TIBCO EBX® Rules Portfolio Add-on Documentation

Version 1.7.9
January 2023

TIBCO®

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

ANY SOFTWARE ITEM IDENTIFIED AS THIRD PARTY LIBRARY IS AVAILABLE UNDER SEPARATE SOFTWARE LICENSE TERMS AND IS NOT PART OF A TIBCO PRODUCT. AS SUCH, THESE SOFTWARE ITEMS ARE NOT COVERED BY THE TERMS OF YOUR AGREEMENT WITH TIBCO, INCLUDING ANY TERMS CONCERNING SUPPORT, MAINTENANCE, WARRANTIES, AND INDEMNITIES. DOWNLOAD AND USE OF THESE ITEMS IS SOLELY AT YOUR OWN DISCRETION AND SUBJECT TO THE LICENSE TERMS APPLICABLE TO THEM. BY PROCEEDING TO DOWNLOAD, INSTALL OR USE ANY OF THESE ITEMS, YOU ACKNOWLEDGE THE FOREGOING DISTINCTIONS BETWEEN THESE ITEMS AND TIBCO PRODUCTS.

TIBCO and TIBCO EBX are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. Please see the readme.txt file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (*https://www.tibco.com/patents*) for details.

Table of contents

User Guide

Scripting Language

Release Notes

User Guide

CHAPTER 1

Overview

You can declare and configure business and permission rules in the MDM repository with the TIBCO EBX® Rules Portfolio Add-on. Instead of having to hard-code rules, they are managed through meta data. This approach improves the quality and traceability of data use.

Special notation key	
✔	Important recommendation for use of the add-on.
✘	This feature is not yet available in the current release.

CHAPTER 2

Categories and types of rules

This chapter contains the following topics:

2.1 Managed rule categories

The add-on manages two rule categories:

- 'Business rules' are used to validate data and are executed using triggers or the data validation service.
- 'Permission rules' are used to manage user permissions and are implemented in compliance with the permission scheme applied to data and services.

2.2 Business rules

There are four types of 'Business rules':

- An 'Automated rule' can modify user data. This type of rule uses triggers set in tables to initiate execution and throws an error if it fails.
- A 'Validation rule' cannot modify user data. This type of rule uses triggers or constraints set on tables to initiate execution and throws an error (when executed via triggers or constraints) or a warning (only when executed via constraints) if it fails. Depending on the configuration of constraint classes, record modification may be blocked if it fails.
- A 'Manual validation rule' cannot modify user data. This type of rule is manually executed through the validation service. Depending on the success level, it validates, raises an error, or a warning.

- A 'Table set rule' can modify user data. This type of rule is manually executed using the 'Execute rules' service.

	Automated rule	Validation rule	Manual Validation rule	Table set rule
Can update the system	✔	✖	✖	✔
Run by the validation service	✖	✖	✔	✖
Run by triggers	✔	✔	✖	✖
Run by constraints	✖	✔	✖	✖
Run by 'Execute rules' service	✖	✖	✖	✔
Is warning	✖	✔	✔	✔
Is error	✔	✔	✔	✔

- If the automated rule fails then the record modification[(1)] is blocked
- When runing by triggers, if the validation rule fails then it blocks the record modification[(1)]
- When running by constraints, if the validation rule fails then it may block the record modification[(1)]
- The manual validation rule does not block the record modification[(1)]

[(1)] 'record modification' refers to create, update and delete operations

Automated rule

An automated rule can modify data. The output value is Boolean and depends on whether the rule executed successfully or failed. This type of rule executes when triggered by the creation, modification or deletion of a record.

If the automated rule raises an error:

- The trigger is interrupted.
- An error message displays.
- The transaction is aborted.

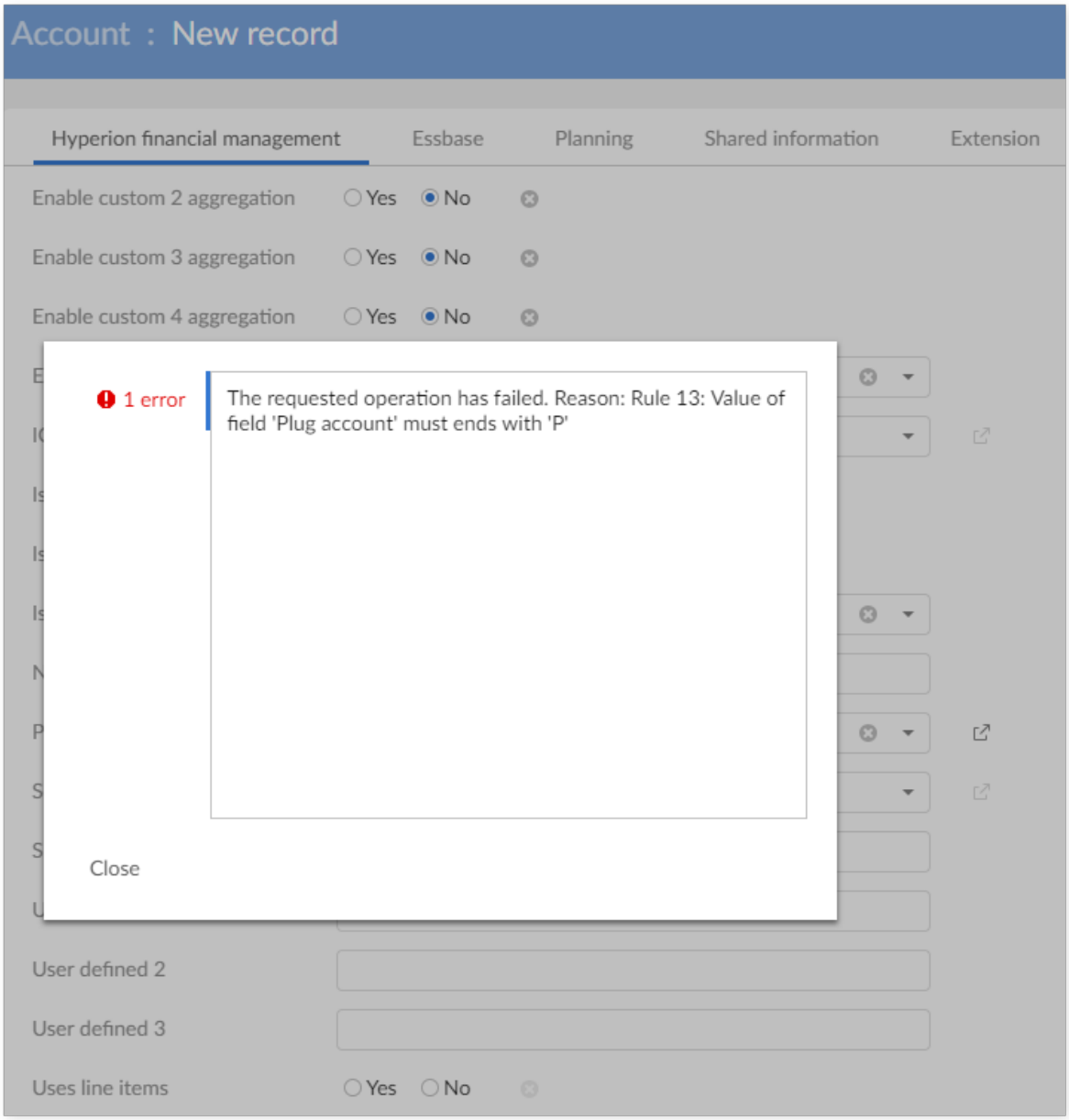
Validation rule

A validation rule is an assertion without any impact on the system and outputs a Boolean value depending on whether execution succeeded or failed. This type of rule can execute upon record creation, deletion or modification using a trigger or constraint on a field.

When the validation rule is executed by a trigger and raises an error:

- The trigger is interrupted.
- An error message displays.

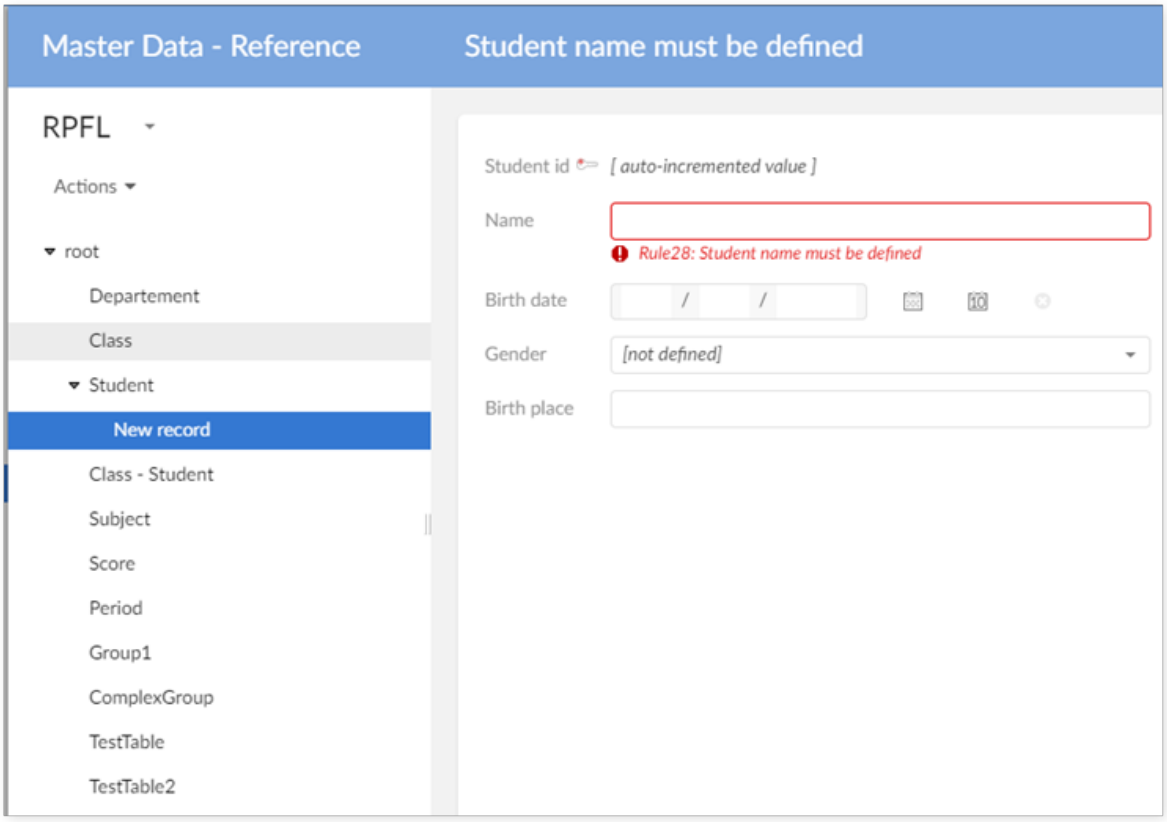
- The transaction is aborted.



When the validation rule executes via a constraint and raises an error, or warning:

- An error message displays.

- Depending on the constraint class configuration, record modification may be blocked.

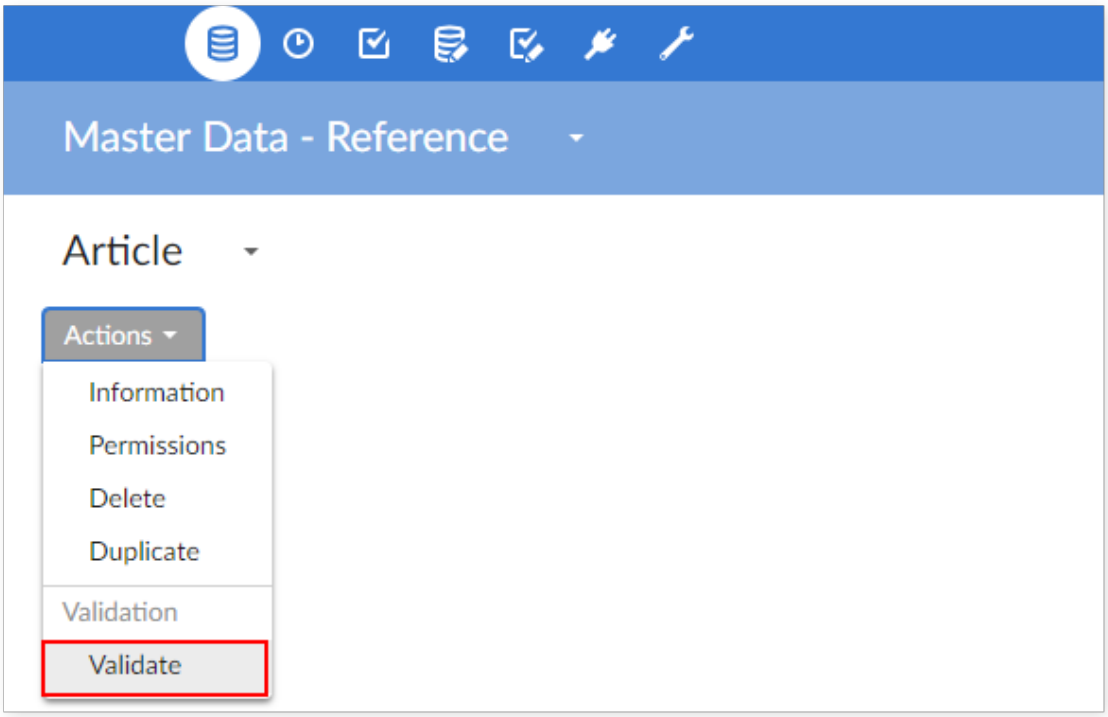


Manual validation rule

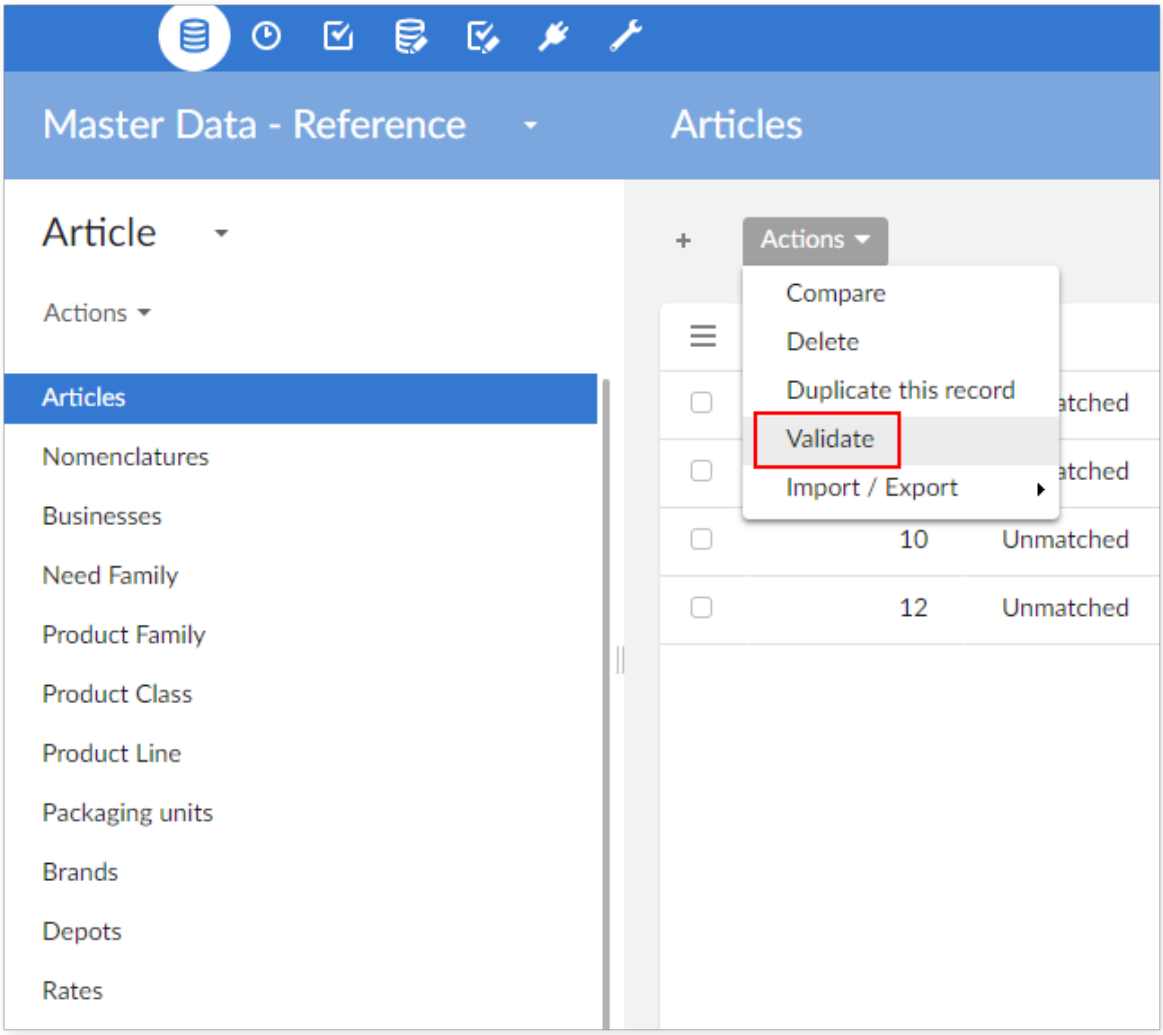
A manual validation rule is an assertion without any impact on the system and outputs a Boolean value depending on whether execution succeeded or failed. This type of rule can be executed at the data set or table level.

Depending on the set level of execution, the standard validation service executes this type of rule in the following ways:

- Execution at the data set level: the add-on executes all manual validation rules configured on each table included in the data set.



- Execution at the table level: the add-on performs all manual validation rules applied to the selected table.



Any error and warning messages that are generated display in the standard validation report.

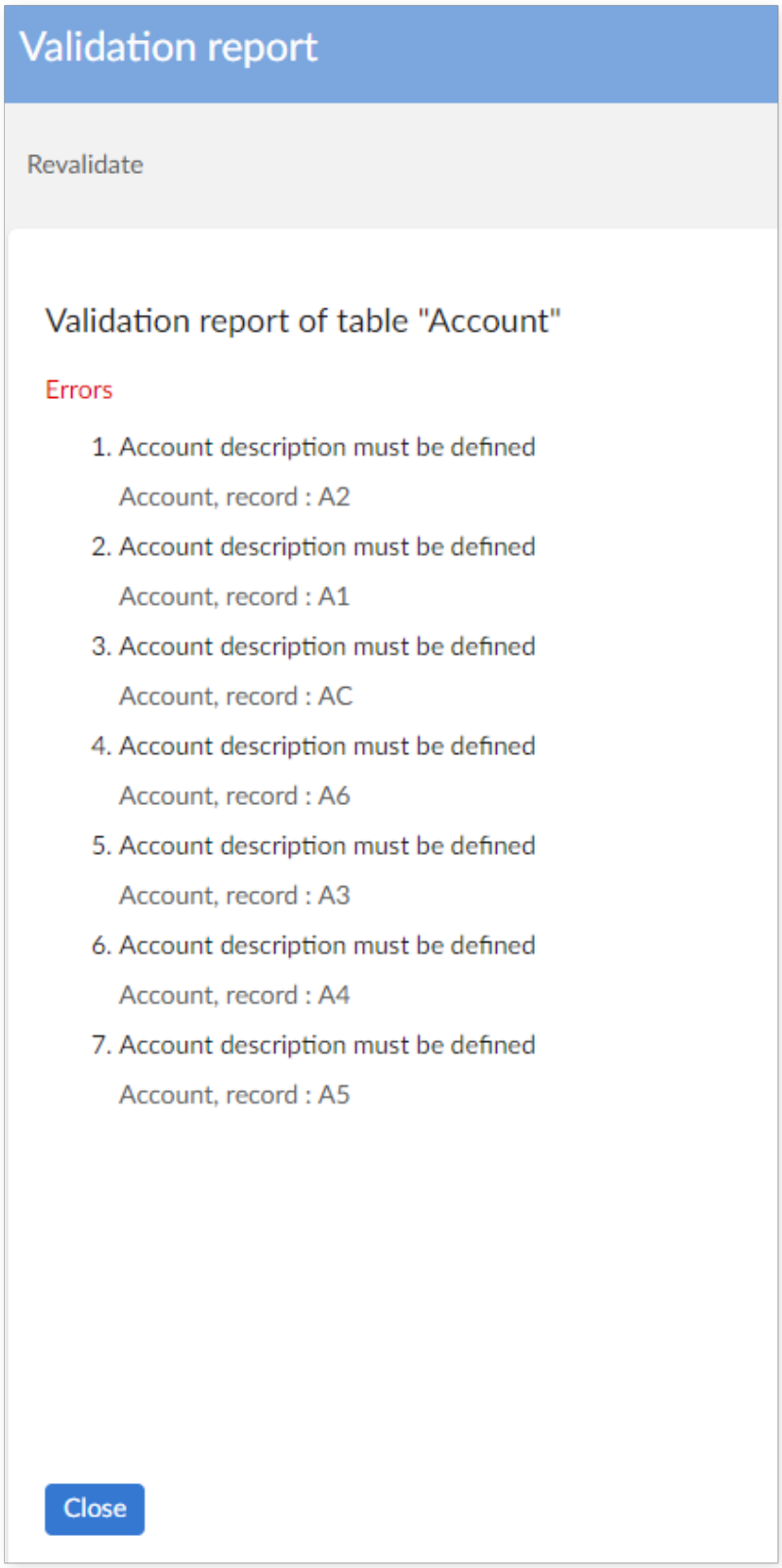
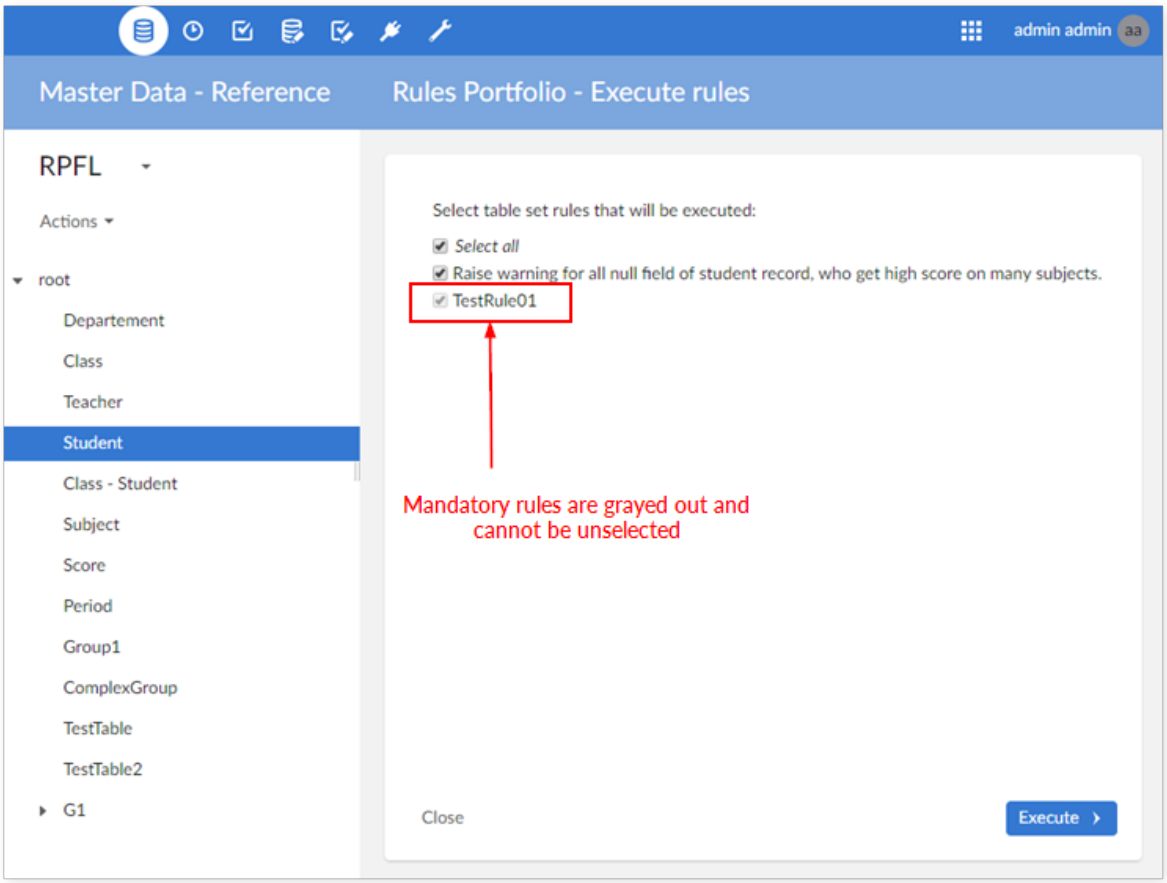


Table set rule

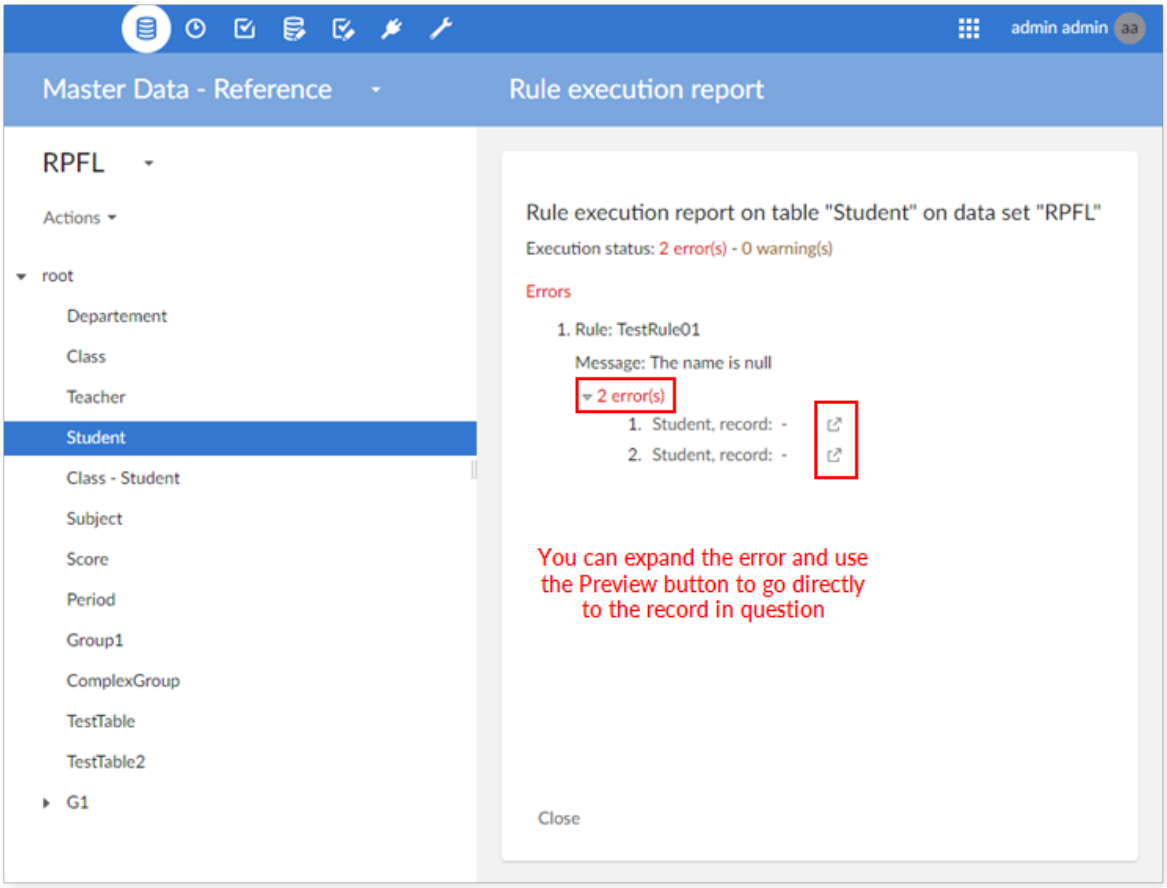
A table set rule can create, modify or delete data. Additionally, it can be an assertion without any impact on the data value. These rules return a Boolean value that indicates success or failure of rule logic execution.

