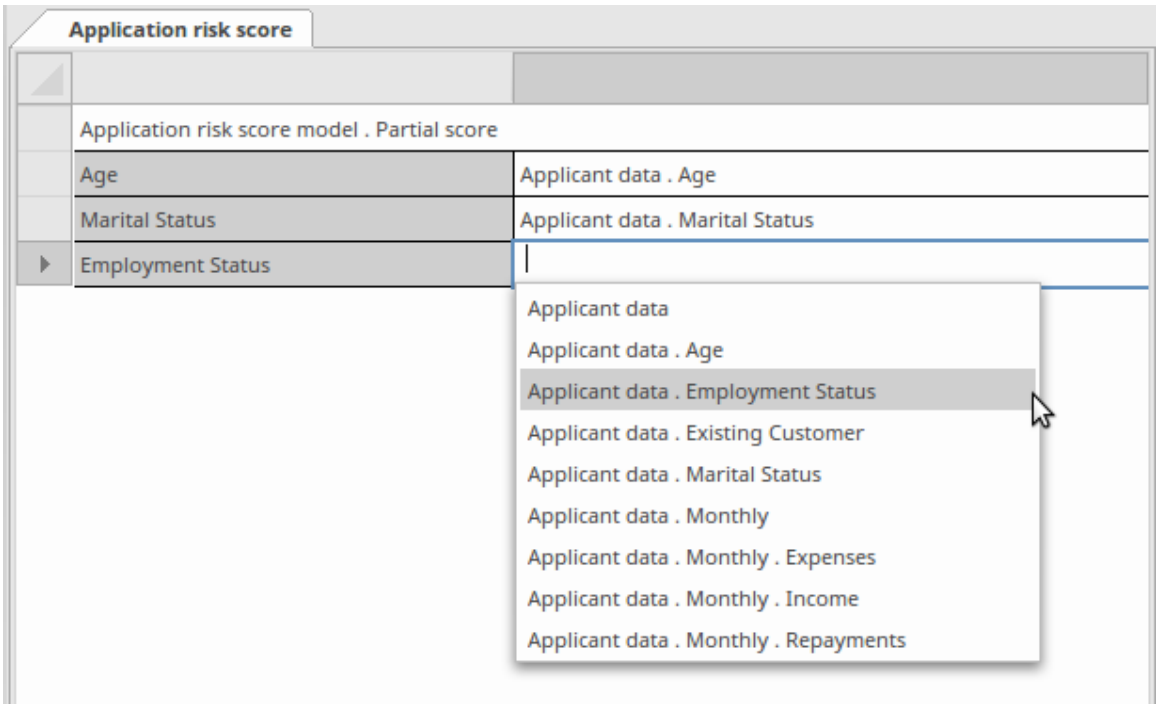


Application risk score model | Input Parameter Values for Simulation

(Age, Marital Status, Employment Status)

C+	Age	Marital Status	Employment Status	Partial score
	[18..120]	S,M	UNEMPLOYED,STUDENT,EMPLOYE...	
1	[18..21]	-	-	32
2	[22..25]	-	-	35
3	[26..35]	-	-	40
4	[36..49]	-	-	43
5	>=50	-	-	48
6	-	S	-	25
7	-	M	-	45
8	-	-	UNEMPLOYED	15
9	-	-	STUDENT	18
10	-	-	EMPLOYED	45
11	-	-	SELF-EMPLOYED	36

The Decision *Application risk score* is implemented as an Invocation to bind the Input Data's 'leaf' components to the BKM's parameters.



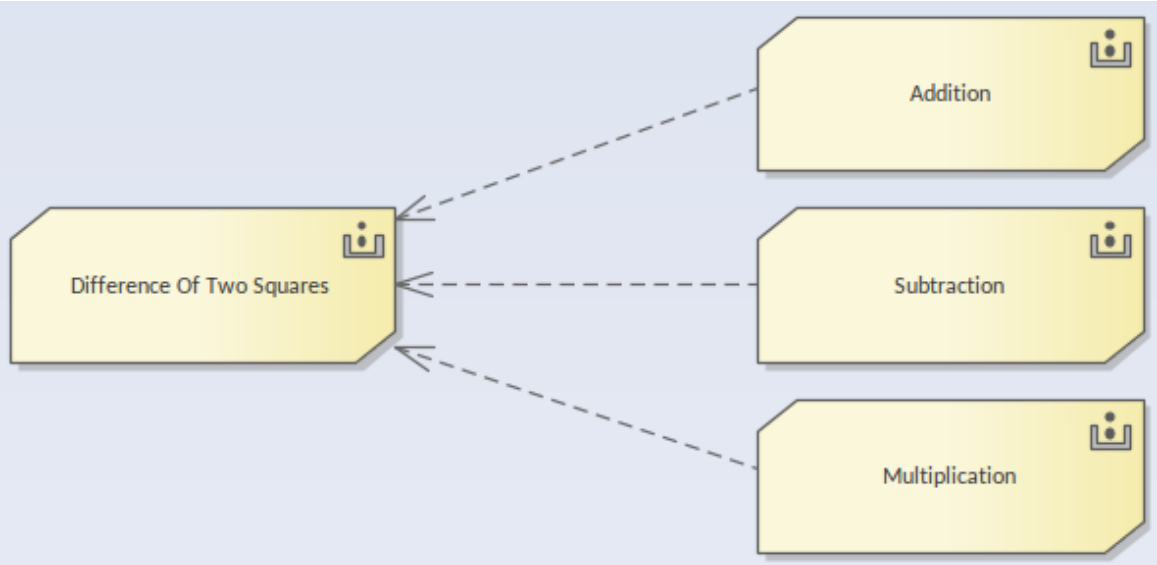
In order to make the binding easier, Auto-Completion is supported for the binding expression.

The full modeling and simulation instructions are available

in the Pattern's documentation.

Example 2 - Bind Context Entry variables to Business Knowledge Model

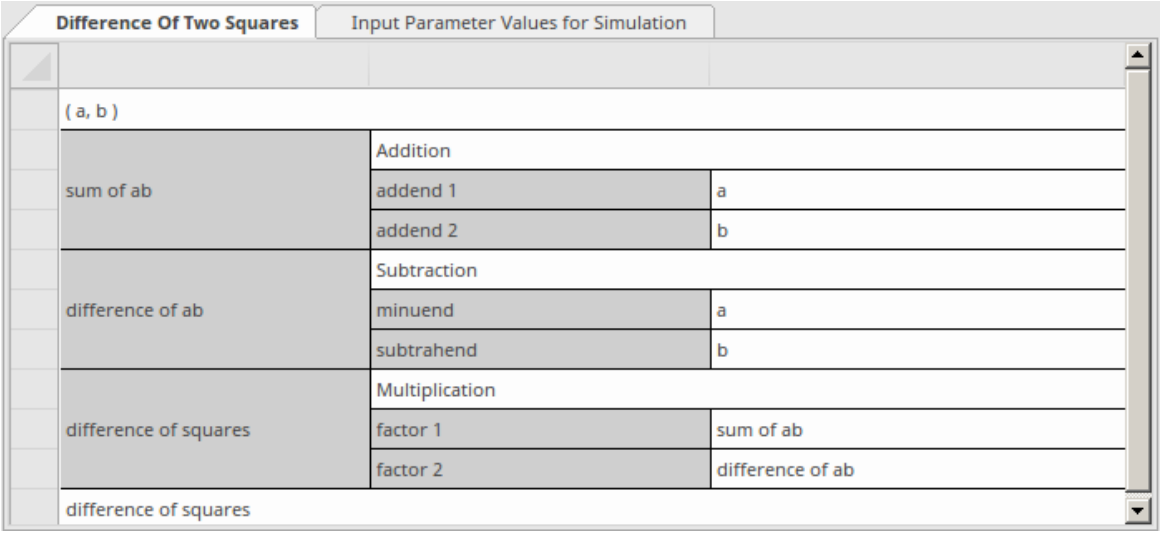
A full example can be created with a Model Pattern (in the ribbon, select 'Simulate > Decision Analysis > DMN > Apply Perspective > DMN Business Knowledge Model Examples > Business Knowledge Model Invocation : Create Model(s)).



In this example, the Business Knowledge Model (BKM) *Difference Of Two Squares* is implemented as Boxed Context:

- The variable *sum of ab* is implemented as an invocation by binding parameters *a* and *b* to BKM *Addition*
- The variable *difference of ab* is implemented as an invocation by binding parameters *a* and *b* to BKM *Subtraction*
- The variable *difference of squares* is implemented as an

invocation by binding local variables *sum of ab* and *difference of ab* to BKM *Multiplication*



Difference Of Two Squares | Input Parameter Values for Simulation

(a, b)		
sum of ab	Addition	
	addend 1	a
	addend 2	b
difference of ab	Subtraction	
	minuend	a
	subtrahend	b
difference of squares	Multiplication	
	factor 1	sum of ab
	factor 2	difference of ab
difference of squares		

In order to make the binding easier, auto-completion is supported for the binding expression.

The full modeling and simulation instructions are available in the Pattern's documentation.

Edit DMN Expression Dialog

The 'Edit DMN Expression' dialog is used for setting expressions in the Boxed Content, Invocation and Literal Expression element types. It provides Intelli-sense support for constructing expressions based on the FEEL grammar, as well as the code languages that can be used for code generation of the model.

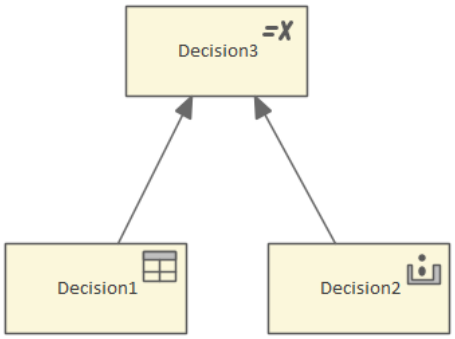
DMN Expression Editor and Intelli-sense support

To help you edit expressions with less typing and fewer mistakes, Enterprise Architect provides Intelli-sense support for editing the expressions.

Note that the parameter and Context Entry variable names can contain spaces, according to the FEEL language specification. This feature is intended to make each expression easy to read.

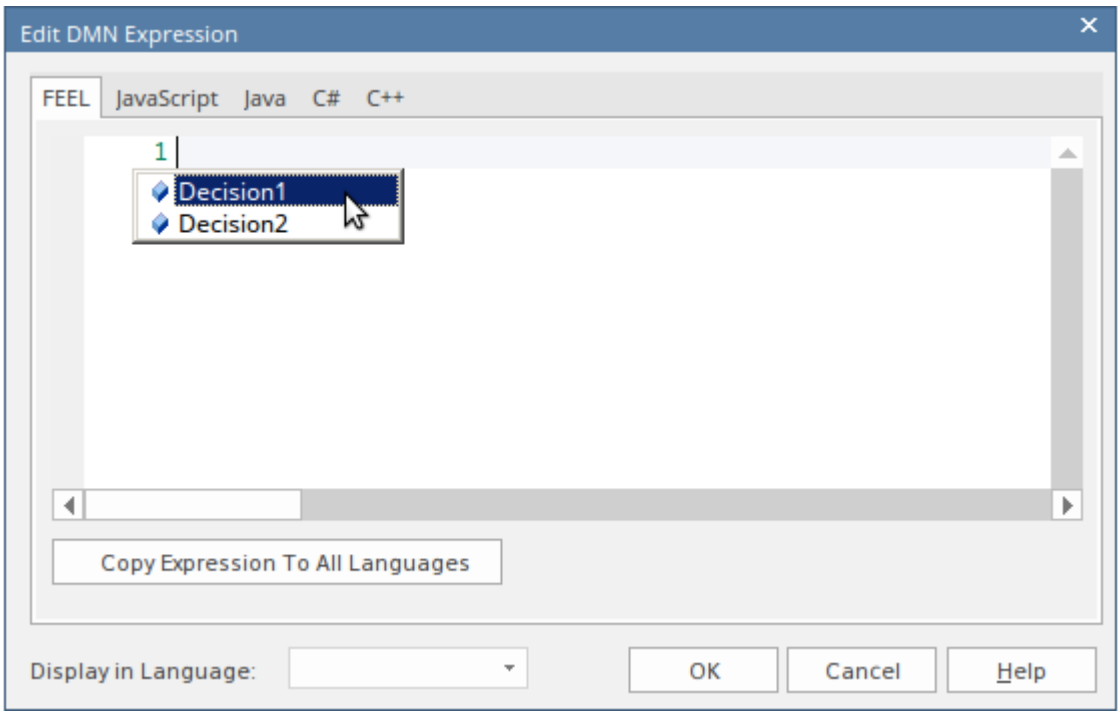
Examples

Given this decision hierarchy, the expression in 'Decision3' is able to use the outputs from the two referenced Decisions.



To open the 'Edit DMN Expression' dialog:

1. Double-click on the Decision element in the diagram, to display the DMN Expression window.
2. Right-click on the expression line and select the menu option 'Edit Expression'. The 'Edit DMN Expression' dialog displays.
3. Click on a line and press Ctrl+Spacebar to show the Intelli-sense menu:

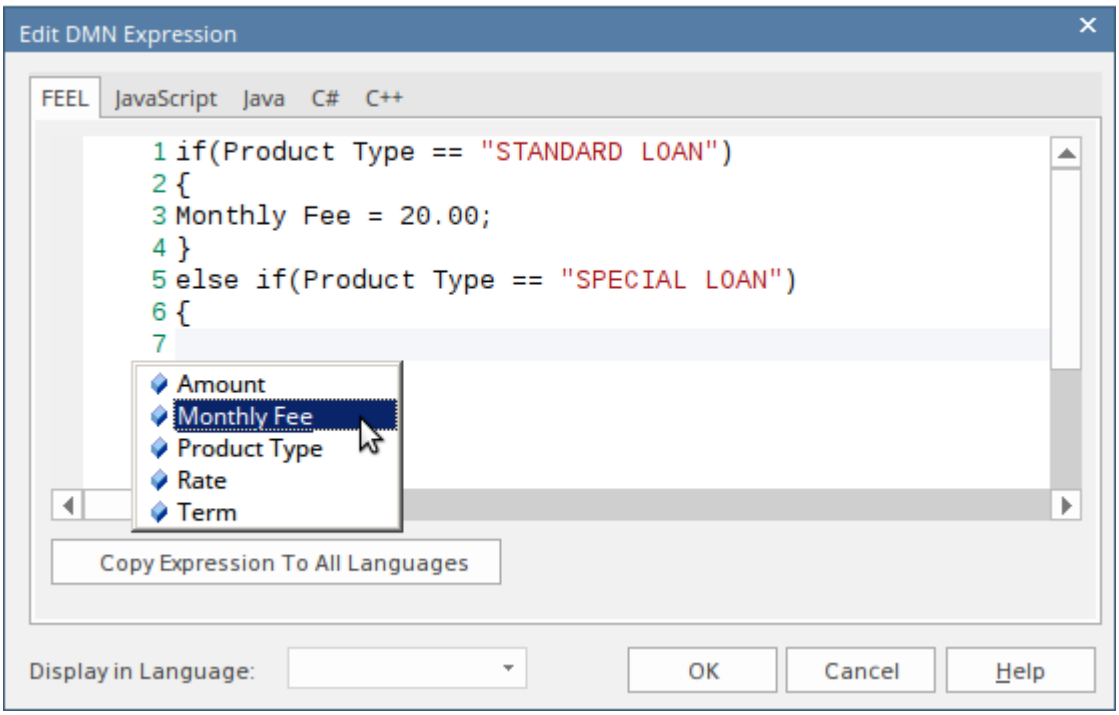


- For a BusinessKnowledgeModel expression all the parameters will be included
- For Decision expression all the required Decisions will be

included

- All Context Entry Variables earlier than the current one will be included (Context Entry Variables later than the current one are excluded)

In this example, editing a BKM Boxed Context expression, the Input Parameters are shown in the Intelli-sense menu:



Language selection

The DMN Model can be generated as source code in JavaScript, Java, C# or C++. As the syntax differs between the languages, Enterprise Architect provides language-override pages for each language. If no override code is specified for a language, the expression that is defined for the FEEL language will be used.

Note: In the generated code, the space inside a variable name will be replaced by an underscore.

DMN Expression Validation

DMN defines many expressions, such as FunctionDefinition, DecisionTable, Boxed Context, Invocation and Literal Expression. The parameters, arguments and logic of these expressions are implemented largely by 'text'.

To make modeling easier and more reliable, Enterprise Architect provides two features: Auto Completion and Validation.

- Validation: Identifies modeling errors caused by typos, logic incompleteness, inconsistency, and so on
- Auto Completion: You can select a text string from a list of enumerations rather than type the text in

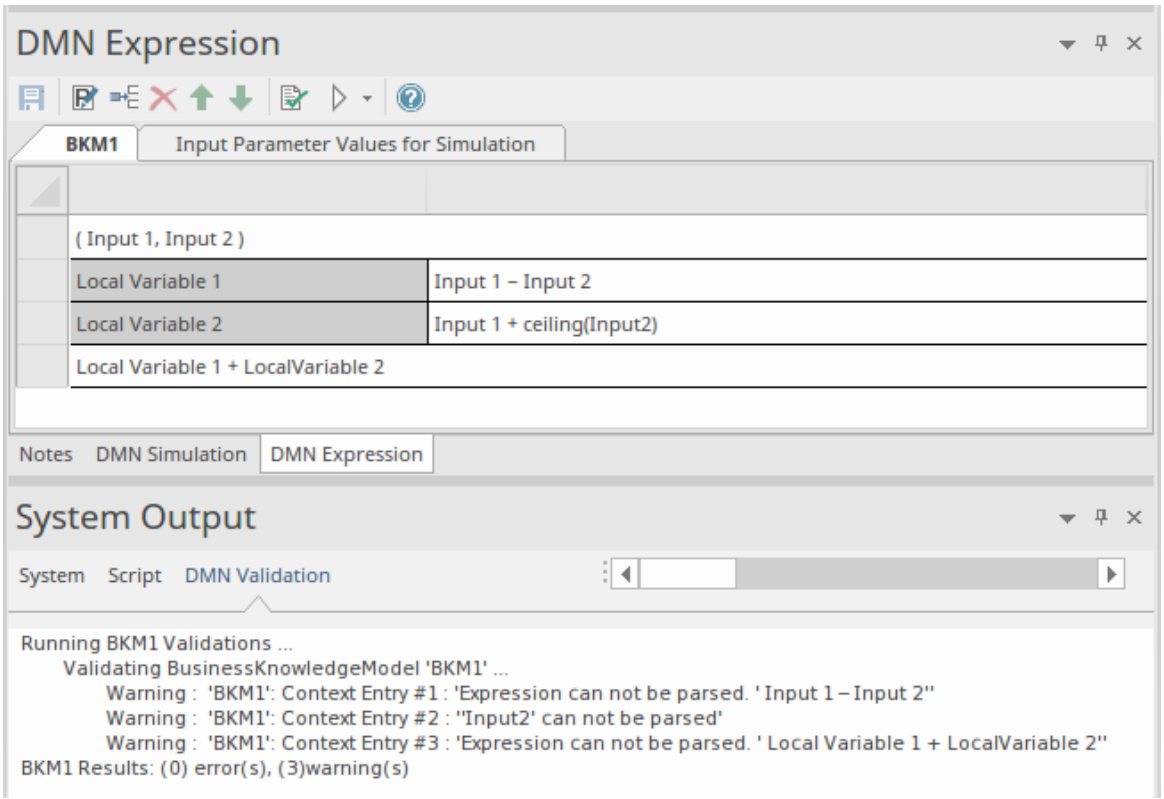
In this topic, we will show you how to validate a DMN Expression.

Access

DMN Expression Window	Simulate > Decision Analysis > DMN > DMN Expression : Validate button
DMN Simulation Window	Simulate > Decision Analysis > DMN > Open DMN Simulation > Simulate : Validate icon

Common Validations

Variable Name Validation



In this example, the Boxed Context Business Knowledge Model BKM1 defines two parameters, 'Input 1' and 'Input 2', and two local variables, 'Local Variable 1' and 'Local Variable 2'. The expression has been validated, and the results output to the 'DMN Validation' tab of the System Output window.

- Context Entry #1 failed because there is a typographic error; it should be operator '-', but the user typed or copied in '–'
- Context Entry #2 failed because there is no space between 'Input' and the number 2; note that the function 'ceiling()' is defined in the DMN Library so it can be successfully

parsed

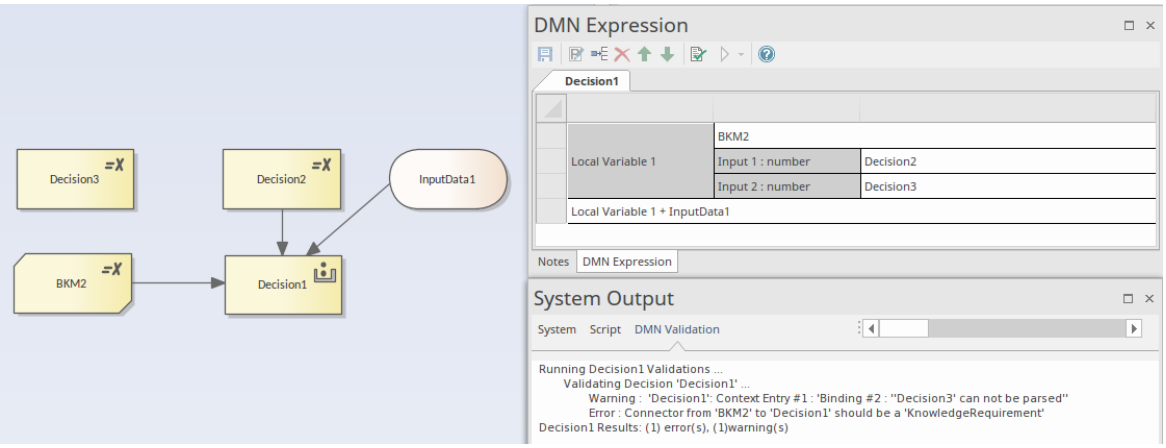
- Context Entry #3 failed because there is no space between 'Local' and 'Variable'

It is hard to visually identify these kinds of error. Running validation can help identify errors and then you can easily perform a correction.

Dependency Validation

A decision might require other decisions, input data and business knowledge models; these relationships are identified by InformationRequirement and KnowledgeRequirement connectors.

When the graph is getting complex, it is quite possible that some connectors are missing or the wrong connector type is being used.

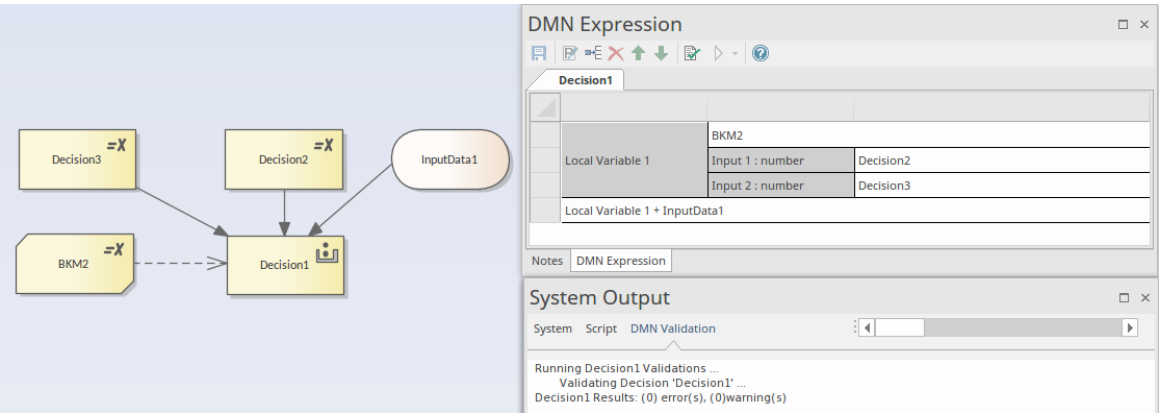


In this example, click on the Validate button, Enterprise Architect will show that:

- 'Decision3' is used by 'Decision1' by binding to a parameter of the called BKM2; however, it is not defined - an InformationRequirement connector is missing
- The Invocation defined in 'Decision1' is not valid; the

connector type from 'BKM2' to 'Decision1' should be a KnowledgeRequirement

After fixing these problems, run the validation again:



DMN Expression Auto Completion

DMN defines many expressions, such as FunctionDefinition, DecisionTable, Boxed Context, Invocation and Literal Expression. The parameters, arguments and logic of these expressions are implemented largely by text.