You must execute table set rules manually using the 'Execute rules' service located in the 'Actions' menu of a data set, or table. Running the service at the data set level executes rules applied to all tables in the data set. When run at the table level, execution includes only rules applied to the current table. Note that if a rule is not designated as mandatory, you will have the option to include, or exclude it from this execution. The steps below briefly outline this process:

- Run the 'Execute rules' service at the data set level.
- Choose which rules you want to execute. Notice in the image below that the Name cannot be null rule's checkbox is grayed out. This indicates execution of the rule is mandatory. This behavior can be set in each table set rule's configuration options using the 'Mandatory' property.



- After clicking 'Execute' a report shows the execution result including any errors or warnings.



2.3 **Permission rule**

There are two types of permission rules:

- An 'Access permission rule' manages user permissions on data sets, tables or fields.
- An 'Action permission rule' determines whether or not a user has permission to use the specified service.

These rules are called by the SchemaExtension class and the ServicePermission class. They are unknown at the trigger level.

	Access permission rule	Action permission rule
Service	✗	✓
Data set	✓	✗
Table	✓	✗
Field	✓	✗
Record	✓	✗

Access permission rule

An 'Access permission rule' manages user permissions on a node, namely a data set, table or field. This type of rule determines whether or not the node is Hidden, Read-Only or Read-Write for a given user.

Action permission rule

An 'Action permission rule' manages service permissions. This type of rule controls whether or not the given service can be used in the current context.

CHAPTER 3

Key concepts

This chapter contains the following topics:

3.1 Concepts overview

The add-on manages rules as real assets. This enables a better understanding of which rules are executed on which tables. Additionally, time and effort is saved by eliminating the need for the IT department to hand-code rules. At the same time the ability to apply business audit operations is enforced. Overall, MDM agility and transparency are greatly increased.

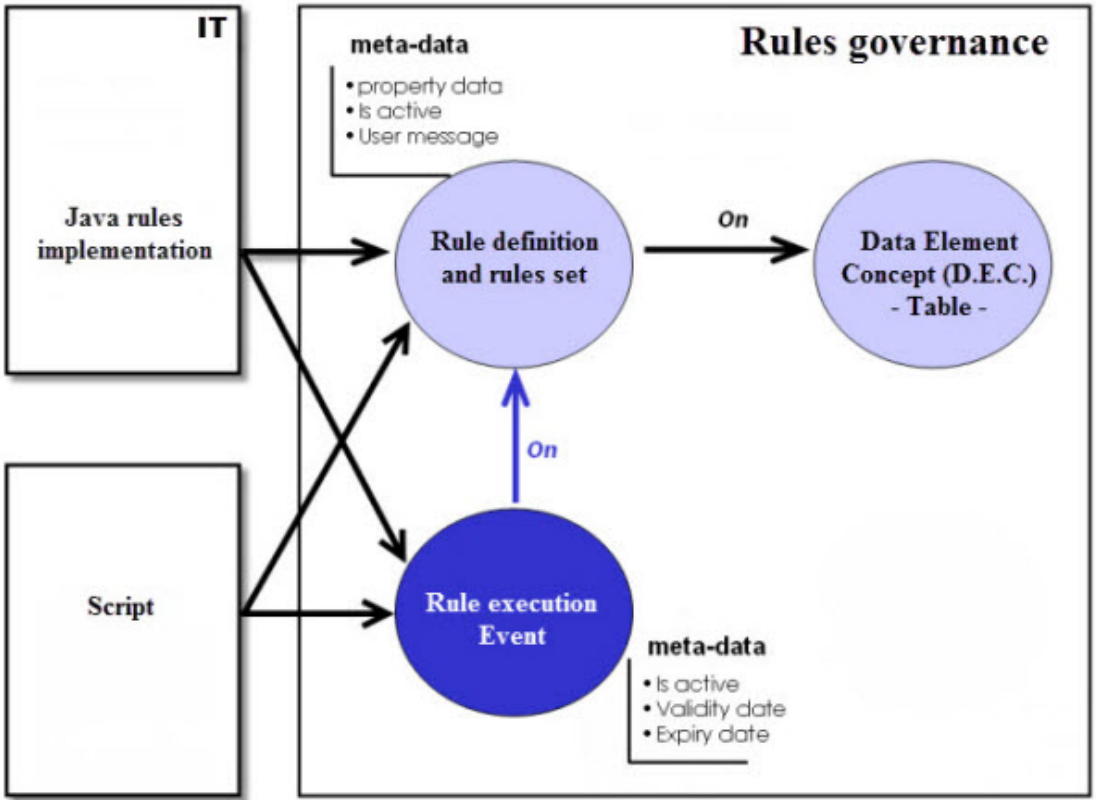
The add-on manages both business rules and permission rules. The portfolio of rules automatically displays through a Java introspection of the rules implementation layer (refer to the Java doc to apply the standard Rule Interface when deploying a new rule). After selecting a rule in the portfolio, you can configure and adapt its behavior to meet your specific needs. To achieve optimal configuration, a metadata set are available such as: period of activation (from date to date), user messages, input parameters, contexts of execution, etc.

You can configure a rule to execute on a type of business object such as a table. This table is identified with the generic term 'Data Element Concept' (D.E.C.). A D.E.C. type is not limited to a table; it can be any asset, such as a table, a field, a workflow, a data space, a data set or a service. This is an ISO-IEC 11179 concept already used in the TIBCO EBX® Information Governance Add-on and TIBCO EBX® Insight Add-on for the management of data quality indicators.

Special notation	
✔	The scripting language included in the add-on should be used in two situations due to the response time: • Inline validation rules and not mass validation treatments • During the prototyping of your data model with business users to make clear the specification, then translation into Java language when needed. This Java implementation can be done within the add-on to benefit from the context management such as the on-off rules, staging, loose coupling, etc.

3.2 **Global architecture**

From an implemented rule (Java implementation) one to many *Rule definition(s)* are declared in the add-on. Alternatively, you can use a script to create a rule definition.



Every rule definition configures a metadata set, including the parameters published by the rule implementation, the user message in case of error, etc.

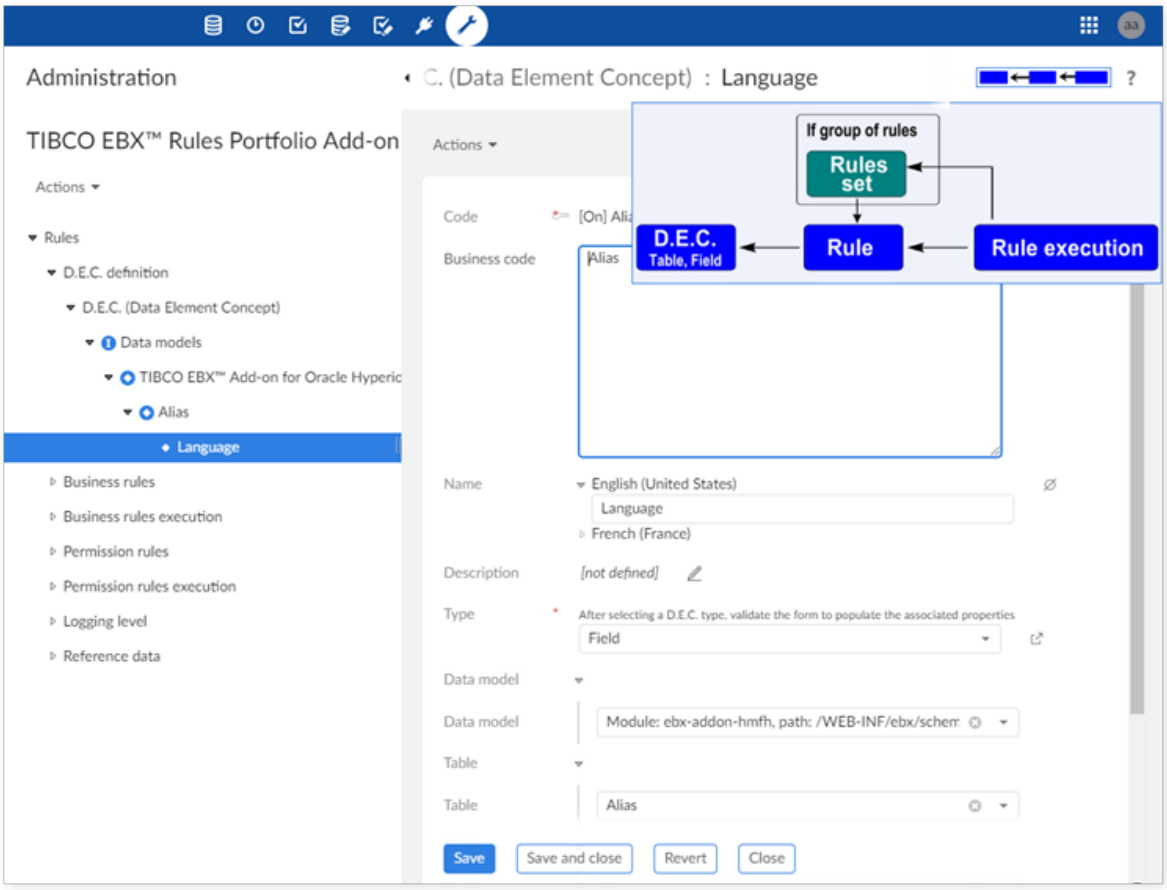
The rule definition is attached to a *Table* on which this rule must be executed. A permission rule type can be applied to other D.E.C.(s). such as a field, a service, a data space or a data set.

A *Rule execution* can be configured and declares the conditions that must be met before a rule can be executed. For example, conditions that must be met can include: an event such as "on demand" (user activation), the creation of a record, a validity date, etc.

In earlier versions of the add-on, you could configure a *Simple expression* when complex business conditions beyond an event or record creation are required for rule execution. However, from version 1.3.0, you use a Java implementation or a script to define business conditions for rule execution instead of using a simple expression.

3.3 **Online quick view**

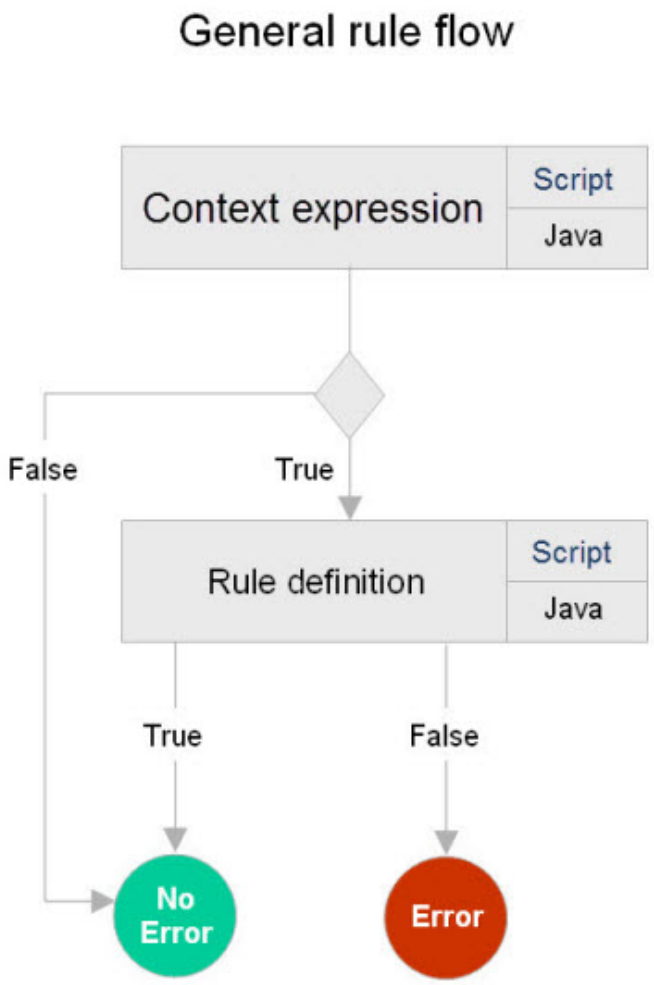
During the process of creating rules, it can be helpful to view reminders of key concepts used in their configuration. To show these reminders, it includes the 'quick view' icon on the top-right of the tabular view. The icon 'quick view' is located at the top-right of the tabular view as illustrated below.



3.4 **How context expressions and rule definitions interact**

The following figure outlines how a context expression and rule definition work. The rule executes if the context expression returns 'True'. You can configure the context expression and the rule using

a Java implementation or by writing a script. For the rule definition, you can use a predefined rule provided with the add-on, or implement your own bespoke rule using Java or a script.



In the example below, a context expression declares that a 'Name' must start with an uppercase letter. This expression can be implemented using Java or a script.

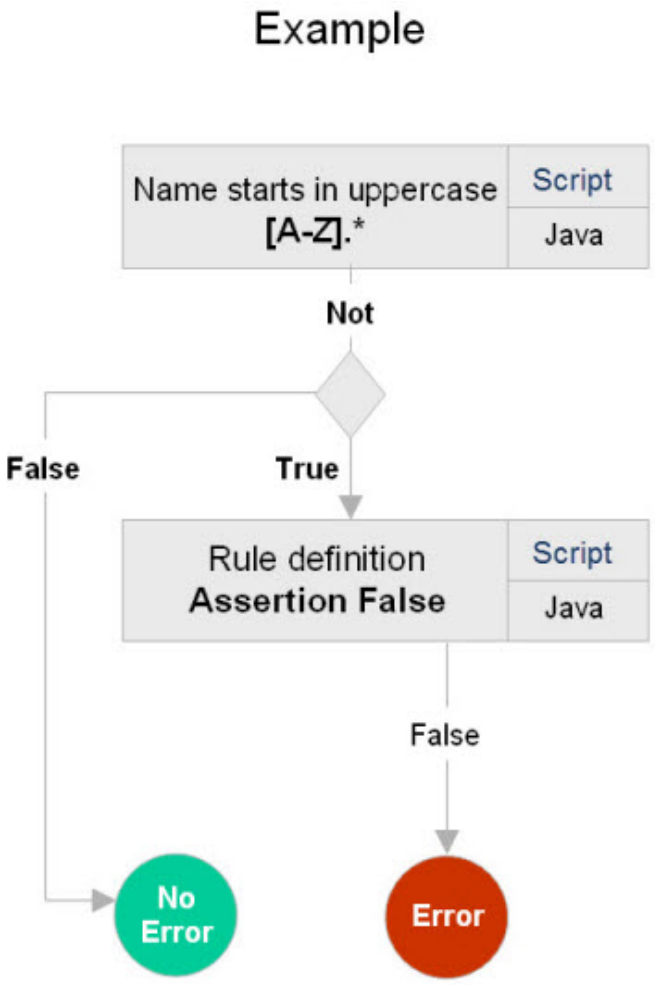
If a name starts with an uppercase:

- According to the assertion this would be *not* 'False'.
- The 'Rule definition' is bypassed.
- No error is raised.

If a name does not start with an uppercase:

- According to the assertion this would be a *not* 'True' scenario.
- The rule executes.

- A value of 'False' is returned.



CHAPTER 4

Data model configuration

Before using the add-on you have to configure the data model by adding classes to it that call the add-on. The default classes you have to add depend on how you plan on configuring rules. The Use Case section in this document provides some example configurations. From version 1.4.0. of the add-on you can add classes automatically through schema compilation, or manually. This section provides the steps to configure your data model.

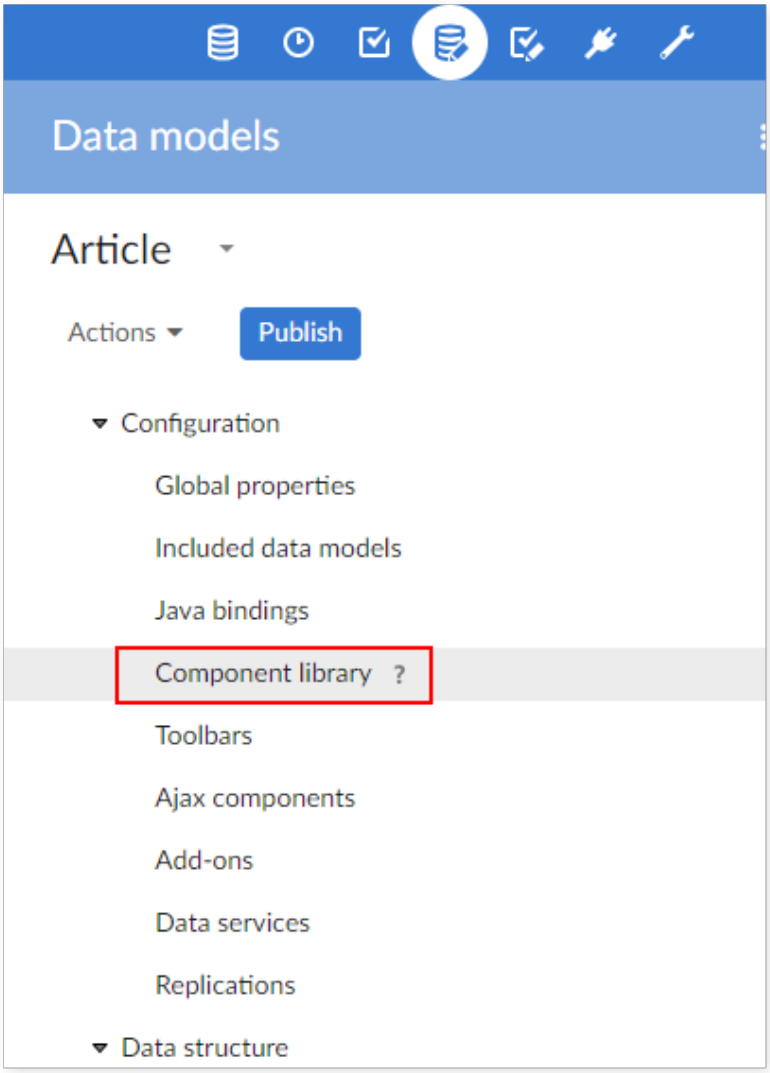
This chapter contains the following topics:

4.1 Adding default classes manually

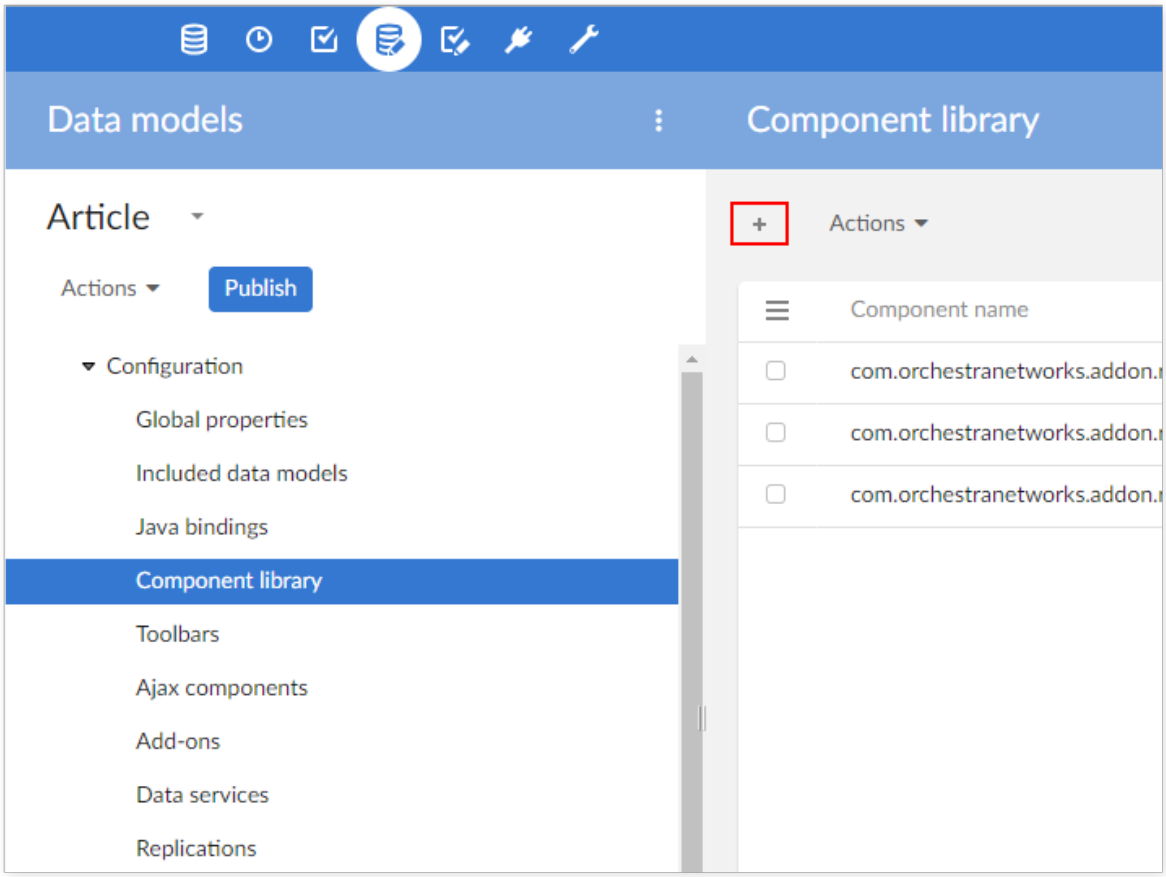
When you add classes manually, you can make use of all rule types. Keep in mind that you have to repeat this process each time you update an existing rule, or create a new rule that does not rely on a default class already part of the data model's component library.

To manually add the classes:

- Open your data model and navigate to 'Configuration' → 'Component library'.

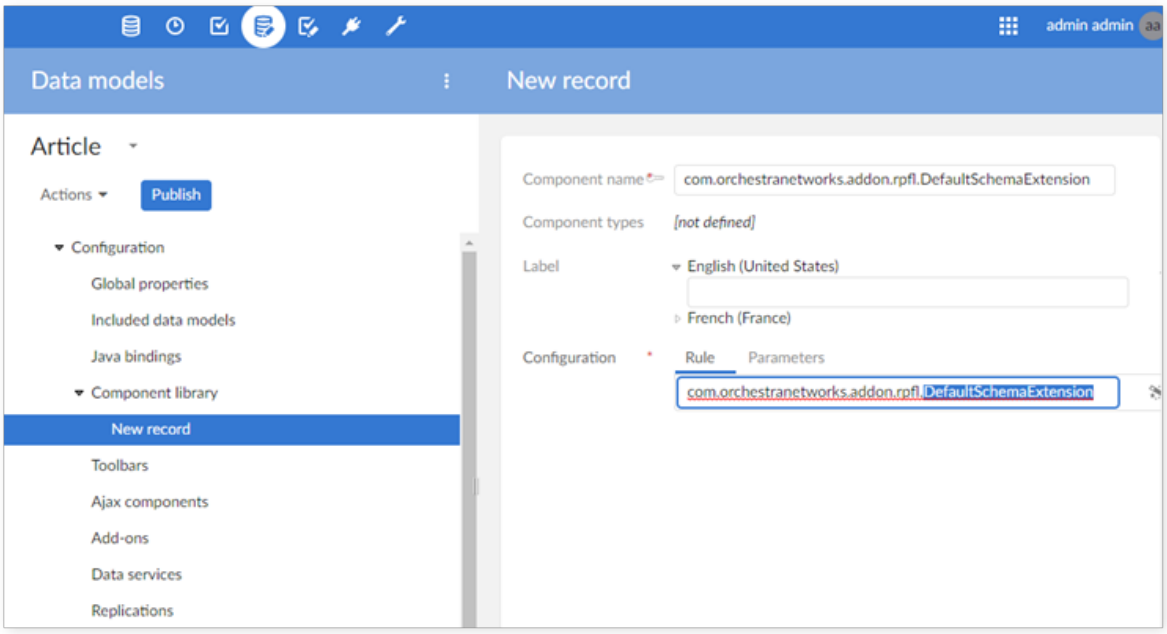


- Add a new component by clicking on the '+' icon.



- After providing a name and optional label, enter one of the following classes in the 'Configuration' field.
  - com.orchestranetworks.addon.rpfl.DefaultSchemaExtension → allows you to use the access permission rules.
  - com.orchestranetworks.addon.rpfl.DefaultTableTrigger → allows you to use the validation rules and automated rules.
  - com.orchestranetworks.addon.rpfl.DefaultConstraintOnTable → allows you to use the manual validation rules.
  - com.orchestranetworks.addon.rpfl.DefaultConstraint → allows you to apply the validation rules on fields.
  - com.orchestranetworks.addon.rpfl.DefaultConstraintOnNull → allows you to apply the validation rules on fields.

- com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission → allows you to use the action permission rules. This default class is applied on the service that you want to set permission to.



- After saving and closing, the class you added displays in the 'Component library' table. You can repeat this process as needed to add the necessary classes to your data model.