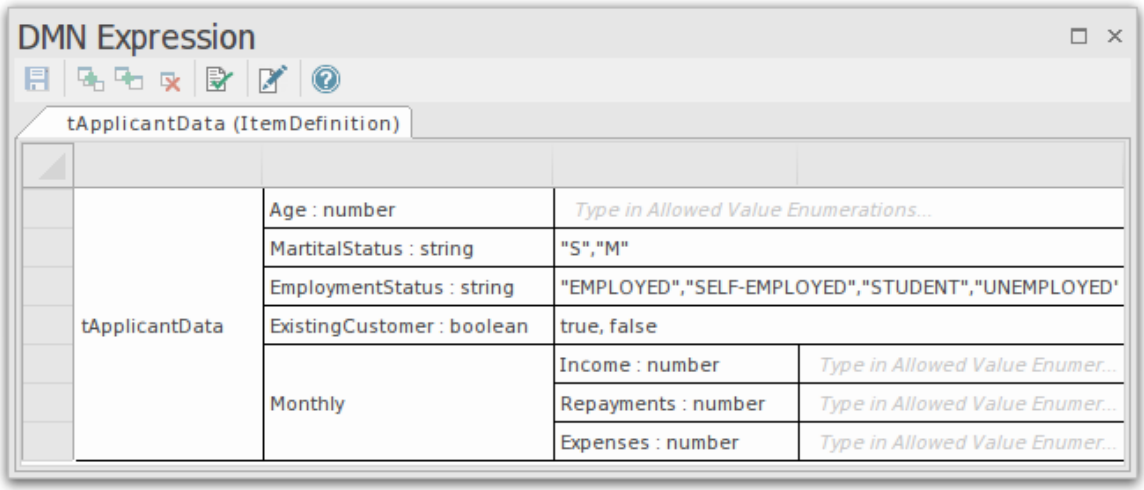
To make modeling easy and reliable, Enterprise Architect provides an Auto Completion facility, helping provide the:

- Allowed Values of ItemDefinition
- Input/Output Entries of a Decision Table
- InformationRequirement

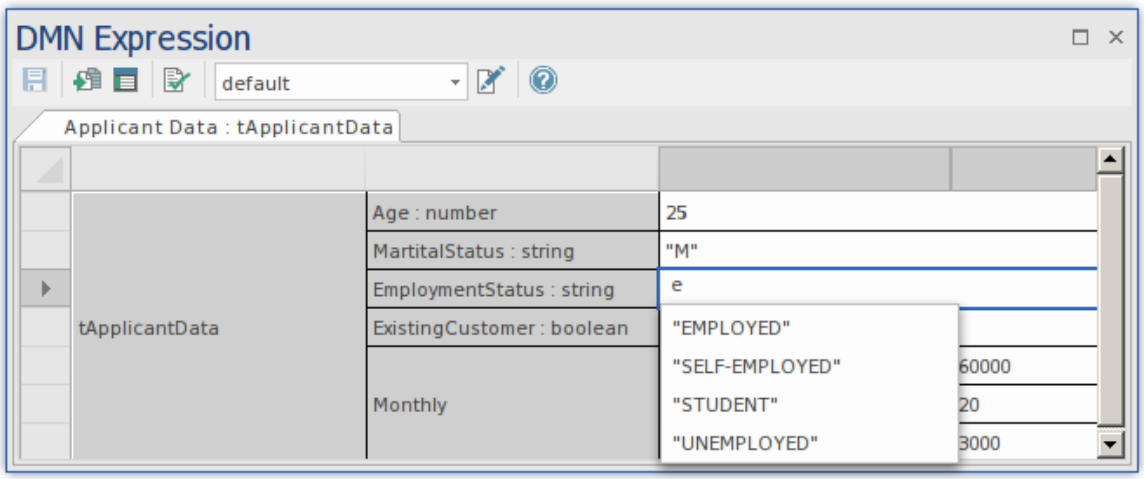
Allowed Values of ItemDefinition

The idea is to define allowed value enumerations in ItemDefinition, then compose a list for selection whenever these values are requested.

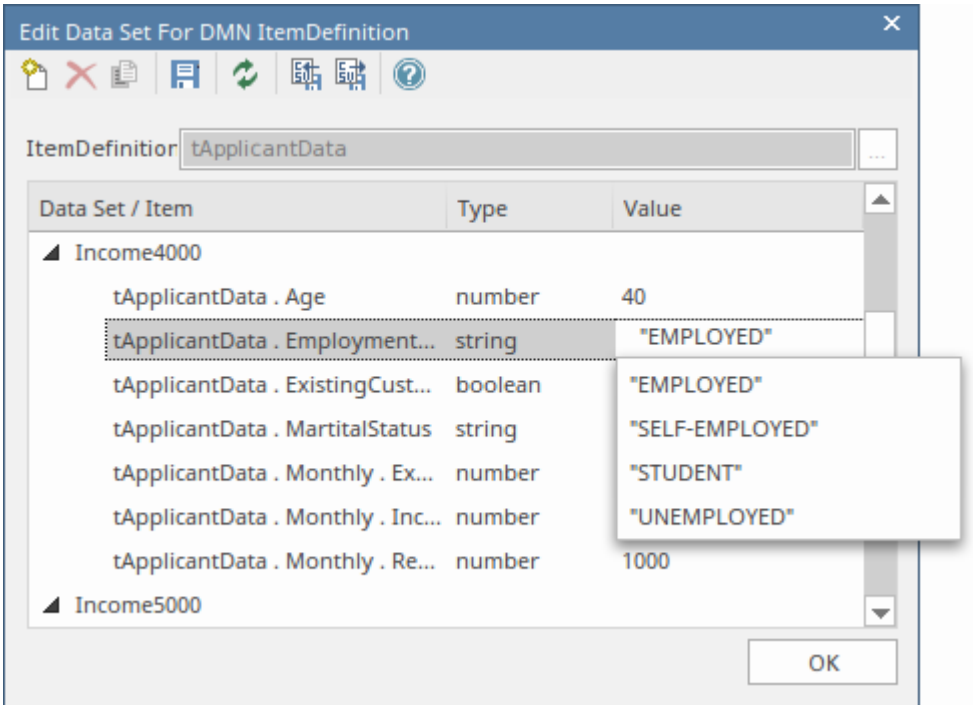
In this example, ItemDefinition 'Applicant data . Employment Status' defines an enumeration of allowed values.



When editing values for the InputData typed to this ItemDefinition, press the Spacebar on the keyboard to display a list of values to select from.

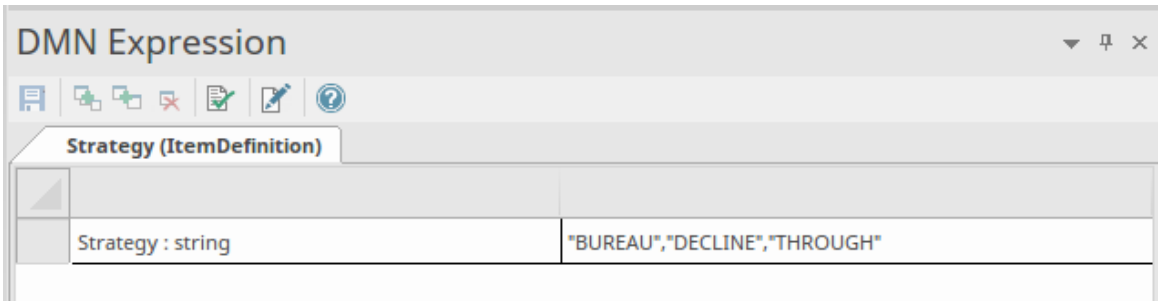


We could also define multiple data sets for the InputData, as the Auto Completion feature is available on this dialog.

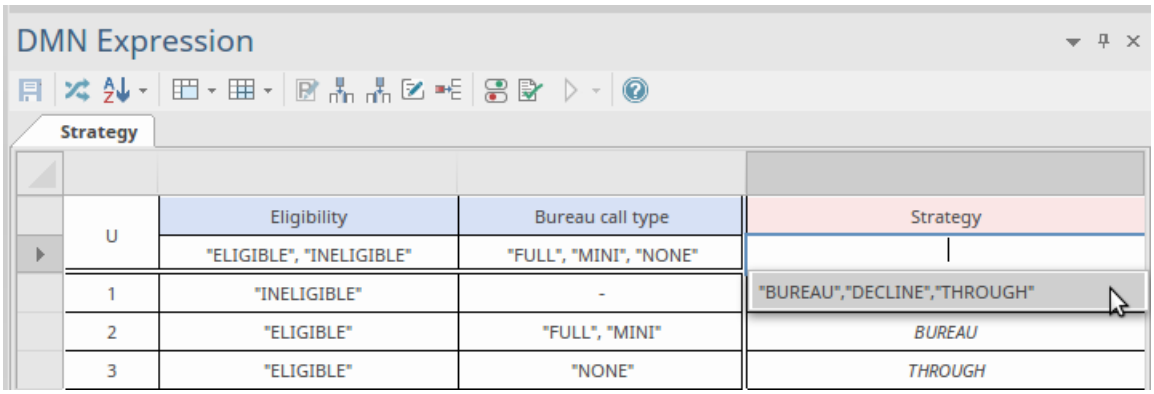


Input/Output Entries of a Decision Table

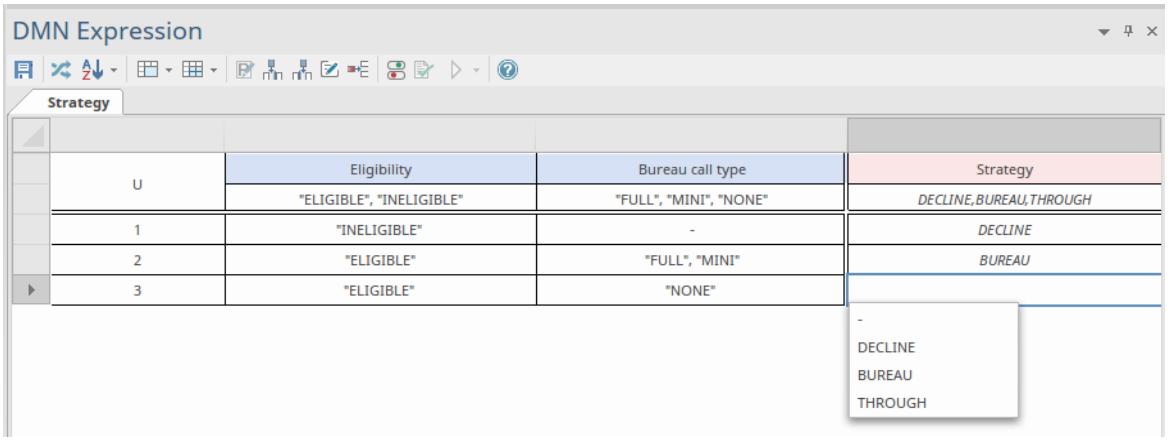
Take the 'Strategy' ItemDefinition as an example:



We can quickly fill the 'Allowed Values' field for a Decision Table by selection:



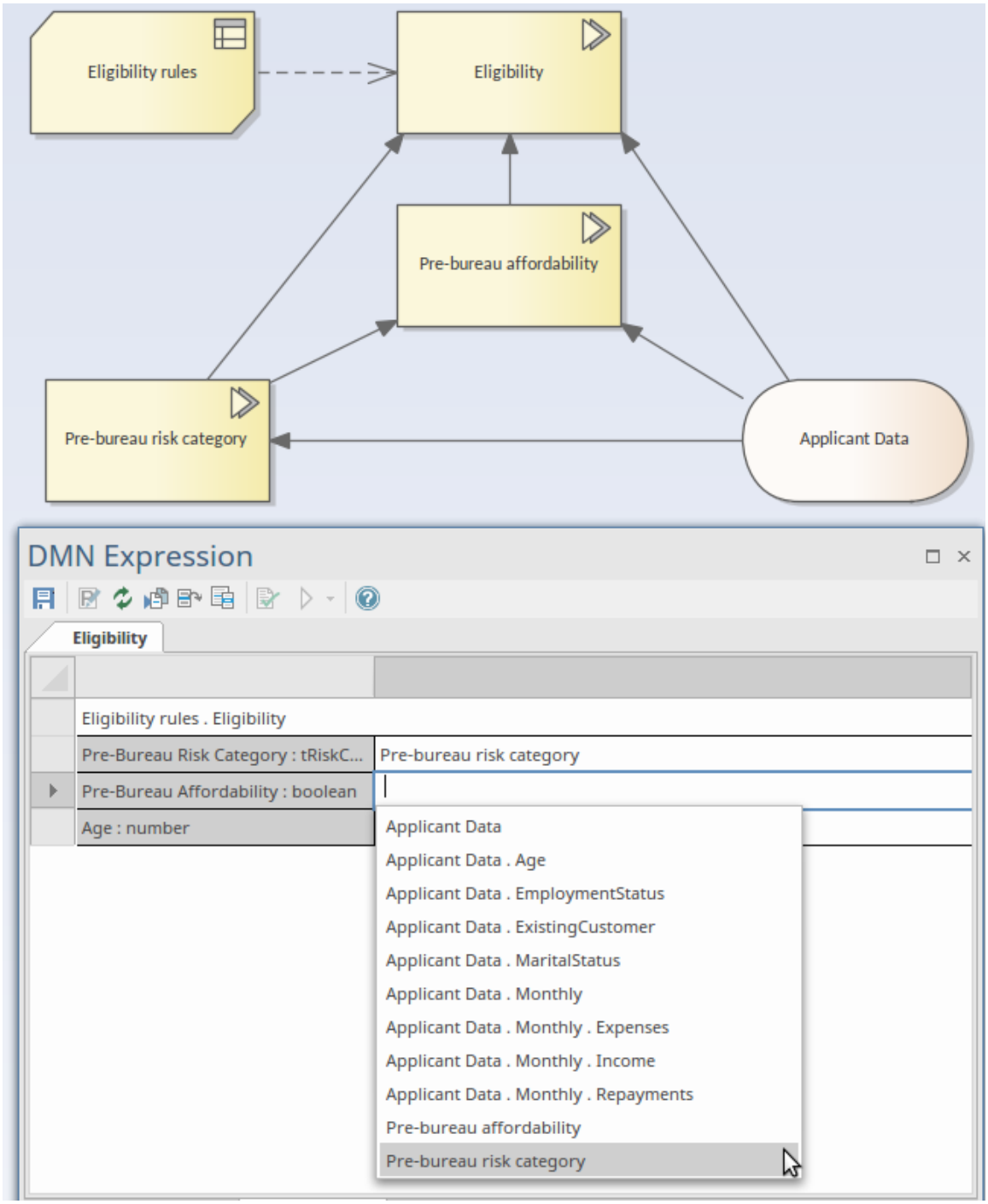
Then we can quickly fill the Decision Table rules by selection:



Note: the default '-' denotes 'Undefined'.

Information Requirement

On a decision hierarchy, a decision might access required decisions and input data; these required elements form a list of variables that can be used by the decision.



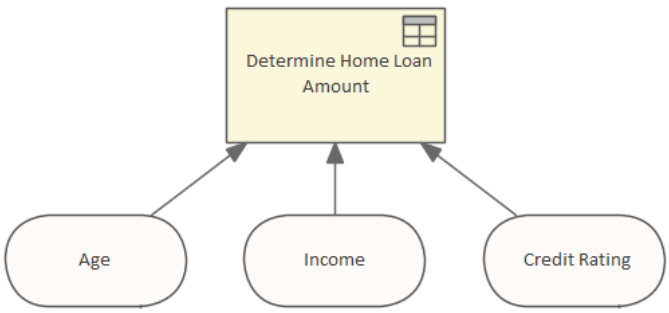
In this example, Decision 'Eligibility' requires two decisions - 'Pre-bureau risk category' and 'Pre-bureau affordability' - and one Input Data item 'Applicant data'.

When setting the binding values for the invoked BusinessKnowledgeModel 'Eligibility rules', an Auto Completion list will prompt for selection. In this list, there are sub-decision names - leaf components of the input data.

With this feature, you can easily set up an invocation.

Modeling with DMN

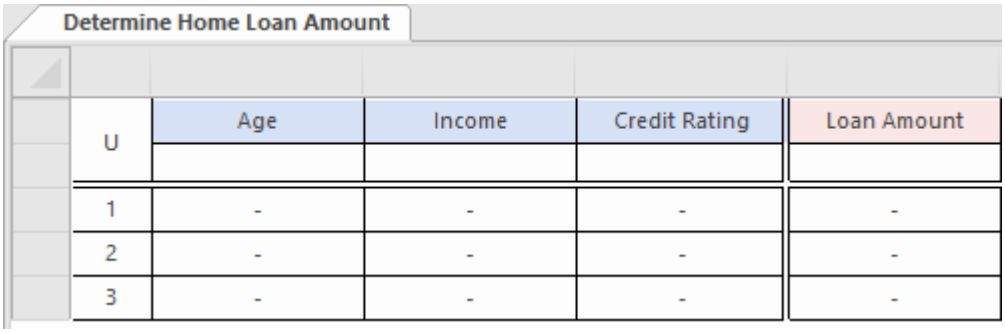
This topic introduces you to the most important elements that you need to create Decision models. As discussed in earlier topics there are two fundamental parts of a decision model namely the Decision Requirements diagram and the decision logic. Creating a decision diagram is straight forward and probably the most onerous part of the exercise will be unraveling the way an organization makes decisions and what the inputs to these decision are. The diagrams will typically contain decisions that are chained together describing the fact that one decision can provide input to another decision and so on.



Simple Decision Requirements diagram showing three inputs into a Decision using a Decision Table.

The logic of the decision is described using a number of devices but the most commonly used and accessible form is the Decision Table. The Decision Table contains rows and columns like a spreadsheet and covers all the possible combinations of inputs to produce a number of outputs. For example if an applicant is older than 21 and younger than 65 and earns $60,000 per year with a good credit rating the

bank will lend them $300,000 for a home loan.



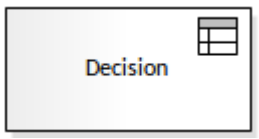
Determine Home Loan Amount

U	Age	Income	Credit Rating	Loan Amount
1	-	-	-	-
2	-	-	-	-
3	-	-	-	-

A Decision Table showing three inputs and one output, rows would be added to define the rules.

Decision

A Decision element is used to evaluate an output based on one or more inputs. The logic that determines the output is either defined within that Decision element or it invokes the decision logic contained in a Business Knowledge Model that is connected to the Decision.



Inputs

A Decision can have any number of inputs, including the option to define the input values in the element. The most common input is to use an Input Data Element.

Output

A Decision can have zero or one output. The output can be a complex data set.

Value Expressions

The output of a Decision element is determined using a Value Expression. The Value Expression contains the element's decision logic and can take one of four forms: Decision Table, Literal Expression, Invocation or Boxed Context. Value Expressions are defined and edited using

the DMN Expression editor, which displays one of four formats according to the type of expression being used.

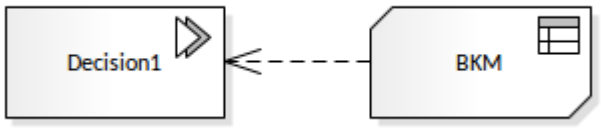
When displayed on a diagram, the Decision element shows an icon in the top-right corner that indicates which type of value expression it is using.

Type	Description
	A Decision Table is a tabular representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.
	A Literal Expression is the simplest form of DMN expression. It is commonly defined as a single-line statement or an if-else conditional block.
	A Decision Invocation requires that a Business Knowledge Model element is referenced using a Knowledge Requirement connector. The Decision element simply contains the parameters that provide the context for evaluating the Business Knowledge Model (BKM). Part or all of the result returned from the BKM can be set to be passed as the output of the Decision.

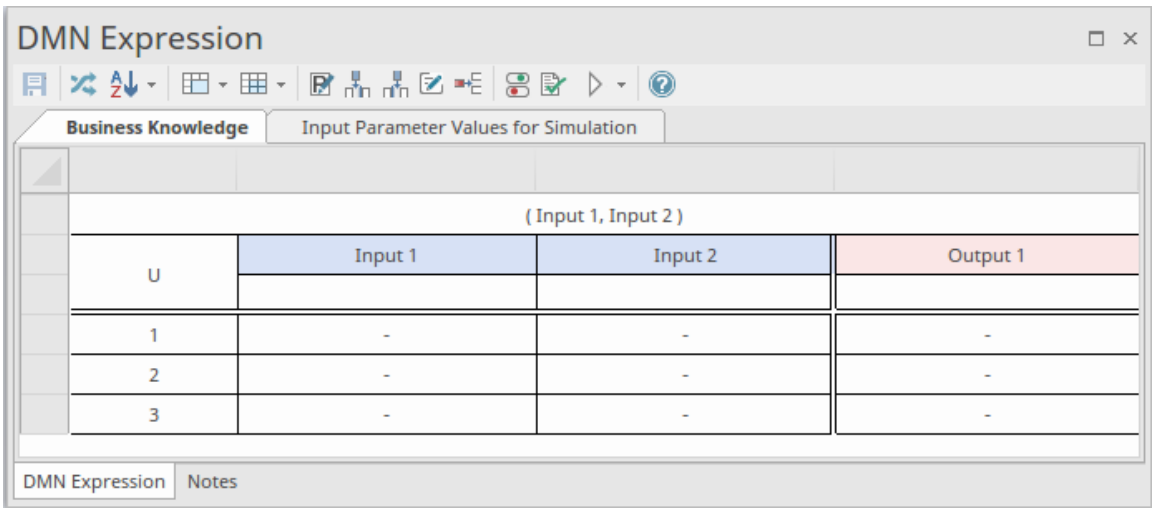
	A Boxed Context is a collection of context entries. Each context entry consists of a variable and an expression. The Context also has a result value.

Business Knowledge Model

A Business Knowledge Model (BKM) element represents a reusable piece of decision logic. Typically, it is connected to a Decision element that invokes the BKM and passes on a set of inputs. The BKM, using its internal logic, evaluates an output that is passed back to the Decision.



Unless a BKM is working on fixed values, it usually requires defining a set of input parameters, as well as the definition of an output. The parameters and the decision logic are defined using the DMN Expression window.



Inputs and Output

When used in a decision model, a BKM must be connected via a KnowledgeRequirement to a Decision or another BKM, through which it receives its inputs . The input parameters are defined using the icon. These can be set as

a simple type or a complex type defined using an ItemDefinition. The naming of the input parameters influences the naming within the Value Expression.

Output

A BKM output is via a KnowledgeRequirement which must be an input to a Decision or to another BKM. The output is defined using:

- The icon for a Literal Expression
- Output column(s) in the DMN Expression table for a Decision Table, Boxed Content and Invocation.

An output can be a simple type or a complex type defined using an ItemDefinition.

Value Expressions

To define a means for evaluating an output, based on the decision logic, a Business Knowledge Model (BKM) element contains a Value Expression. This is defined and edited using the DMN Expression window, which has four formats, the format being determined by the type of Value Expression that you want to use.

The BKM element can be set with these structures for the Value Expression. Each is shown in the model with an icon.

Type	Description
	A Decision Table is a tabular

	representation of a set of related input and output expressions, organized into rules indicating which output entry applies to a specific set of input entries.
	A Literal Expression is the simplest form of DMN expression. It is commonly defined as a single-line statement or an if-else conditional block.
	A Decision Invocation requires that a Business knowledge model element is referenced using a Knowledge Requirement connector. It simply contains the parameters that provide the context for the evaluating a business knowledge model.
	A Boxed Context is a collection of context entries. Each context entry consists of a variable and an expression. The Context also has a result value.

Validation and Testing

To ensure a BKM element is able to produce a correct output it can be validated using the Validation icon . A BKM can also be tested as a unit to ensure it is operative

using the Simulation button. For more details see the *Input Parameter Values for Simulation* Help topic.