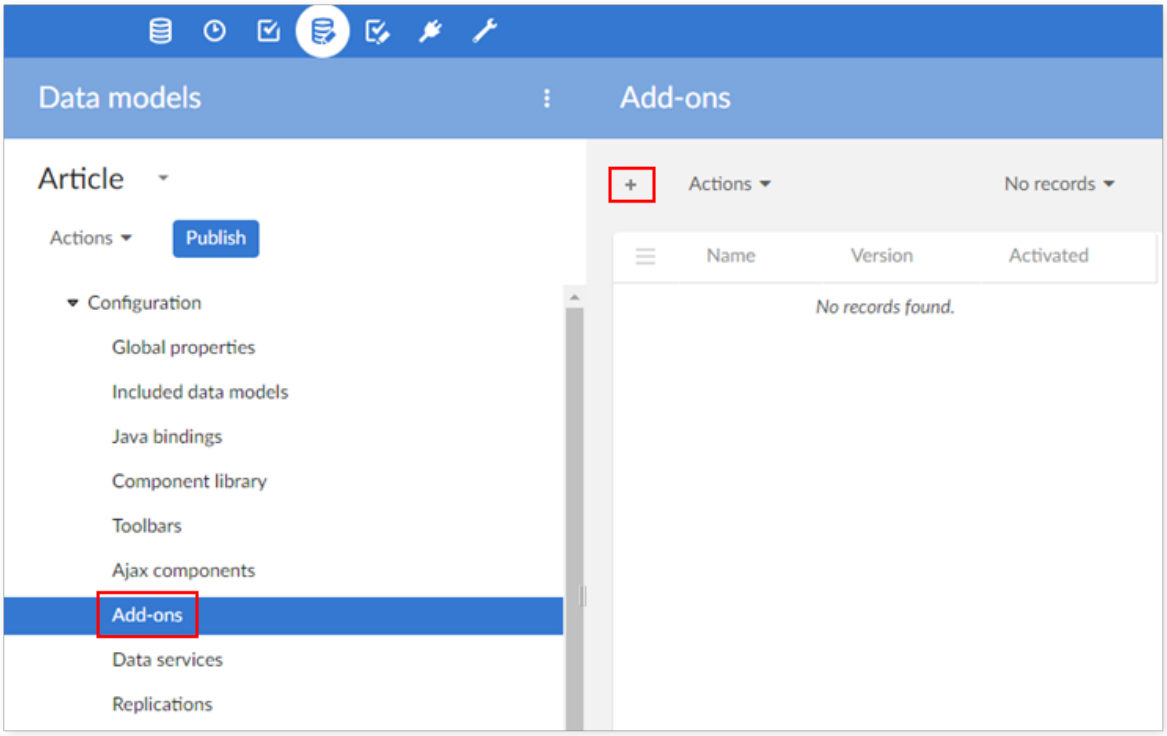
4.2 Automatically adding default classes

From version 1.4.0 of the add-on, you can add classes to a data model by simply activating the add-on in the model. Thus, alleviating the need to manually add classes one at a time. When you publish your Rules Portfolio the add-on automatically adds the required classes, triggers and constraints to the data model based on rules configured for this model. However, the following limitations exist:

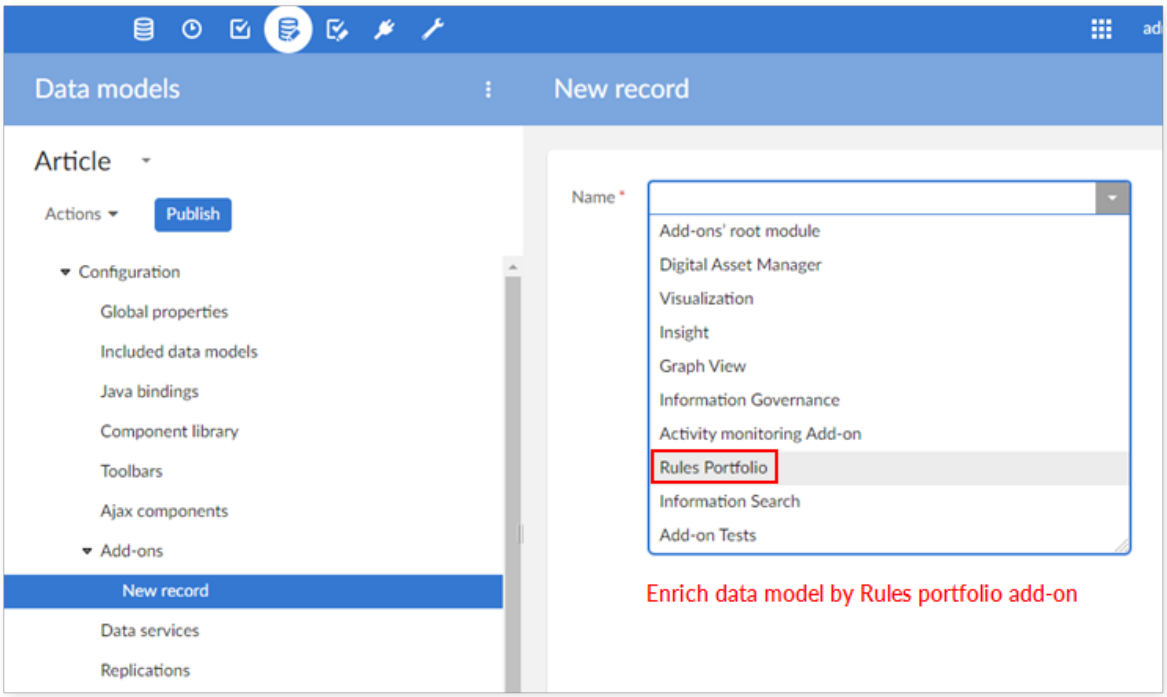
- Action permission rules are not supported. You have to manually add these using the method described in the 'Adding default classes manually' section above.
- Default classes - available to be added - automatically include the default schema extension. However, one data model can only have one schema extension. If you have your own schema extension in the data model, and you want to use a default class, you have to first remove your own schema extension from the data model. To reuse your programmatic access rules, you have to invoke them into the add-on using the method described in the appendix. See the 'Classic access rule declaration' section for more information.
- You must keep your D.E.C.'s current. This is especially true if you add or remove the mandatory attribute from a data model field. To ensure successful rule operation, update the D.E.C. configuration before publishing the data model. For instance, if you update a 'Short label' field to a mandatory field, then you have to find the D.E.C. field corresponding to the updated field and set the 'Mandatory' field to 'Yes' and vice versa.

The following steps show you how to configure your data model to automatically add default classes:

- Open the data model and navigate to 'Configuration' → 'Add-ons'.



- Click '+' to create a new record and select the Rules Portfolio.



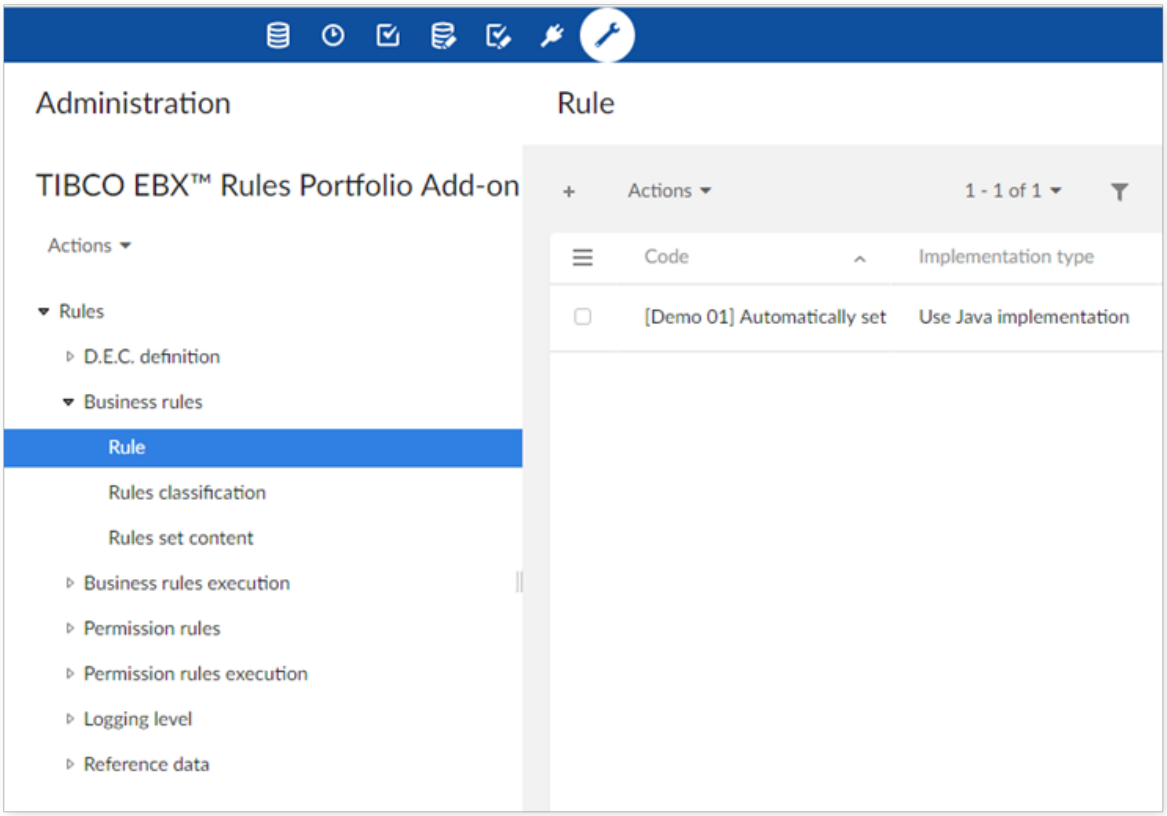
CHAPTER 5

Rules configuration

This chapter contains the following topics:

5.1 **Administration domain**

The 'TIBCO EBX® Rules Portfolio Add-on' data space is located in the Administration tab and allows you to configure rules and how they execute.



- The 'D.E.C. definition' group contains tables to declare the Data Element Concept (D.E.C.). The D.E.C. is an ISO-IEC 11179 concept that is used in other add-ons such as EBX® Information Governance Add-on and EBX® Insight Add-on. Rules can be executed on a D.E.C. which can be either a table, a field, a service, etc.
- The 'Business rules' group contains the business rules definitions and their packaging in the form of 'Rules set content' and 'Rules classification'.
- The 'Business rules execution' group contains all tables defining the context for rules set execution.
- The 'Permission rules' group contains all tables defining permission rules, permission rules set and permission rule classification.
- The 'Permission rules execution' group contains all tables defining contexts to execute a permission rule or a permission rules set.
- The 'Logging level' group contains the configuration for the logging mechanism.

- The 'Reference data' group contains the reference tables' definition.

Special Notation	
✔	Rules do not execute through add-on configuration. They execute on a publication of the rule configuration. Before executing rules, you have to run the 'Publish rule portfolio' service to publish the current configuration.

5.2 D.E.C. Definition

The 'D.E.C. definition' group contains tables that declare a Data Element Concept (D.E.C.). The D.E.C. is an ISO-IEC 11179 concept that is used in other EBX® add-ons such as EBX® Information Governance Add-on and EBX® Insight Add-on. Rules can be executed on a D.E.C. which can be either a table, a field, a service, etc.

D.E.C. (Data Element Concept)

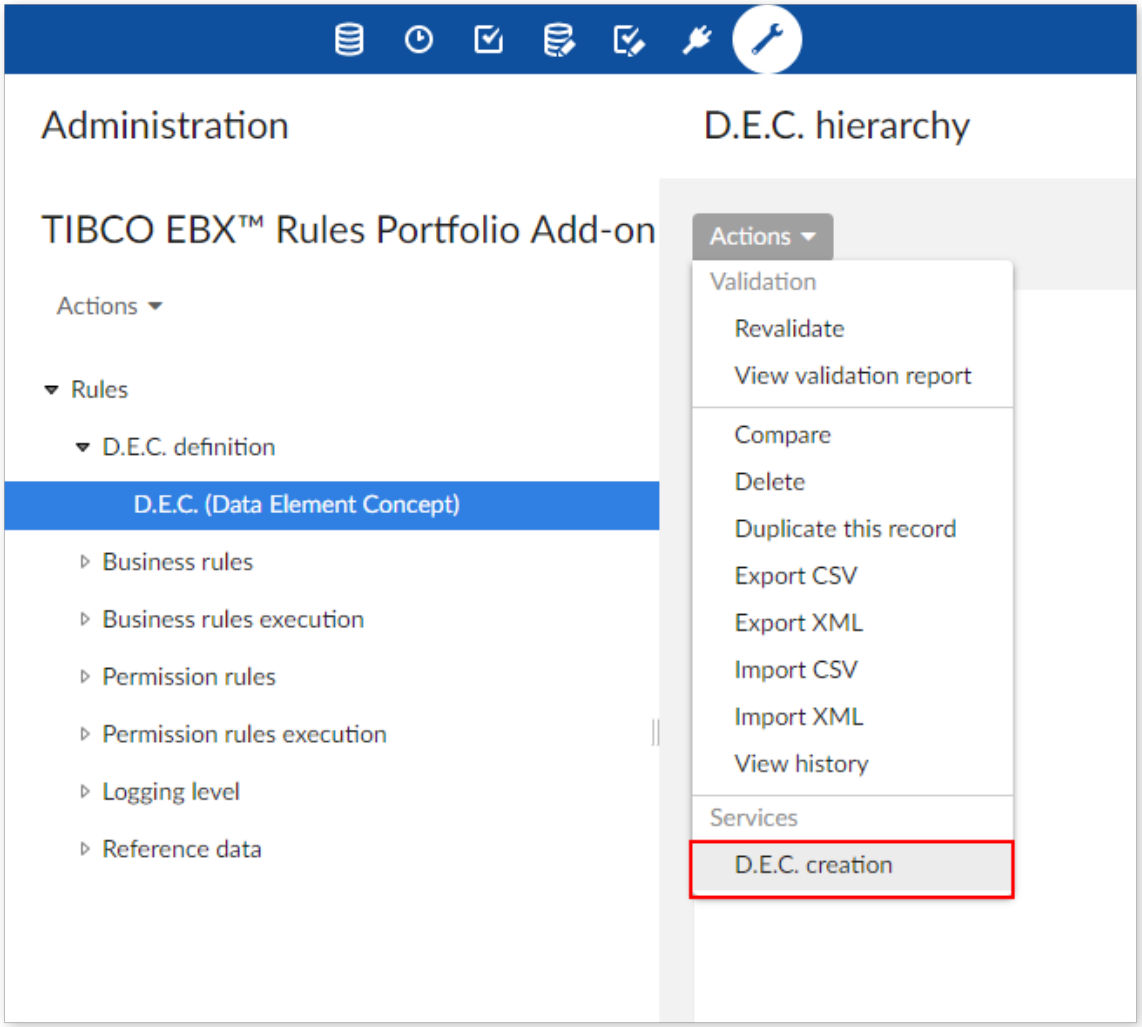
A Data Element Concept (D.E.C.) is a generic term defining any type of data asset that can be associated with rules. The primary D.E.C. used is 'Table'.

You can define the D.E.C. properties shown in the following table:

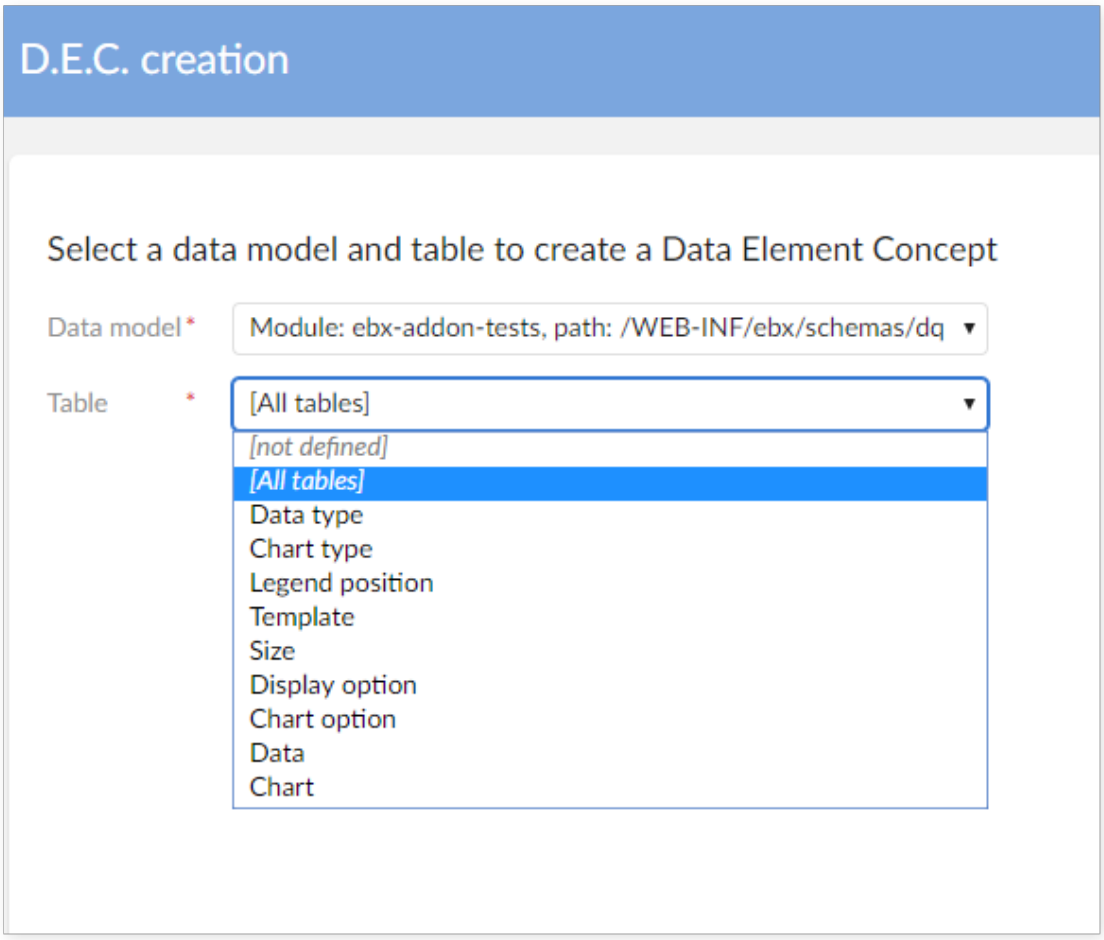
Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined Data Element Concepts. Example: [ON] Account
Business code	A business identifier.
Name	The D.E.C. name. Any naming convention is valid.
Description	The D.E.C. Description. Any naming convention is valid.
Type	The list of D.E.C. available types which is supplied by the reference table 'D.E.C. type' (Service, Data model, Table, Field). Note: After selecting a D.E.C. Type, you need to validate (Click 'Save') the form to display a type's associated properties.
If 'Type' is set to 'Data model', 'Table' or 'Field', the following properties are available:	
Data model	The data model to which the Data Element Concept belongs.
If 'Type' is set to 'Table' or 'Field', the following property is available:	
Table	A list of available tables contained in the selected 'Data model'.
If 'Type' is set to 'Field', the following property is available:	
Field	A list of available fields contained in the selected 'Table'.
Mandatory	If 'Yes': this field is mandatory. If 'No': it is not.
Computed value	If 'Yes': this field is a computed value. If 'No': it is not.
Multi-valued	If 'Yes': this field is a multi-valued or under a multi-valued group. If 'No': it is not.
If 'Type' is 'Service', the following property is available:	
Service	A field that allows you to enter a 'Service' name.

D.E.C. creation service

You can automatically create all D.E.C.(s) for a selected data model. The 'D.E.C. creation' service is available in the D.E.C. table as illustrated below.



In configuration part of 'D.E.C. creation' service, you have to choose data model and table. The 'D.E.C. creation' service generates D.E.C.s based on your input data.



5.3 'Business rules' group

This group allows you to define business rules and their packaging.

Configuring a business rule

You can base rule configuration on business rules and implement their configuration using a script or Java. Predefined Java-implemented rules are ready-to-use and require no specific programming knowledge from end-users. However, using a script requires scripting language knowledge for implementation and requires end-users to possess at least basic programming knowledge. For more information about the scripting language see the Scripting Language.

The 'Rule' table contains settings that allow you to define rules. This table's view defaults to a hierarchy of D.E.C.'s under which associated rules display. You can use the 'View' drop down menu to change views. The following points outline the rule creation process:

- Choose whether to use a script or Java based rule.
- Name the rule and select the D.E.C. on which the rule executes.
- Specify the rule type and the conditions that must be met for successful execution.

- Document the message displayed when the rule cannot execute successfully.

The following table describes the 'Rule' table's properties that allow you to configure a rule:

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined rules. Example: [ON] Rule 01
Implementation type	There are 2 ways to implement a business rule: -Use script: you use a script to define the rule. -Use Java implementation: you use Java implementation to define the rule.
Name	The rule name. You can use any naming convention.
Description	The rule description. You can use any naming convention.
Table	The Data Element Concept (D.E.C.) on which the rule executes. By default, this is a data model table. A control is applied by the add-on to check if the selected table is compliant with the rule definition.
Field attachment	The field on which the rule executes. The field specified by this property is only used if a constraint prompts execution of a validation rule. When you specify a field, the rule can only execute from a constraint attached to the specified field in the table selected above. The system returns either a warning, or an error based on the level of severity chosen. If this property is set to undefined, the rule can execute via constraint on any field in the selected table. The rule can execute via trigger, regardless of the field specified here.
Type	A rule's type. This value can be: Validation rule, Manual validation rule, Automated rule or Table set rule • A 'Validation rule' is an assertion without any impact on the system and outputs a Boolean value that indicates whether rule execution succeeded or failed. The 'Validation rule' executes upon creation, modification or deletion of a record. • A 'Manual validation' rule is an assertion without any impact on the system and outputs a Boolean value that indicates whether rule execution succeeded or failed. This type of rule is executed through an action taken by a user either at the data set or table level using the standard validation service. • An 'Automated rule' can modify data and outputs a Boolean value that indicates whether rule execution succeeded or failed. This type of rule executes upon creation, modification or deletion action of a record. If the automated rule raises an error, it interrupts the trigger, displays an error message and aborts the transaction. • A 'Table set' rule can create, modify or delete data. Alternatively, it can be an assertion without any impact on the data value. This type of rule is executed through the 'Execute rules' service either at the level of a data set or a table and outputs a Boolean value that indicates whether execution succeeded or failed.
Active	If set to 'True': the rule can execute. If set to 'False': the rule is deactivated and cannot execute even if all other execution conditions are valid (execution context, rule set is active).
Mandatory	• The field Mandatory is displayed only if the rule type is "Table set rule". • The default value of this field is "No". • If the value is set to "No", user can select/unselect the rule.

Property	Definition
	• If the value is set to "Yes", the created table set rule is checked automatically and user cannot edit.
Severity	The type of error message displayed, either Warning or Error.
Message	The message displayed when an error or warning is raised.
Script	You enter the script for your rule here. The Scripting Language used by the add-on is based on JavaScript and simplifies the rule definition process. However, when running against a high volume of data, performance may be reduced. So, careful consideration must be put into the decision on whether to use Java or a script.
Java Implementation This group field defines: • The implementation in java • This implementation's input/output • This implementation's parameters	
Rule name	The list of available rules. This list populates automatically through a Java introspection of the rules implementation (refer to the Java doc).
Object class	Business object on which the rule can be applied. This information is for documentation only. This is not the table name. For example, the "Employee" Object class can be related to several tables, such as: Employee, Third party, Salaried employee, etc.
Input data	Read-only parameters controlled and used by the add-on to execute the rule.
Output data	The read-only result data.
Property data	These are parameters that can be modified at the configuration level, and used as input data for rule execution.
Dependency This group defines all dependencies for this field.	
Mode	You can choose from three dependency modes: • Local dependency: In this mode, the current constraint's validation result depends on the underlying node's local value. Only modification of this local value affects the validation result. • Explicit dependencies: In this mode, the current constraint's validation result depends on source node updates that you explicitly specify. Only local updates and not external events affect the validation result. • Unknown dependencies (default mode): In this mode, the current constraint's validation result depends on unknown sources or events (for example, data stored in an external system). This is the default mode and enabled if you do not use either of the other dependency types, or if the element containing this constraint is computed by a value function.
Explicit dependencies This group allows you to define explicit dependencies. You can add multiple dependencies by clicking the '+' icon.	

Property	Definition
Dependency	This group is used for defining explicit dependencies only. It displays when you select the 'Explicit dependencies' mode.
Event	You can use the following seven events that define an explicit dependency: • Dependency to modify node: Specifies that the validation result of the current constraint depends on the value of the specified source node in the same record. • Dependency to insert and delete record: Specifies that the validation result of the current constraint depends on the insertions and deletions of the specified table node. • Dependency to insert delete and modify node: Specifies that the validation result of the current constraint depends on insertions, modifications and deletions on the specified node. • Dependency to insert and delete in other instance: Specifies that the validation result of the current constraint depends on the insertions and deletions of the specified table node of a given instance. • Dependency to insert delete and modify in other instance: Specifies that the validation result of the current constraint depends on the insertions, modifications and deletions on the specified node in a given data set. • Dependency to insert and delete in specific data space: Specifies that the validation result of the current constraint depends on the insertions and deletions of the specified table node of a given data set in a specific data space. • Dependency to insert delete and modify in a specific data space: Specifies that the validation result of the current constraint depends on the insertions, modifications and deletions on the specified node in a given data set in a specific data space. The following four possible fields display depending on the event you select: 'Data space', 'Data set', 'Table' and 'Field'.
Data space	If you set the 'Event' field to one of values presented below, this field displays: • Dependency to insert and delete in other instances • Dependency to insert, delete and modify in other instances • Dependency to insert, and delete in a specific data space • Dependency to insert, delete and modify in a specific data space
Data set	If you set the 'Event' field to one of values presented below, this field displays: • Dependency to insert and delete in other instances • Dependency to insert, delete and modify in other instances • Dependency to insert and delete in a specific data space • Dependency to insert, delete and modify in a specific data space
Table	If you set the 'Event' field to one of values presented below, this field displays: • Dependency to insert and delete record • Dependency to insert, delete and modify node • Dependency to insert and delete in other instances • Dependency to insert, delete and modify in other instances • Dependency to insert and delete in a specific data space • Dependency to insert, delete and modify in a specific data space
Field	If you set the 'Event' field to one of values presented below, this field displays: • Dependency to modify node

Property	Definition
	• Dependency to insert, delete and modify node • Dependency to insert, delete and modify in other instances • Dependency to insert, delete and modify in a specific data space

Classifying business rules

Rules can be organized into a classification scheme making their management easier. A rule can be associated to one to many domain of rules.

Property	Definition
Domain of rules	A domain of rules configured in the reference 'Domain of rules' table.
Rule	A rule configured in the 'Rule' table. This rule is attached to the domain.

Defining Rules set content

You can group individual rules into a 'Rules set' and execute multiple rules at the same time. The 'Rules set content' lets you define rules contained in a 'Rules set'.

Property	Definition
Rules set	A reference to a 'Rules set'.
Rule	A reference to a rule that is integrated into the 'Rules set'. All rules contained in a 'Rules set' must be attached to the same Data Element Concept (D.E.C.). Therefore, the list of rule selections is limited to those related to a single D.E.C. The referenced D.E.C. is determined by the first rule in the list.
Order	The order of rules contained in the 'Rules set'. This order manages the rule's execution sequence. You can rearrange rules using the 'move up', 'move down' and 'move to position' services in the 'Rule set hierarchy' data hierarchy.
Active rule	This property is inherited from the 'Is active' property configured in the 'Rule' table. It can be overwritten to adapt the property in the context of the rules set content. If set to 'True': the rule can be executed. If set to 'False': the rule is deactivated and cannot be executed even if all other execution conditions are valid (execution context, rule set is active).

5.4 'Business rules execution' group

The tables in the 'Business rules execution' group allow you to define the context for rule execution.

Using the 'Rule execution' table to configure business rule execution

A record in the 'Rule execution' table applies to either a business rule or multiple rules contained in a 'Rules set'. You can configure an event that determines when rule execution occurs (Ex. On

demand, After create, etc.). The D.E.C. on which the rules execute is defined in the rule definition. The following table lists 'Rule execution' properties and their definitions:

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined rule execution. Example: [ON] Create Account
Table	The Data Element Concept (D.E.C.) on which the rule executes. The value of 'Rule' and 'Rule set' will be filtered accordingly.
Rule	The rule to execute. If this property is configured, you cannot define the 'Rules set' property. After you select a rule, the 'Event' field's value filters accordingly.
Rules set	The 'Rules set' to execute. If this property is configured, you cannot define the 'Rule' property.
Execution context	This field allows you to choose: 'No context', 'Use Script' and 'Use Java implementation' for the execution context. The value 'Use simple expression' only pertains to old data and displays after the migration process.
Active	If set to 'True': the *Rule execution* can execute. If set to 'False': the *Rule execution* is deactivated and cannot execute, even if all other execution conditions are valid.
Validity date	The first date that the *Rule execution* can be executed.
Expiry date	The date that you can no longer execute the *Rule execution*, even though its 'Is active' property is set to 'True'.
Event	The event that launches the *Rule execution* (refer to the reference table 'Event'). These events are aligned with triggers: • Before create → HandleBeforeCreate • Before update → HandleBeforeModify • Before delete → HandleBeforeDelete • After create → HandleAfterCreate • After update → HandleAfterModify • After delete → HandleAfterDelete This means that 'before' is used to modify a record before the save, and 'after' is used to execute operations after the save. But if both are in same transaction and an error is raised the save is aborted. In the current version of the add-on, you manually have to select the event. In a future version, automatic selection of possible events will be available. To select the event: • 'After' for the 'Automated rule' rule type. This rule type can modify the record before it is saved by the system, and any other information beyond the current record. If the rule raises an error, the save is aborted. • 'Before' for the 'Validation rule' rule type. This type of rule cannot modify the record and is executed before the system saves the record. If a validation rule raises an error, the save is aborted.

Property	Definition
	• 'On demand' is used with the 'Manual validation rule' and 'Table set rule' rule types. This event allows you to execute the rules through the ' validation service' and the 'Execute rules' service, respectively. • 'On constraint' is used with the 'Validation rule'. This event allows you to execute validation rules using a constraint set on a field. Setting 'On demand' on a 'Automated rule' or 'Validation rule' type ensures that this rule is not automatically executed. Setting 'After' or 'Before' on a 'Manual validation rule' or 'Table set rule' type ensures that this rule is not automatically executed. When rule execution is configured in a rules set that includes different rule types, the event to select must be prioritized as shown below: • The 'Event' is selected by the priority first of the 'Automated rule', and then of the 'Validation rule'. • For example, a rules set can contain all four rule types: 'Manual validation rule', 'Validation rule', 'Automated rule' and 'Table set rule'. If the 'Event' in this case, is After or Before then only 'Automated rule' and 'Validation rule' are executed. If the 'Event' is 'On demand' then only 'Manual validation rule' and 'Table set rule' are executed.
Data set execution	One to many data sets can be selected as a execution context. This means that rule execution happens only if the execution is requested in the defined data set(s). It is possible to use the special tokens '[All data spaces]' and '[All data sets]'.
User profile	This property allows you to restrict rule execution for the selected user-profiles. This is not applied to 'Manual validation' rule types that are executed through the validation report service.
Script	You enter the script for your rule here. The Scripting Language used by the add-on is based on JavaScript and simplifies the rule definition process. However, when running against a high volume of data, performance may be reduced. So, careful consideration must be put into the decision on whether to use Java or a script. See the Scripting Language for more information.
Java Implementation This group field defines: • The implementation in java • This implementation's input/output • This implementation's parameters	
Rule name	The list of available rules. This list populates automatically through a Java introspection of the rules implementation (refer to the Java doc).
Object class	Business object on which the rule can be applied. This information is for documentation only. This is not the table name. For example, the "Employee" Object class can be related to several tables, such as: Employee, Third party, Salaried employee, etc.
Input data	Read-only parameters controlled and used by the add-on to execute the rule.
Output data	The read-only results data.

Property	Definition
Property data	You can modify these parameters at the configuration level and use them as input data for rule execution.

Context

A context is used to associate a *Rule execution* with one to many *Simple expression(s)*. Only simple expressions related to the same Data Element Concept as the 'Rule execution' can be selected in the configuration.

Special Notation	
✕	From version 1.3.0. of the add-on, the 'Context' table is only used for backward compatibility. Its permission level is set to read-only if it previously contained data, if not, it is hidden.

Property	Definition
Rule execution	A rule execution on which the context is defined.
Simple expression	A simple expression that defines a constraint applied to the fields of the Data Element Concept declared for rule execution.
Logical operator	A logical operator used to combine two simple expressions. The logical operator of the first simple expression is configured to 'Void' or 'Not'.
Order	Displays the simple expression order. When more than one simple expression is configured, their order and associated 'Logical operator' determine flow. You can manage simple expression order in the data hierarchy view 'Context by D.E.C. and Rule execution'.

Simple expression

A 'Simple expression' is used by the 'Context' table and defines the conditions under which rules can be executed.

Special Notation	
✖	From version 1.3.0. of the add-on, the 'Simple expression' table is only used for backward compatibility. Its permission level is set to read-only if it previously contained data, if not, it is hidden.

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined simple expressions. Example: [ON] Age greater than 50
Name	Any naming convention.
Table	The Data Element Concept on which the simple expression is built. By default, this is a table.
Field	A Data Element Concept field that is used to build the first part of the simple expression.
Comparison operator	A list of comparison operators. The selections populated in the list depend on the 'Field' selected.
Value	The 'Value' property can be used to build the second part of the simple expression and is a predefined value to compare against the 'Field' value. You cannot use the 'Value' property and the 'Field 2' property in the same simple expression.
Field 2	The 'Field 2' property can be used to build the second part of the simple expression and compares the value of the 'Field' property with the value specified in 'Field 2'. You cannot use the 'Value' property and the 'Field 2' property in the same simple expression.
Implementation - Sometimes configuration requirements for a simple expression go beyond using the 'Field', 'Comparison operator' and 'Field 2' properties. In this type of situation you can configure a rule as a simple expression. The rule's output data enforces a standard interface (refer to the Java doc).	
Rule name	The name of the selected rule. This list populates automatically through a Java introspection of the rules implementation (refer to the Java doc).
Object class	Business object on which the rule can be applied. This information is for documentation only. This is not the table name. For example, the "Employee" Object class can be related to several tables, such as: Employee, Third party, Salaried employee, etc.
Input data	Read-only parameters used to execute the rule that are under the full control of the add-on.
Output data	Read-only results data.

Property	Definition
Property data	Modifiable parameters used as input data for rule execution.

5.5 Logging level

This group contains logging mechanism configuration settings and is hidden in the rule portfolio snapshot (publication of rules' configuration).

Business rules logging

This record contains the properties to configure business rules logging.

Property	Definition
Trace activation	If set to 'True': Logging is active. If set to 'False': Logging is not active. This property's default value is 'False'.
Automatic archive	If set to 'True': archives are automatically created when the maximum number of logs allowed is reached. If set to 'False': the system will not create automatically archives file after creating the log data. This property's default value is 'False'.
Stack trace size (in records)	The maximum number of logs allowed before the automatic archive process begins archiving logs or overwriting older ones.

Permission rules logging

This record contains the properties to configure permission rules logging.

Property	Definition
Trace activation	If set to 'True': Logging is active. If set to 'False': Logging is not active. This property's default value is 'False'.
Automatic archive	If set to 'True': archives are automatically created when the maximum number of logs allowed is reached. If set to 'False': the system will not create automatically archives file after creating the log data. This property's default value is 'False'.
Stack trace size (in records)	The maximum number of logs allowed before the automatic archive process begins archiving logs or overwriting older ones.

5.6 'Permission rules' group

The tables in the 'Permission rules' group allow you to define permission rules, permission rules sets and permission rule classification.

Configuring a permission rule

You can base rule configuration on permission rules and use Java or a script to implement the configuration. Predefined Java-implemented rules are ready-to-use and require no specific programming knowledge from end-users. However, using a script requires Scripting Language knowledge for implementation and users should possess at least basic programming knowledge. See the Scripting Language for more information.

A 'Rule' allows you to set Hidden, Read-Only or Read-Write permissions and apply them to a table, record, field, data set or service.

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on to provide predefined rules. Example: [ON] Rule 01
Implementation type	There are 3 implementation types: • Use script: you create the rule using a script. • Use Java implementation: you use Java implementation to create the rule. • Use Classic access rule: you use your own programmatic access rule to create a rule. This type is used to create access permission rule only.
Name	A required, editable field that displays the 'Rule' name. You can use any naming convention.
Description	An optional, editable field where you can enter a description for the 'Rule'.
D.E.C. attachment	The selected Data Element Concept (D.E.C.) on which the rule executes.
Scope	Scope of permission rule.
Type	There are 2 supported types for permission rules: • An 'Access permission rule' manages node permissions, namely a data set, table or field. This type of rule determines whether or not the node is Hidden, Read-Only or Read-Write. • An 'Action permission rule' manages service permissions. This type of rule controls whether or not a given service can be used in the current context (displays or hides the service).
Active	If set to 'True': the rule can execute. If set to 'False': the rule cannot execute even if all execution conditions are valid (execution context, rule set is active).
Script	You enter the script for your rule here. The Scripting Language used by the add-on is based on JavaScript and simplifies the rule definition process. However, when running against a high volume of data, performance may be reduced. So, careful consideration must be put into the decision on whether to use Java or a script.
Java Implementation This group field defines: • The implementation in java • This implementation's input/output • This implementation's parameters	
Rule name	A list of the available rules. This list populates automatically through a Java introspection of the rule implementation (refer to the Java doc).

Property	Definition
Object class	Displays the name of the business object class on which the rule implementation is applied. This information is for documentation only. This should not be a specific name of a table, field, record, data set or service on which the rule is applied, but a generic identification, namely: Table, Field, Service, etc.
Input data	These read-only parameters are used for rule execution and are fully under control of the add-on.
Output data	Read-only result data.
Property data	These parameters can be modified at the configuration level and used as input data for rule execution.
Classic access rule Enter the class path of the programmatic access rule you want to execute. Note: Your classic access rule has to satisfy the following conditions: -It has to implement the AccessRule interface. -It must have a no-argument constructor.	
Access rule	Class path of your programmatic access rule.

Rule classification

A rule can be associated with one to many 'Domain of rules'. This enables you to organize rules into any classification scheme and simplifies their management.

Property	Definition
Domain of rules	A domain of rules configured in the 'Domain of rules' reference table .
Rule	A rule configured in the 'Rule' table. This rule is attached to the 'Domain of rules'.

Rules set content

You can group individual rules into a 'Rules set' which allows multiple rules to execute at the same time. The 'Rules set content' lets you define the rules contained in a 'Rules set'.

Property	Definition
Rules set	The name of the referenced 'Rule set'.
Rule	The name of the 'Rule' that is added to the rules set. All rules in a 'Rules set' are attached to the same Data Element Concept (D.E.C.). Therefore, you can only select rules that are attached to the first listed rule's D.E.C.
Order	The order of rules contained in the 'Rules set'. This order manages the rule's execution sequence. You can rearrange rules using the 'move up', 'move down' and 'move to position' services in the 'Rule set hierarchy' data hierarchy.
Active rule	This property is inherited from the 'Is active' property configured in the 'Rule' table. It can be overwritten to adapt to the property in the context of the rules set. If set to 'True': the rule can be executed. If set to 'False': the rule is deactivated and cannot be executed even if all other execution conditions are valid (execution context, rule set is active).

5.7 **'Permission rules execution' group**

The 'Permission rules execution' group allows you to define contexts to execute a permission rule or a permission rules set.

Using the 'Rule execution' table to configure permission rule execution

A record in the 'Rule execution' table can be an executable permission rule or a 'Rules set'. The rule definition determines the D.E.C. on which the rules are executed.

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined rule execution. Example: [ON] Create Account
D.E.C. attachment	The Data Element Concept (D.E.C.) on which the rule executes. The value of 'Rule' and 'Rules set' will be filtered accordingly.
Rule	Displays the rule to execute. This property cannot be used in combination with the 'Rules set' property.
Rules set	Displays the 'Rules set' to executed. This property cannot be used in combination with the 'Rule' property.
Execution context	This field allows you to choose: 'No context', 'Use Script' and 'Use Java implementation' for the execution context. The value 'Use simple expression' only pertains to old data and displays after the migration process.
Active	If set to 'True': this configuration can execute. If set to 'False': this configuration is deactivated and cannot execute even if all other conditions of the execution are valid.
Validity date	The date that you can execute the "Rule execution".
Expiry date	The date that you can no longer execute the "Rule execution", even if its 'Is active' property is set to 'True'.
Data set execution	One to many data set(s) can be selected as an execution context. This means that the rule execution happens only if the execution is requested in the defined data set(s). It is possible to use the special tokens '[All data spaces]' and '[All data sets]'.
User profile	This property allows you to restrict the execution of rules based on the selected user-profile. This does not apply to 'Manual validation' type rules that are executed through the validation report service .
Script	You enter the script for your rule here. The Scripting Language used by the add-on is based on JavaScript and simplifies the rule definition process. However, when running against a high volume of data, performance may be reduced. So, careful consideration must be put into the decision on whether to use Java or a script. See the Scripting Language for more information.
Java Implementation This group field defines: • The implementation in java • This implementation's input/output	

Property	Definition
• This implementation's parameters	
Rule name	The list of available rules. This list populates automatically through a Java introspection of the rules implementation (refer to the Java doc).
Object class	Business object on which the rule can be applied. This information is for documentation only. This is not the table name. For example, the "Employee" Object class can be related to several tables, such as: Employee, Third party, Salaried employee, etc.
Input data	Read-only parameters controlled and used by the add-on to execute the rule.
Output data	The read-only results data.
Property data	These parameters that can be modified at the configuration level, and used as input data for rule execution.

Context

A context is used to associate a *Rule execution* with one to many *Simple expression(s)*. Only simple expressions related to the same Data Element Concept as the 'Rule execution' can be selected for configuration.

Special Notation	
✕	From version 1.3.0. of the add-on, the 'Context' table is only used for backward compatibility. Its permission level is set to read-only if it previously contained data, if not, it is hidden.

Property	Definition
Rule execution	The rule execution on which the context is defined.
Simple expression	The simple expression defining a constraint applied to the fields of the Data Element Concept that has been declared for rule execution.
Logical operator	A logical operator used to combine two simple expressions. The logical operator of the first simple expression is configured to 'Void' or 'Not'.
Order	The order of the simple expression. When many simple expressions are configured, their flow is defined by the order and the associated 'Logical operator' (refer to the next property). The order is managed through the data hierarchy view 'Context by D.E.C. and Rule execution'. The 'move up', 'move down' and 'move to position' are available from this view.

Simple expression

A 'Simple expression' is used by the 'Context' table to define the rule execution conditions.

Special Notation	
✕	From version 1.3.0. of the add-on, the 'Simple expression' table is only used for backward compatibility. Its permission level is set to read-only if it previously contained data, if not, it is hidden.

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined simple expressions. Example: [ON] Age greater than 50
Name	Any naming convention.
Table	The Data Element Concept on which the simple expression is built. By default, this is a table.
Field	The Data Element Concept field used to build the first part of the simple expression.
Comparison operator	A list of comparison operators automatically populated depending on the type of D.E.C. field selected.
Value	The 'Value' property is used to build the second part of the simple expression. This is a predefined value to compare with the 'Field' value. When this property is used then the 'Field 2' property is no longer available.
Field 2	The 'Field 2' property is used to build the second part of the simple expression. The value of the 'Field' property is compared with the 'Field 2' value. When this property is used then the 'Value' property is no longer available.
Implementation - Sometimes configuration requirements for a simple expression go beyond using the 'Field', 'Comparison operator' and 'Field 2' properties. In this type of situation you can configure a rule as a simple expression. The rule's output data enforces a standard interface (refer to the Java doc).	
Rule name	A list of the available rules. This list populates automatically through a Java introspection of the rules implementation (refer to the Java doc).
Object class	Business object on which the rule can be applied. This information is for documentation only. This is not the table name. For example, the "Employee" Object class can be related to several tables, such as: Employee, Third party, Salaried employee, etc.
Input data	Read-only parameters used for rule execution and are fully controlled by the add-on.
Output data	Read-only results data.
Property data	These parameters can be modified at the configuration level and used as input data for rule execution.

5.8 Reference data

The 'Reference data' group contains the reference tables' definition.

Domain of rules

You can make rule management easier by using a 'Domain of rules' to classify rules. A 'Domain of rules' is differentiated from a 'Rules set' in that it is used only as a classification scheme and not at run-time to execute multiple rules at the same time.

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined 'Domain of rules'. Example: [ON]Hyperion Account dimension
Name	The 'Domain of rules' name. You can use any naming convention.

Domain of rules hierarchy

You can use the 'Domain of rules hierarchy' to arrange 'Domain(s) of rules' and further facilitate classification.

Property	Definition
Parent domain	The 'Domain of rules' selected as the parent domain out of the available 'Domains of rules'.
Child domain	The 'Domain of rules' child. When there are many children, a child's position is configured through the data hierarchy view 'Domain hierarchy view' available on the 'Domain of rules' table. Their position can be adjusted using the 'move up', 'move down' and 'move to position' services.

Business rules set

A 'Business rules set' is a defined group of rules that can be executed together. 'Business rules set' content is defined in the 'Rules set content' table.

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined rules sets. Example: [ON] Validate account
Name	The name of 'Business rules set'. You can use any naming convention
Description	Text defining the 'Business rules set' description. You can use any naming convention.
Active	If set to 'True': the rule set is active and the rules are executed if the execution context is valid. If set to 'False': the rule set is inactive and cannot be executed even if the execution context is valid.

Permission rules set

A 'Permission rules set' defines a group of rules that can be executed together. 'Permission rules set' content is defined in 'Rules set content' table.

Property	Definition
Code	The name of the 'Rules set code'. Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined rules sets. Example: [ON] Validate account
Name	The name of 'Permission rules set'. You can use any naming convention
Description	Text defining the 'Permission rules set' description. You can use any naming convention.
Active	If set to 'True': the rule set is active and the rules are executed if the execution context is valid. If set to 'False': the rule set is inactive and cannot be executed even if the execution context is valid.
Restriction mode	A property that allows you to combine the results of the rule set's permission rules: • Most restricted - takes the most restricted result in the rule set. • Most unrestricted - takes the most unrestricted result in the rule set.

Data set

You can use a data set to limit rule execution. The following table lists data set properties and their definitions:

Property	Definition
Code	The data set name. You can use any naming convention.
Data space	The data space in which the rules are executed. This property is used as a context of the rules execution. Refer to the 'Rule execution' table.
Data set	The data set in which the rules are executed. This property is used as a context of the rules execution. Refer to the 'Rule execution' table.
Description	A description of the data set.

Event

An event initiates rule execution. The predefined events provided by the add-on allow you to manage any type of need. Refer to the 'Rule execution' table to see 'Event' definitions.

Property	Definition
Code	Any naming convention can be used except the prefix '[ON]' which is reserved by the add-on for predefined events. Example:[ON] Before create
Name	The 'Event' name. You can use any naming convention.
Description	Text defining the 'Event' description. You can use any naming convention.

Rule type

Refer to the online help for more information on the 'Rule type' table.

Property	Definition
Code	Any naming convention can be used except the prefix [ON] that is reserved by the add-on to provide predefined rule types. You can choose from four predefined rule types: [ON] Validation rule: executed by a trigger. This type of rule cannot modify data. [ON] Manual validation rule: executed by the user through the validation service. [ON] Automated rule: executed by a trigger. This type of rule can modify data. [ON] Table set rule: executed by the user through the 'Execute rules' service. This type of rule can modify the data. [ON] Access permission rule: manages node (data set, table or field) permissions. This type of rule controls whether or not the node is Hidden, Read-Only or Read-Write. [ON] Action permission rule: manages service permissions. This type of rule determines whether or not the given service can be used (displayed or hidden) in the current context.
Name	The 'Rule type' name. You can use any naming convention.
Description	Text defining the 'Rule type' description. You can use any naming convention.

Severity

The type of error message displayed, either Warning or Error.

Property	Definition
Code	There are two severity levels: • Warning: the error message displays as a warning. Only manual validation rules can rely on this error type. • Error: the error message displays as an error. Any type of rule can raise an error.
Name	You can use any naming convention.
Description	You can use any naming convention.

D.E.C. type

This table defines the type of data element concept.

Field	Definition
Code	Any naming convention can be used except the prefix '[ON]' that is reserved by the add-on to provide the default D.E.C. types. Example: [ON] Table
Name	Name of the D.E.C. type. Example: Table
Description	Text defining the 'D.E.C. type' description. You can use any naming convention.

Comparison operator

Defines the operator which is used to compare values.

Field	Definition
Code	Any naming convention can be used except the prefix '[ON]' that is reserved by the add-on to provide the predefined comparison operators. Example: [ON] Greater than
Name	The 'Comparison operator' name. You can use any naming convention.
Description	The 'Comparison operator' description. You can use any naming convention.

Logical operator

Defines the operator that is used to combine the logic value.

Field	Definition
Code	Any naming convention can be used except the prefix '[ON]' that is reserved by the add-on to provide predefined logical operators. Example: [ON] and
Name	The 'Logical operator' name. You can use any naming convention.
Description	The 'Logical operator' description. You can use any naming convention.

5.9 **Deleting contexts and simple expressions**

The 'Context' and 'Simple expression' tables remain to ensure backwards compatibility with versions of the add-on prior to the GA 1.3.0 release. Due to read-only or hidden access permissions, you cannot

create new records in these tables. However, you can still delete these records as existing contexts and expressions become obsolete.

To delete a context or simple expression, navigate to the relevant table, open the 'Actions' drop-down menu, and select either the 'Delete context' or 'Delete simple expression' service.

CHAPTER 6

TIBCO EBX® Rules Portfolio Add-on - Production

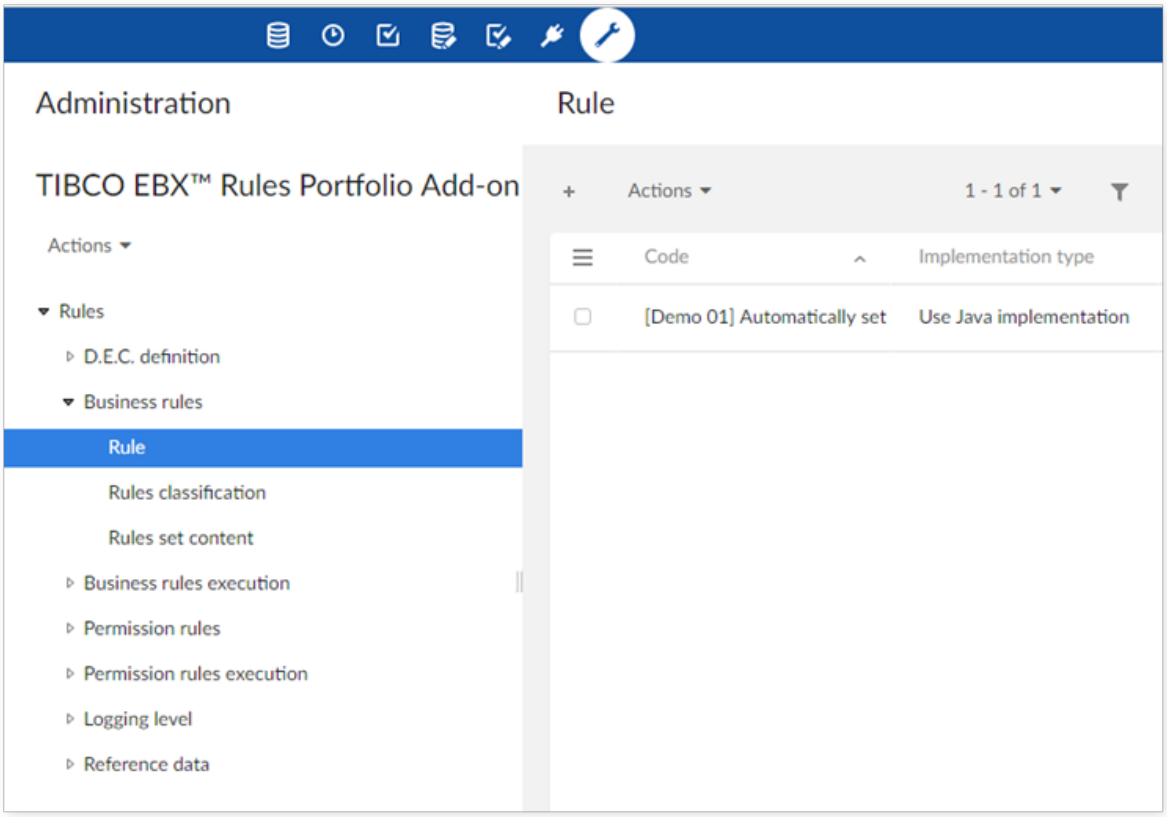
This chapter contains the following topics:

6.1 Administration domain

You can find the 'TIBCO EBX® Rules Portfolio Add-on - Production' data space under the 'Administration' tab. This data space allows you to make a stable rule configuration environment for running a rule. From the version 1.3.0 release, all rules run on the snapshot of a rule configuration instead of running directly on the rule configuration. The 'TIBCO EBX® Rules Portfolio Add-on - Production' data space stores the snapshot information of rules configuration and the mapping between the snapshot version and the data space where the rule is applied.

- The 'Rule publication' table stores the published snapshot information.

- The 'Rule repository' table stores the mapping of the data space which is applied by rules and snapshot version.



6.2 The 'Rule publication' table

The 'Publish rules portfolio' service creates a snapshot of the 'TIBCO EBX® Rules Portfolio Add-on' data set. One rule publication can be applied to many data spaces (or all data spaces). The rule publication properties are shown in the following table:

Property	Definition
UUID	The generated unique id.
Published snapshot	The publication of rules configuration. This is a snapshot of the 'TIBCO EBX® Rules Portfolio Add-on' data set.
Creation user	User who publishes the rule configuration.
Name&Description	The name and description of rules portfolio publication.
Creation date	The date and time when the rules configuration is published.

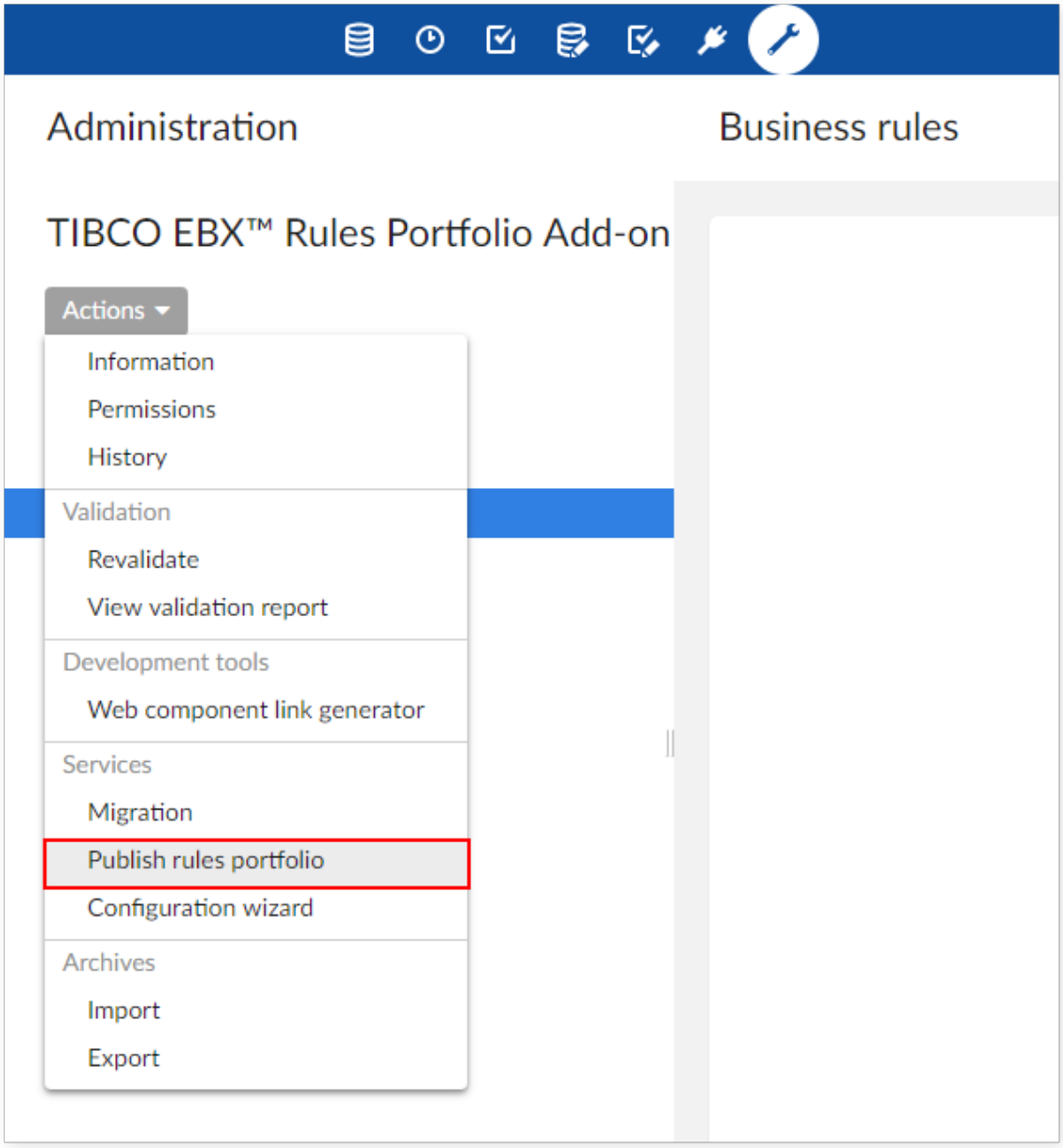
6.3 **The 'Rule repository' table**

A rule repository is the mapping between rule publications and data spaces. If the data space does not have any associated rule publication, the system will use the rule publication which is applied to [All data spaces]. The rule repository properties are shown in the following table:

Property	Definition
Data space	If there is an existing publication applied to the current data space, the system uses this publication to execute rules on this data space; otherwise, the system uses the publication applied to [All data spaces] to execute rules.
Rule publication	The rules portfolio publication which will be applied to the data space.
Creation user	Inherited from the Rule publication table.
Creation date	Inherited from the Rule publication table.