BKM Parameters

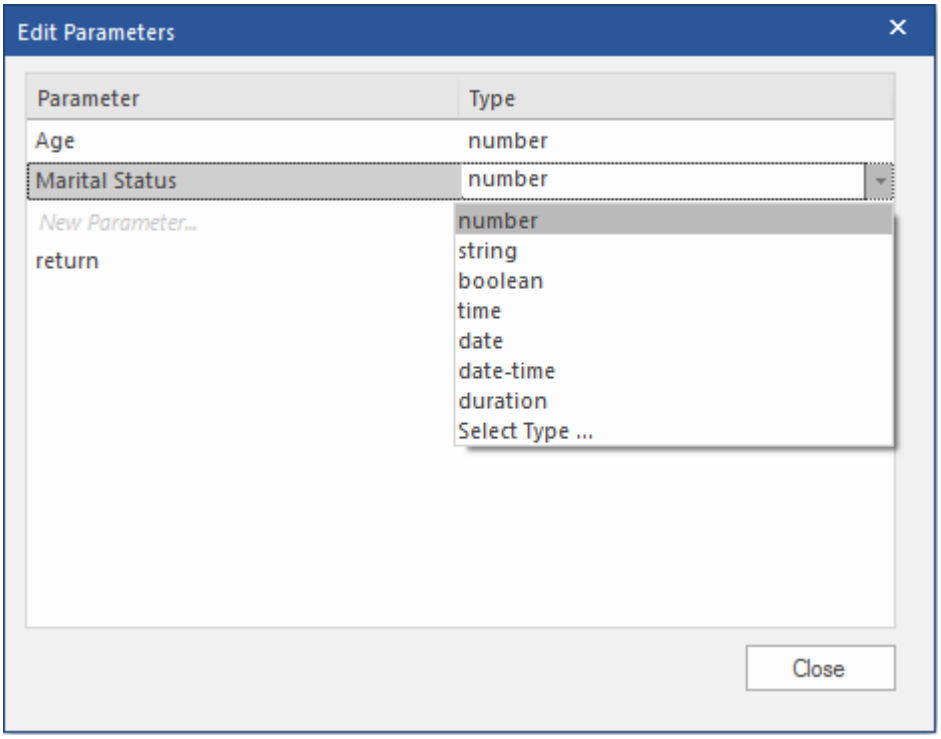
A Business Knowledge Model (BKM) is implemented as a function definition, with parameters and a DMN expression as its body (such as Decision Table, Boxed Context or Literal Expressions).

As a BKM is intended to function stand-alone, and be called by Decisions or other BKMs, it is necessary to define any input parameters. Also, for Literal Expressions, you must define the output parameter.

When defining any input Parameters you can set them with default values for testing. After creating a BKM, to verify that it functions correctly, you can run a simulation based on these default values.

Parameters of a Business Knowledge Model

To open the 'Edit Parameters' dialog, in the DMN Expression window, click on the Edit Parameters button :



Note: this is an example for a Literal Expression that includes a *return* type.

Edit Parameters

You can perform these actions on the parameters:

Action	Description
	Add a new parameter by typing in the 'New Parameter...' row.
	Modify the name of the existing parameter by in-place editing in the cell.
	Delete an existing parameter using the context menu.

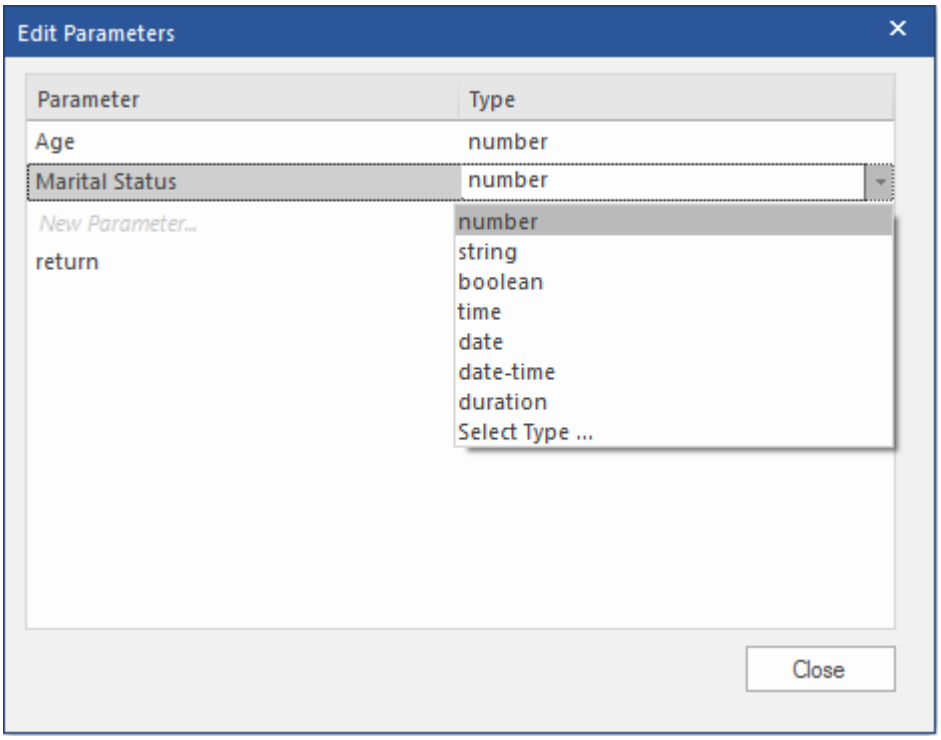
	Click on the Type to enable a drop-down. Select a type for the parameter from the drop-down. **Set an Item Definition Type** When changing the type of Parameter there is an option to select a pre-defined type from an ItemDefinition. The option for this is 'Select Type ...'. When this option is selected it will open a dialog for selecting an ItemDefinition.

Input Parameter Values for Simulation

As a Business Knowledge Model is self-contained, it is possible to perform a simulation 'Unit Test' by providing a default set of values as an input for its parameters. These values can be defined in the *Input Parameter Values for Simulation* tab in the DMN Expression window.

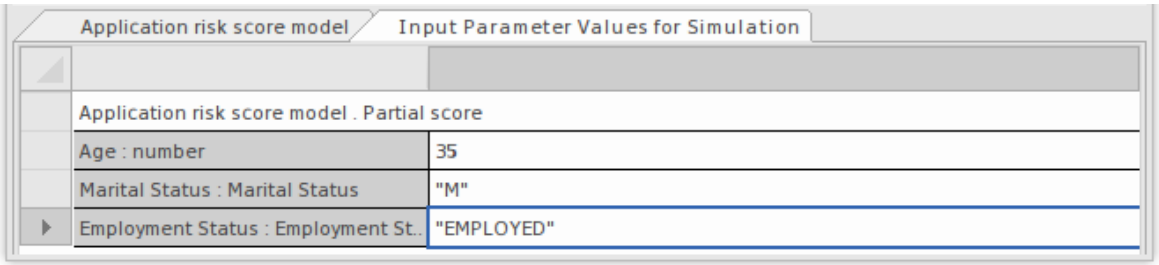
Parameters of a Business Knowledge Model (BKM)

Parameters for a BKM are accessed from the DMN Expression window, using the Edit Parameters button on the toolbar:



A default set of values for these parameters, that can be used in a simulation of the BKM, are defined in the 'Input Parameter Values for Simulation' tab on the DMN

Expression window:



With these parameters set the BKM can be tested using the Simulation button.

Simulation Examples

These are two examples of using the *Input Parameter Values for Simulation.*

Type	Description
Decision Table	An example simulation of a BKM Decision Table element based on values set in the *Input Parameter Values for Simulation* tab.
Literal Expression	An example simulation of a BKM Literal Expression element based on values set in the *Input Parameter Values for Simulation* tab.

Decision Table Simulation Example

The example Business Knowledge Model (BKM) described in this section is available from the Model Wizard (Ctrl+Shift+M). Select a host Package in your model, invoke the Model Wizard and - from the Perspectives drop-down menu - select 'Requirements | Decision Modeling'.

To access the example used in this section:

- Create a pattern for 'DMN Decision | A Complete Example'
- Navigate in the Browser window to 'A Complete Example | Business Knowledge Models'

It is also available in the Enterprise Architect Example model (EAExample):

- Navigate in the Browser window to 'Analysis and Business Modeling > DMN Examples > A Complete Example > Business Knowledge Models'

Double-click on the 'Eligibility rules' element to open the BKM in the DMN Expression window

When a Decision Table is created for a Business Knowledge Model, we can test this BKM by binding some values:

Eligibility rules | Input Parameter Values for Simulation

(Pre-Bureau Affordability, Pre-Bureau Risk Category, Age)

P	Pre-Bureau Risk Category	Pre-Bureau Affordability	Age	Eligibility
	VERY LOW, LOW, MEDIUM			*INELIGIBLE, ELIGIBLE*
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

We can provide test values such as these:

Eligibility rules | **Input Parameter Values for Simulation**

Eligibility rules . Eligibility	
Pre-Bureau Affordability	true
Pre-Bureau Risk Category	"VERY LOW"
Age	16

Click on the Simulation button on the tool bar to obtain this result:

Eligibility rules | Input Parameter Values for Simulation

(Pre-Bureau Affordability = true, Pre-Bureau Risk Category = "VERY LOW", Age = 16)

P	Pre-Bureau Risk ...	Pre-Bureau Affor...	Age	Eligibility
	VERY LOW	true	16	*INELIGIBLE*
1	*DECLINE*	-	-	*INELIGIBLE*
2	-	false	-	*INELIGIBLE*
3	-	-	<18	*INELIGIBLE*
4	-	-	-	*ELIGIBLE*

- The runtime parameter value will take the place of 'Allowed Values' in simulation mode
- Valid rule(s) are highlighted
- Since this Decision Table's Hit Policy is P (Priority) the final result is determined by the order of Output Values; since 'INELIGIBLE' and 'ELIGIBLE' are the output values and 'INELIGIBLE' comes ahead of 'ELIGIBLE',

rule #3 will give the final result and this applicant is 'INELIGIBLE'.

Literal Expression Simulation Example

The Business Knowledge Model (BKM) described in this section is available from the Model Wizard (Ctrl+Shift+M). Select a host Package in your model, invoke the Model Wizard and - from the Perspectives drop-down menu - select 'Requirements | Decision Modeling'.

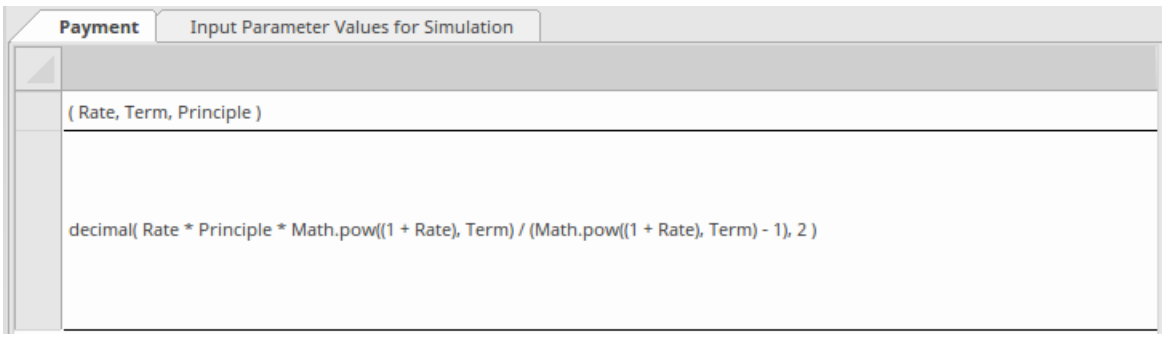
To access the example used in this section:

- Create a pattern for 'DMN Business Knowledge Model > Business Knowledge Model Literal Expression'
- Navigate in the Browser window to 'Business Knowledge Model Literal Expression > Payment'

It is also available in the Enterprise Architect Example model (EAExample):

- Navigate in the Browser window to 'Model Simulation > DMN Models > Business Knowledge Model > Business Knowledge Model Literal Expression'

Double-click on the 'Payment' element to open the BKM in the DMN Expression window.

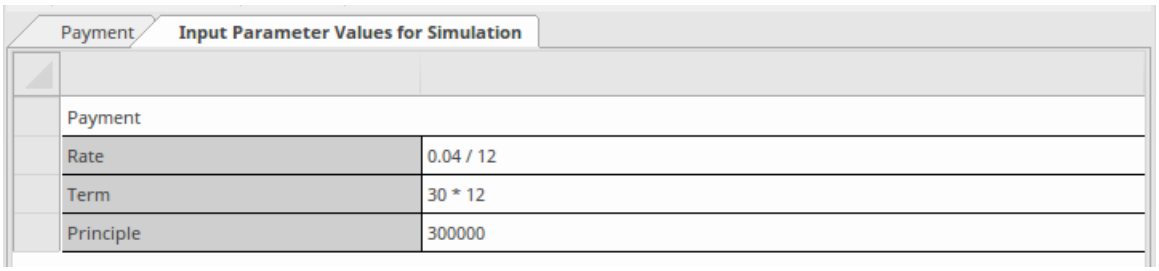


Similar to a Decision Table, the Business Knowledge Model

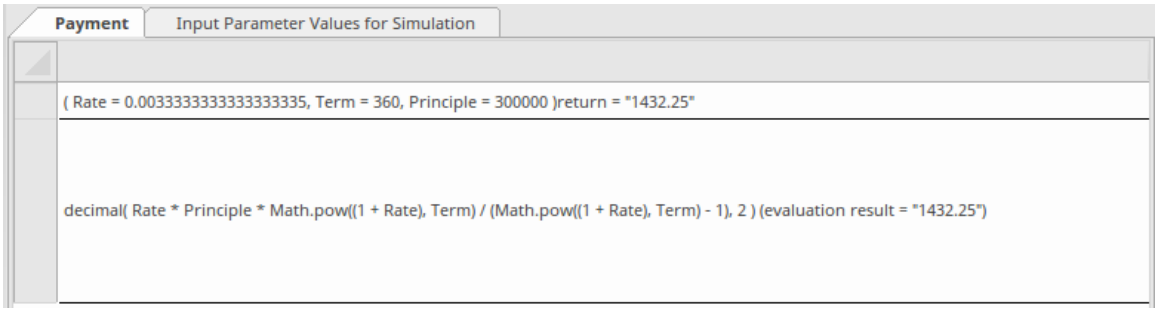
implemented as a Boxed Expression can be tested as well.

Take the 'Payment' element as an example. This BKM will calculate the monthly repayment based on interest rate, number of terms and principal amount.

We could provide test values such as these:



Click on the Simulation button on the tool bar; this result is obtained:



The runtime parameter and return values will be displayed with an equals sign '=' followed by the runtime value. This value is also displayed as a label against the element on its parent diagram.

In this example, given an annual Rate of 4% for 30 years and a principal of $300,000, the monthly repayment is $1,432.25.

Note: The DMN Library already has a PMT function defined; this example mainly demonstrates how Literal Expression works and how to test it with a set of arguments.

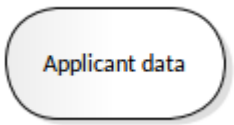
InputData

An InputData element is used to input into Decisions a set of values that originate outside the model. That set of values is used for evaluating Decisions. It derives its type and a set of values from an ItemDefinition.

Overview

InputData elements are created by dragging an

icon from the Toolbox onto a DMN diagram.



The name of the InputData element must be unique and not duplicate the name of any other Decision, InputData, Business Knowledge Model, Decision Service, or Import in the decision model.

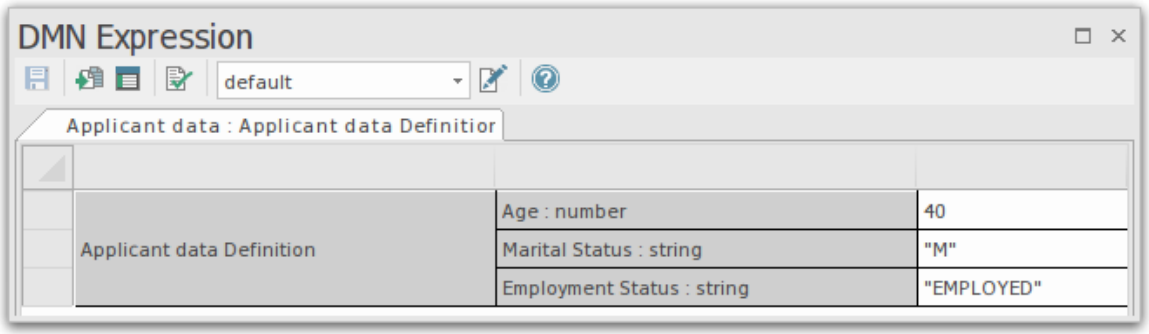
Referencing an ItemDefinition

The structure of the data, as well as sets of values for an InputData element, are defined in an ItemDefinition element. A DMN InputData element must be referenced (typed) by an ItemDefinition by either:

- Clicking on the icon on the DMN Expression window of the InputData element or
- Selecting the InputData element and pressing Ctrl+L to select the ItemDefinition from the dialog

InputData properties

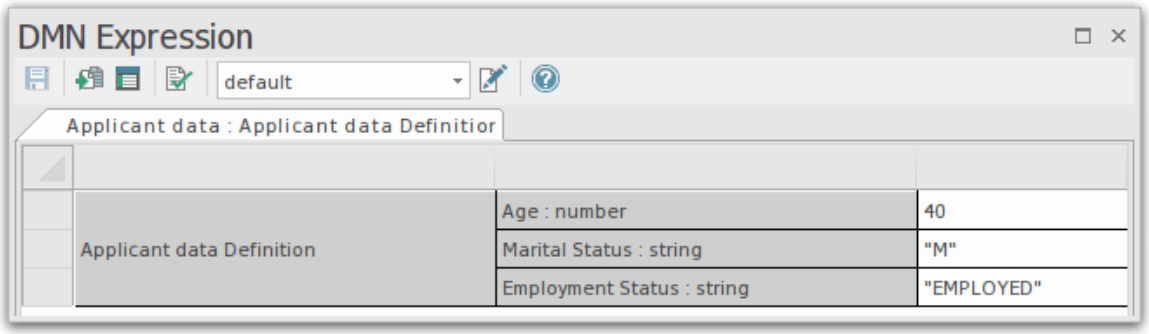
The properties of an InputData element are accessible via the DMN Expression window. Double-click on the InputData element to open this window.



The DMN Expression window provides a view of the data structure as well as access to Data Sets that can be used in simulations.

InputData DMN Expression

The DMN Expression window provides a view of an InputData's data structure, options to alter the value of Items, and access to Data Sets that can be used in simulations.



Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select / create an InputData
Other	Double-click on a DMN InputData element

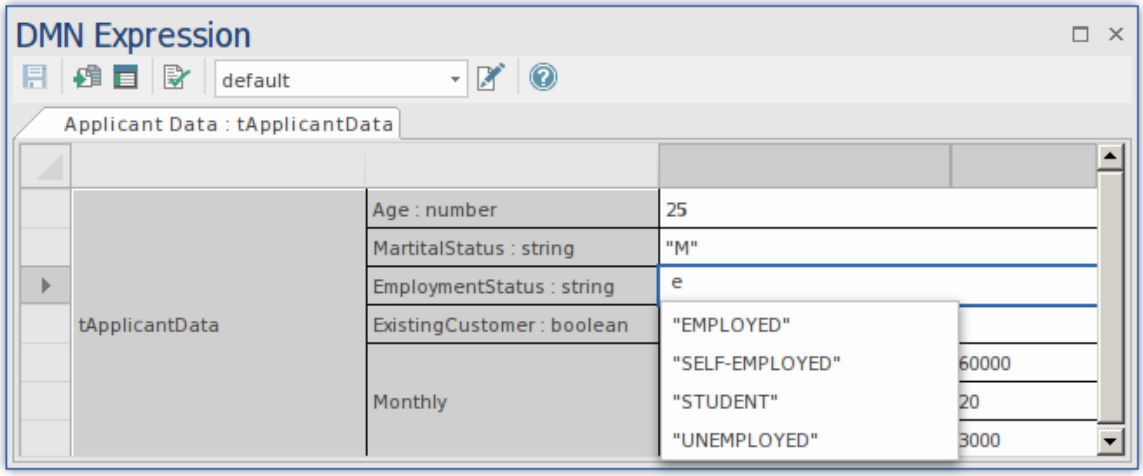
Toolbar Options

Option	Description

	Saves the configuration to the current InputData element.
	Sets the InputData's type by selecting a reference to an ItemDefinition.
	Opens the ItemDefinition element that is referenced by this InputData as its type definition.
	Runs a validation of the InputData. Enterprise Architect will perform a series of validations to help you identify errors in the InputData.
	Option to select a Data Set as defined in the ItemDefinition that references this InputData.
	Opens the dialog for editing data sets for this input data. Each InputData can define multiple data sets. With this feature, the DMN Simulation can quickly test the results of a decision by choosing different data sets.

Auto Completion

If the InputData has a field with 'Allowed Value' defined, then the field can be populated by selecting the field, pressing on the Spacebar, then selecting an option from the drop-down.



Data Sets

Data Sets are defined in the ItemDefinition referenced by the InputData element. Using the toolbar drop-down you can select a data set from the ItemDefinition. Once a set is selected you can alter the values of the items. You can also add new Data Sets by opening the Edit Data Set window using the icon.

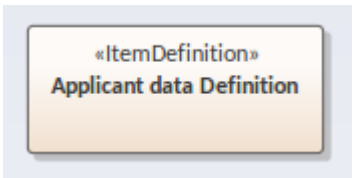
ItemDefinition

Fundamental to creating Decision Models is the definition of the structure of data items used within the model. An ItemDefinition is used to define the structure of the input data and, optionally, to restrict the range of allowable values of the data. ItemDefinitions can range from a simple single type through to a complex structured type.

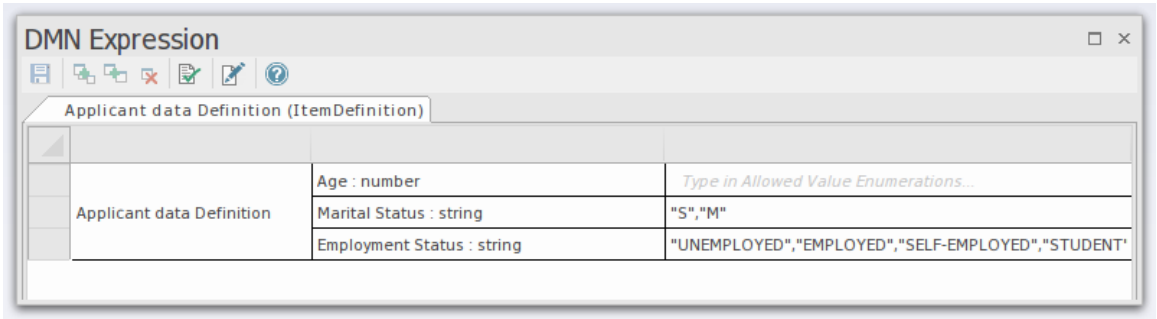
Overview

ItemDefinition elements are created by dragging a

icon from the DMN Toolbox page onto a DMN diagram.



The core properties of an ItemDefinition element are accessed via the DMN Expression window.



Access

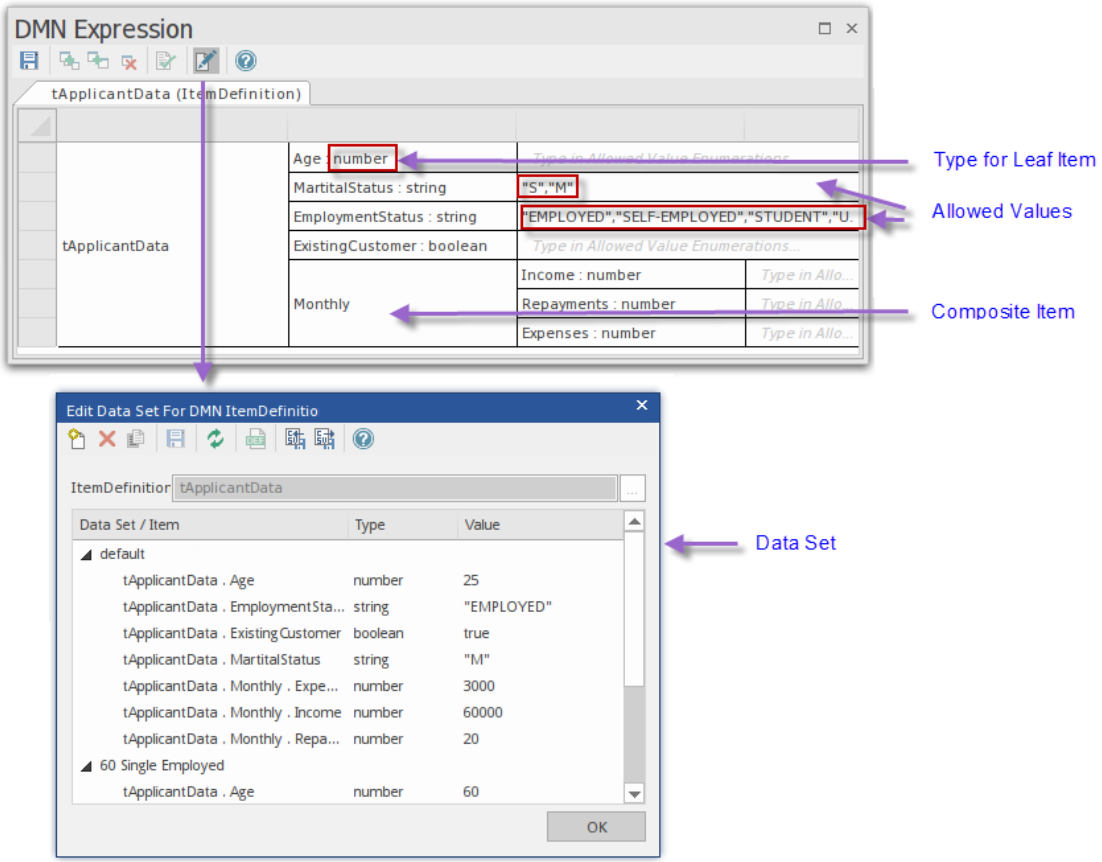
To open the DMN Expression window for an ItemDefinition

Element:

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression, then select or create an ItemDefinition
Other	Double-click on a DMN ItemDefinition

DMN Expression and Data set

This image is an overview of the DMN Expression window, showing a complex data item and the layout of the key fields used in the definition of the data. Included is a view of a Data Set defined using this ItemDefinition. A Data Set is an 'instance' of data conforming to an ItemDefinition, which contains a set of values to be used in the DMN simulation.