6.4 The 'Publish rules portfolio' service

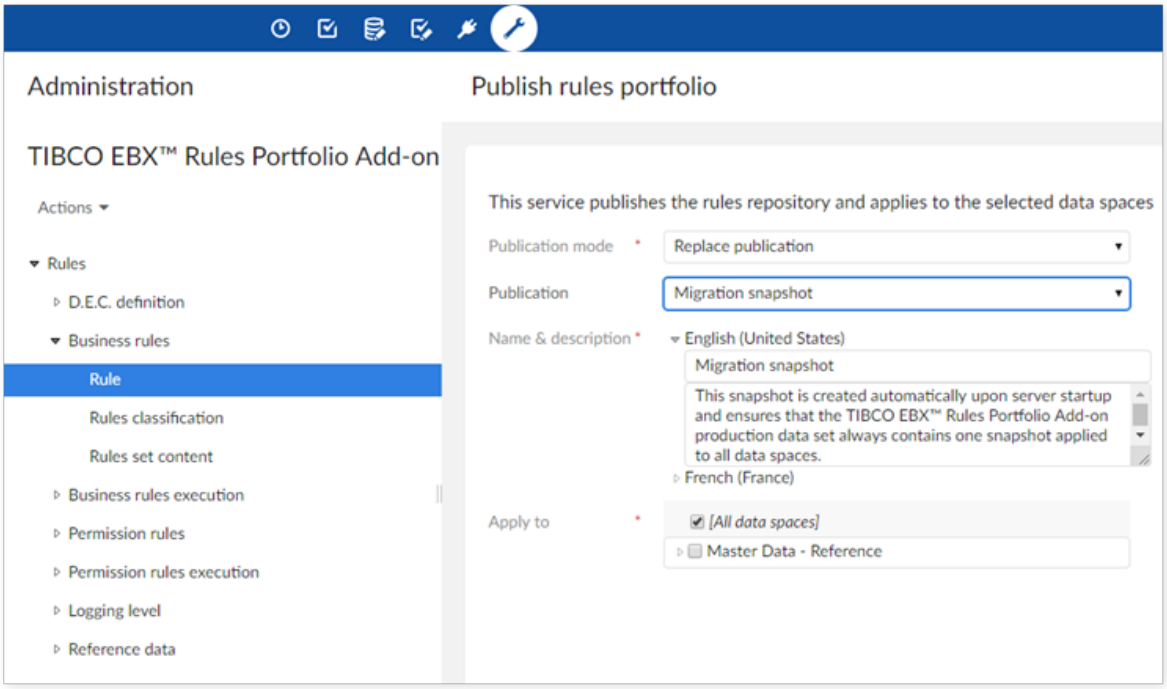
As shown below, the 'Publish rules portfolio' service is located in the 'TIBCO EBX® Rules Portfolio Add-on' data set's 'Actions' menu. This service publishes the current rules portfolio configuration.



Once you have run the service, you can use the publication mode option to either create a new, or overwrite an existing publication. If creating a new publication, input a name, description, and choose

the data space(s) that this publication applies to. If overwriting an existing publication, specify the publication to overwrite and the required fields.

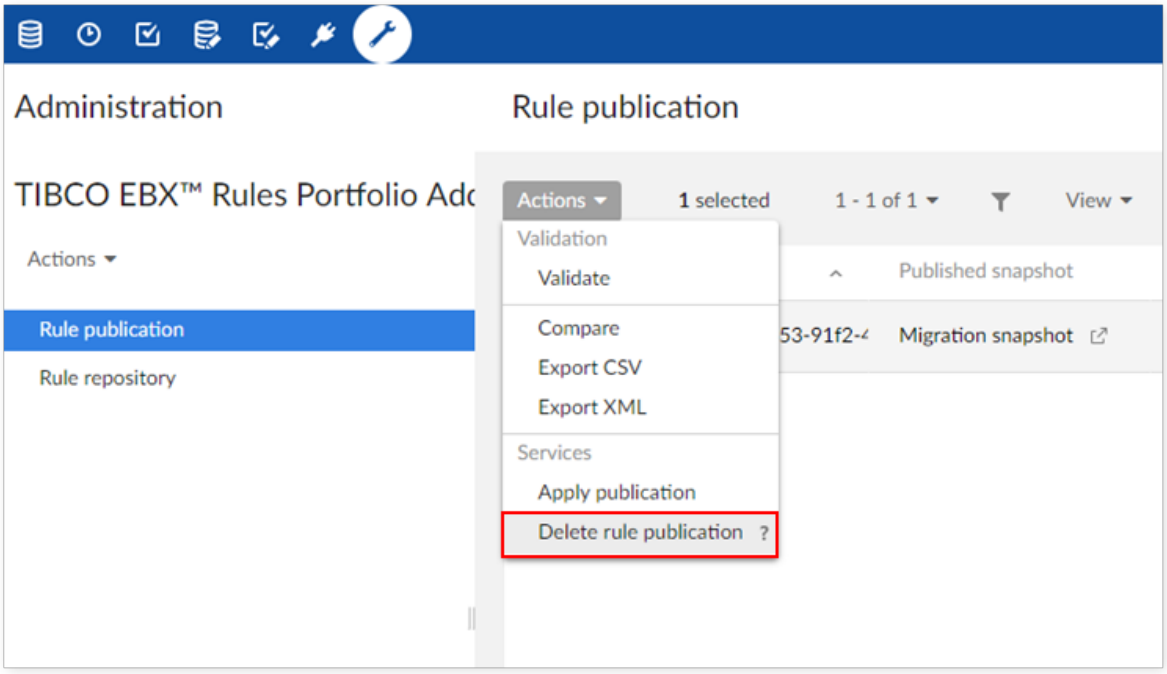
Special Notation	
✔	"[All data spaces]" is a special value. The rule publication applied to [All data spaces] is only used when there is no existing publication applied to current data space.



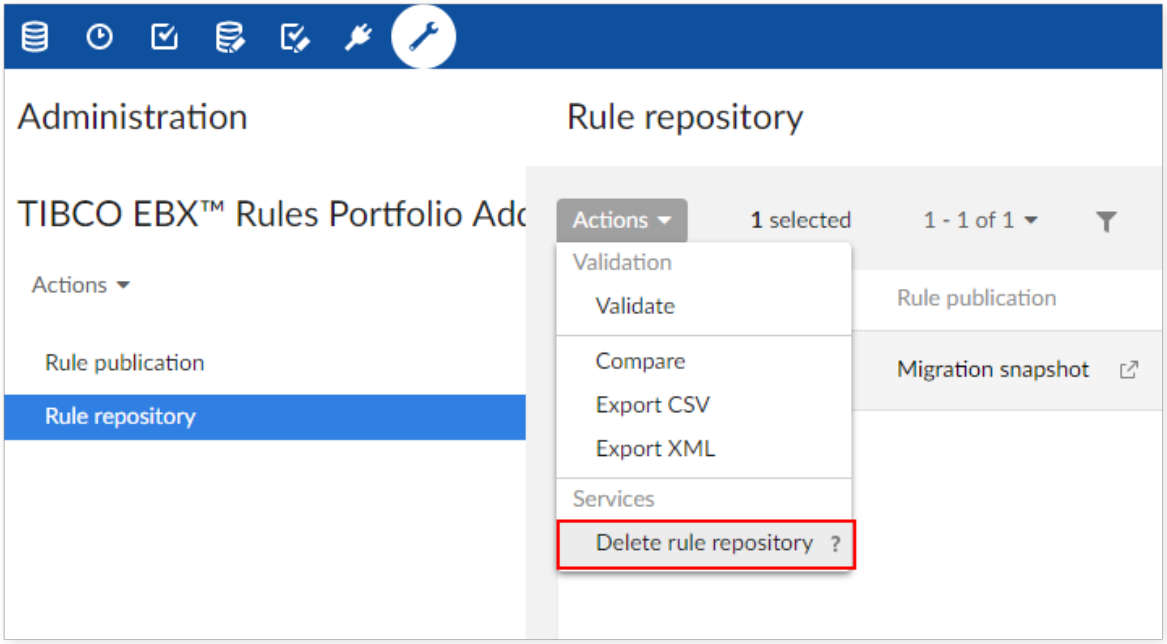
6.5 The 'Delete rule publication' and 'Delete rule repository' services

The access permission for the 'TIBCO EBX® Rules Portfolio Add-on - Production' data set is read-only, so it is unable to manually create or delete any record. The 'Publish rule portfolio' service creates records on both tables in the 'TIBCO EBX® Rules Portfolio Add-on - Production' data set. Records in the 'Rule publication' table store a snapshot's information and records and the 'Rule repository' table stores the mapping. The 'Delete rule publication' and 'Delete rule repository' services were created and activated on their respective tables to enable deletion.

To delete a rule publication, you only need to choose the publication that you want to remove and then select the 'Delete rule publication' service (shown in the image below). This service deletes not only the chosen rule publication but also the associated rule repository.

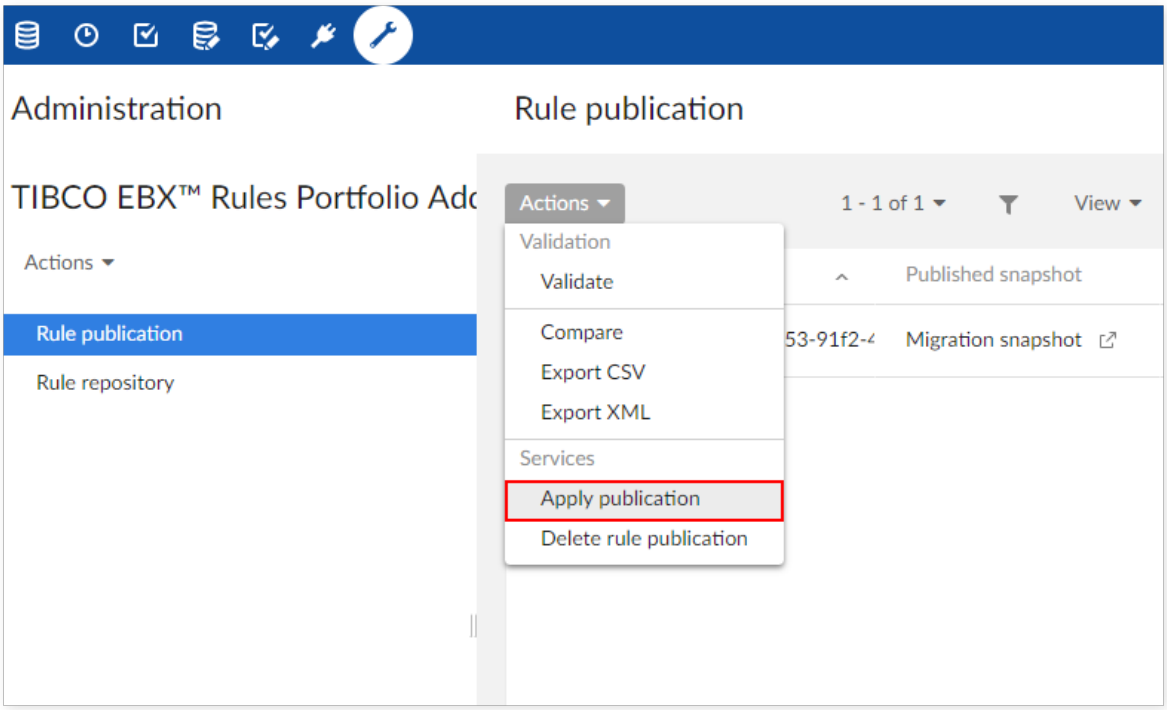


Similar to 'Delete rule publication' service, the 'Delete rule repository' service removes all the chosen rule repositories.

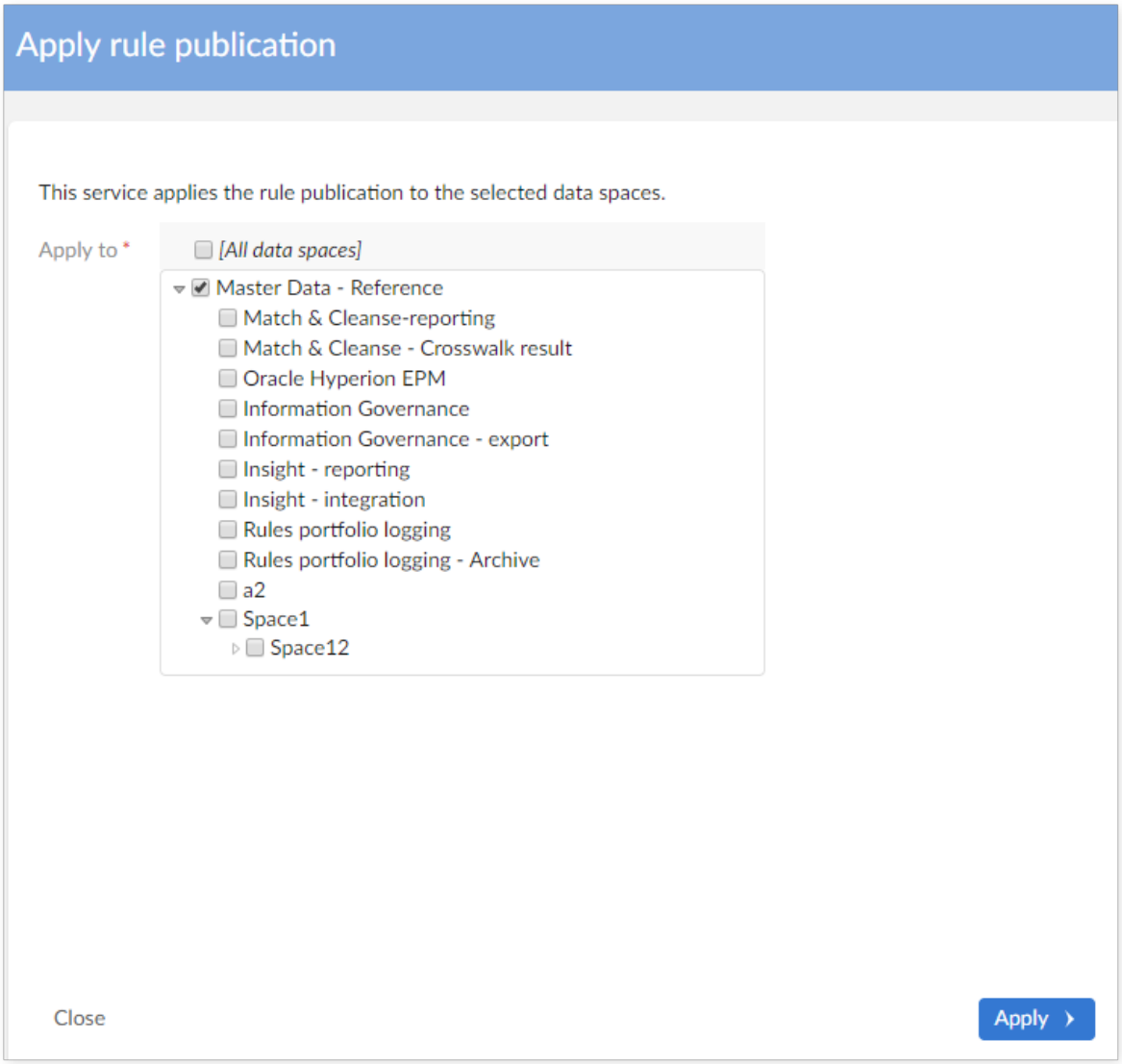


6.6 The 'Apply publication' service

The 'Apply publication' service is located at the 'Rule publication' table level. This service allows you to apply existed rule publication to the selected data spaces except for the data spaces that it already applies.



As shown below, you only need to choose the data space(s) that your existed rule publication applies to. The data spaces that already be applied by that rule publication will have the blur check box and the value of that check boxes cannot be changed.



CHAPTER 7

Use case

This chapter contains the following topics:

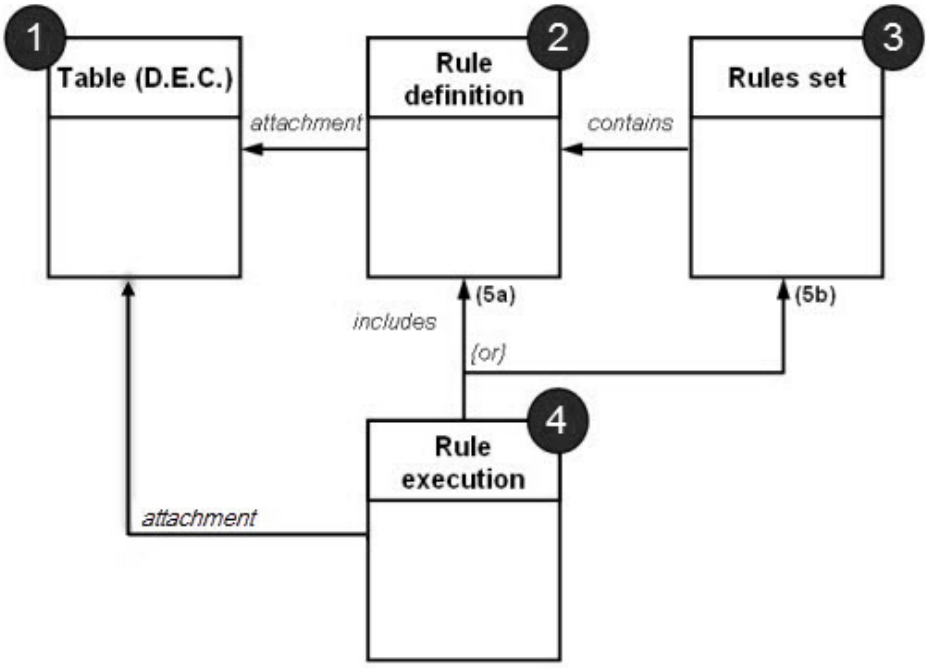
7.1 Overview of use cases

This section contains use cases that demonstrate how the EBX® Rules Portfolio Add-on can meet the following business requirements:

- Prevent deletion of records that have a child relationship, or an association to a record from another table.
- Ensure that a field's value does not exceed a specified number and if it does, auto-correct the value.

The examples in this section do not show the Java code used for rule implementation. However, it does include add-on specific scripting language examples where applicable. See the add-on's Java API documentation, if you would like more information on coding your own Java implementation classes.

The diagram below outlines the rule creation process. A 'Rule execution' declares information about the rule such as the event on which the rule executes, a validity date range or any other context required for rule execution. A 'Rules set' allows you to declare multiple rules and attach them to a definition.

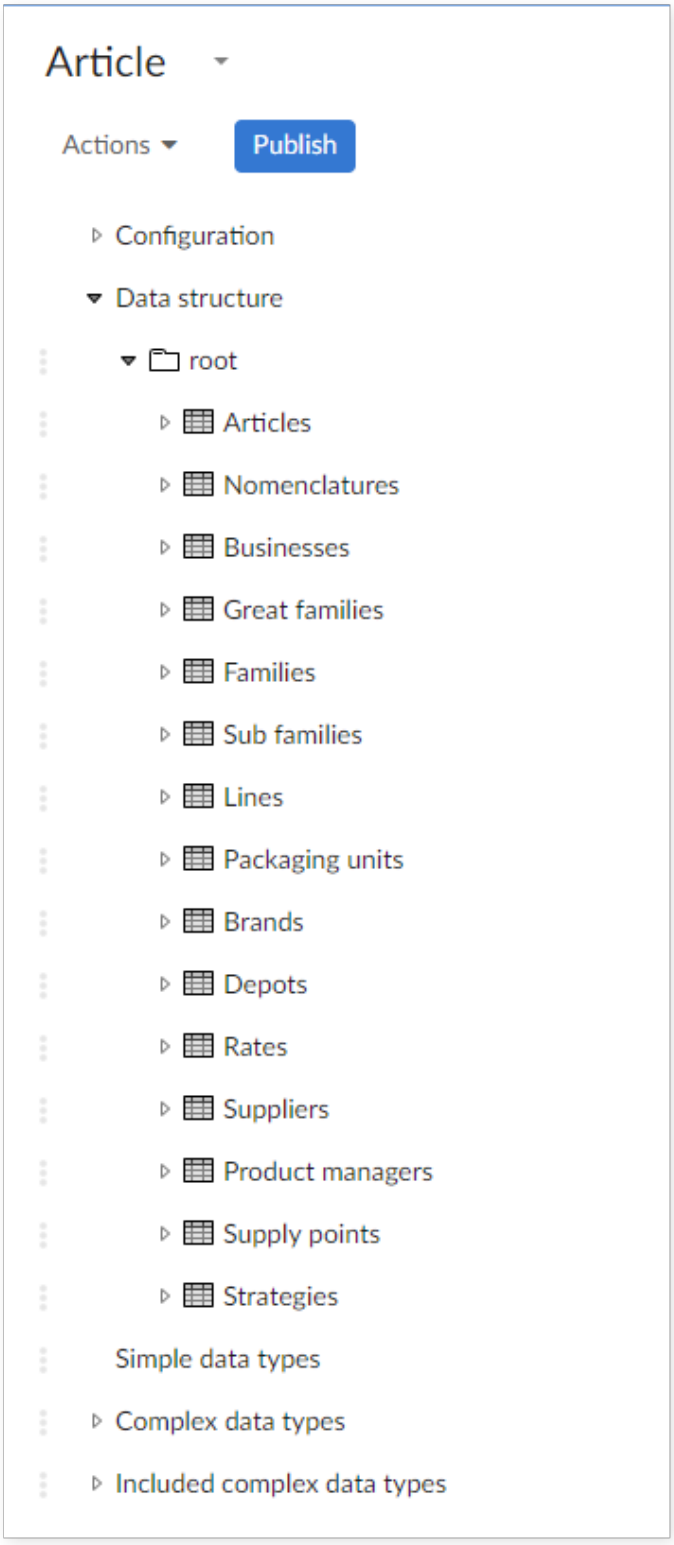


7.2 **Use case: Preventing record deletion when relationships exist**

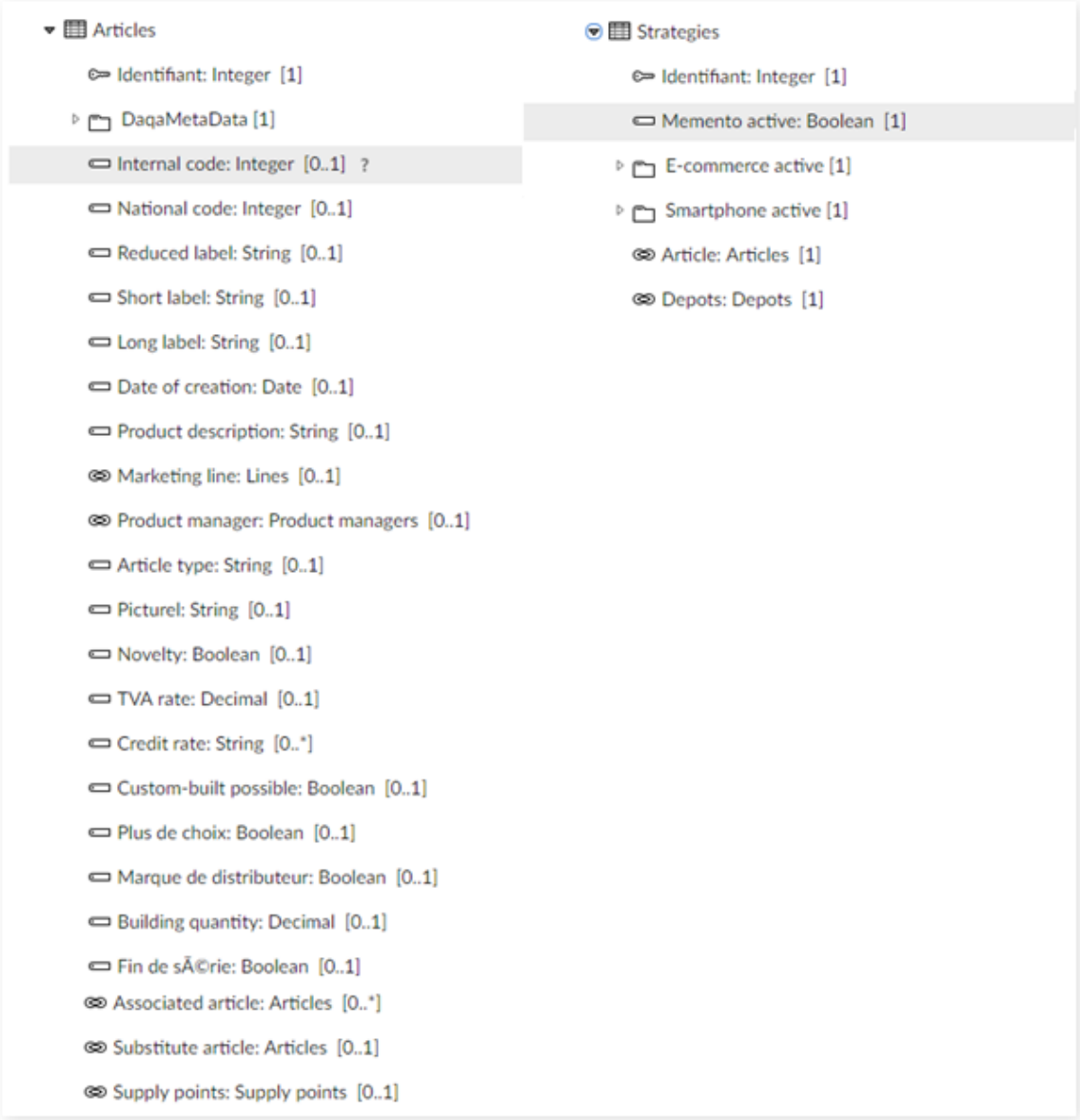
This use case uses the 'Article' data model (shown below) and demonstrates how you can prevent users from deleting a record when:

- A record has a child.

- A record has an association to a record in another table.



In this case, the rule's business logic relates to the 'Articles' and 'Strategies' tables.



Configuring the data model

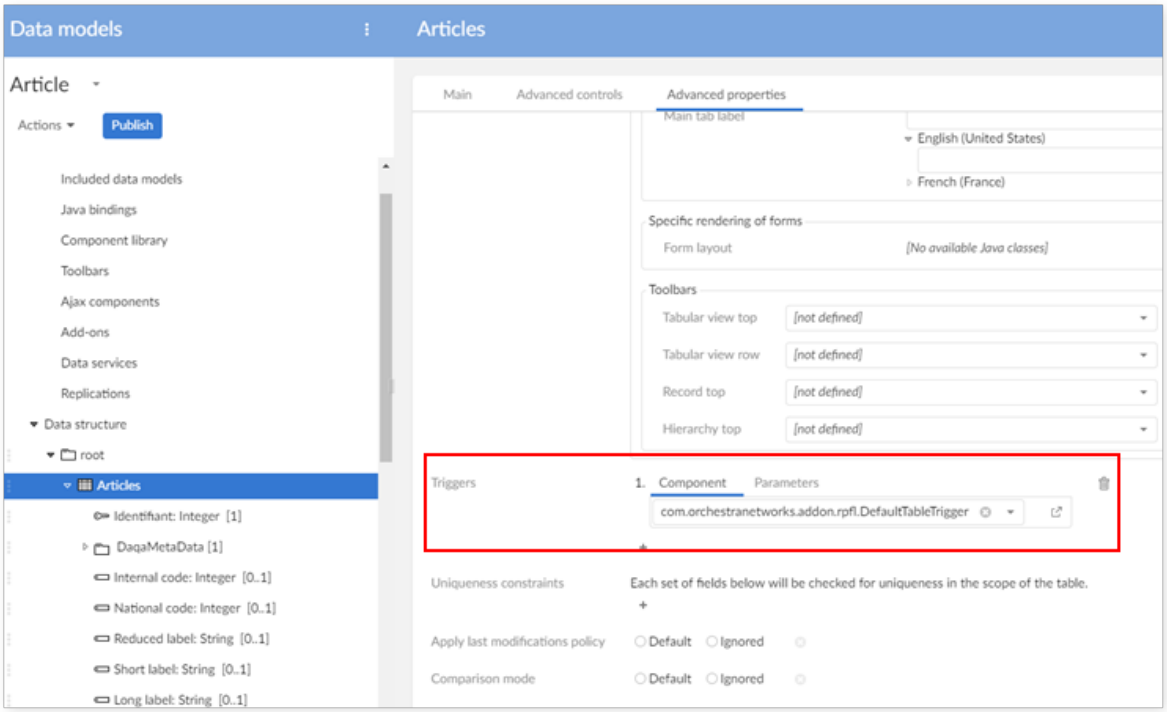
Complete the following steps to prepare the data model for rule execution:

- Add the classes shown in the following table to the data model's 'Configuration' → 'Component library'. These classes call the EBX® Rules Portfolio Add-on.