As ItemDefinitions are foundation elements in the model, it is recommended that they are validated before going on to use them in the model. This will ensure that any issues are resolved early on in the process of creating a complex model.

For more details on setting up ItemDefinitions, see these Help topics:

- *DMN Item Definition, Data Set and Input Data*
- *Types of Component*
- *ItemDefinition Allowed Value*
- *DMN Expression Auto Completion*
- *DMN Expression Validation*

Item Definition Toolbar

This table provides descriptions of the features accessible in the DMN Expression window when an ItemDefinition element is selected.

Toolbar Options

Option	Description
	Saves the configuration of the current ItemDefinition.
	Creates a new data component as a child of the selected component.
	Creates a new data component as a sibling of the selected component.
	Deletes the selected data component.
	Validates the ItemDefinition; Enterprise Architect will perform a series of validations to help you identify any errors in the ItemDefinition.
	Opens the 'Edit Data Set' dialog, in which you can create and edit 'instances' of the

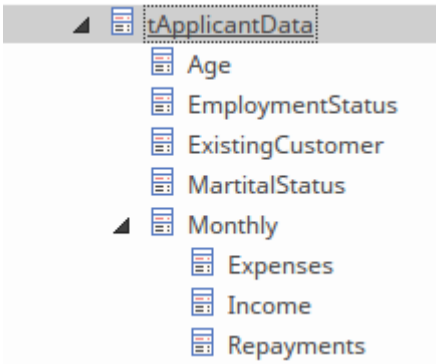
	ItemDefinition for use by InputData elements.

ItemDefinitions and Data Sets

An ItemDefinition describes the types and structures of data items used in a Decision model. It serves as the data type definition for InputData elements, Decision elements and Business Knowledge Model parameters. An ItemDefinition can also define data sets that provide sets of values for use in DMN Simulations. Switching between different data sets provides the ability to do 'what-if' analysis using the Decision model.

ItemDefinition Structure

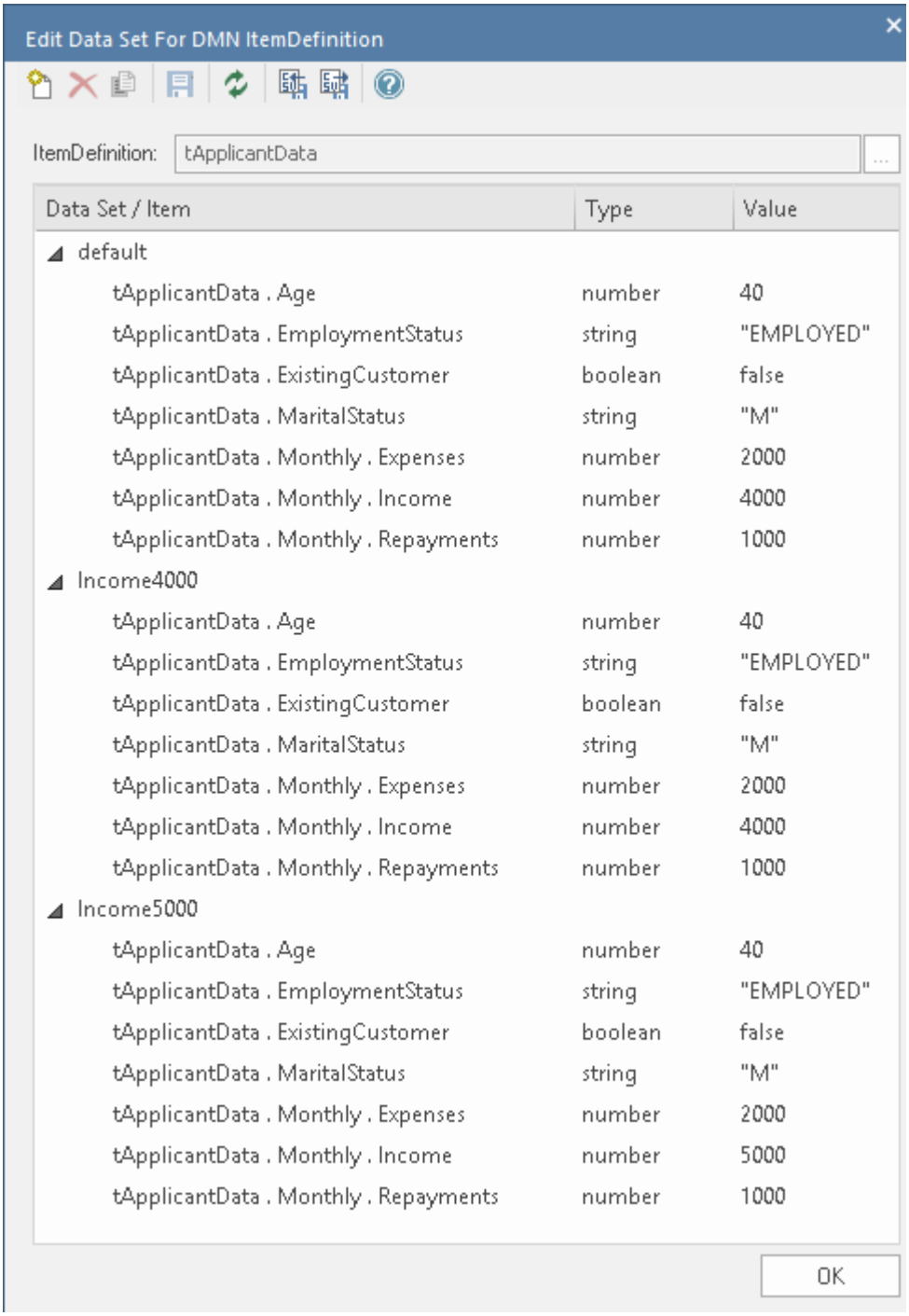
A complex ItemDefinition consists of nested elements. For example, *tApplicantData* is structured as:



The tApplicantData ItemDefinition example is a composite type of five child items. 'Monthly' is composed of three children (Expenses, Income and Repayments). The Leaf components (non-composite), will have a primitive type such as number, string or Boolean.

Data Set

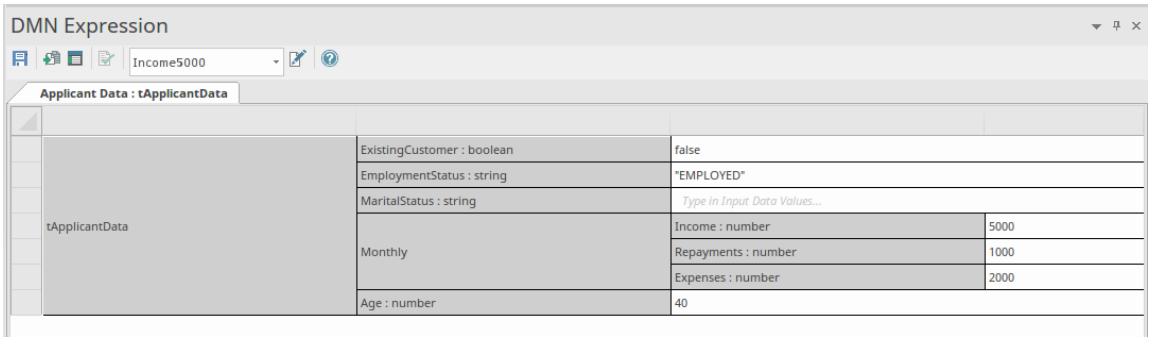
The ItemDefinition's Data Set can be viewed and edited using the icon on the Toolbar. With the 'Edit Data Set' dialog, you can add, delete and duplicate the data sets. There is also support for CSV import and export of data sets.



As shown in the example, the ItemDefinition for *tApplicantData* defines three data sets:

- default
- Income4000
- Income5000

Each data set can be viewed in an InputData element that is typed to the ItemDefinition. For example the 'Applicant Data' InputData element is typed to the 'tApplicantData' ItemDefinition. The DMN Expression window for 'Applicant Data', illustrated here, shows the data values according to the data set selected in the drop-down list in the window toolbar (*Income5000* in this case).



Setting a Reference to an ItemDefinition

A DMN InputData element is set to be referenced (typed) by an ItemDefinition using either:

- The icon on the DMN Expression window of the InputData element or
- Selecting the InputData element and pressing Ctrl+L to select the ItemDefinition from the dialog

There are other cases of using ItemDefinitions; for instance, when setting the type for an Input Parameter in a BKM or an output parameter in a Decision Table.

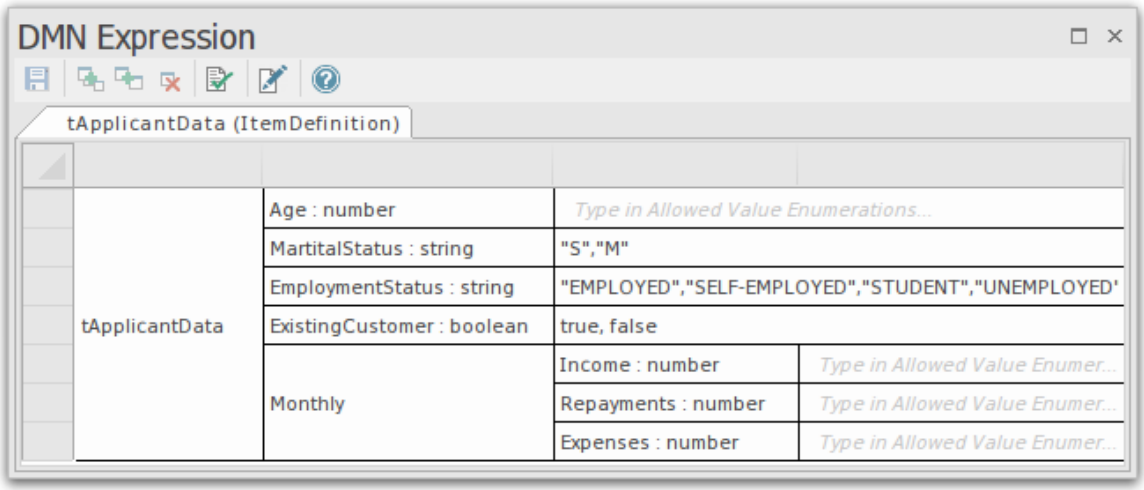
Types of Component

An ItemDefinition element can be defined as a tree of components that consists of only one of either:

- A built-in type or
- A Composition of ItemDefinition elements

In this tree of components, if a component is a 'leaf' that has no child components, it must be set as a built-in type. If an ItemDefinition has child components, it is those child/leaf components that are set as a built-in type.

For example *Applicant Data* and *Monthly* are compositions, whereas *Age* and *Expenses* are leaves set to a built-in type:



The FEEL language has these built-in types:

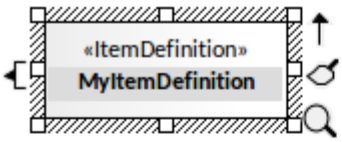
- **number**
- **string**
- **boolean**
- days and time duration
- years and months duration

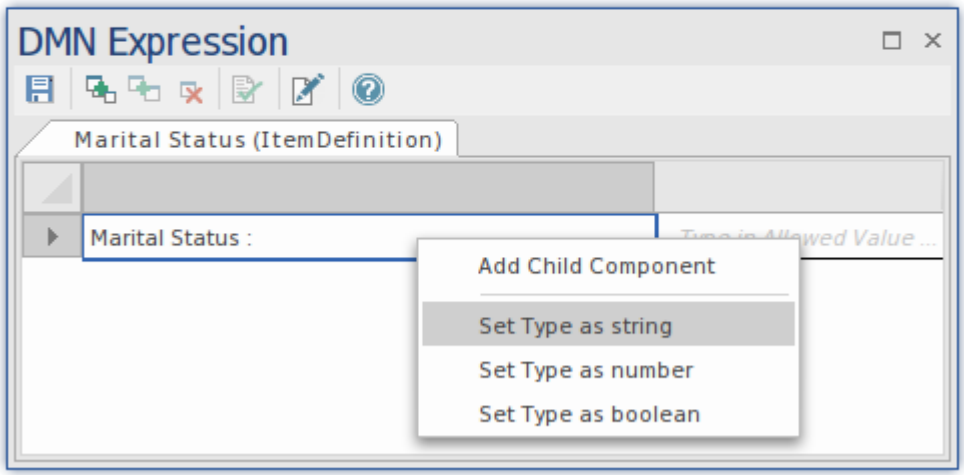
- time
- date and time

Note: 'number', 'string' and 'boolean' are supported by Enterprise Architect for simulation.

To set a type for a 'leaf' ItemDefinition, you can use one of three methods:

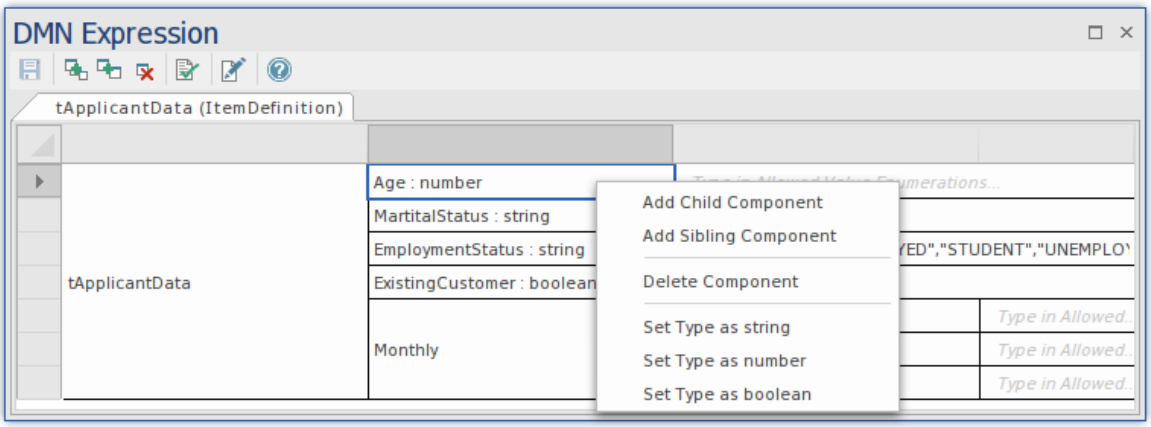
- Select the appropriate context menu option in the DMN Expression window (Recommended)





- Type ': string', ': boolean' or ': number' after the name in the cell in the DMN Expression window
- Type 'string', 'boolean' or 'number' as the value of the tag 'Type' in the Properties window for the ItemDefinition

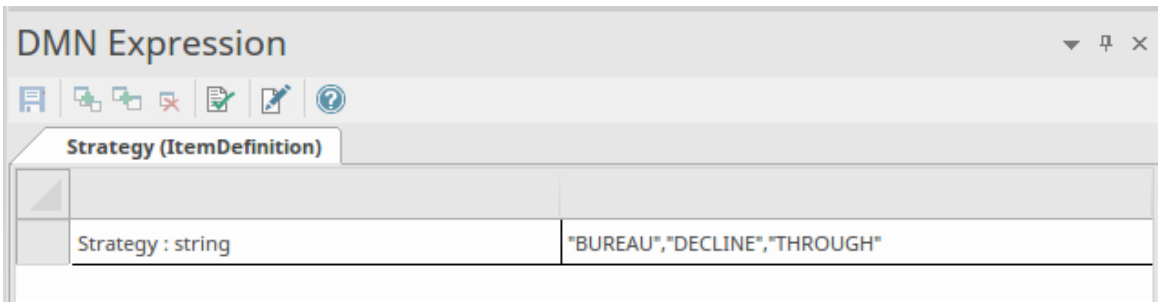
For composite ItemDefinitions, the context menu also offers options to create a child or a sibling component, or to delete the selected item:


DMN Expression
tApplicantData (ItemDefinition)
tApplicantData
Age : number
MartitalStatus : string
EmploymentStatus : string
ExistingCustomer : boolean
Monthly
Add Child Component
Add Sibling Component
Delete Component
Set Type as string
Set Type as number
Set Type as boolean
Type in Allowed...

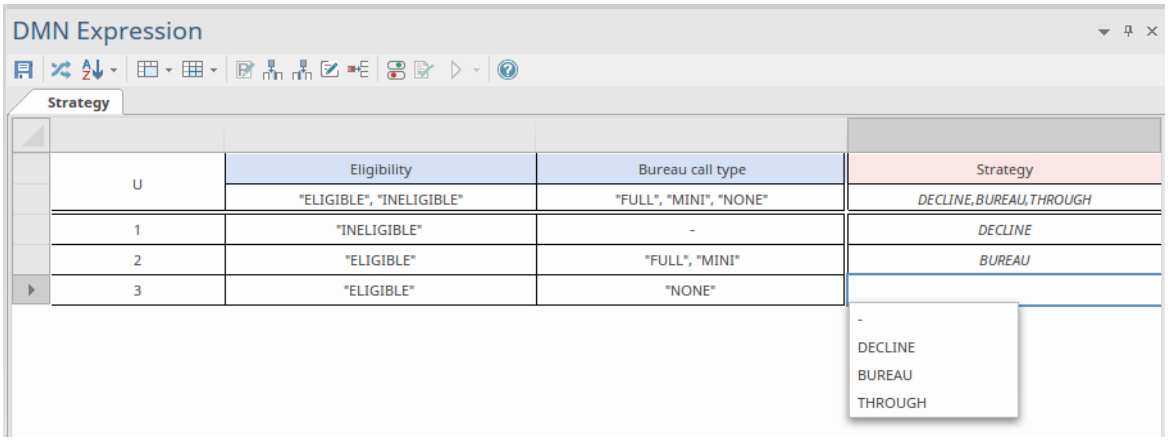
Allowed Value Enumerations

When defining data inputs for a Decision, it is common to want to restrict the set of allowable values for an input. For example, you might want to restrict the allowed values for Marital Status to just two options, 'Single' and 'Married'.

You can specify the allowed values for any leaf component of an ItemDefinition. Initially, the data field for a leaf component contains the text *Type in Allowed Value Enumerations*. You simply type over this text with the allowed values. For example, the ItemDefinition *Strategy* has three allowed values - BUREAU, DECLINE and THROUGH.



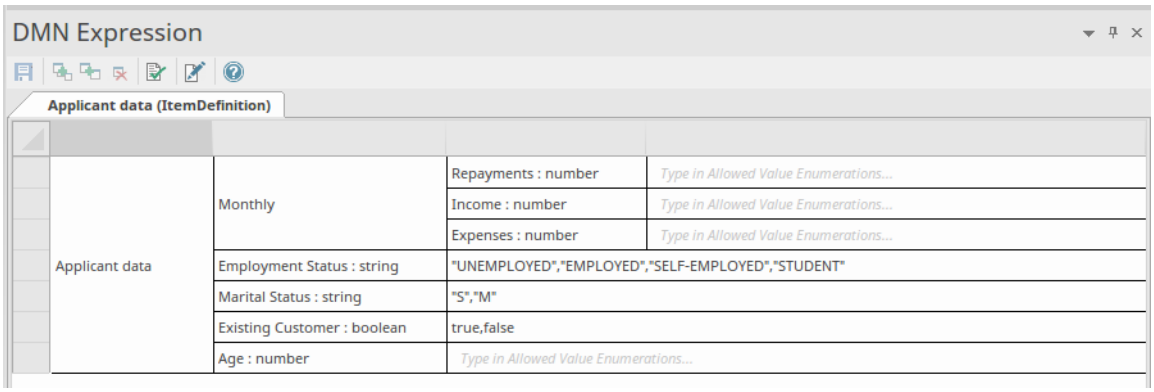
Allowed Value Enumerations are also used to support Auto Completion. When specifying values for an InputData element or an input parameter that references an ItemDefinition in which Allowed Values have been defined, the user can simply press the Spacebar and choose a value from the list.



You can also autocomplete by typing the first letter of the option you want to enter.

The input parameters and Output Clauses of Decision Tables also support the specification of allowable values. This restricts the values that can be used when defining the rules in the table, but also allows the user to fast fill the rules by pressing the Spacebar then selecting the required item.

A more complex ItemDefinition can include a number of Allowed Value Enumerations; for example:



Data Sets

Each InputData element typed by an ItemDefinition has a set of components, and multiple data sets can be defined to provide different sets of values for those components. With this feature, a user performing a DMN Simulation can quickly test the result of a decision by choosing different data sets. The data sets are associated with and based on the ItemDefinition, but you can also work on them via the InputData element.

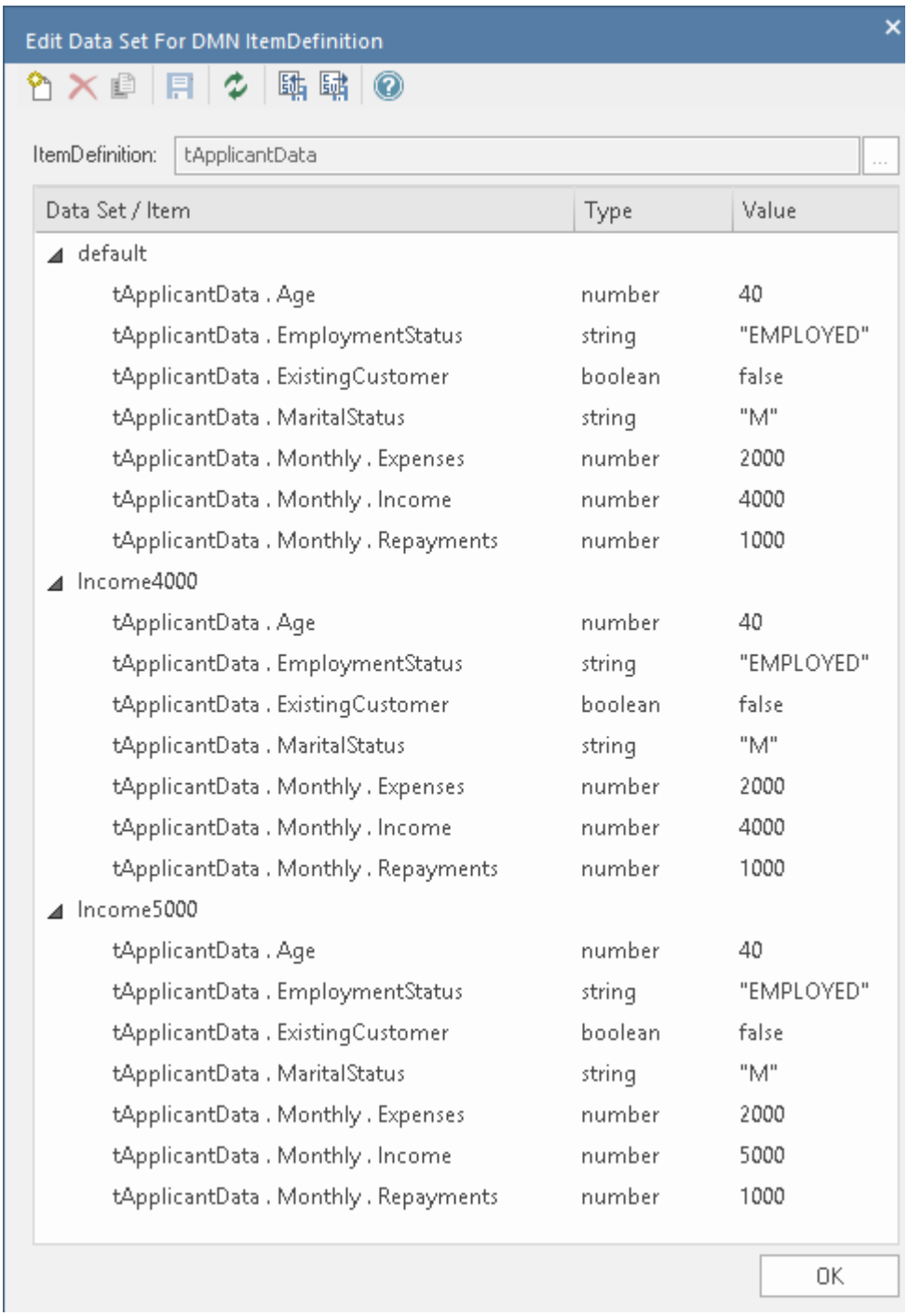
You add or update data sets using the 'Edit Data Set' dialog, which you invoke from the DMN Expression window for either the ItemDefinition or the InputData element. Initially, the 'Edit Data Set' dialog shows a single set of components with no values, under the set name 'default'. You can either leave this set with no values, or provide values; either way, you can use this as a template to duplicate for new data sets. You cannot delete the 'default' data set.

When you access an InputData element in the DMN Expression window, the values in the 'default' data set are shown against the components of the element. You can then click on the drop-down arrow in the toolbar and select any other data set from the list. Note that if you leave the 'default' data set untouched, you can create a duplicate 'default' data set and assign values to it, and that 'default' set will provide the values when you initially access the InputData element.

You can duplicate and delete any other data set that you create, export the data sets to a CSV file and import them

from a CSV file.

Note that if you create a data set and do not enter values, it is discarded when you close the dialog.



Data Set / Item	Type	Value
default		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	4000
tApplicantData . Monthly . Repayments	number	1000
Income4000		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	4000
tApplicantData . Monthly . Repayments	number	1000
Income5000		
tApplicantData . Age	number	40
tApplicantData . EmploymentStatus	string	"EMPLOYED"
tApplicantData . ExistingCustomer	boolean	false
tApplicantData . MaritalStatus	string	"M"
tApplicantData . Monthly . Expenses	number	2000
tApplicantData . Monthly . Income	number	5000
tApplicantData . Monthly . Repayments	number	1000

Access

Ribbon	Simulate > Decision Analysis > DMN > DMN Expression > click on InputData item : icon
Other	In a diagram, double-click on the DMN InputData element : icon.

Toolbar Options

Option	Description
	Click on this button to create a new data set.
	Click on this button to delete the selected data set.
	Click this button to duplicate the selected data set.
	Click on this button to save the data sets to the InputData.
	Click on this button to reload the data sets for the InputData.

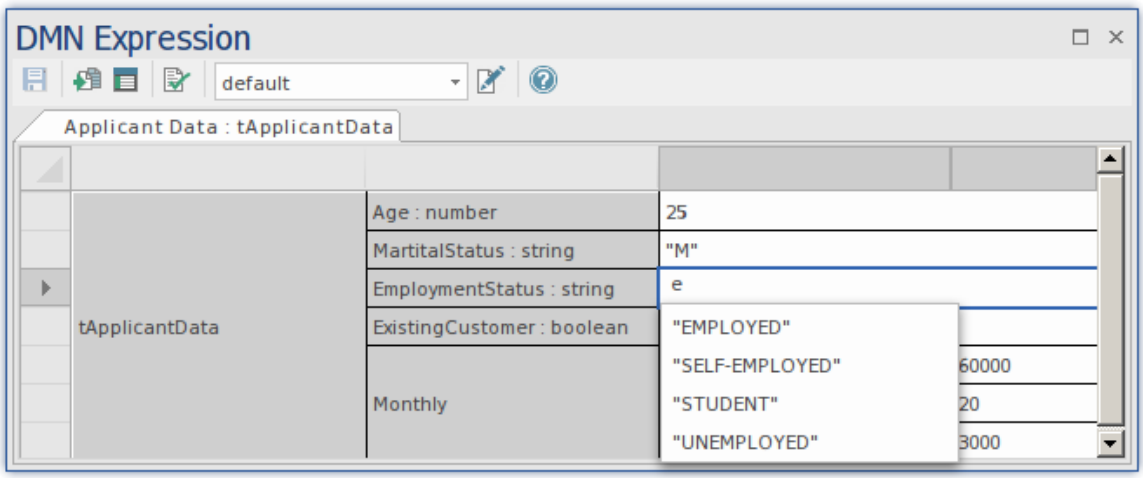
	Click on this button to import data sets from a CSV file.
	Click on this button to export the datasets to a CSV file.

Exchange Data Sets using DataObjects

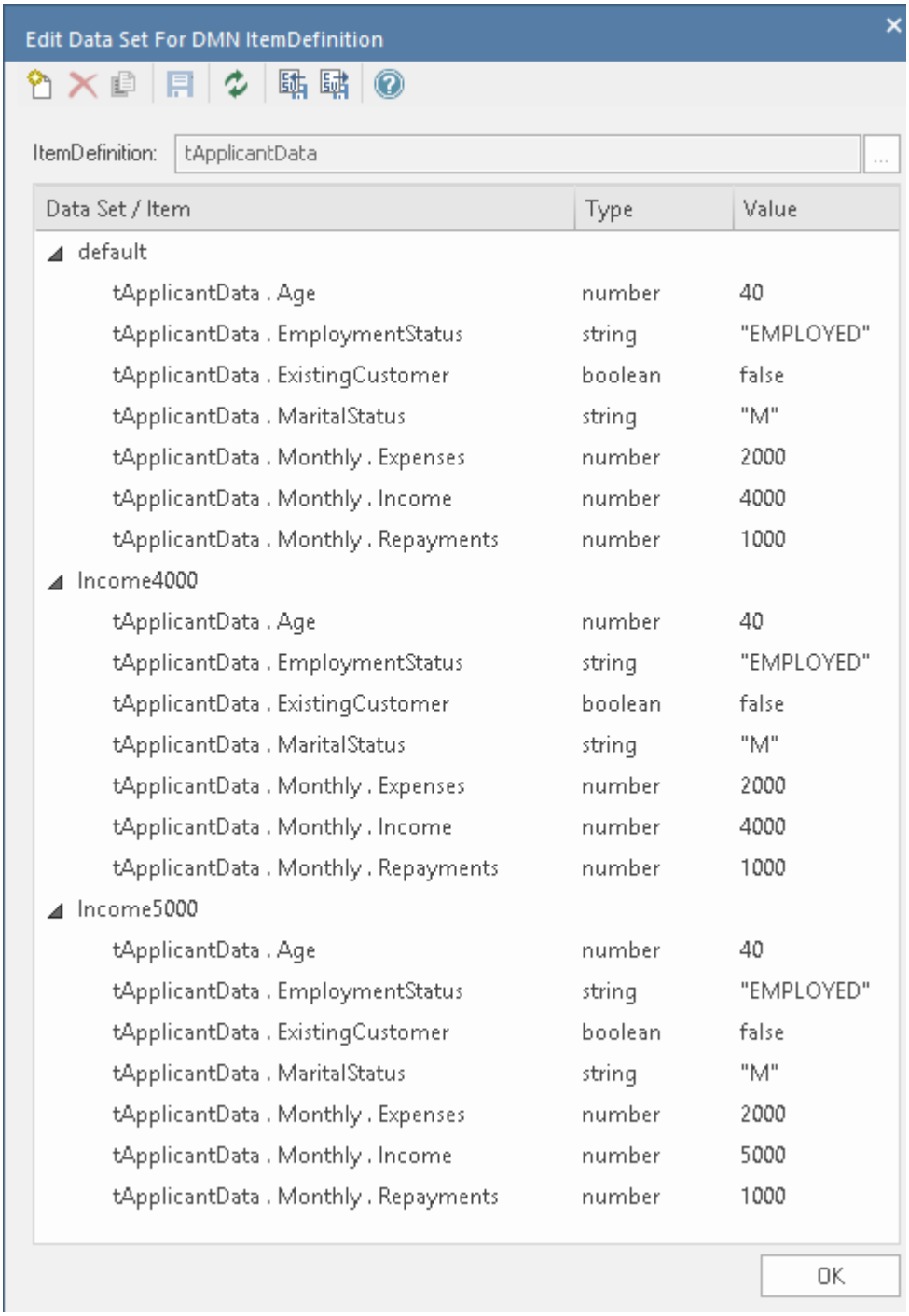
When testing code generated from a DMN model, or when simulating Business Process Model and Notation (BPMN) models that call DMN models, you need a means of exchanging data sets. For example, in a BPMN call of a DMN model, a BPMN DataObject is used to store the set of variables that will be passed on to the DMN model that it is calling. This DataObject needs to be populated with data fitting the DMN InputData's data structure ready to be passed to that InputData object. This same BPMN DataObject is used when testing the code generated from a DMN model.

This topic describes the process of creating BPMN DataObjects from DMN Data Sets.

A Data Set is stored in a DMN InputData element and can be accessed using the icon on the DMN Expression window.



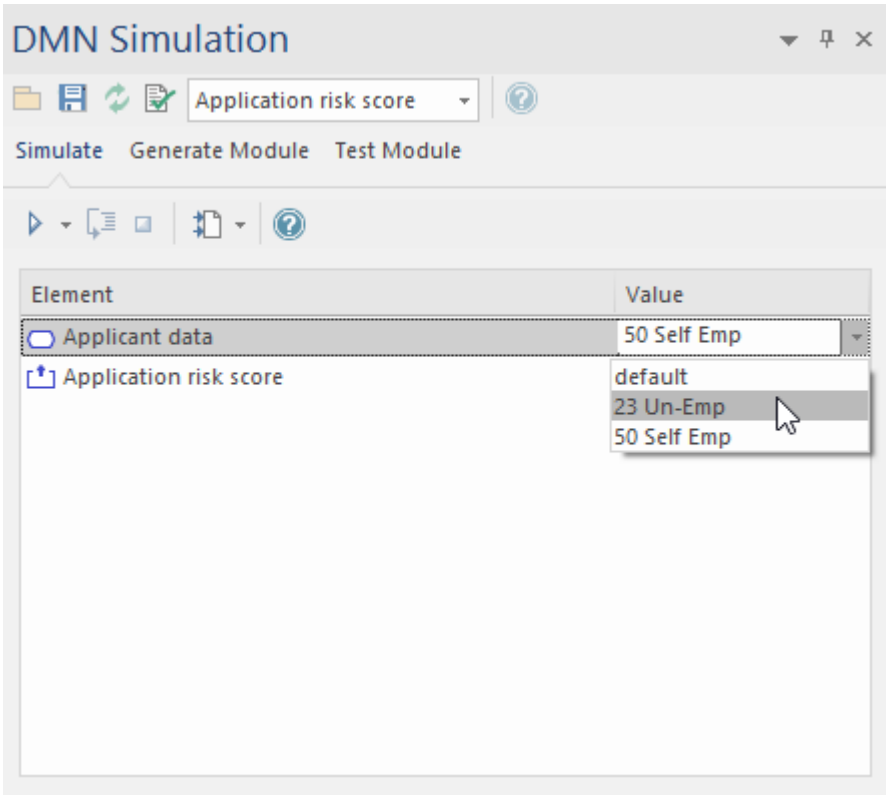
This opens the InputData's 'Edit Data Set' dialog, which can contain multiple sets of values:



There are two options to transfer the Data Set to a DataObject:

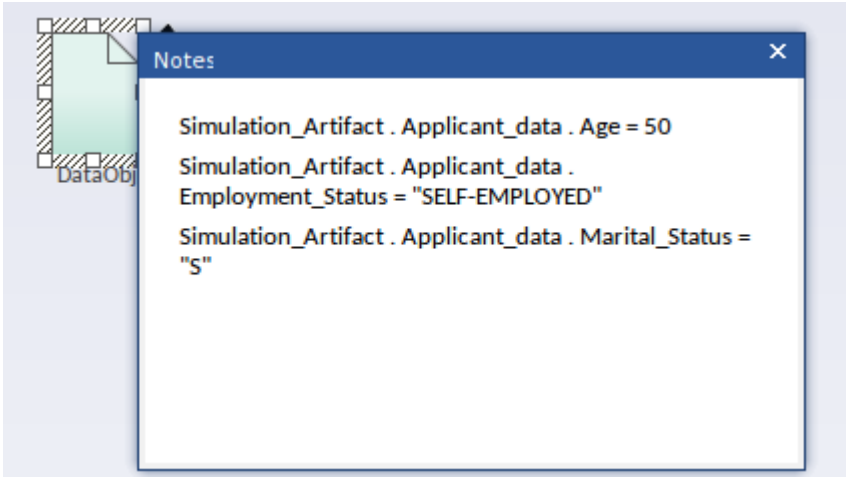
1. Direct

- Create a BPMN DataObject under a Package in the Browser window.
- Open the DMN Simulation window



- Select a Data Set from the 'Value' drop-down
- Click on the icon on the DMN Simulation window; this opens the 'Select Element' dialog
- Select the BPMN DataObject element
- Click on the OK button

The Data Set is now viewable in the Notes of the DataObject.

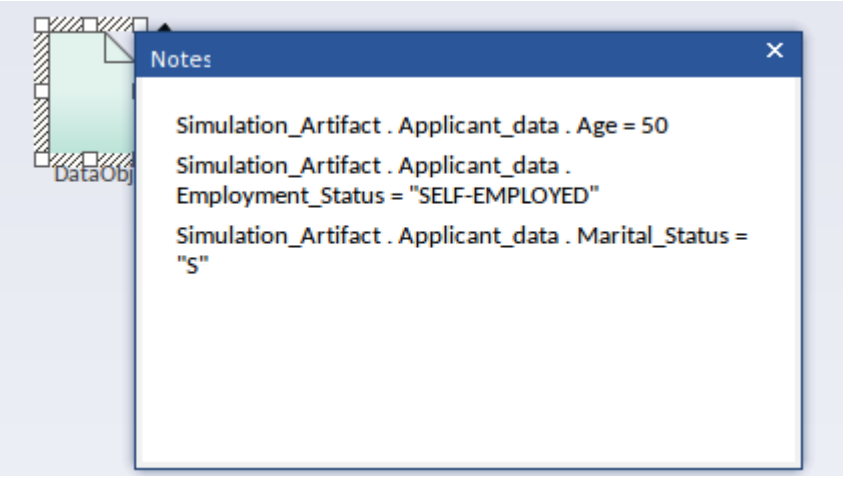


2. Manual

To manually exchange this Dataset:

- Open the DMN Expression window for the InputData element
- Click on the Edit DataSet icon ; this opens the 'Edit Data Set' dialog
- Use the CSV Export icon to export these details to a file

The text in the CSV file can be added as text in the Notes of a BPMN DataObject element.



Decision Service

Portions of this topic have been used verbatim or are freely adapted from the DMN Specification, which is available at: https://www.omg.org/spec/DMN. This site contains a full description of the DMN and its capabilities.

A Decision Service exposes one or more decisions from a Decision model as a reusable element, which might be invoked internally by another decision in the Decision model, or externally by a task in a BPMN process model.

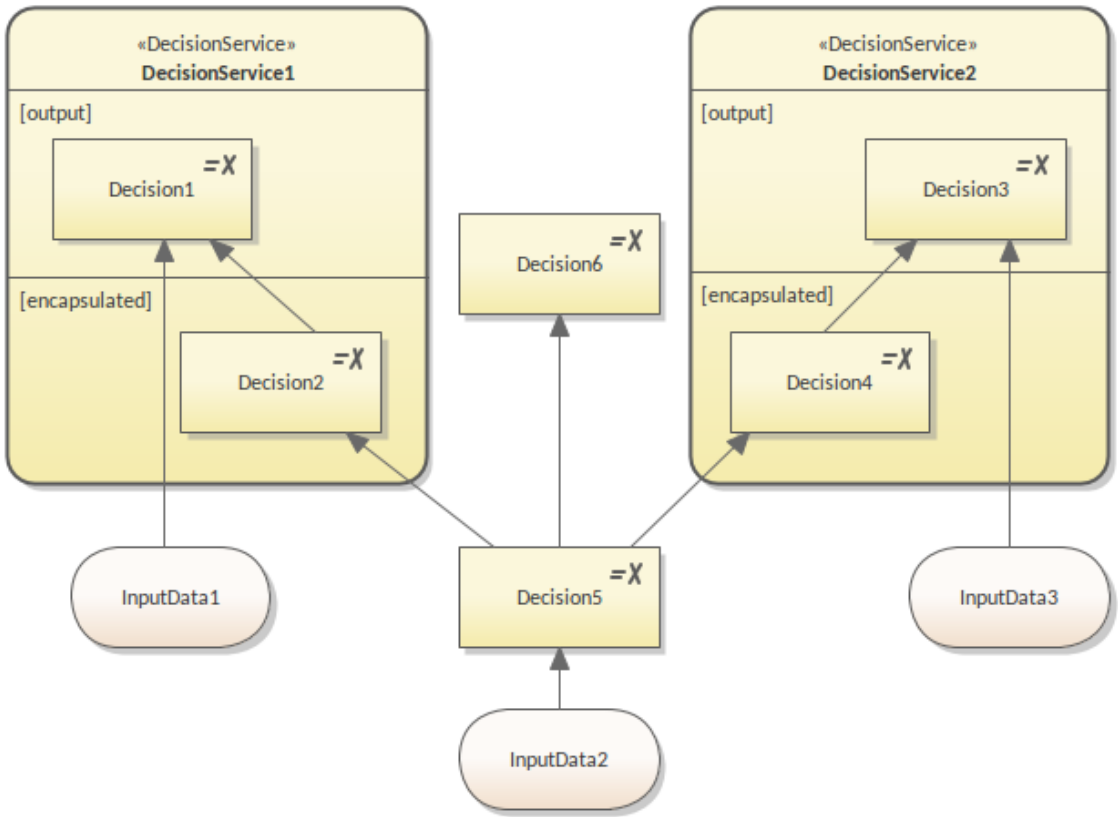
When the Decision Service is called with the necessary input data and input decisions, it returns the outputs of the exposed decisions.

The Interface of a Decision Service

The interface to the Decision Service consists of:

- Input data - instances of all the input data required by the encapsulated decisions
- Input decisions - instances of the results of all the input decisions
- Output decisions - the results of evaluating (at least) all the output decisions, using the provided input decisions and input data

When the Decision Service is called with the necessary input data and input decisions, it returns the outputs of the exposed decisions.



This figure shows a Decision model that includes six decisions and three items of input data.

For DecisionService1, the:

- Output decision is {Decision1}
- Input decision is {Decision5}, and
- Input data is {InputData1}

As Decision1 requires Decision2, which is not provided to the service as input, the service must also encapsulate Decision2; therefore the encapsulated decisions are {Decision1, Decision2}.

It is obvious from the figure that Decision6, Decision3, Decision4 and InputData3 are not required by any decisions from DecisionService1. What about InputData2? Although it is required by Decision5, which is required by DecisionService1, InputData2 is actually not required by

DecisionService1. This is because Decision5 is defined as the Input Decision. From the point of view of a Decision Service, we ignore any decisions or input data required by an Input Decision.

For DecisionService2, the:

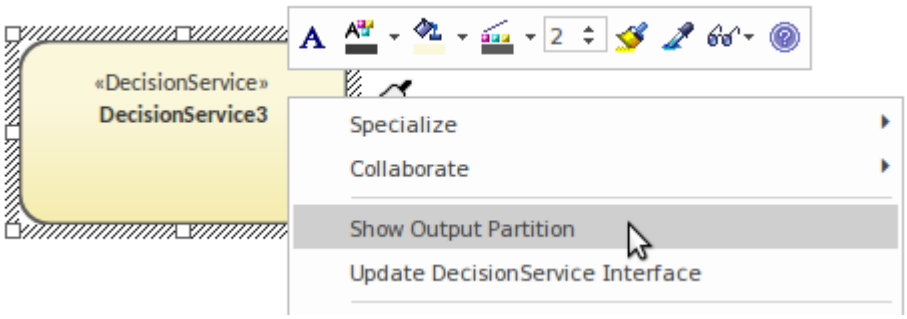
- Output decision is {Decision3}
- Input decision is {Decision5}, and
- Input data is {InputData3}

As Decision3 requires Decision4, which is not provided to the service as input, the service must also encapsulate Decision4; therefore the encapsulated decisions are {Decision3, Decision4}.

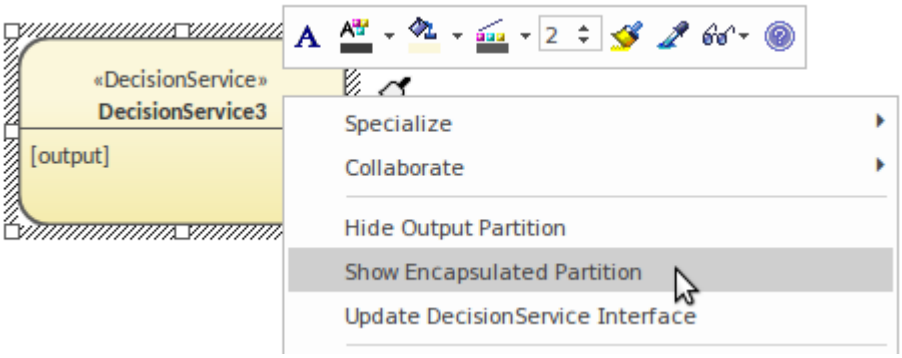
It is good practice to create a separate diagram for each Decision Service. In this way, the diagram will only contain the interface elements and encapsulated decisions for the Decision Service; the elements that are not relevant will not appear on the diagram.

Modeling a Decision Service

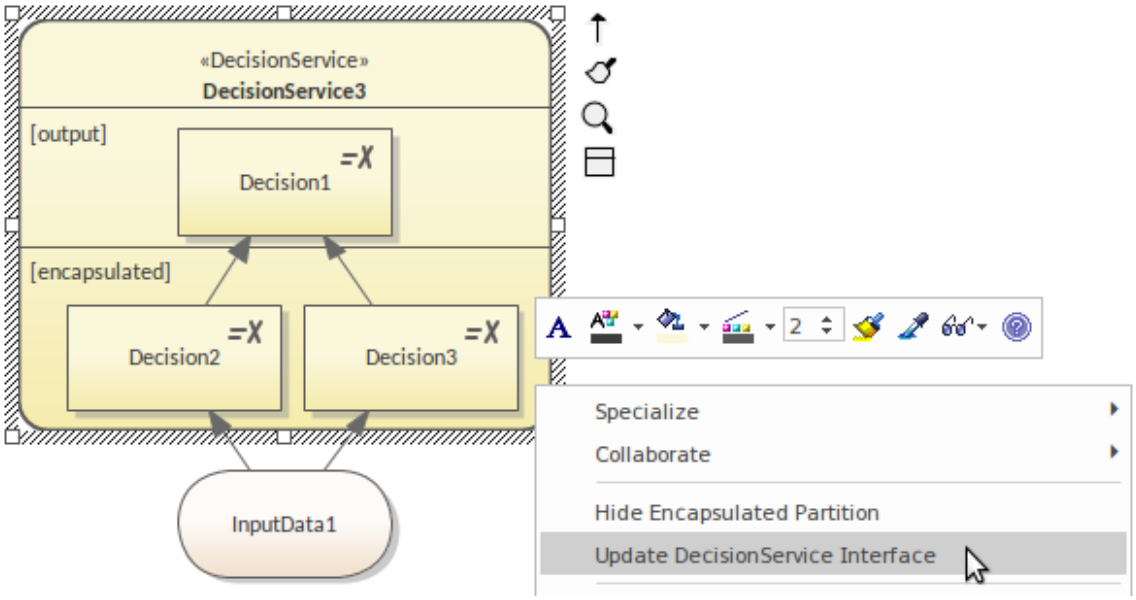
We can create a Decision Service element from the DMN pages of the Diagram Toolbox, and toggle [output] and [encapsulated] partitions from the context menu.



You can only show an [encapsulated] partition when an [output] partition is shown.



Once the decisions and input data are put in the correct partition(s), you must run the 'Update DecisionService Interface" command from the context menu to update the model.