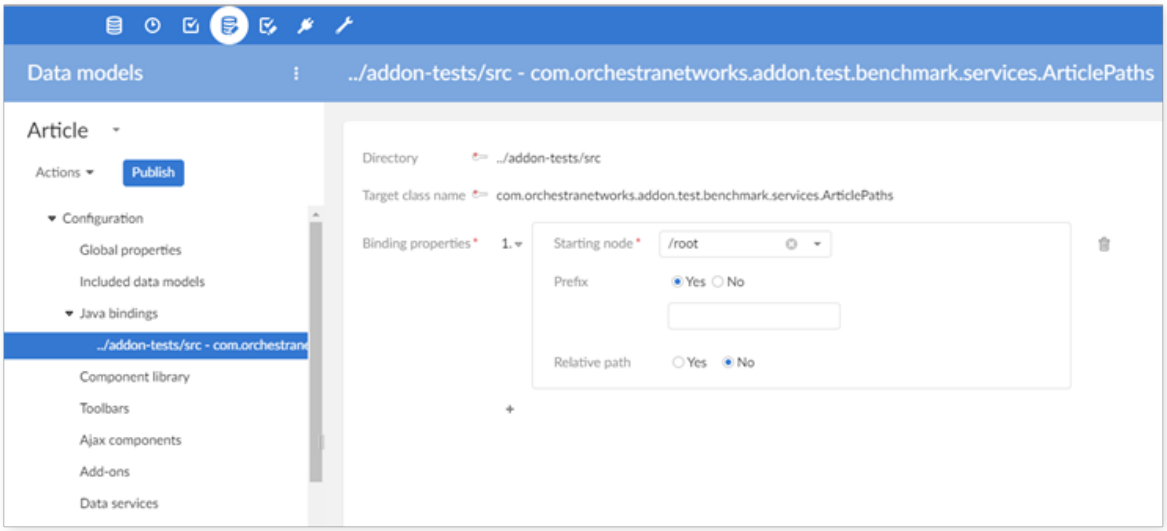
Class	Description
com.orchestranetworks.addon.rpfl.DefaultTableTrigger	Enables use of validation and automated rules.
com.orchestranetworks.addon.rpfl.DefaultConstraintOnTable	Enables manual validation rules.
com.orchestranetworks.addon.rpfl.DefaultSchemaExtension	Enables access permission rules.
com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission	Enables action permission rules if the service is declared in the schema. If the service is declared in the module.xml file, create a class to extend DefaultRulesServicePermission and add is to the module.xml services declaration.

- Navigate to 'Configuration' → 'Data model properties' and add the 'DefaultSchemaExtension' to the 'Model properties' → 'Special extensions' attribute.
- Select the data model's 'Articles' table and apply the following components:
  - On the 'Advanced controls' tab, add the 'DefaultConstraintOnTable' component, as shown below.

- • On the 'Advanced properties' tab, add the 'DefaultTableTrigger' component, as shown below.



- Use the 'Configuration' → 'Java bindings' to specify what Java types the model generates. This use case implements the configuration shown in the following image:



- You can publish the data model and create a data set.

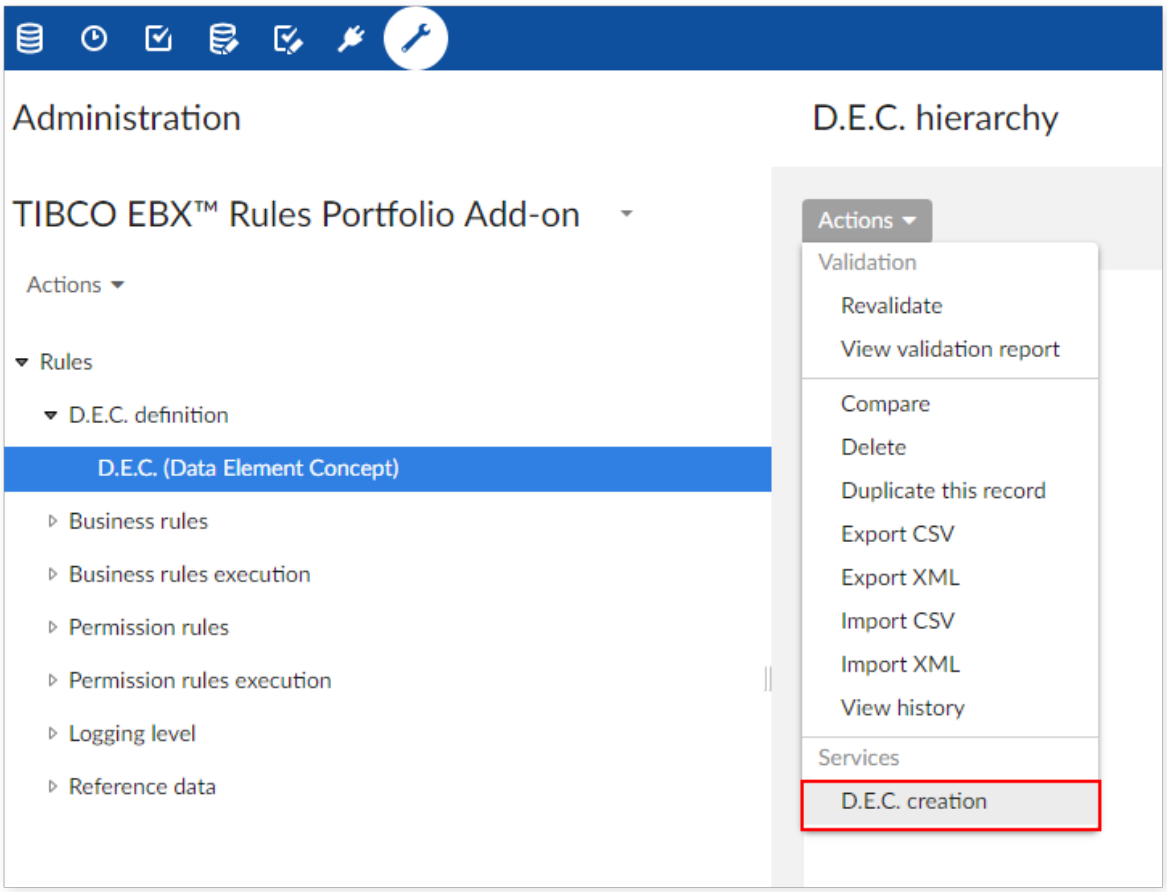
The next section describes how to create D.E.C.'s (Data Element Concepts) in the add-on.

Creating the D.E.C.s

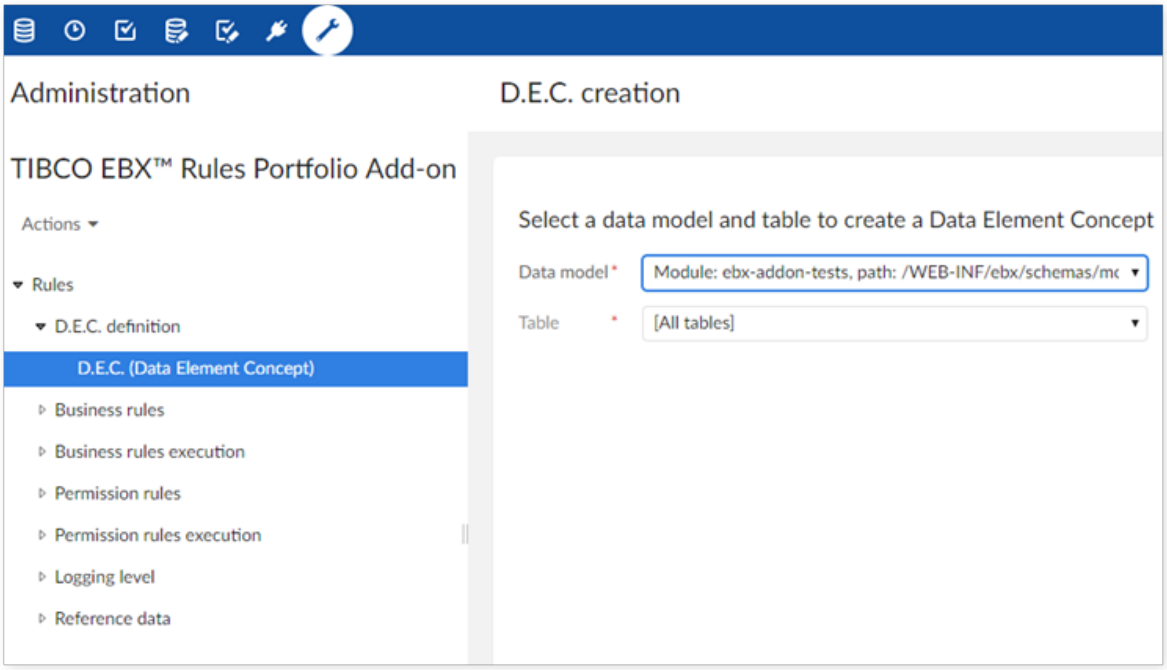
A D.E.C. associates a specific business concept, such as a table, or a field with a rule. You can add each D.E.C. manually, or use the 'D.E.C. creation' service to automatically create them. However, manual creation can easily lead to duplication and is not a recommended practice.

The following demonstrates using the service to auto-generate the D.E.C.s:

- Navigate to the 'Administration' tab → 'Business rules' → 'TIBCO EBX® Rules Portfolio Add-on' → 'D.E.C. definition' → 'D.E.C. by type table' view and from the 'Actions' menu select 'D.E.C. creation'.



- Enter the desired 'Data model' and 'Table' information and click 'Execute'.



Implementing the rule using Java

The requirements for this use case stem from the following two scenarios:

- An user tries to delete a record when it has a child record.
- An user tries to delete a record when it has an associated record.

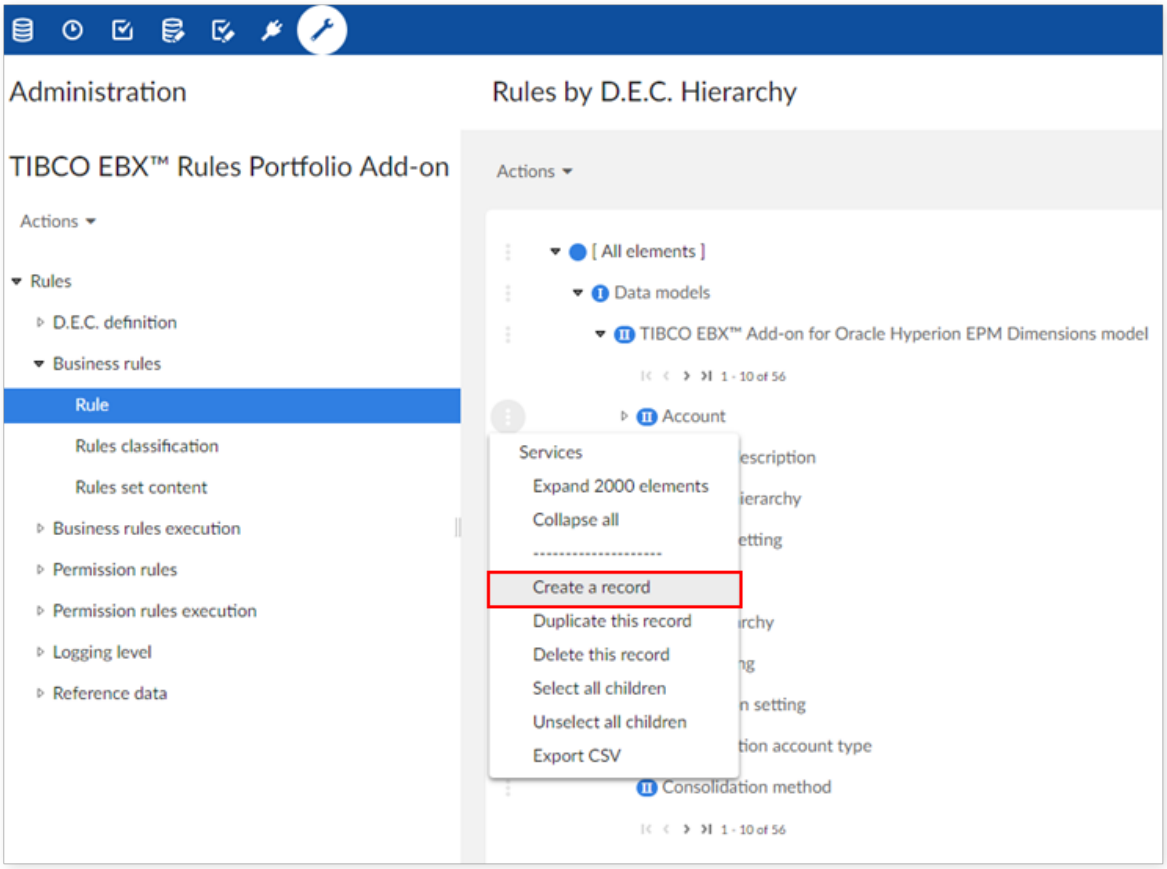
Each of the above scenarios requires its own validation type business rule, relies on a Java implementation and applies to the same data model table ('Articles' in this case). Each Java implementation contains two classes. One class defines the rule and the other defines how the rule executes. The following table outlines these two classes:

Class	Description
The rule definition class	This class has to implement the 'BusinessRuleDefinition' interface and defines the following rule information: • Label • Input • Output • Properties • Rule type • Implementation class
The rule implementation class	This class must implement the 'Rule' interface and it contains the logic to execute the rule.

Note that the following steps only demonstrate creating one of the two rules. Just repeat the same set of steps to add the second rule.

To create the business rules:

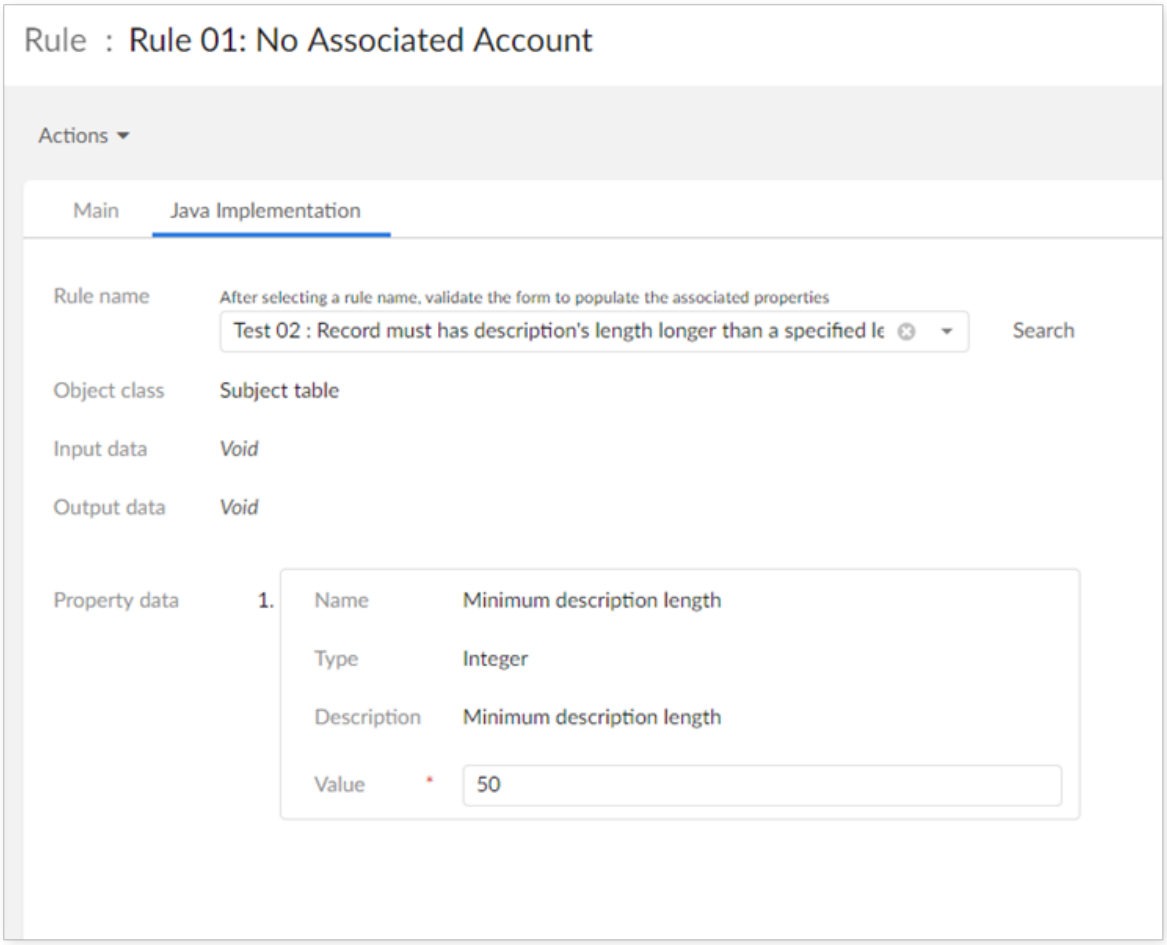
- Navigate to the 'Administration' tab → 'Business rules' → 'TIBCO EBX® Rules Portfolio Add-on' → 'Business rules' → 'Rule' table and 'Create a record' on the appropriate D.E.C. This table's view defaults to 'Rules by D.E.C.' and allows you to easily select the D.E.C. to which you want to add the rule.



- Fill in the required information. Properties of note are:
  - 'Implementation type' - Set this property to 'Use Java implementation' to use a Java implemented rule.
-
  - 'Type' - This property defines the rule type. For this use case, the type is set to 'Validation rule'.
  - 'Severity' - This property should be set to 'Error'.
  - 'Message' - Our first rule's message is set to "You cannot delete an article when it is substitute article." Our second rule's error message says, "You cannot delete an article if it has associated strategies."

After saving the record, the 'Java implementation' tab displays.

- Click on the 'Java implementation' tab and use the 'Rule name' drop-down to select the first business rule. The 'Rule name' property only populates with rules that have been registered with the add-on. See the API documentation for more information.

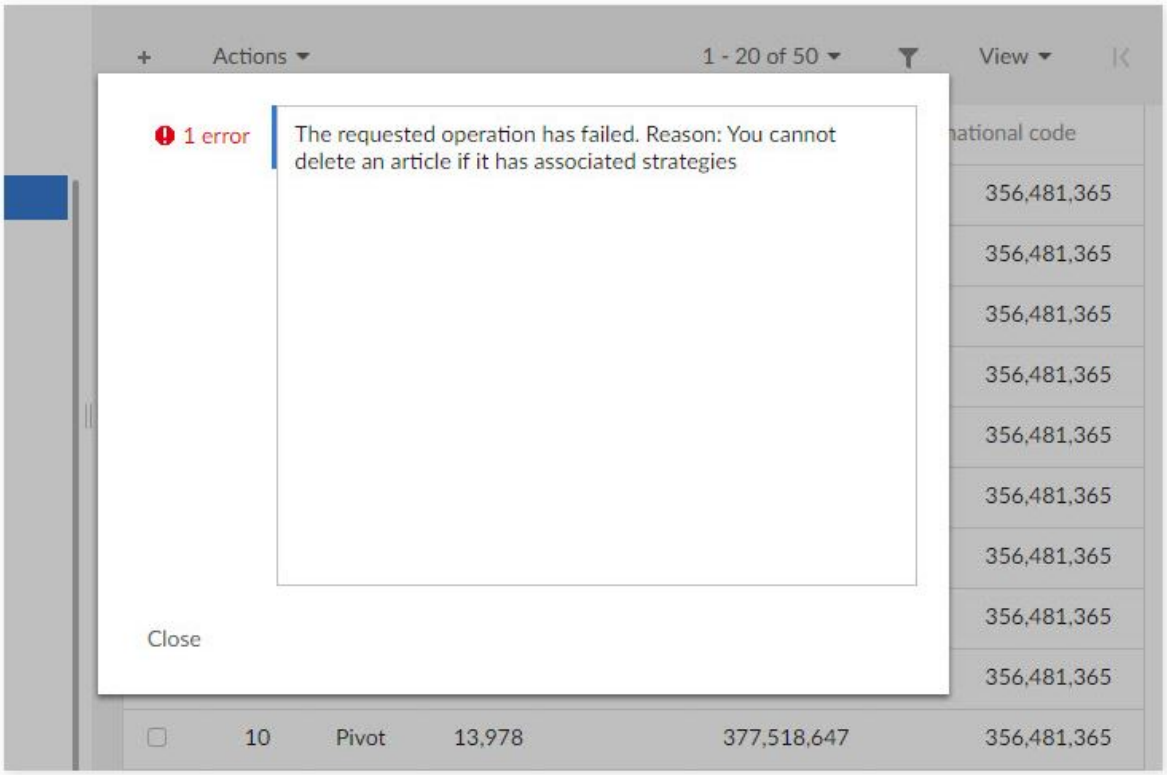


- To create a rule execution, navigate to 'Business rules execution' → 'Rule execution' and add a record to the same D.E.C. used for the rule definition. Among other things, a rule execution determines whether the associated rule is active, a rule validity time period and what type of event initiates rule execution. Ensure the required fields have values, including the 'Event' field which should be set to 'Before delete'. The 'Rule' field should be set to the business rule you want to apply this execution to. Save and close.

Repeat steps one through five to create the second business rule and rule execution. Once both rules are complete, you can publish the rules portfolio and test the rules.

- To publish the rules portfolio, open the 'Actions' menu at the 'TIBCO EBX® Rules Portfolio Add-on' data set level and select 'Publish rules portfolio'.
- Give the publication a unique name and be sure to select the data space containing the correct data model and data set. Click 'Publish'.

The following images show the error messages that display when we attempt to delete an 'Article' that is a substitute 'Article' and an 'Article' with a related 'Strategies', respectively.



Implementing the rule using a script

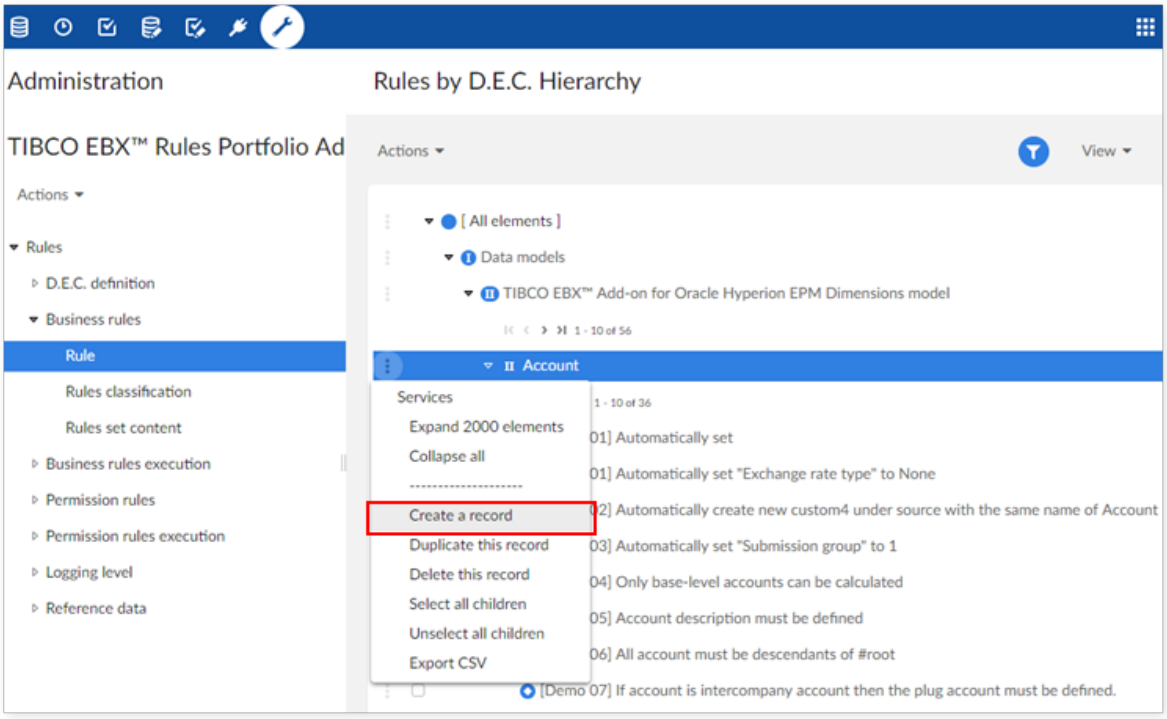
This section demonstrates using a script to meet the following requirements for this use case:

- A user tries to delete a record when it has a child record.
- A user tries to delete a record when it has an associated record.

The following steps provide script examples for both rules, but only demonstrate creating one of the two rules. Just repeat the same set of steps to add the second rule.

To create the business rules:

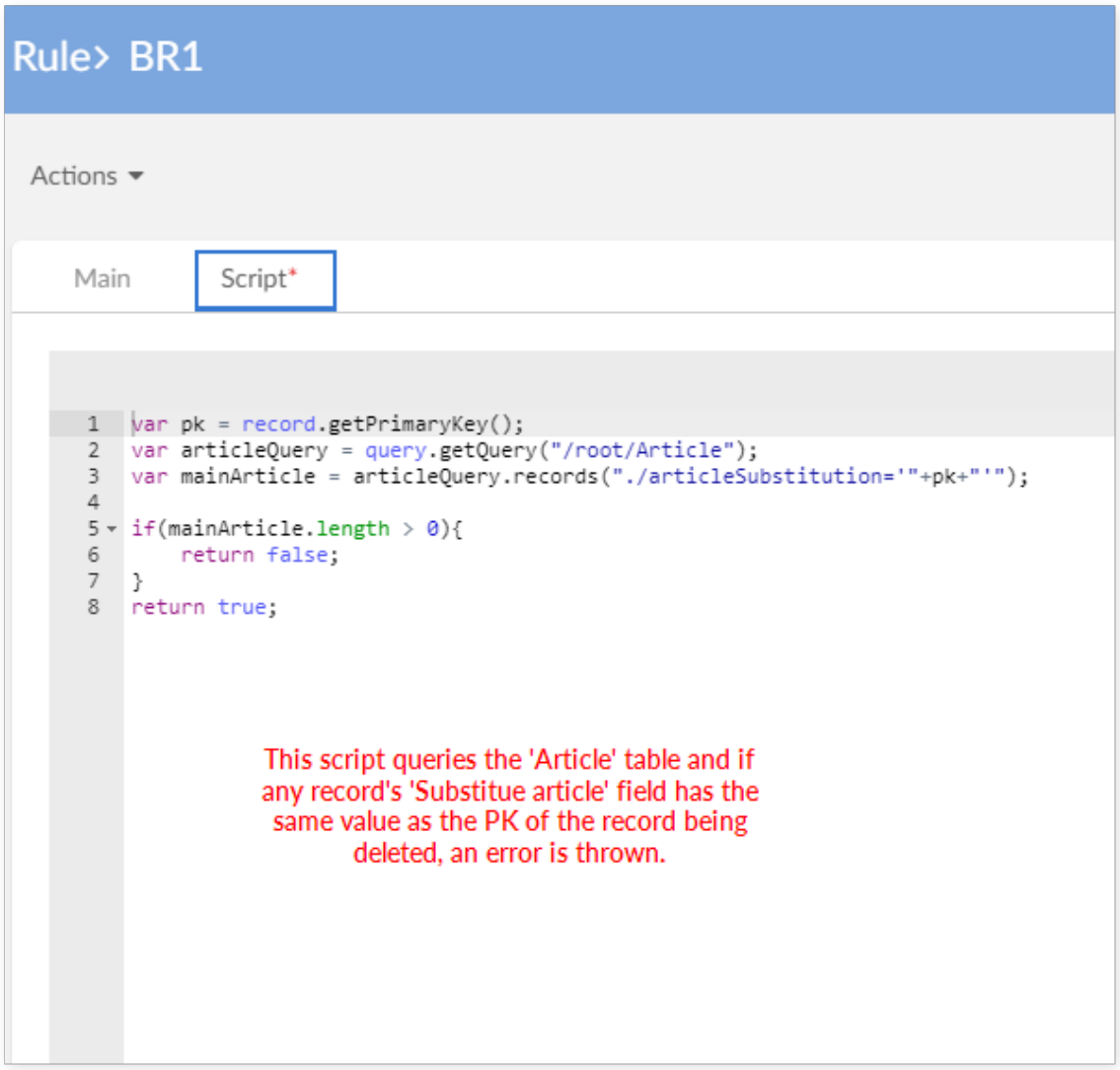
- Navigate to the 'Administration' tab → 'Business rules' → 'TIBCO EBX® Rules Portfolio Add-on' → 'Business rules' → 'Rule' table and 'Create a record' on the appropriate D.E.C. This table's view defaults to 'Rules by D.E.C.' and allows you to easily select the D.E.C. to which you want to add the rule.