Important: in order for the DMN simulation to work

properly, please update the Decision Service interface whenever you:

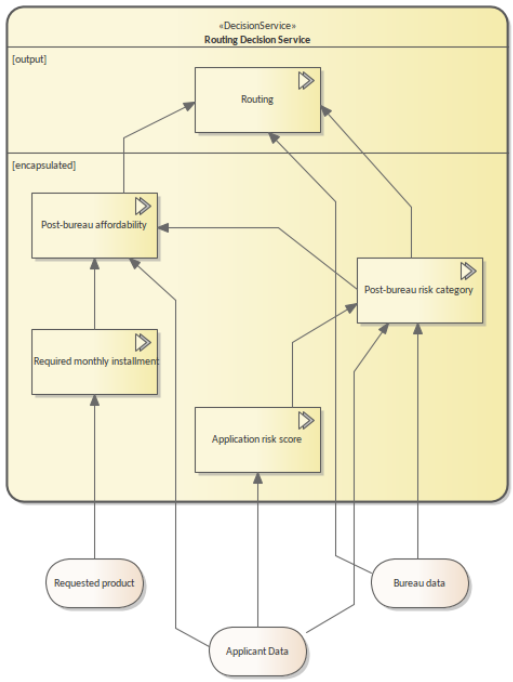
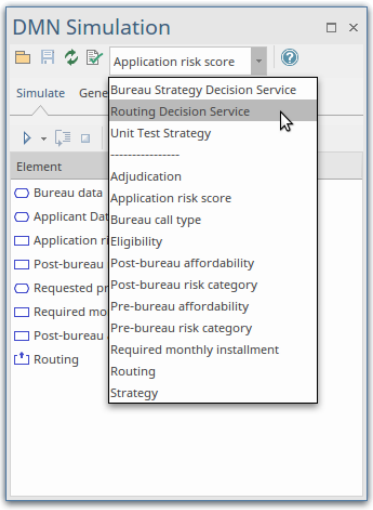
- Show/Hide the decision service partition(s)
- Add a decision to the decision service
- Remove a decision from the decision service
- Move a decision between partitions
- Add/Remove Decision Service Inputs: Input Data or Input Decisions

Simulating a Decision Service

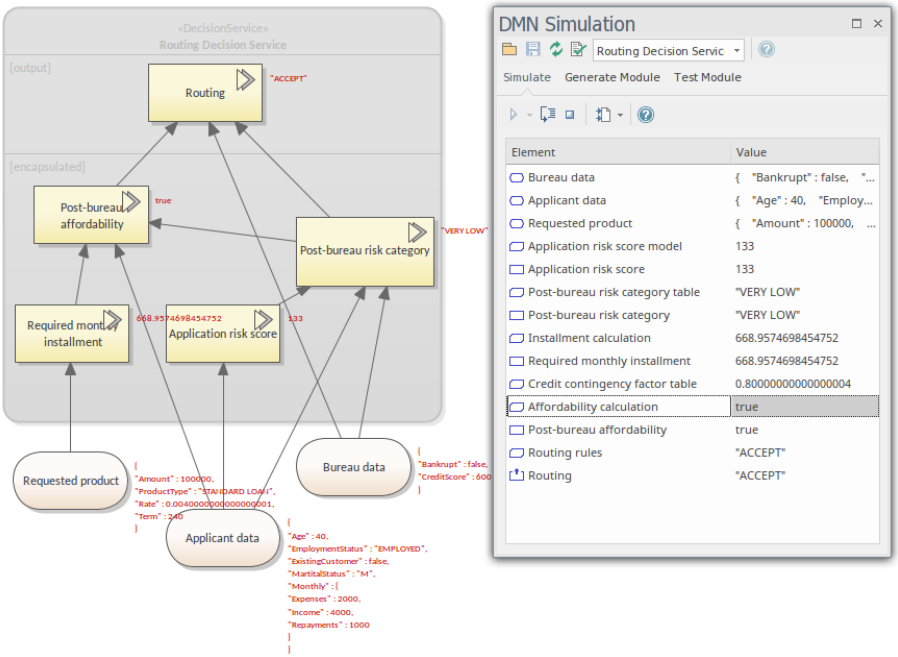
It is possible to perform a model simulation on a Decision Service.

Decision Service Simulation

To perform a model simulation on the Decision Service, work through these steps:

Step	Description
1	Drag a Simulation Configuration Artifact element onto a diagram from the 'DMN Components' page of the Toolbox, and double-click on it to open it in the DMN Simulation window.   By default, all Decision Service elements and each single decision are listed for

	selection in the drop-down field in the dialog toolbar.
2	Select a Decision Service element on which to run the simulation. In the example we chose 'Routing Decision Service', so three input data items and five encapsulated decisions (including one output decision) are loaded in the simulation list. Important: This list is drawn from the internal data of the Decision Service; make sure you run the 'Update DecisionService Interface' command from the context menu whenever the Decision Service model diagram is changed. Reload the Decision Model by clicking the 'Refresh' icon (third from the left) on the DMN Simulation window toolbar.
3	The input data and decisions are in the correct execution order. For example, 'Application risk score' will be executed before 'Post-bureau risk category', 'Post bureau affordability' and 'Routing'. For each Input Data element, click on the drop-down arrow in the 'Value' field and select the Data Sets to provide input data

	values. Validate the input data and decisions, and make any necessary corrections using the DMN Expression window. On the DMN Simulation window, click on the Save icon and on the button on the toolbar.
4	The runtime execution result is shown both in the list and on the diagram. You can also click on the 'Step-through' icon on the toolbar to debug the DMN model.  A good practice is to keep the DMN Expression window open while debugging. The run time status of the expression (such as Decision Table, Boxed Context, Literal Expression or Invocation) will show the details of the

	logic encapsulated by the Decision or invoked Business Knowledge Model.

Code Generation and Test Module

After a Decision model is created and simulated, you can generate a DMN module in Java, JavaScript, C++ or C#. That DMN module can be used with the Enterprise Architect BPSim Execution Engine, Executable StateMachine, or your own project.

Enterprise Architect also provides a 'Test Module' page, which is a preprocess for integrating DMN with BPMN. The concept is to provide one or more BPMN2.0::DataObject elements, then test if a specified target Decision can be evaluated correctly or not.

If any error or exception occurs, you can create an Analyzer Script to debug the code of the DMN module and Test Client.

After this 'Test Module' process, Enterprise Architect guarantees that the BPMN2.0::DataObject elements will work well with the DMN Module.

You then configure BPSim by loading DataObjects and assigning DMN module Decisions to BPSim Properties, which will be further used as conditions on the Sequence Flows outgoing from a Gateway.

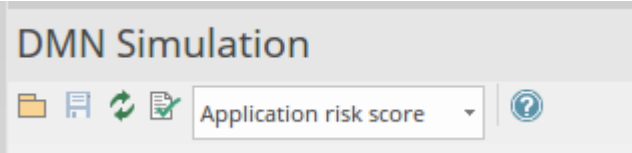
Access

Ribbon	Simulate > Decision Analysis > DMN > Open DMN Simulation > Generate

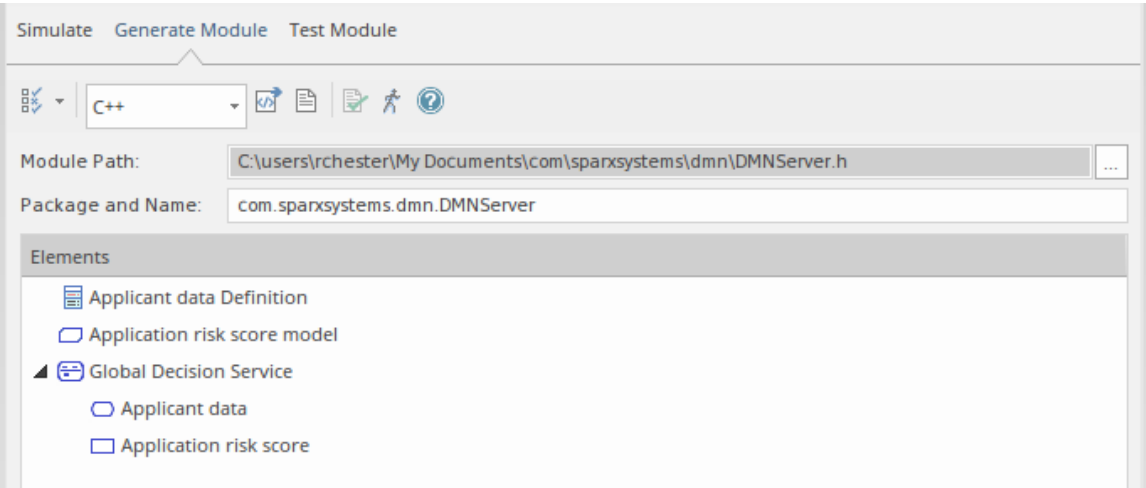
	Module

DMN Module: Code Generation

On the DMN Simulation window, select the DMN structure you want to generate the module from, in the data entry field of the Toolbar.



Click on the 'Generate Module' tab, and then Ctrl+click on the names of the DMN elements you want to generate to the server.



In the data entry field in the tab toolbar, select the language to generate in, and in the 'Module Path' field click on the ... icon and browse to the path location to generate the module into (note, for Java the path has to match the Package structure).

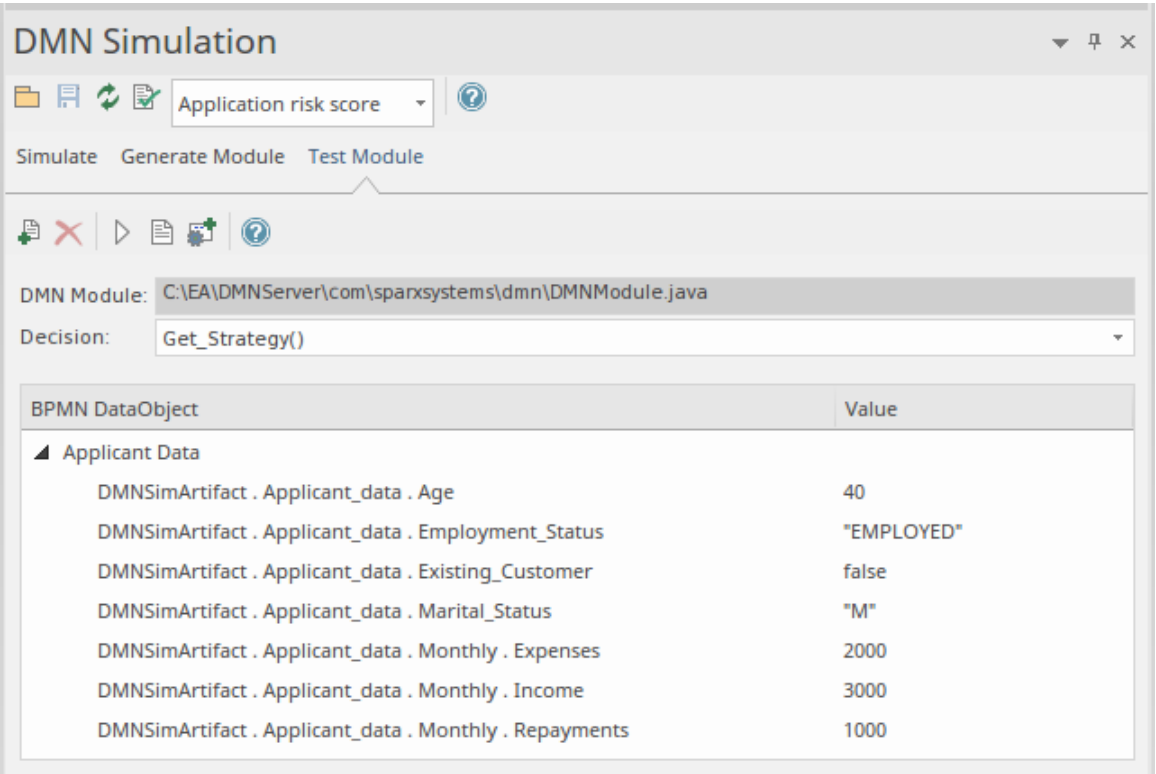
Click on the Generate button ( ).

When the generation is complete, click on the button to open the 'Test Module' tab for the generated module.

DMN Server: Test Module

When you use the button to select the 'Test Module' tab, the 'DMN Module' field will be filled automatically with the generated DMN Server path of the module you most recently generated on the 'Generate Module' tab. If necessary, on the 'Decision' field click on the drop-down arrow and select the required Decision.

Click on the Add BPMN DataObject button ( ) in the Toolbar and select one or more (Ctrl+Click) BPMN2.0 DataObject(s) to add to the list in the main panel.



Now click the Run button on the toolbar. In the System Output window, this message indicates the DMN Server and

BPMN2.0 DataObject can work well with each other to evaluate the selected decision:

Running Test Client for DMN Server...

dmnServer.Application_risk_score: 133.0

Result : 133.0

The Running completed successfully.

If there are errors, create an Analyzer script by clicking the toolbar button and use the script to fix the issue.

Important: This 'Test Module' step is recommended before integrating DMNServer.java with the Enterprise Architect BPSim Execution Engine. See the *Integrate a DMN Module Into BPSim for Simulation* Help topic.

Code Generation & Connect to BPMN

- Generate the DMN Server in Java, JavaScript, C++, or C#
- Run/Debug tests of the Java version of the DMN Server
- Connect the DMN Server to the Enterprise Architect BPSim Execution Engine

Common Errors & Solutions

- Variable Types: as DMN models use the FEEL language (Simulate with JavaScript), typing variables is not compulsory; however, when generating code to languages that are compiled, you do have to type a variable - there are context menu options and tag values for setting the

type of a variable

- Since a DMN expression allows for spaces, in order to clarify the composite Input Data there must be a space before and after the '.' in the expression; for example, 'Applicant data . Age' is valid, whereas 'Applicant data.Age' is not valid
  Note that when using the Auto Completion feature this issue will not arise
- Running validation will help you locate most of the modeling issues; do this before simulation and code generation

Notes

- Compiling with Java requires full read-write access to the target directory; compilation will fail if the module path is set to just 'C:' or 'C:\Program Files (x86)'

Integrate Into BPSim for Simulation

The strength of DMN is its ability to describe business requirements through the Decision Requirement diagram and to encapsulate the complicated logic in versatile expressions such as the Decision Table and Boxed Context.

Equally, the strength of BPMN is its ability to describe business processes with a Sequence Flow of tasks and events, or to describe collaborations of processes with Message Flows.

The Decision Requirements diagram forms a bridge between Business Process models and decision logic models:

- Business Process models define tasks within business processes, where decision-making is required
- Decision Requirements diagrams define the decisions to be made in those tasks, their interrelationships, and their requirements for decision logic
- Decision logic defines the required decisions in sufficient detail to allow validation and/or automation

DMN provides a complete Decision model that complements a Business Process model by specifying in detail the decision-making carried out in process tasks.

The two examples demonstrated in this topic can be accessed from:

- EAExample Model | Model Simulation | BPSim Models
- Perspective | Business Modeling | BPSim | BPSim Case

Studies

There are two ways in which BPSim expressions use a DMN model:

- DMN's Decision Service - demonstrated by the Loan Application Process
- DMN's BusinessKnowledgeModel - demonstrated by the Delivery Cost Calculation

The process of integrating a DMN model with a BPSim model includes:

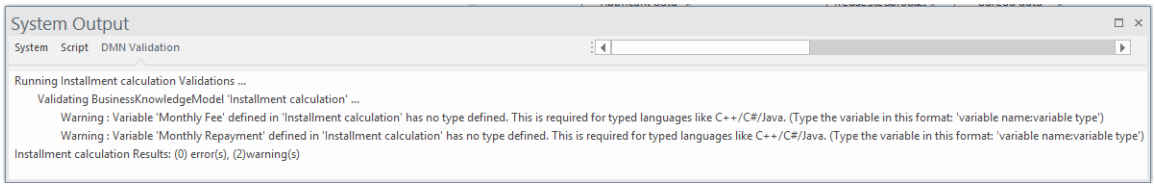
- DMN Model Validation, Simulation, Code Generation and Testing on the generated module
- Set up a usage dependency from the BPSim Artifact to the DMN Artifact
- Generate or update the BPMN DataObject from the DMN DataSet
- Create Property Parameters in BPSim to be used on tasks and Sequence Flows out going from Gateways
- Bind the DMN interface to BPSim Property Parameters

DMN Model Validation for Compiled languages such as Java

When you create a DMN model and simulate it in Enterprise Architect, the code driving the simulation is JavaScript; this means that the variables do not need to be explicitly typed (the variable type is inferred from the value assigned to it).

However, for languages such as C++, C# and Java, the

compiler will report an error that a variable does not have a type.

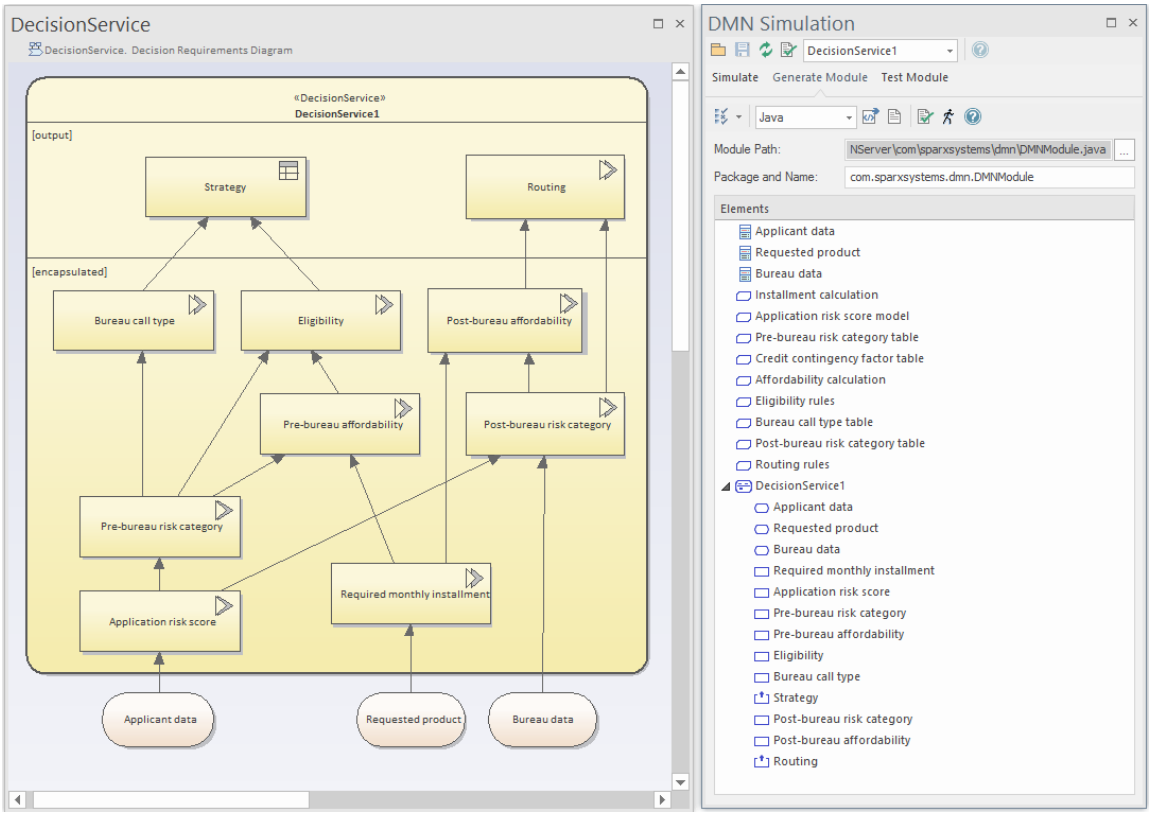


For generation to these languages you must run validation on the model and use the results to find variables that need their type set. For example:

- Business Knowledge Model parameter - select the BKM element to view in the DMN Expression window, click on the second button to open the 'Parameter' dialog, specify a type for the parameter
- Decision type - select the Decision element, open the Properties window, for the property 'variableType' select from the 'Value' field
- Decision Table Input/Output clauses - on the Decision Table Input/Output clause, right-click to display the context menu and choose the type
- Boxed Context variables - refer to the Boxed Context Help topic

DMN Code Generation In Java

After using validation to fix any variable type issues, we can proceed to the 'Generate Module' page in the DMN Simulation window.



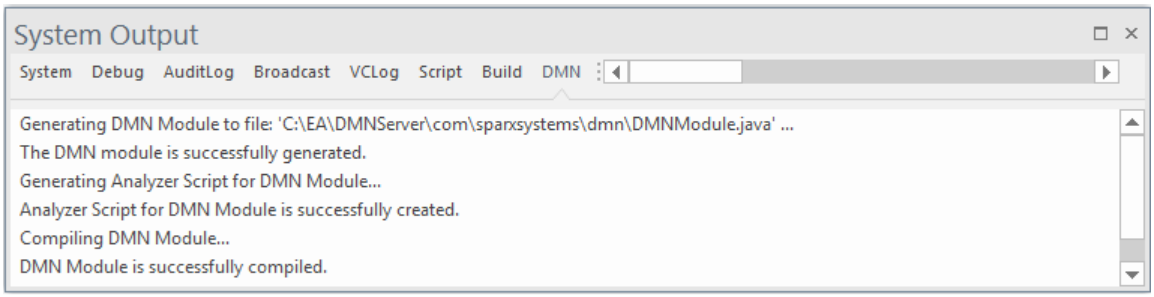
- Select *DecisionService1* in the top toolbar data entry field; all the elements involved in *DecisionService1* will now be included in the list
- Item Definition and Business Knowledge Model are global elements
- Input Data and Decisions are encapsulated in the DecisionService element
- The supported languages are C++, C#, Java and JavaScript; note that for JavaScript the generated .js file is the same as the simulation script ('Simulate' tab | Run button drop down menu | Generate New Script (Scripting Window)) except that the simulation-related codes are omitted
- For Java, the 'Module Path' value must match the Package structure; in this example, the DMNModule.java must be generated to a directory to form a file path that ends with

'\com\sparxsystems\dmn\DMNModule.java' - you have to manually create the directory structures for now

Click on the Generate Code button ( ) on the toolbar. This example will use Java; however, C++ and C# are the same. These actions are performed:

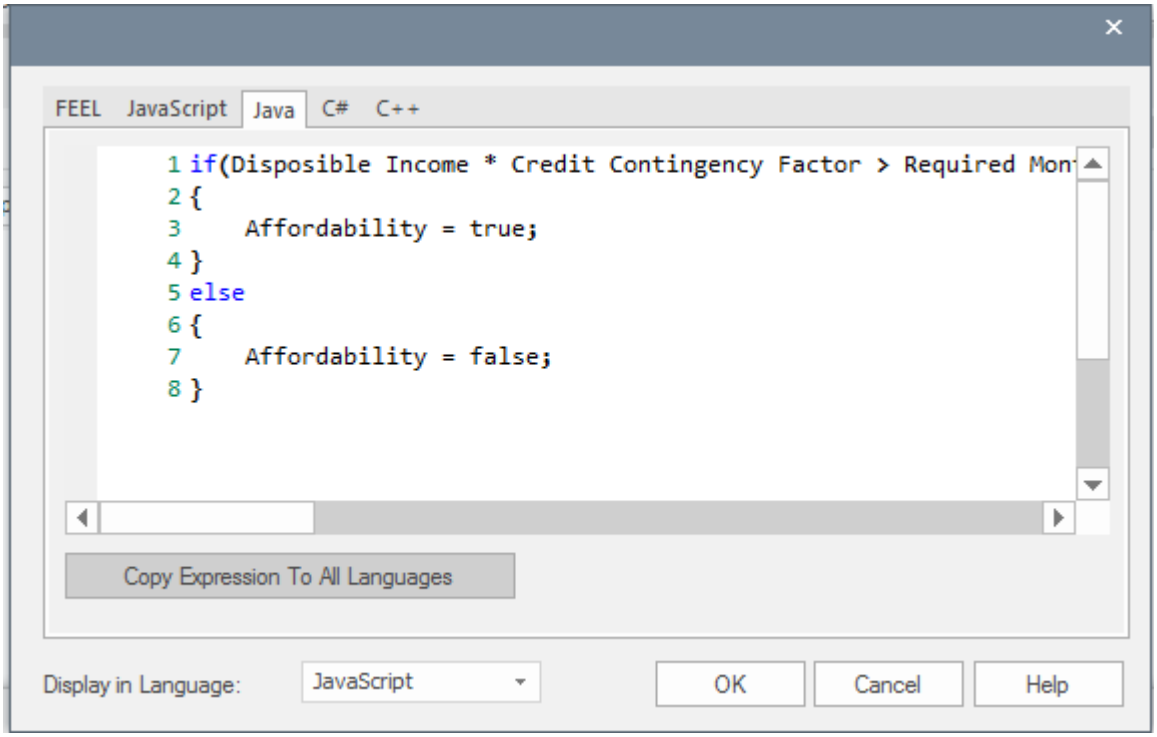
- The .java file is generated to the path specified
- An Analyzer Script (Build script) for this Artifact is created
- The Build Script for this Analyzer Script is executed
- Progress messages are reported in the System Output window

If the model is valid, this process will return the message:



If there are compiling errors, you can open the generated .java file by clicking the button next to the button on the toolbar, manually fix the issue, and compile with the generated script until you are successful.

One common reason for a compile failure is that languages can have different grammars for an expression. You might need to provide a value for a language to overwrite the default (right-click on a DMN Literal Expression | Edit Expression).

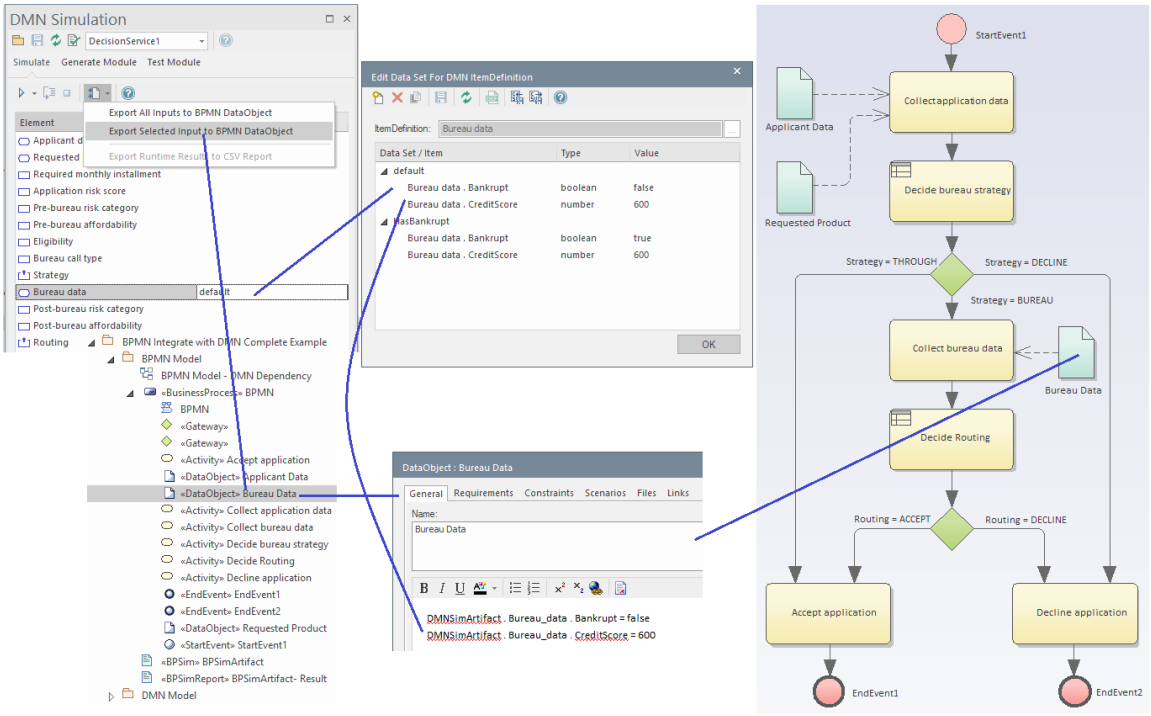


Testing DMN Modules before external Use

Having generated the model to java code and successfully compiled it we now want to:

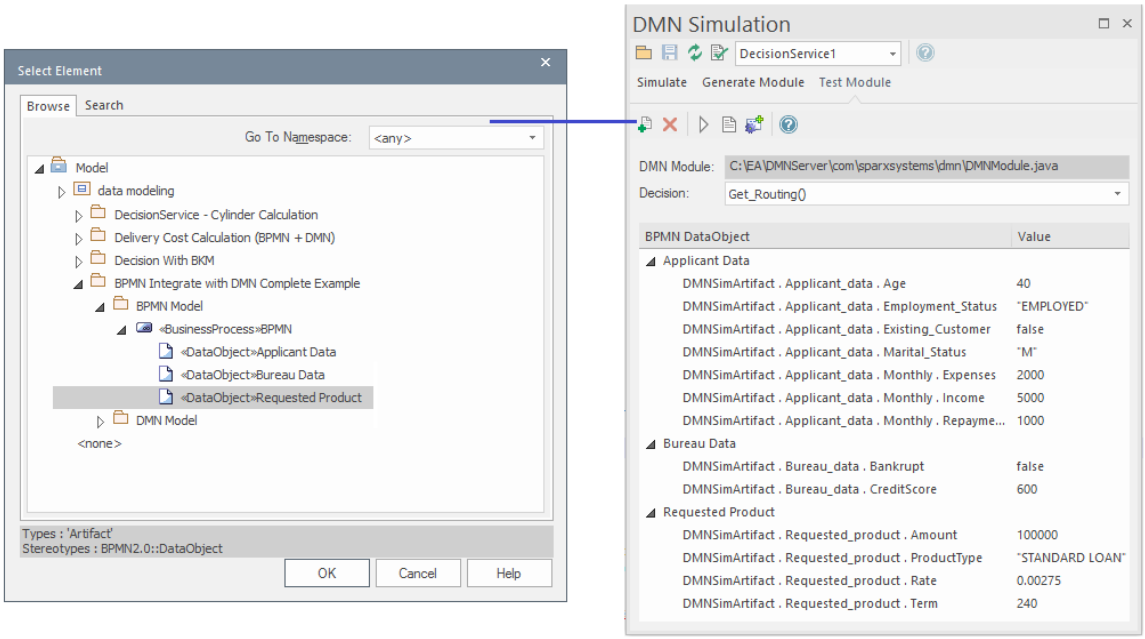
- Test this module's correctness
- Provide it with inputs
- Get the output Decision values

Generate BPMN DataObject



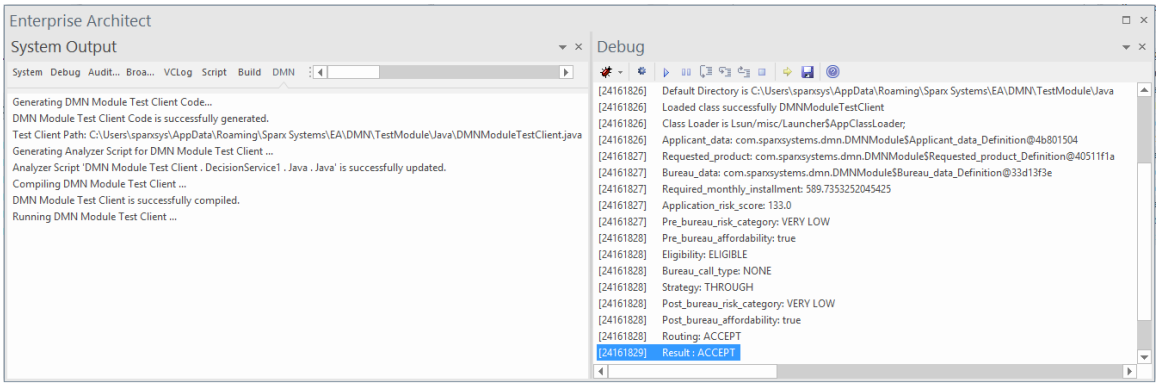
The data carried by the selected data set will be generated to the BPMN DataObject's 'Notes' field.

- Click the button (2nd to the right on the toolbar of the 'Generate Module' tab) to open the 'Test Module' tab



- Click the on the toolbar to select the input BPMN DataObject elements
- Select the available outputs from the 'Decision' combo

box, such as Get_Routing(), and click on the Run button on the toolbar



The execution result will be displayed in the Debug window. You can also open the test module file, set a breakpoint on the line and debug in the DMN Module to do line-level-debugging.

We highly recommend you test your DMN Module with this window to guarantee that the DMN Module is functional with the given inputs (from the BPMN DataObjects) and that it will successfully compute the result of the output.

Note: The DMN Module path is saved in the DMNSimConfiguration Artifact's 'Filepath' property.

Now, it is time to integrate the DMN module with the BPSim model.

The first step is to set up the usage dependency between the BPSim Artifact and the DMN Simulation Artifact.

Note: A BPSim Artifact can use multiple DMN modules if necessary. This is supported by simply putting all DMN Artifacts on this diagram and drawing a Dependency

connector from the BPSim Artifact to each DMN Simulation Artifact.

These Help topics provide two examples of using these methods. See:

- *Example: Integrate DMN Decision Service into BPSim Data Object and Property Parameter*
- *Example: Integrate DMN Business Knowledge Model into BPSim Property Parameter*

Example: Integrate DMN Decision Service into BPSim Data Object and Property Parameter

An example of integrating a DMN Decision service into the BPSim model is provided in the Model Wizard for BPSim.

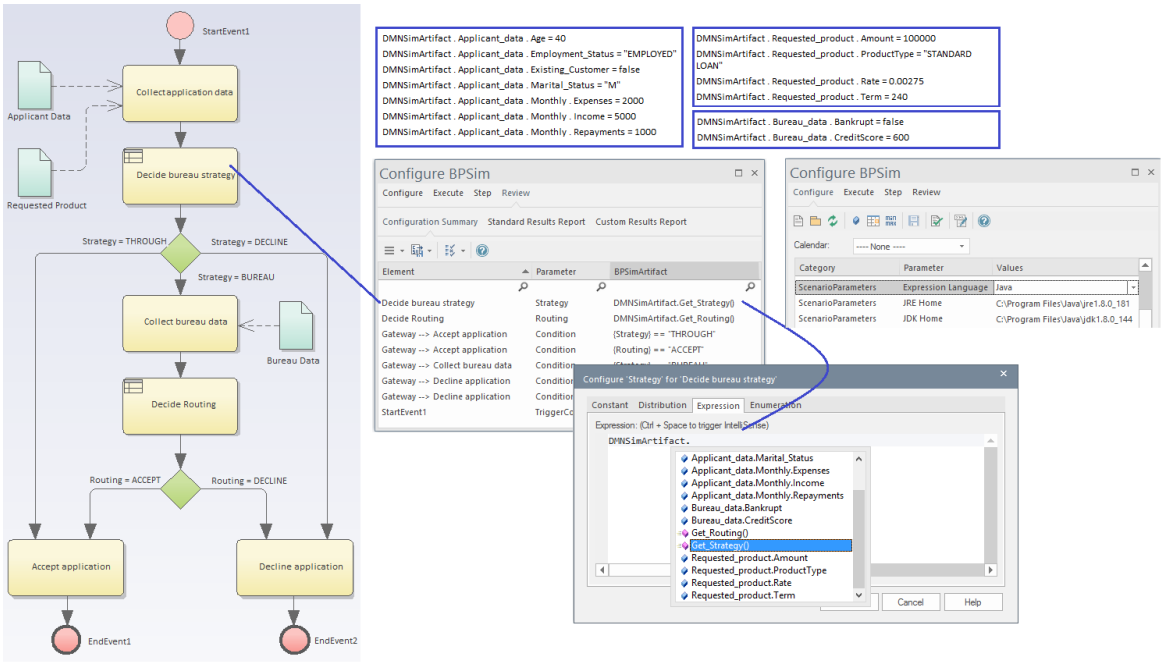
To access this:

- Set the *Perspective* to *Business Modeling > BPSim*
- Open the Model Wizard (Start Page 'Create from Pattern' tab)
- From the BPSim Case Studies group select *BPMN Integrate with DMN Complete Example*
- Click on the Create Model(s) button.

This will create BPMN and DMN models configured to simulate a call to a DMN model from the BPMN model.

Note: In order to integrate the DMN Module, the Expression Language must use Java and the JRE and JDK must be configured correctly (the minimum version of Java is 1.7). See *Install the BPSim Execution Engine* in the Help topic BPSim Business Simulations.

In this BPMN diagram there are three DataObjects (aqua) connected to BPMN Activities. These DataObject elements carry input data, generated from the DMN Simulation window.



- When the simulation is running it will automatically load all DataObjects connecting to the task when the simulation token passes through
- The second Business Rules task 'Decide bureau strategy' is configured to set the property 'Strategy' to the value 'DMNSimArtifact.Get_Strategy()'; you don't need to type this in - press Ctrl+Space to help you edit the expression

When these are set, click on the 'Execute' tab and simulate the model. You can then view the report or go to the 'Step' page to do step debugging of the BPSim model.

Example: Integrate DMN Business Knowledge Model into BPSim Property Parameter

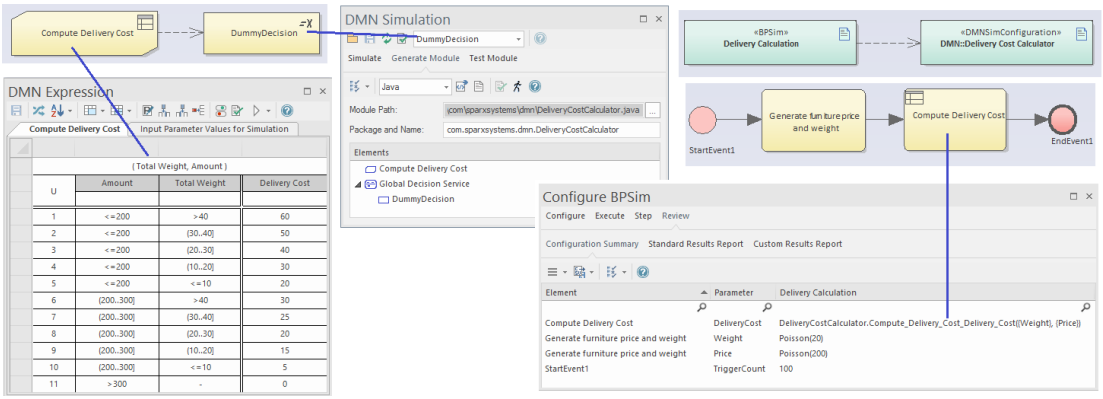
In some cases, you might want just to design a Decision Table to use in a BPMN model. If so, there is no need to go through the processes of creating a Decision Service, Decision, Input Data or even Item Definition, as a Business Knowledge Model (BKM) can be directly interfaced.

An example of integrating a DMN BKM into the BPSim model is provided in the Model Wizard for BPSim.

To access this:

- Set the *Perspective* to *Business Modeling > BPSim*
- Open the Model Wizard (Start Page 'Create from Pattern' tab)
- From the BPSim Case Studies group select *BPMN Integrate with DMN - Delivery Cost Calculation*
- Click on the Create Model(s) button

1. Create a simple Business Knowledge Model as a Decision Table (you can also create other expressions such as Boxed Context or Literal Expressions) with parameters, then model the logic (Input Clause, Output Clause, rules) and test it (the 'Input Parameter Values for Simulation' tab on the DMN Expression window).



2. Connect the BKM to a Decision with a Knowledge Requirement connector. This Decision serves as a group name for a number of BKM functions; you can simply input a number such as '10' to the expression. For example, if you want to generate Java code with only five BKMs (considering your model might have over one hundred BKMs), you can connect these five BKMs to a Decision and select this Decision in the DMN Simulation window, then all five BKMs will be included automatically.
3. Generate Java code and (assuming everything is correct) the compile will be successful.
4. In the BPSim configuration, we simply use Intelli-sense to construct the expression for task 'Compute Delivery cost'.

In this example, the 'Generate furniture price and weight' task will generate random values to the properties 'Weight' and 'Price', then the 'Compute Delivery cost' task will pass the value to the Business Knowledge Model and the result will be carried back to the property 'DeliveryCost'.

You can now execute the simulation, and step through the

debug process to observe, for example, the attribute value changes.

Integrate Into UML Class Element

After a Decision Model is created and simulated, you can generate a DMN Module in Java, JavaScript, C++ or C# and test it.

The DMN Module can be integrated with a UML Class element, so the code generated from that Class element can reuse the DMN Module and be well-structured. Since a Class element can define a StateMachine, after integration with the DMN Module the Executable StateMachine simulation will generically be able to use the power of the DMN Module.

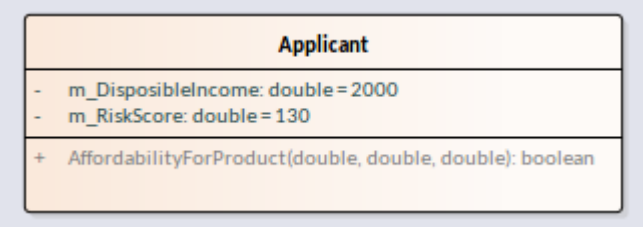
In this topic, we will explain the process of integrating a DMN Model with a UML Class element, considering the:

- Class element's requirement
- DMN Models
- DMN Binding to Class & Intelli-sense
- Code Generation on the Class element

Class Element's Requirement

Suppose we have a Class *Applicant* with an operation *AffordabilityForProduct* that evaluates whether the applicant can afford a loan product.

A simplified model resembles this:



The Class *Applicant* contains two attributes, which are actually calculated from more basic data such as the applicant's monthly income, expenses, existing repayments, age and employment status.

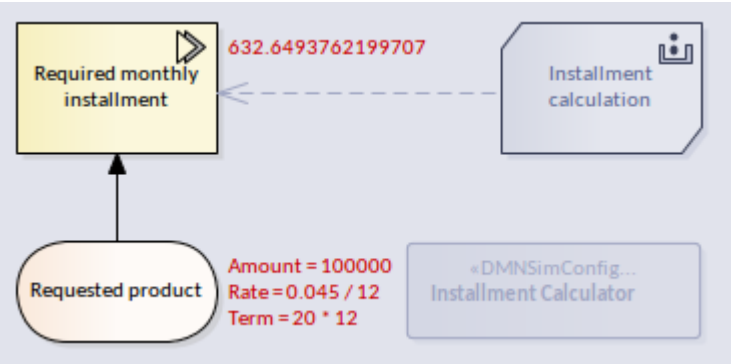
In this example, however, we simplify the model by skipping these steps and providing disposable income and risk score directly.

DMN Models

In this example, we have two disjoint DMN Models to show that a UML Class can integrate multiple DMN Models.

Installment Calculator

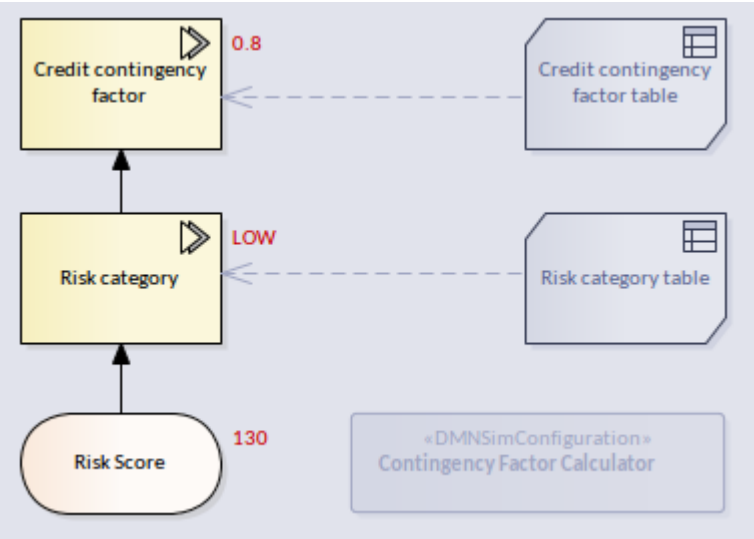
This DMN model computes the monthly repayment based on amount, rate and terms. It is composed of an InputData, a Decision and a Business Knowledge Model.



Credit Contingency Factor Calculator

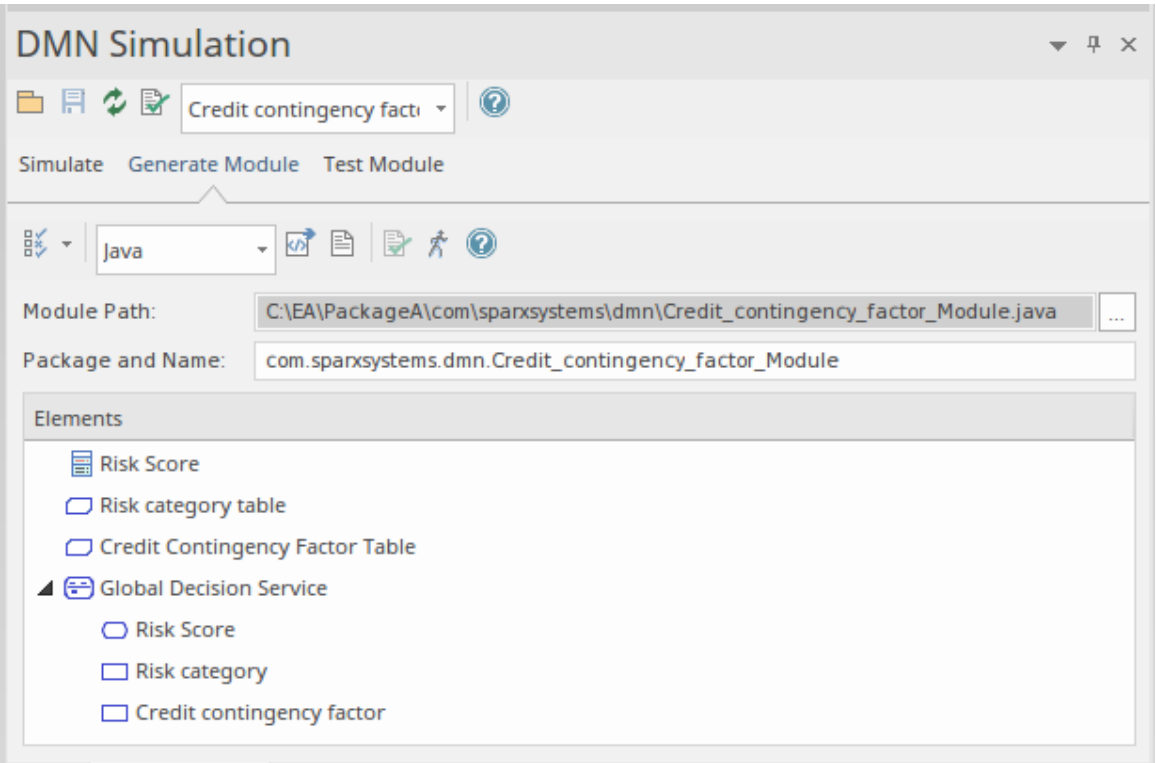
This DMN model computes the credit contingency factor

based on the applicant's risk score. It is composed of an InputData, two Decisions and two Business Knowledge Models.



Note: In this example, we focus on how to integrate DMN modules into a Class element; the DMN elements' detail is not described here.