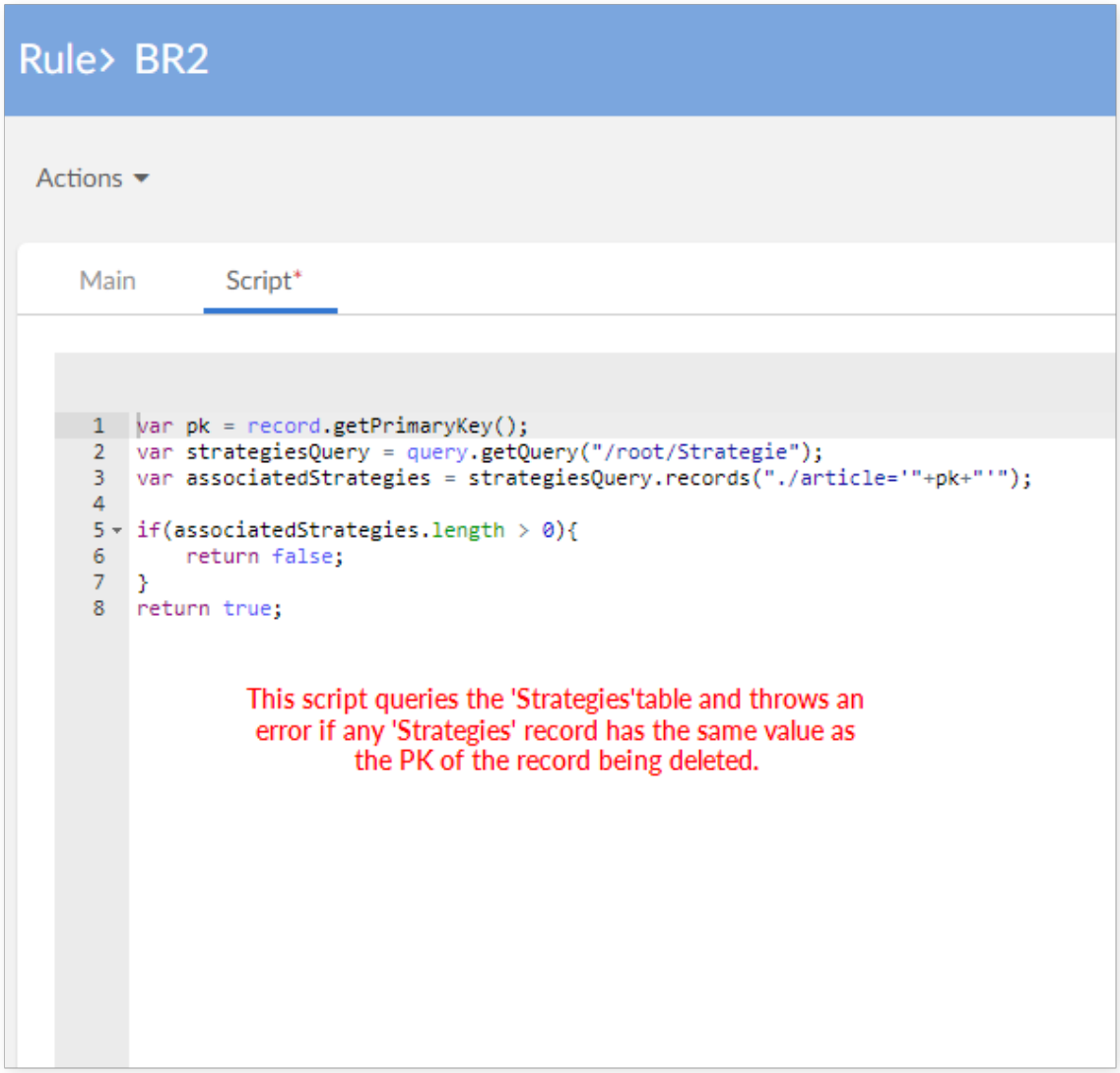


- Fill in the required information. Properties of note are shown below:
  - 'Implementation type' - Set this property to 'Use Script'. You can add the script to the 'Script' tab after saving.
  - 'Type' - This property defines the rule type. For this use case, the type is set to 'Validation rule'.
  - 'Severity' - This property should be set to 'Error'.
  - 'Message' - Our first rule's message is set to "You cannot delete an article when it is substitute article." Our second rule's error message says, "You cannot delete an entity if it has associated strategies."

After ensuring the above properties are correct, save the record and the 'Script' tab displays.

- Click on the 'Script' tab and add the appropriate script. The following two images show the script used for each rule. For a more in-depth explanation of how specific objects work, see the Scripting Language Guide in the online help.



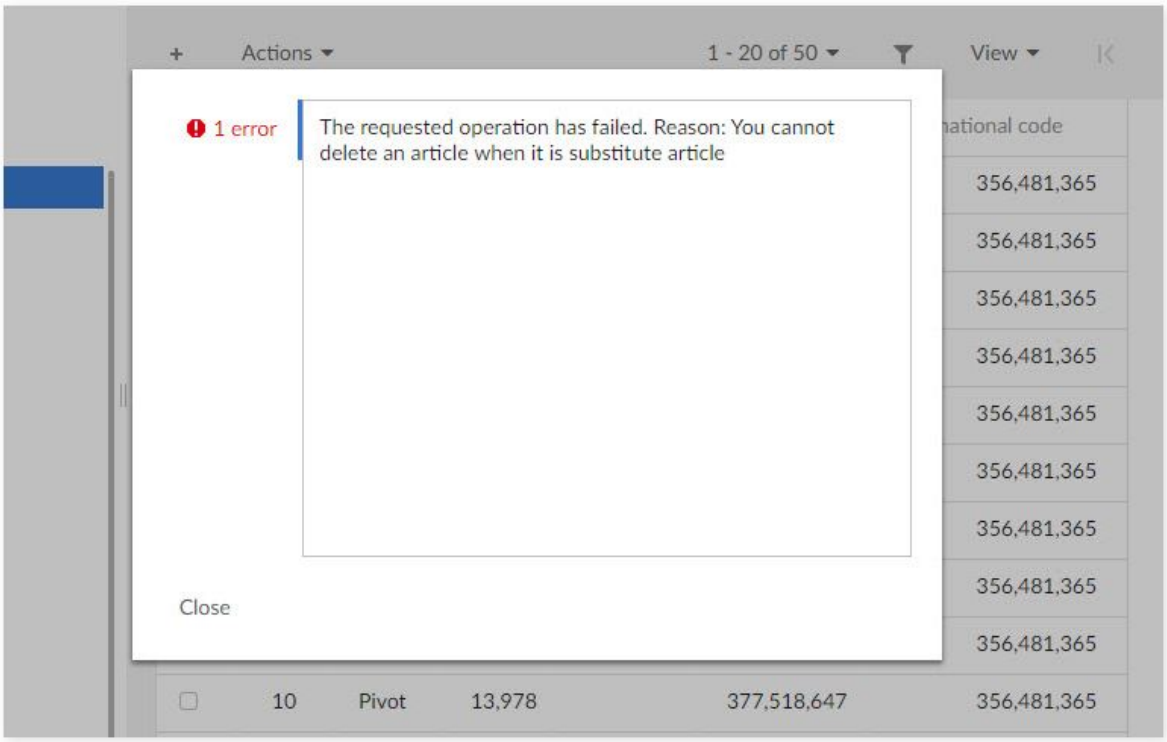


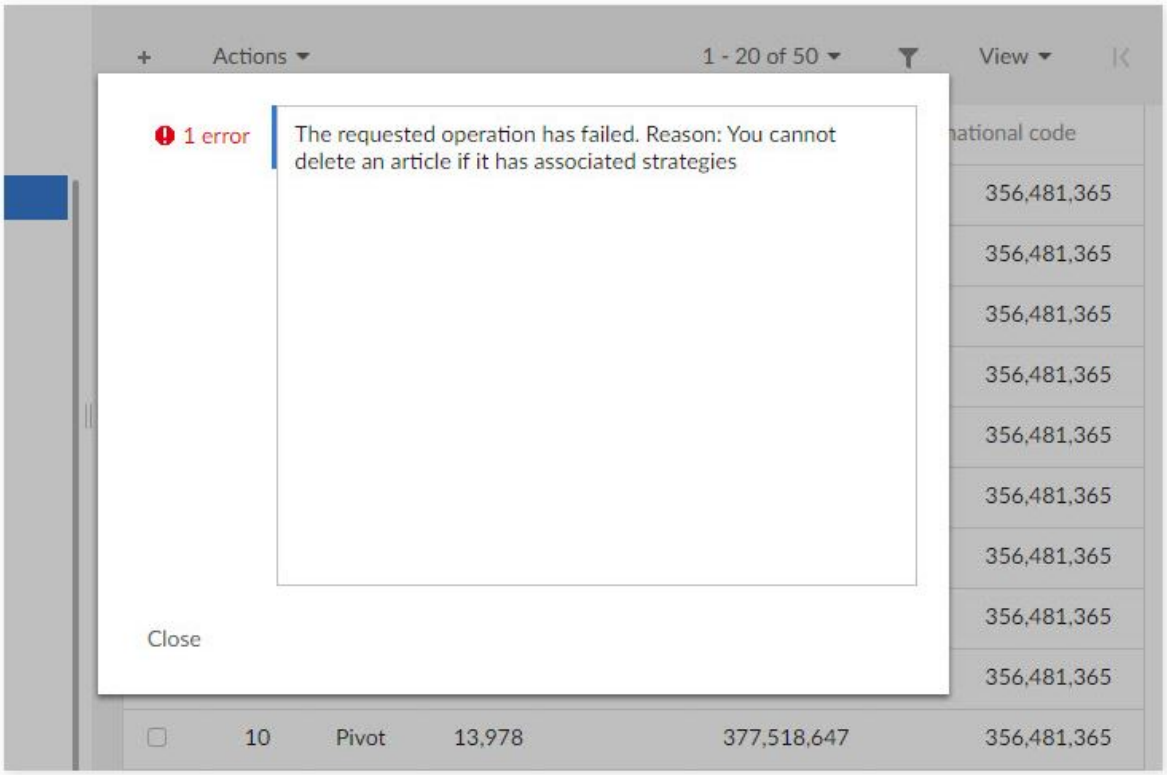
- To create a rule execution, navigate to 'Business rules execution' → 'Rule execution' and add a record to the same D.E.C. used for the rule definition. Among other things, a rule execution determines whether the associated rule is active, a rule validity time period and what type of event initiates rule execution. Ensure the required fields have values, including the 'Event' field which should be set to 'Before delete'. The 'Rule' field should be set to the business rule you want to apply this execution to. Save and close.

Steps one through four are repeated to create the second business rule and rule execution. Once both rules are complete, you can publish the rules portfolio and test the rules.

- To publish the rules portfolio, open the 'Actions' menu at the 'TIBCO EBX® Rules Portfolio Add-on' data set level and select 'Publish rules portfolio'.
- Give the publication a unique name and be sure to select the data space containing the correct data model and data set. Click 'Publish'.

The following images show the error messages that display when we attempt to delete a 'Article' with a child 'Article' and an 'Article' with a related 'Strategies', respectively.

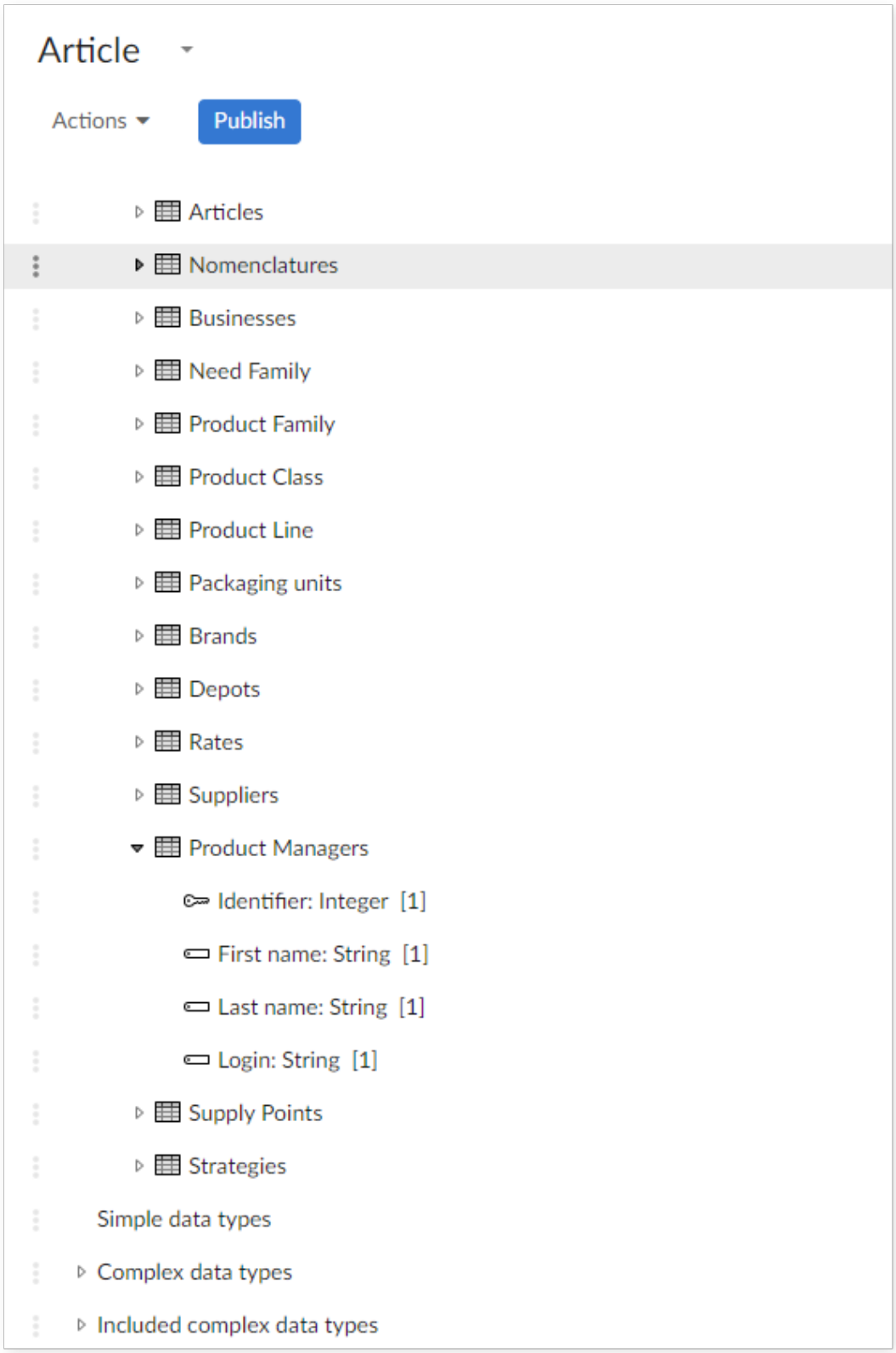




7.3 **Use case: Preventing a field from exceeding a value and auto-correcting**

This use case uses the previously shown 'Article' data model and applies sample rules to the 'Product managers' table. Data model configuration for add-on compatibility and creation of the D.E.C.s

followed the same processes as the previous use case. For more information about these processes, see the above sections 'Configuring the data model' and 'Creating the D.E.C.s'.



The next two topics show how to define a rule using Java and the built-in scripting language. Part of the rule definition includes a condition for rule execution, or 'Execution context', which defines rule requirements. When creating your own rules, not all situations require an 'Execution context'. In these cases you can simply remove the 'Execution context'. This rule enforces the following business requirement:

- If the 'Login' field's value does not have a default prefix, then the add-on automatically adds a default prefix when you manually execute the rule.
- Configuring the rule and execution conditions with Java.

Configuring a Java-based rule and execution conditions

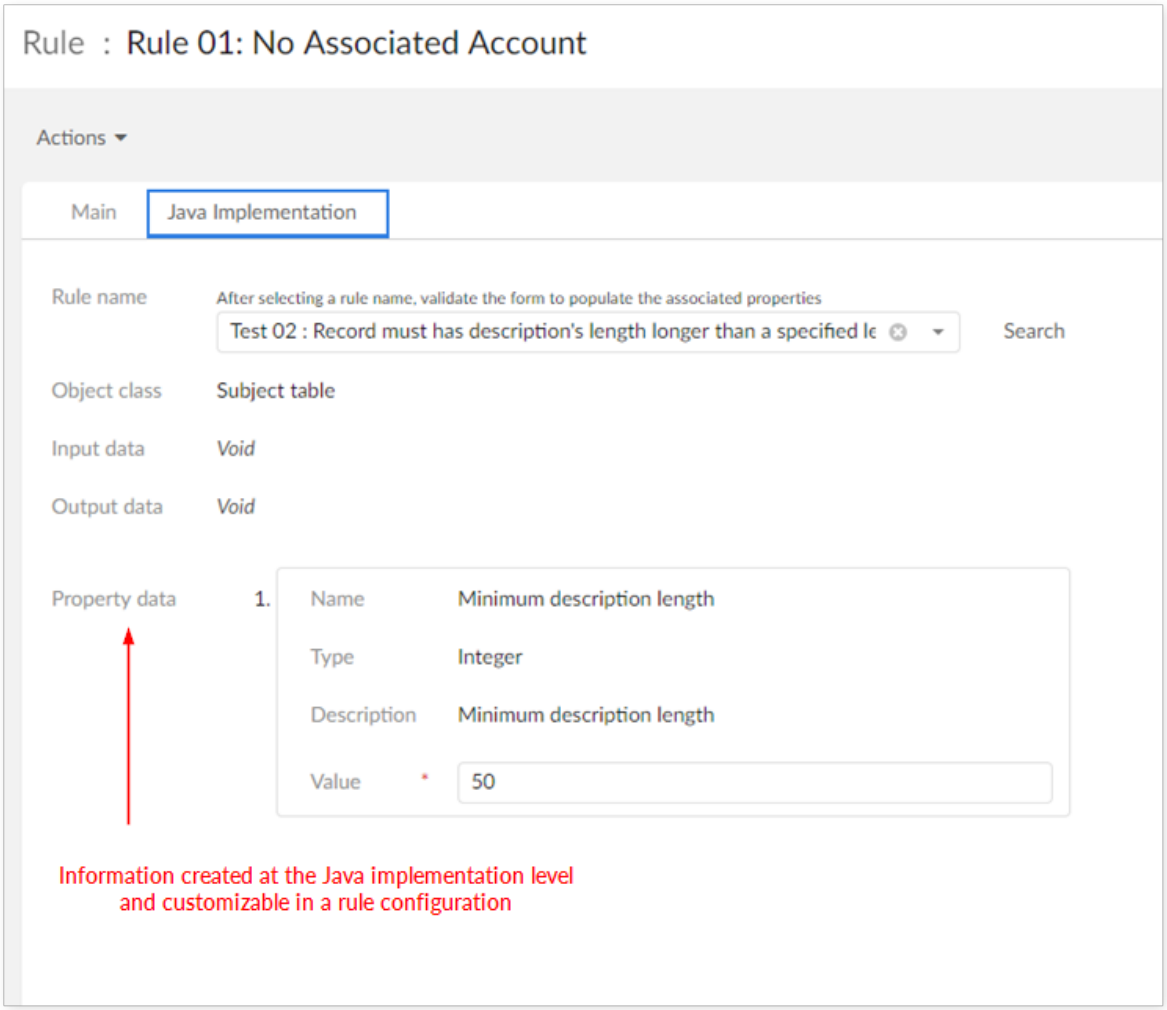
When relying on Java implementation, you select from a list of available rules previously registered with the add-on. These rules already contain programming logic required to analyze data conditions and pass a result to the add-on. Subsequently, an end-user responsible for 'Rule' and 'Rule execution' configuration does not need Java-specific programming knowledge to complete his task. If you are required to create the rules in Java and register them with the add-on, refer to the API documentation for more information.

The following steps demonstrate 'Rule' and 'Rule execution' configuration based on use case requirements:

- Navigate to the 'Administration' tab → 'Business rules' → 'TIBCO EBX® Rules Portfolio Add-on' → 'Business rules' → 'Rule' and create a new record under the 'Product managers' D.E.C.

  Note: If you have not set up a data model or created the required D.E.C.s, see the 'Configuring the data model' and 'Creating the D.E.C.s' sections in the previous use case.
- Enter the required information and click 'Save'. The 'Java implementation' tab displays. Properties of note:
  - 'Implementation type' - Set to 'Use Java implementation' to use a Java implemented rule.
  - 'Type' - This property is set to 'Table set rule'. This type of rule allows modifying existing data-one of the use case requirements. Keep in mind that you need to manually execute a 'Table set rule' rule using the 'Execute rules' service.
  - 'Severity' - Set to 'Error'.
  - In the 'Java implementation' tab, use the 'Rule name' field to select the appropriate rule class and click 'Save'. After saving, the tab displays the 'Property data' defined in the Java class. Note that the 'Value' property, currently set to 'PM', is editable. This function allows you to simply change the value to reuse this rule for other business requirements-without additional software development. You can design any type of 'Property data' at the Java implementation

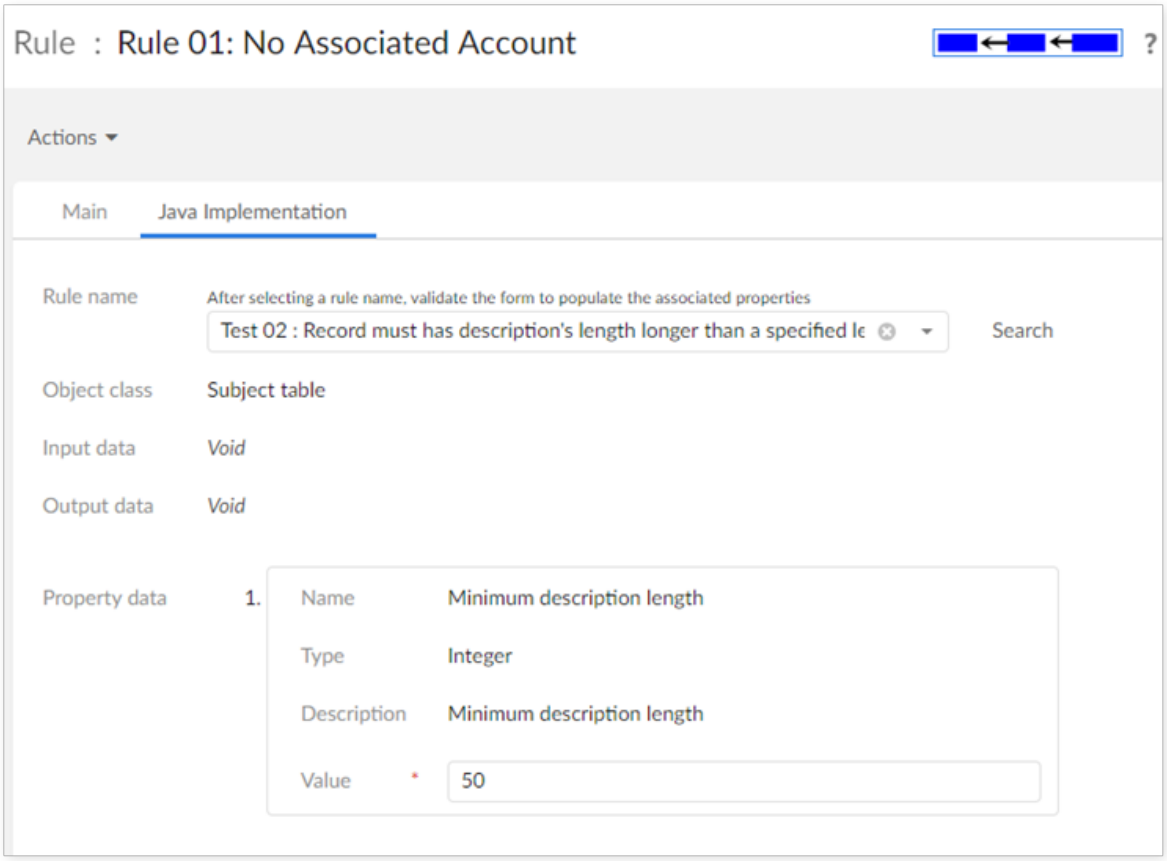
level and customize the values while configuring a rule. Refer to the Java doc for more information.



- Navigate to 'Business rules execution' → 'Rule execution' and create a new record under the 'Product managers' table D.E.C. This is where you define when rule execution occurs and the 'Execution context' that specifies conditions for rule execution. Notice the following properties:
  - 'Rule' - The 'Business rule' that these execution properties apply to.
  - 'Execution context' - This property allows you to specify a condition for rule execution using either Java, or a script. This example uses Java. However, not all scenarios require an 'Execution context'. In these types of cases you can set this property to 'No context'.
  - 'Event' - This property specifies when rule execution takes place. This use case implements a 'Table set rule' and requires you to run the 'Execute rules' service.