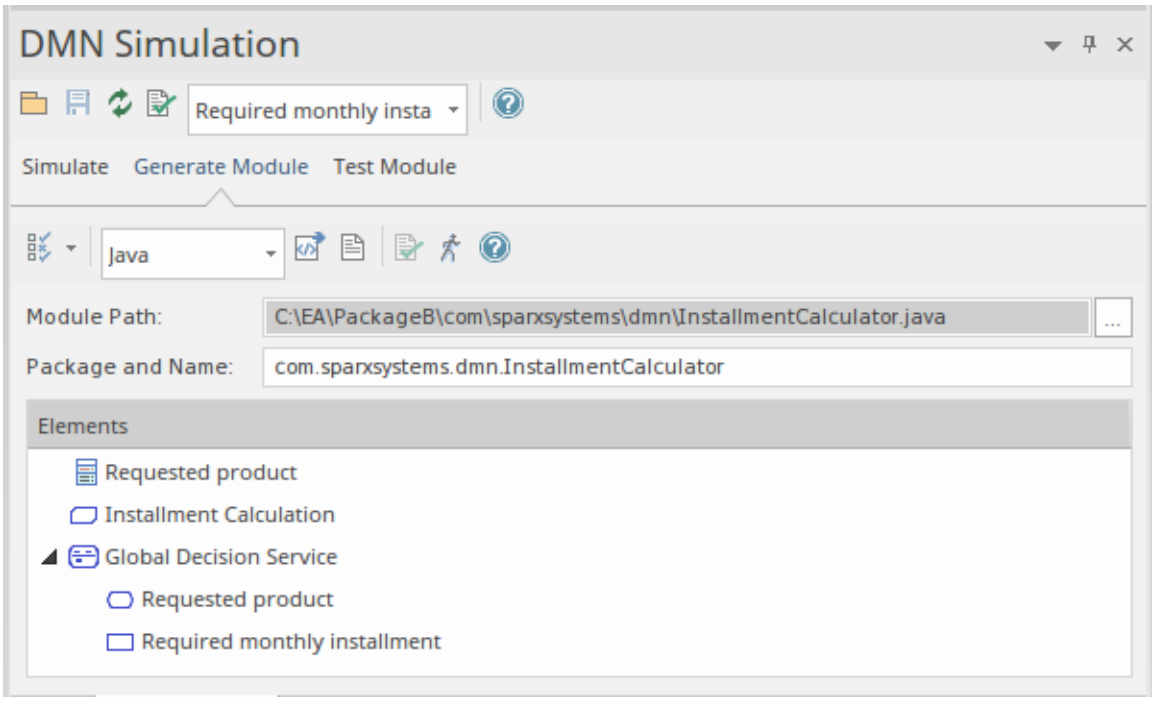
Generate code for both DMN Models



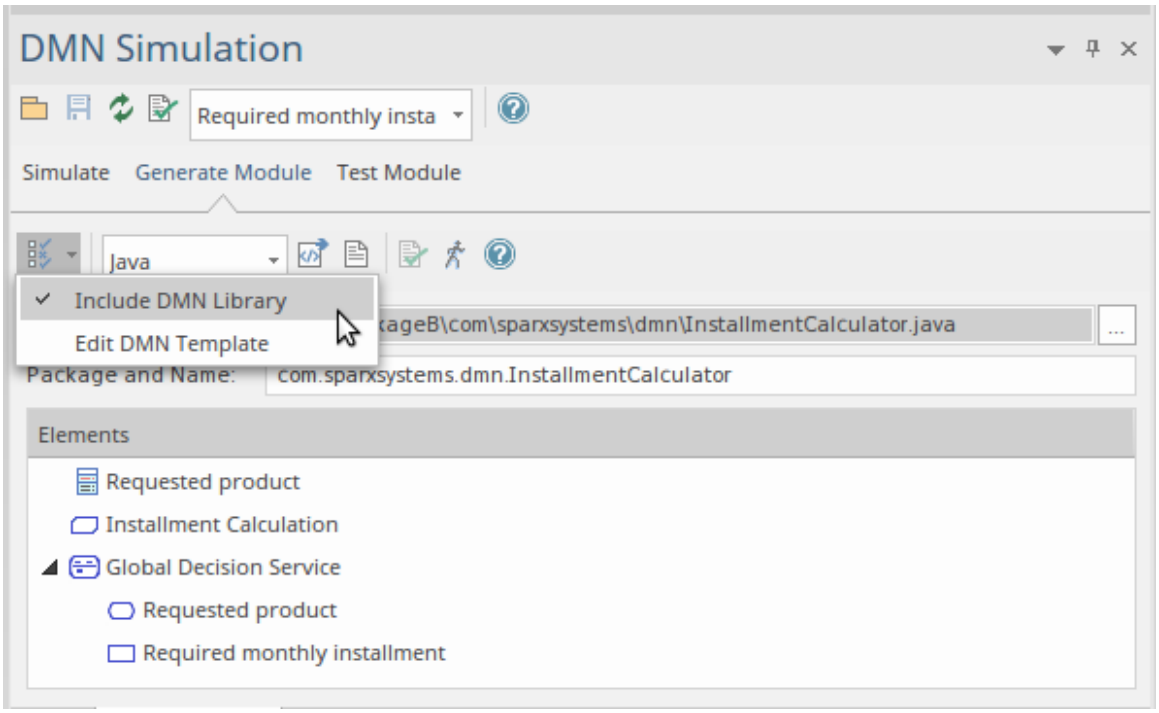
Click on the Generate Code icon, and check that you can see

this string in the System Output window, 'DMN' tab:

DMN Module is successfully compiled.



Note: Since this model uses a built-in function PMT, the DMN Library has to be included:

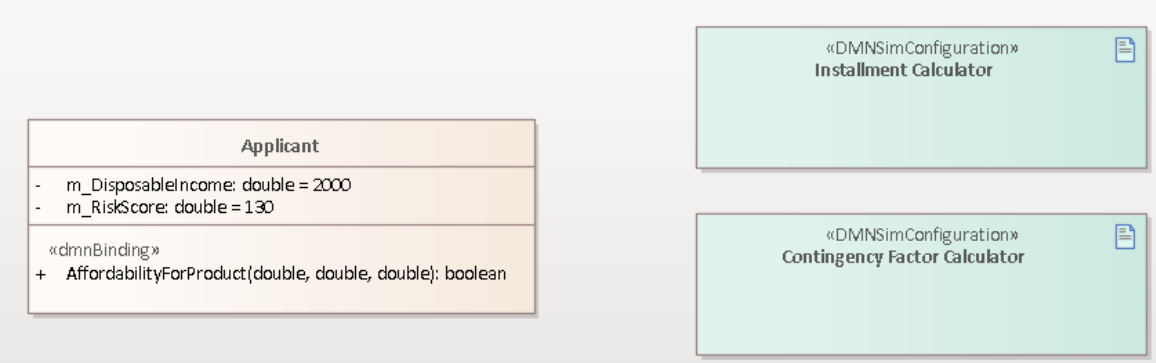


Click on the Generate Code icon, and check that you can see this string in the System Output window, 'DMN' page:

DMN Module is successfully compiled.

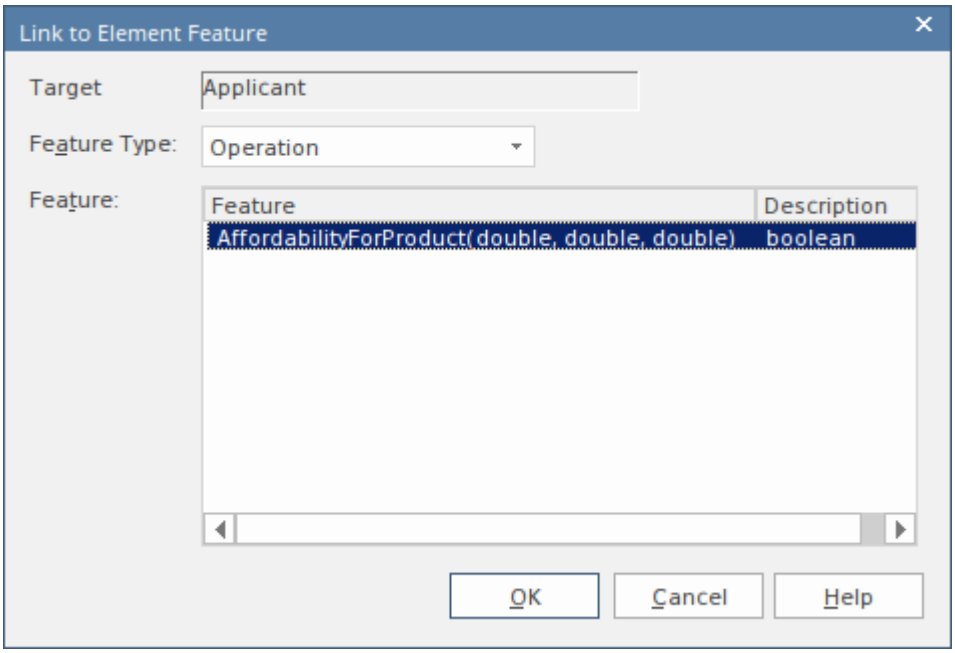
DMN Binding to Class & Intelli-sense

Put the two DMNSimConfiguration Artifacts on the Class diagram.



Use the Quick Linker to create a Dependency connector from the Class *Applicant* to each of the DMN Artifacts.

On creation of the connector, a dialog will prompt you to choose the operation to be bound to the DMN module.

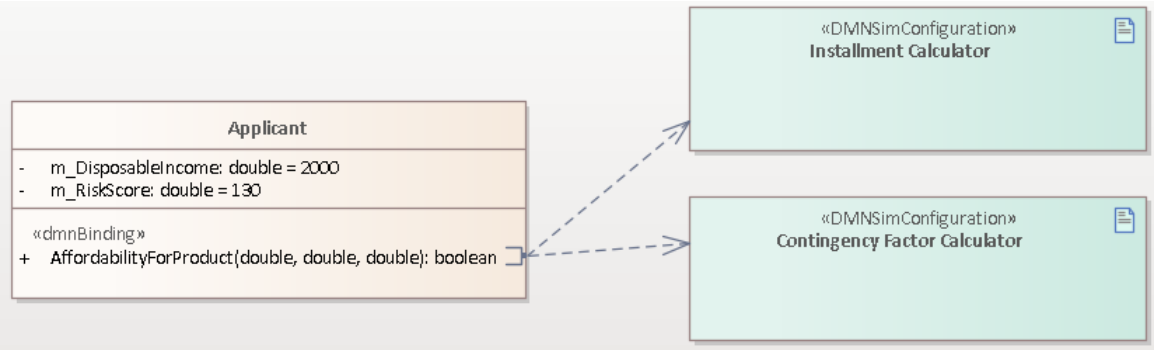


When the DMN module is bound to the operation:

- The operation takes a stereotype <<dmnBinding>>

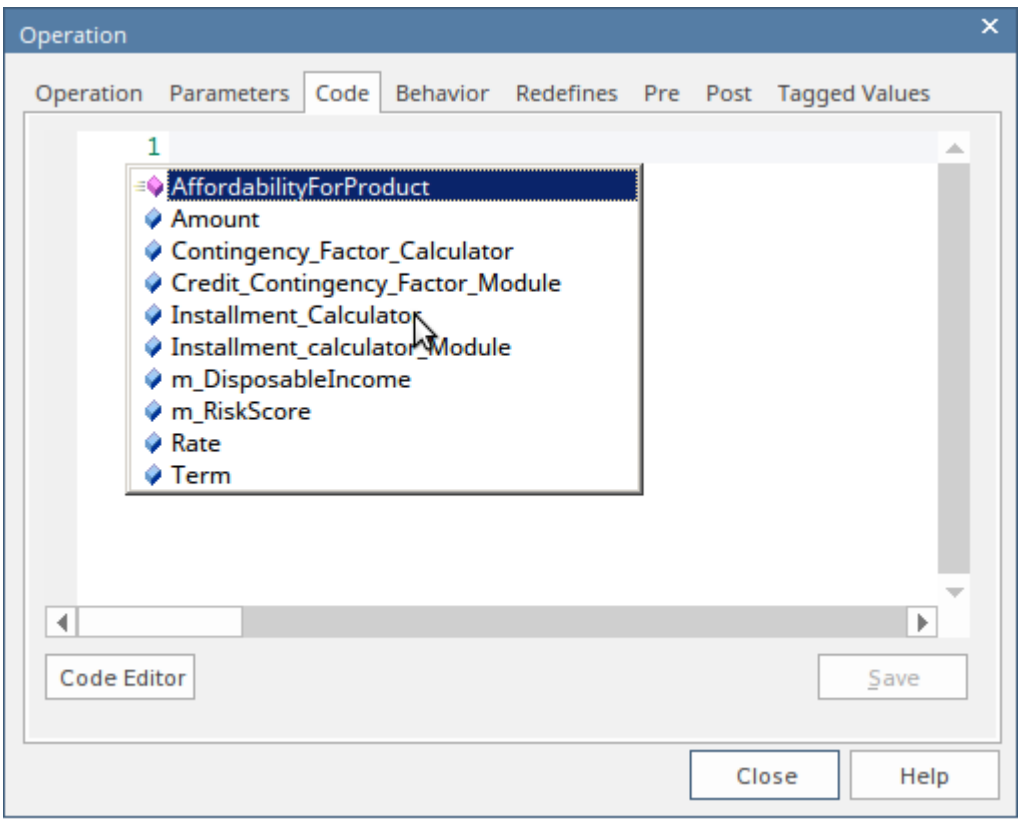
- The Dependency connector is linked to the operation

Multiple DMN Artifacts can be bound to the same operation.



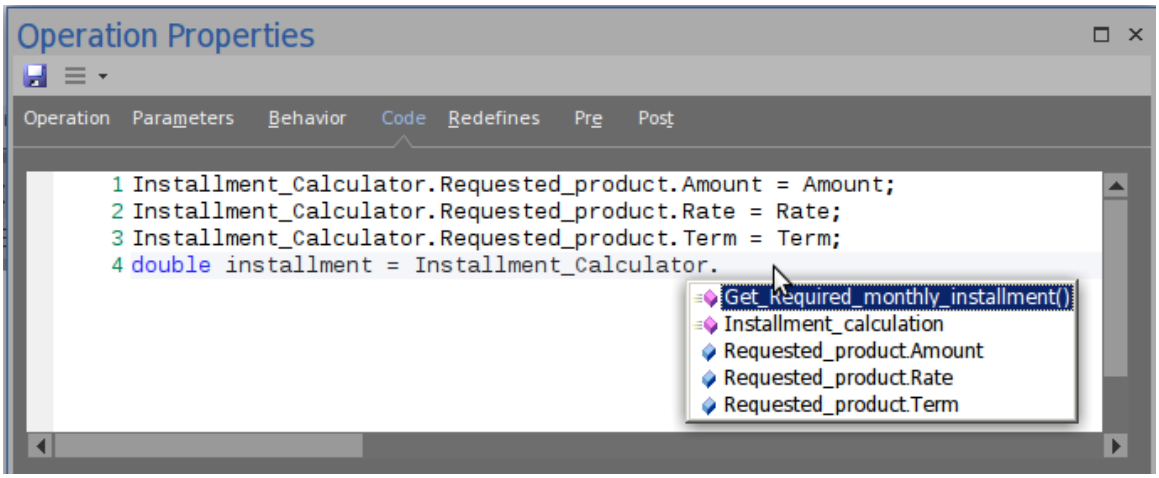
After DMN Bindings, Intelli-sense for the operation's code editor will support DMN Modules. To trigger the Intelli-sense, use these key combinations:

- Ctrl+Space - in most of the cases
- Ctrl+Shift+Space - when Ctrl+Space does not work after a parenthesis '('; for example, a function's arguments, or inside an 'If' condition's parentheses



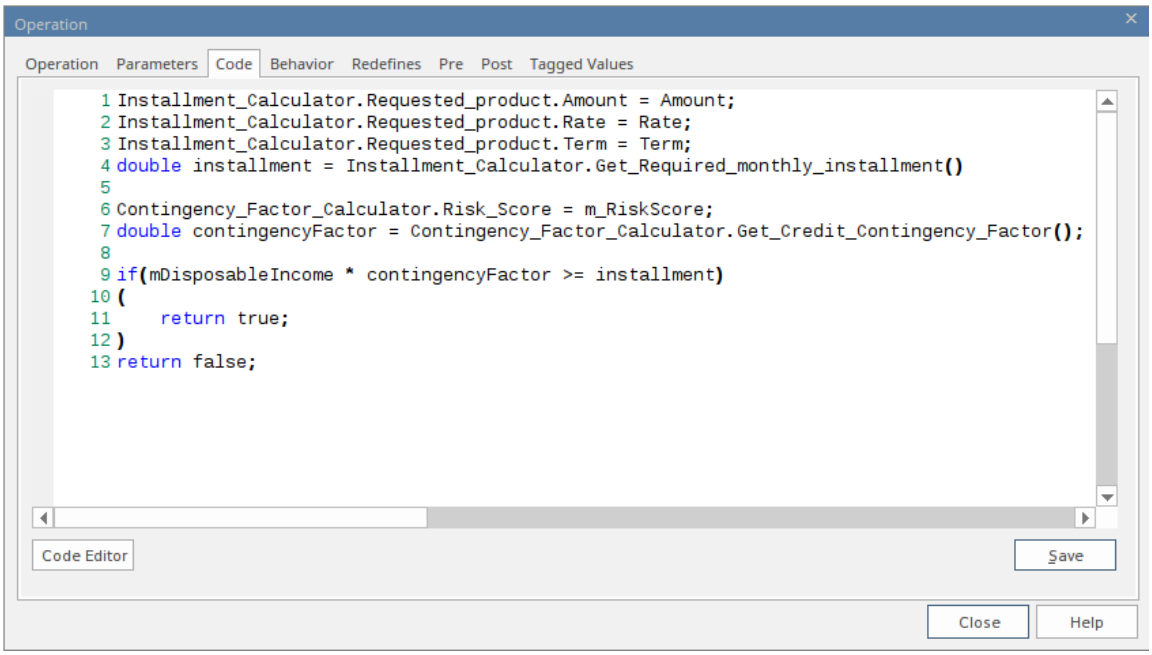
- Class attributes will be listed - m_RiskScore, m_DisposableIncome
- Operation parameters will be listed - Amount, Rate, Term
- Operations will be listed - AffordabilityForProduct
- All bound DMN Modules will be listed - Contingency_Factor_Calculator, Installment_Calculator

It is quite easy to compose the code with Intelli-sense support. On accessing the DMN Module, all the Input Datas, Decisions and Business Knowledge Models will be listed for selection.



This illustration shows that we are selecting 'Get_Required_monthly_installment()' from the Installment_Calculator.

This is the final implementation for the operation.



Code Generation for Class (With DMN Integration)

'Generate Code on Class Applicant' produces this code:

```
8 public class Applicant {
9
10    private double m_DisposibleIncome = 2000;
11    private double m_RiskScore = 130;
12
13    PackageA.ContingencyFactorCalculator Contingency_Factor_Calculator = new PackageA.ContingencyFactorCalculator();
14    PackageB.InstallmentCalculator Installment_Calculator = new PackageB.InstallmentCalculator();
15
16    public boolean AffordabilityForProduct(double Amount, double Rate, double Term){
17        //WARNING: Code in this function will be overwritten when generate from EA because this operation has a flush type of stereotype
18        Installment_Calculator.Requested_product.Amount = Amount;
19        Installment_Calculator.Requested_product.Rate = Rate;
20        Installment_Calculator.Requested_product.Term = Term;
21        double installment = Installment_Calculator.Get_Required_monthly_installment();
22
23        Contingency_Factor_Calculator.Risk_Score = m_RiskScore;
24        double contingencyFactor = Contingency_Factor_Calculator.Get_Credit_contingency_factor();
25
26        if(m_DisposibleIncome * contingencyFactor >= installment) {
27            return true;
28        }
29        return false;
30    }
31 }//end Applicant
```

- The DMN Module(s) are generated as attributes of the Class
- The dmnBinding operation's code is updated

Note: Regardless of whether the generation option is 'Overwrite' or' Synchronize', the operation's code will be updated if it has the stereotype 'dmnBinding'.

Importing DMN XML

Enterprise Architect supports the import of a DMN 1.1 or 1.2 XML file into a project, with both model semantics and diagram-interchange information.

Access

In the Browser window, select the Package into which to import the XML file. Then use one of the methods outlined here to open the 'Import Package from DMN 1.1 XML' dialog.

Ribbon	Publish > Model Exchange > Import > DMN 1.1
Keyboard Shortcuts	Ctrl+Alt+I : Other XML Formats > DMN 1.1

Import DMN 1.1 XML

Step, Step	Action, Action
1	In the 'Filename' field, type in the source file path and name, or click on the ... icon to locate and select the file.

2	Click on the Import button to import the file into the Package.

Import the example from OMG

1. Download the zip file at this link and extract it to your file manager.
2. Browse for the folder *examples/Chapter 11/*.
3. Click on the file *Chapter 11 Example.dmn* and import it as a DMN 1.1 format file.

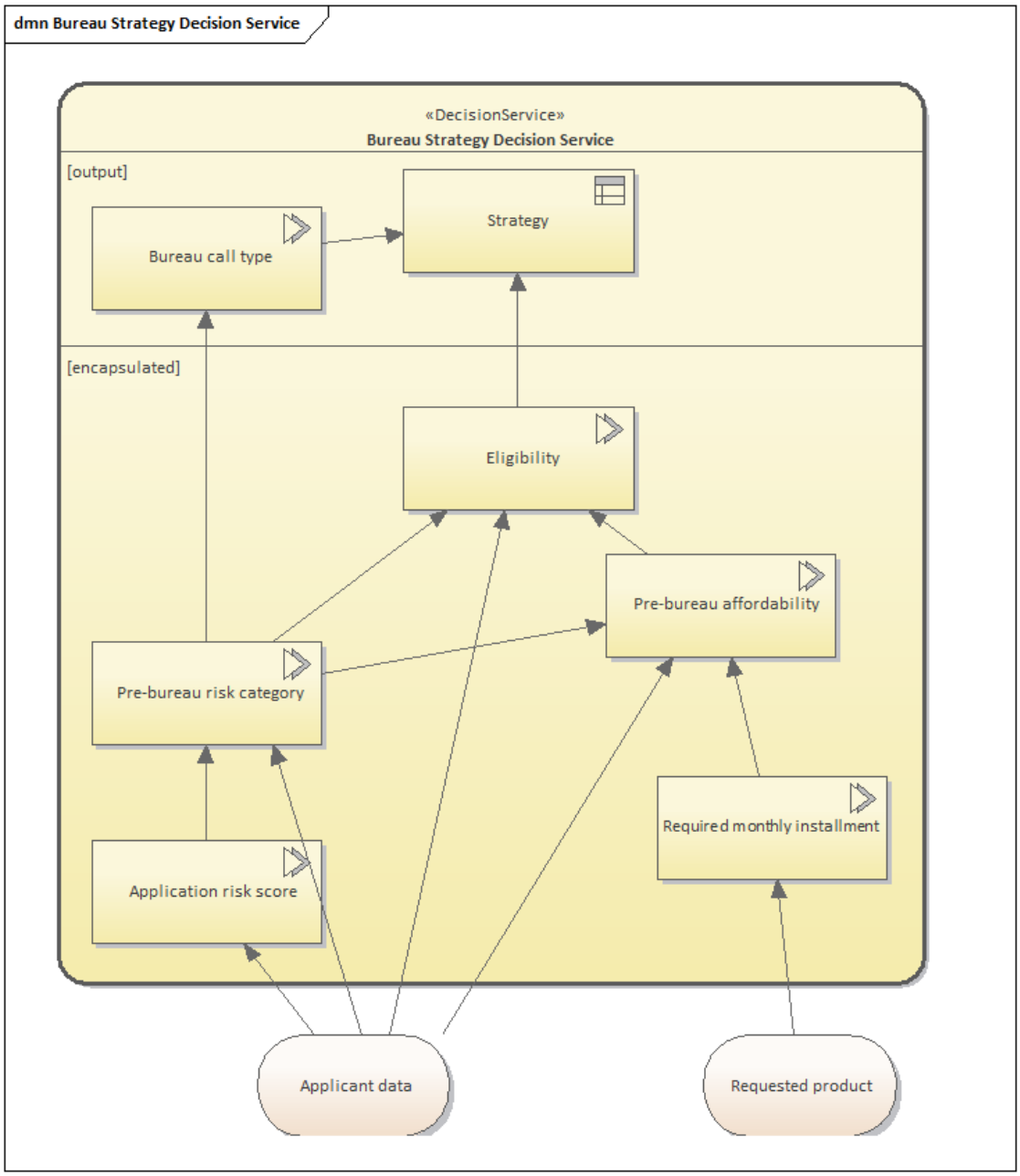
These diagrams are imported to show different perspectives of the model:

- DRD of all automated decision-making
- DRD for the Review Application decision point
- DRD for the Decide Routing decision point
- DRD for the Decide Bureau Strategy decision point

These diagrams are imported to define the Decision Services:

- Bureau Strategy Decision Service
- Routing Decision Service

The 'Bureau Strategy Decision Service' diagram is shown here. It has two Input Data elements (Applicant data, Requested product), two Output Decisions (Bureau call type, Strategy) and five Encapsulated Decisions. Note that the invoked Business Knowledge Models are not shown on the diagram.



In order to generate production code from the model, you might have to run a validation and simulation to ensure that the imported model has the correct expressions.

1. Create a DMN Sim Configuration Artifact on any of the listed diagrams, and double-click on it to open it in the DMN Simulation window.
2. The Decision Services and Decisions are listed in the target drop-down field. Once you specify a target, all the

required elements are listed in the window.

3. Click on the Validate button (4th on the toolbar). If any error or warning messages display, we suggest that you to fix the problems as directed by the error or warning descriptions, before performing the simulation.
4. Provide appropriate values for the inputs, and either run the simulation or step-debug the model.

Note: The 'Bureau Strategy Decision Service' example is also available in the EAExample Model. From the 'Getting Started' diagram, select 'Business Modelling > DMN Examples > Bureau Strategy Decision Service'.

More Information