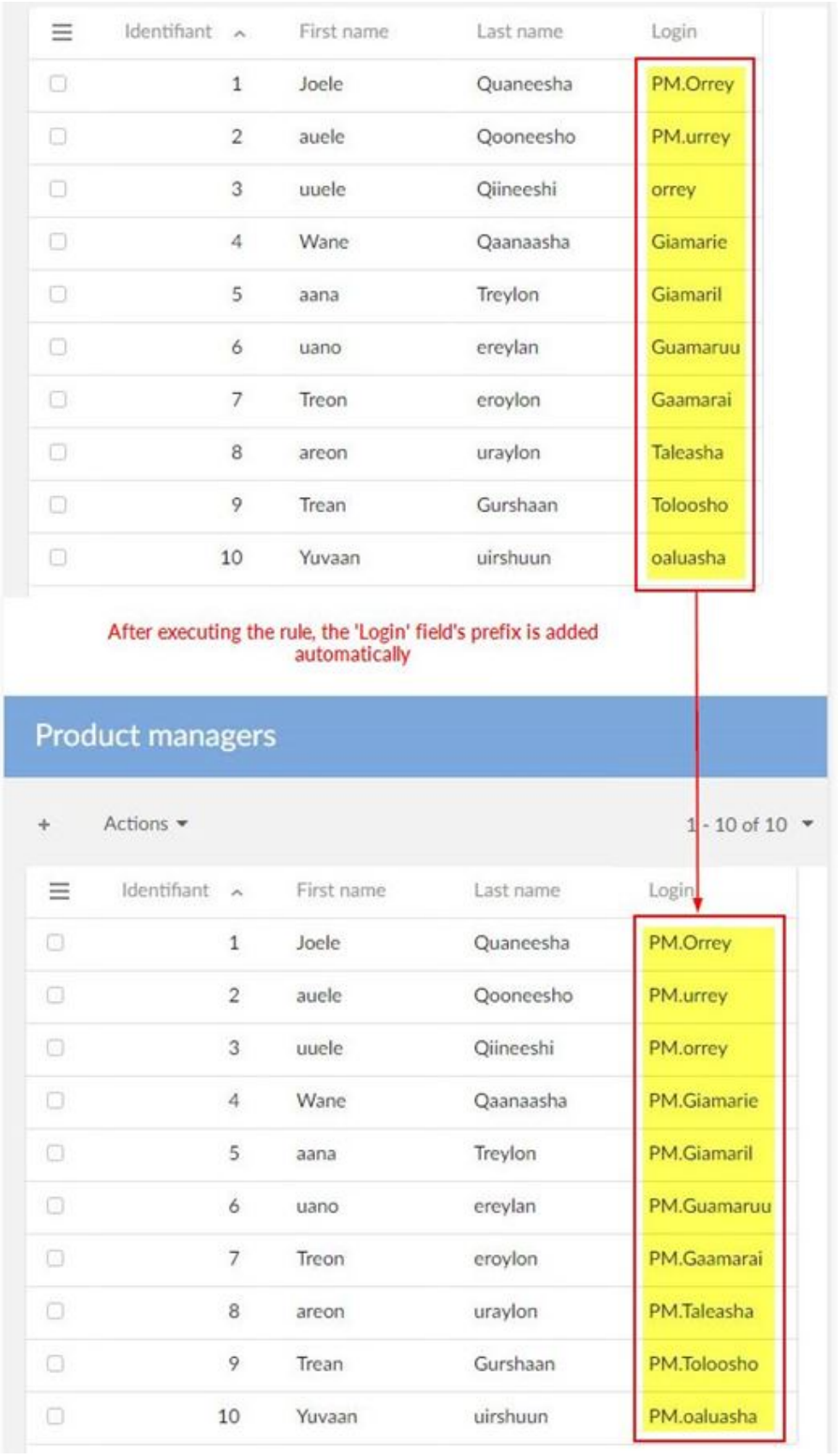
After entering the required information, click 'Save' and the 'Java implementation' tab displays.

- Use the 'Rule name' property to select the desired rule used for this 'Execution context'. 'Save and close'.



- Open the 'TIBCO EBX® Rules Portfolio Add-on' in 'Actions' drop-down menu and select 'Publish rules portfolio'. This creates the add-on's configuration snapshot as the execution environment for your rule. Enter the required information, including snapshot name and description, and the data space(s) to apply to.

- Return to the data set and from the 'Actions' menu, select 'Rules Portfolio' → 'Execute rules'. The following image shows the rule execution result.

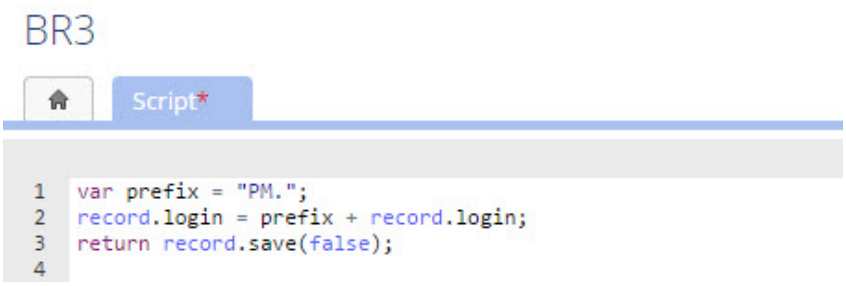

Identifiant
First name
Last name
Login
1 Joele Quaneesha PM.Orrey
2 auele Qooneesho PM.urrey
3 uuele Qiineeshi orrey
4 Wane Qaanaasha Giamarie
5 aana Treylon Giamaril
6 uano ereylan Guamaruu
7 Treon eroylon Gaamarai
8 areon uraylon Taleasha
9 Trean Gurshaan Toloosho
10 Yuvaan uirshuun oaluasha
After executing the rule, the 'Login' field's prefix is added automatically
Product managers
Actions
1 - 10 of 10
Identifiant
First name
Last name
Login
1 Joele Quaneesha PM.Orrey
2 auele Qooneesho PM.urrey
3 uuele Qiineeshi PM.orrey
4 Wane Qaanaasha PM.Giamarie
5 aana Treylon PM.Giamaril
6 uano ereylan PM.Guamaruu
7 Treon eroylon PM.Gaamarai
8 areon uraylon PM.Taleasha
9 Trean Gurshaan PM.Toloosho
10 Yuvaan uirshuun PM.oaluasha

Configuring a script-based rule and execution conditions

When you use a script to define rule and execution logic, it requires the end-user to have some background programming knowledge. However, such a user can rapidly develop and implement rules by leveraging the add-on's scripting feature. This section observes the same use case conditions as the previous section, except it uses a script (examples included) to meet the requirements.

To create a rule that checks the 'Login' field value and add prefix automatically if the value does not have prefix:

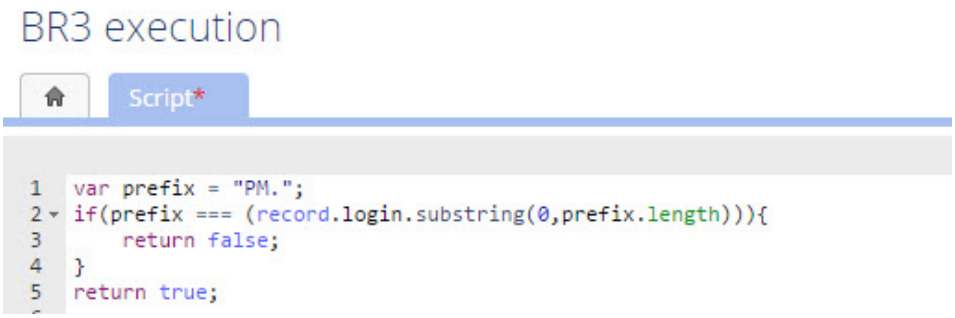
- Navigate to the 'Administration' tab → 'Business rules' → 'TIBCO EBX® Rules Portfolio Add-on' → 'Business rules' → 'Rule' and create a new record under the 'Product managers' D.E.C. Note: If you have not set up a data model or created the required D.E.C.s, see the 'Configuring the data model' and 'Creating the D.E.C.s' sections in the previous use case.
- Enter the required information and click 'Save'. The 'Script' tab displays. Properties of note are:
  - 'Implementation type' - Set to 'Use Script'.
  - 'Type' - This property is set to 'Table set rule'. This type of rule allows modifying existing data-one of the use case requirements. Keep in mind that you need to manually execute a 'Table set rule' rule using the 'Execute rules' service.
  - 'Severity' - Set to 'Error'.
- The following image shows the information entered in the 'Script' tab.



- Navigate to 'Business rules execution' → 'Rule execution' and create a new record under the 'Product managers' table D.E.C. This is where you determine when rule execution occurs and the 'Execution context' that specifies conditions for rule execution. Notice the following properties:
  - 'Rule' - The 'Business rule' that these execution properties apply to.
  - 'Execution context' - This property allows you to specify a condition for rule execution using either Java, or a script-as used in this example. However, not all scenarios require an 'Execution context'. In these types of cases you can set this property to 'No context'.
  - 'Event' - This property specifies when rule execution takes place. This use case implements a 'Table set rule' and requires you to run the 'Execute rules' service.

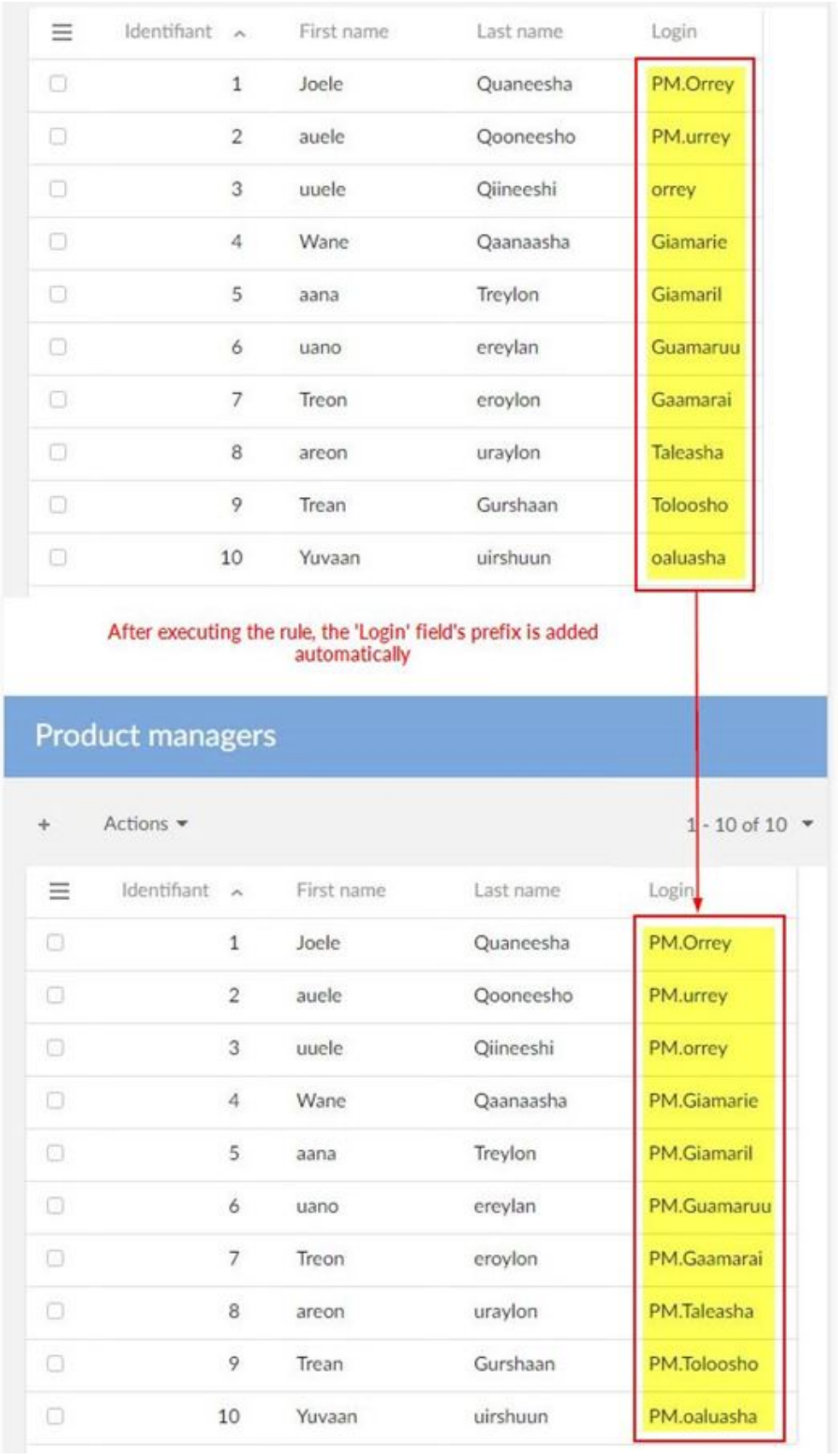
After entering the required information, click 'Save' and the 'Script' tab displays.

- The following image shows the information entered in the 'Script' tab.



- Open the 'TIBCO EBX® Rules Portfolio Add-on' in 'Actions' drop-down menu and select 'Publish rules portfolio'. This creates the add-on's configuration snapshot as the execution environment for your rule. Enter the required information, including snapshot name and description, and the data space(s) to apply to.

- Return to the data set and from the 'Actions' menu, select 'Rules Portfolio' → 'Execute rules'. As shown below, 'PM' prefix is added automatically in all 'Login' field values.


Identifiant
First name
Last name
Login
1 Joele Quaneesha PM.Orrey
2 auele Qooneesho PM.urrey
3 uuele Qiineeshi orrey
4 Wane Qaanaasha Giamarie
5 aana Treylon Giamaril
6 uano ereylan Guamaruu
7 Treon eroylon Gaamarai
8 areon uraylon Taleasha
9 Trean Gurshaan Toloosho
10 Yuvaan uirshuun oaluasha
After executing the rule, the 'Login' field's prefix is added automatically
Product managers
Actions
1 - 10 of 10
Identifiant
First name
Last name
Login
1 Joele Quaneesha PM.Orrey
2 auele Qooneesho PM.urrey
3 uuele Qiineeshi PM.orrey
4 Wane Qaanaasha PM.Giamarie
5 aana Treylon PM.Giamaril
6 uano ereylan PM.Guamaruu
7 Treon eroylon PM.Gaamarai
8 areon uraylon PM.Taleasha
9 Trean Gurshaan PM.Toloosho
10 Yuvaan uirshuun PM.oaluasha

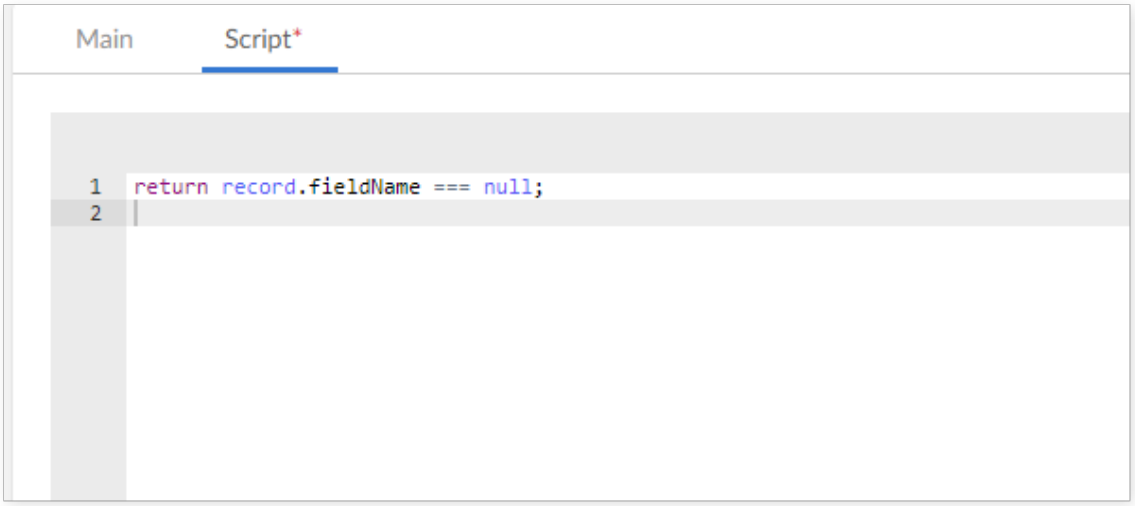
7.4 Use case: Check empty field with predefined assertion rules and the execution condition

Certain situations can lend themselves to using pre-built assertion rules to create new rules. These pre-built rules are script-based and alleviate the need to hand-code Java. The following lists the available pre-built assertion rules (business rule):

- [ON] Assertion true (Manual validation rule, Validation rule and Table set rule)
- [ON] Assertion false (Manual validation rule, Validation rule and Table set rule)

For example, to check if field (F) in a table (T) is empty, configure the following rule execution:

- Create a rule definition on (T) with the pre-built assertion rule '[ON] Assertion false (Manual validation)'. This rule systematically returns a false value.
- Configure a 'Rule execution' with scripting code to check that (F) is empty.



During execution, when the condition of execution is validated (the table field is empty), the rule executes and returns a value of 'False' which logs an error in the validation report. If the field is not empty, then the rule is not executed and no error is logged.

You can extend this type of configuration using a script to implement any execution context.

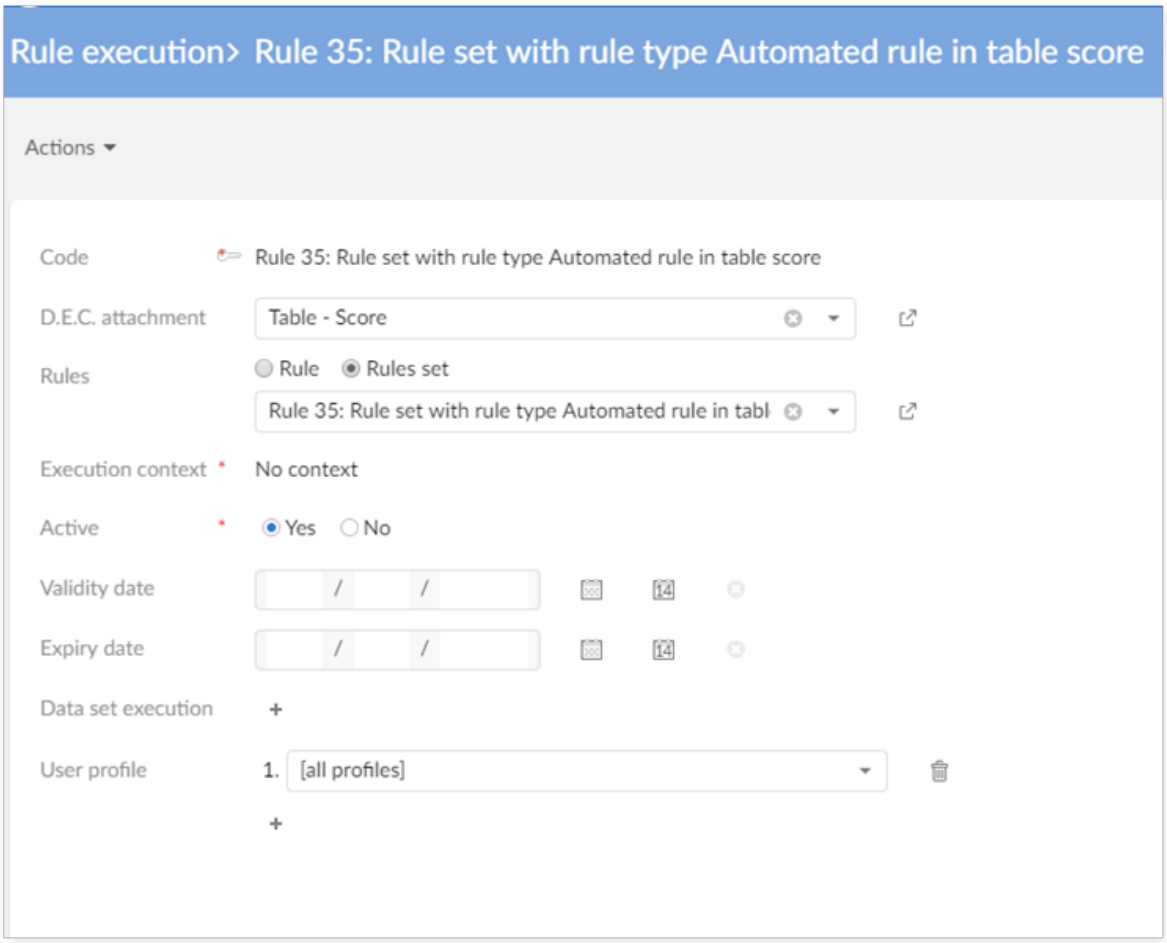
7.5 Use case: Set table permisison for a specific user using predefined assertion rules and execution conditions

The predefined assertion rules also allow you to set permisison on a data set, table, field and record. The available pre-built assertion rules (permission rules):

- [ON] Hidden the data set, table, field, or record.
- [ON] Permission read-only for the data set, table, or field/record.
- [ON] Permission read-write for the data set, table, field, or record.

For example, to set table (T) hidden only for a specific user:

- Create a rule definition on (T) with the pre-built assertion rule '[ON] Hidden the table'. This rule systematically hidden the table (T).
- Configure a 'Rule execution' with the value on 'User profile' belong to the user applied by your rule.



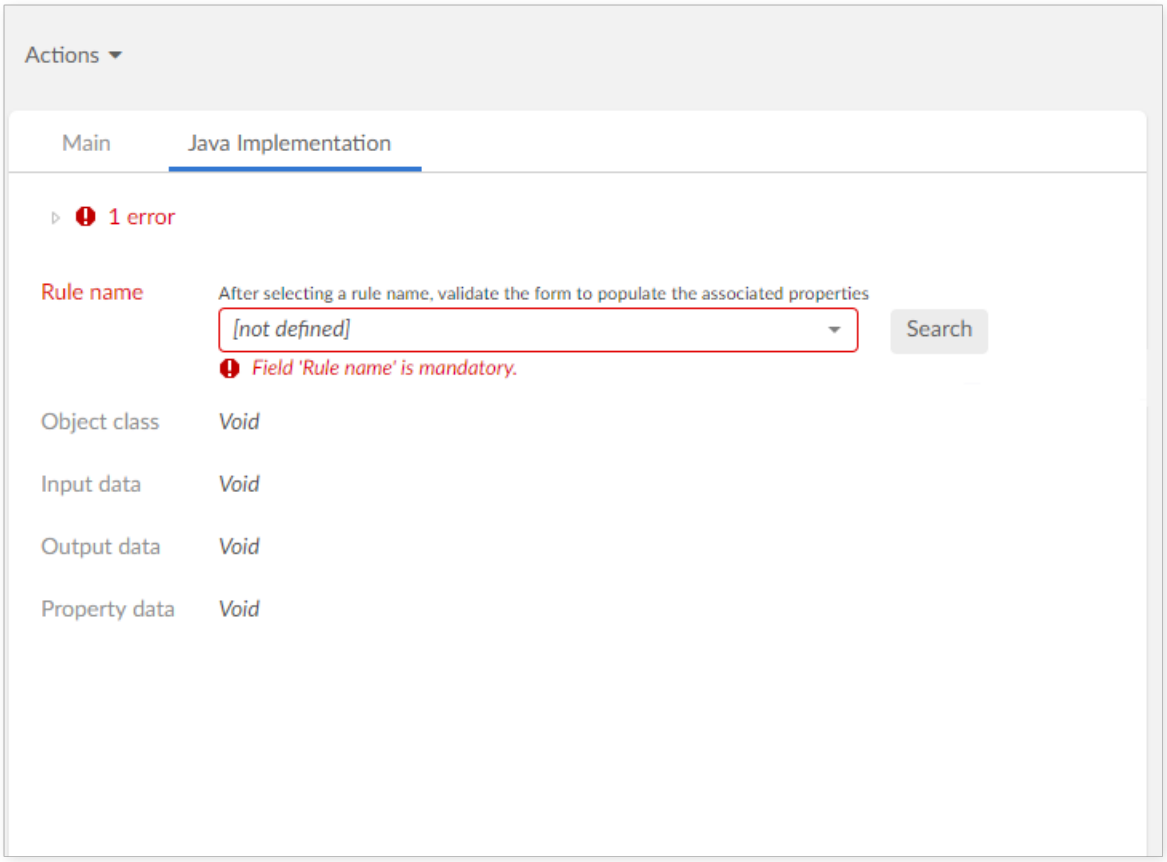
During execution, when the condition of execution is validated (the user profile is correct), the rule executes and set hidden permission of table (T). If the user profile is not correct, then the rule is not executed and the user still has read-writer permission on table (T).

7.6 Use case: Finding an existing Java implementation

When you have many Java implementations - whether execution contexts, or rules - it can be difficult to remember their names. Without the exact name, you may not be able to locate the implementation of interest. To locate an existing implementation you can search for it by defining criteria in the search dialog.

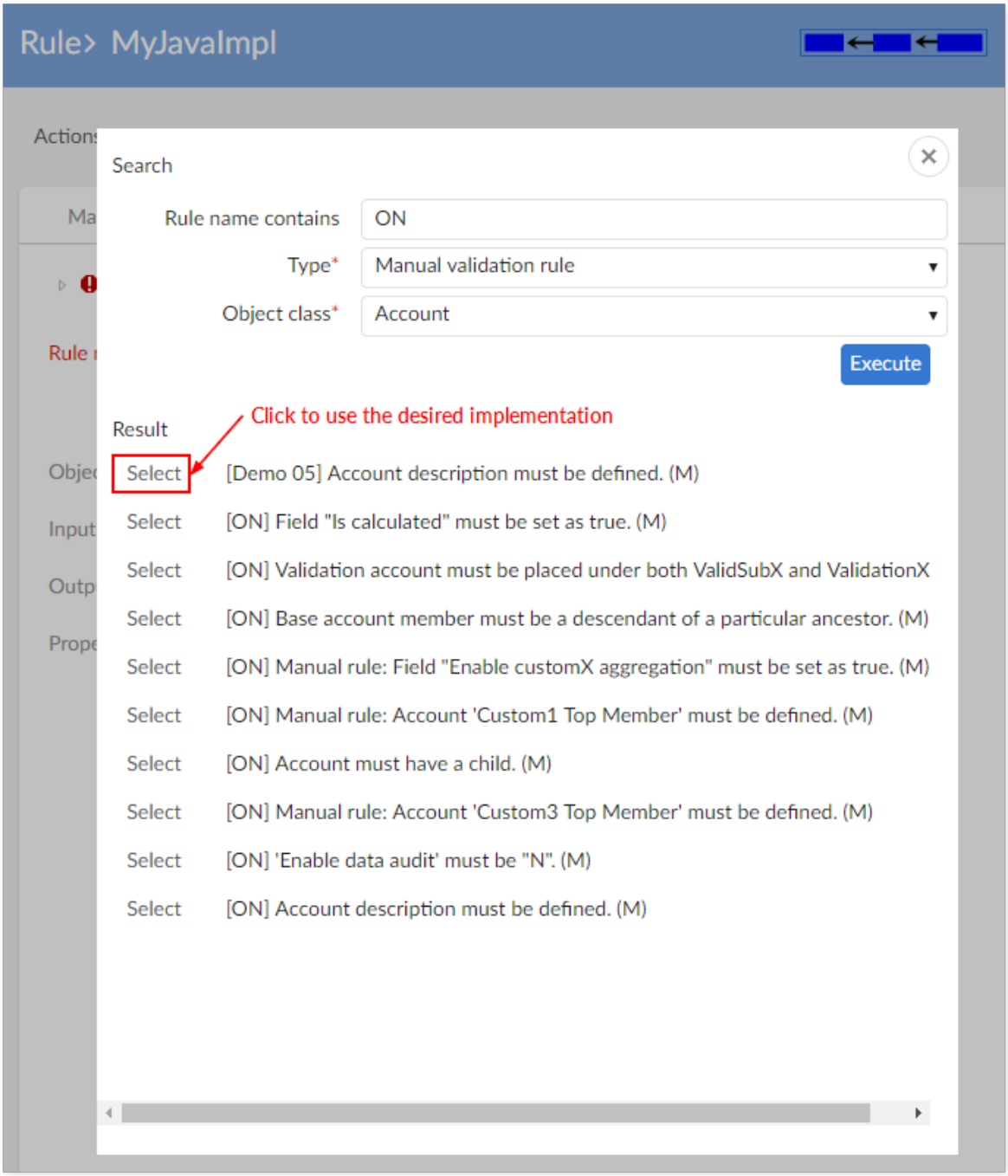
- Create a new, or open an existing rule or execution context that uses a Java implementation.

- Next to the 'Rule name' field click 'Search'.



- In the search box that displays, specify the following (note that the options that display in the search box differ depending on where you start the search):
- Text contained in the implementation name can apply to all searches. If you are not sure of the name, or when entering a name returns limited results, you can use the type, scope and Object Class options to expand the search.
- The type of rule option applies to business rules. If you have indicated a value on the rule's home tab in the 'Type' property, the search will be limited to that type. Otherwise, you will be able to choose a type to search for.
- The permission scope option applies to permission rules. If you have indicated a value on the rule's home tab in the 'Scope' property, the search will be limited to that scope. Otherwise, you will be able to choose a scope to search for.
- The Object Class option is available on all search types. The value specified here searches for types of business objects this rule can be applied to.
- After entering your search criteria, click 'Execute'.

- The 'Result' heading contains the search results. Click 'Select' next to the implementation you want to use.

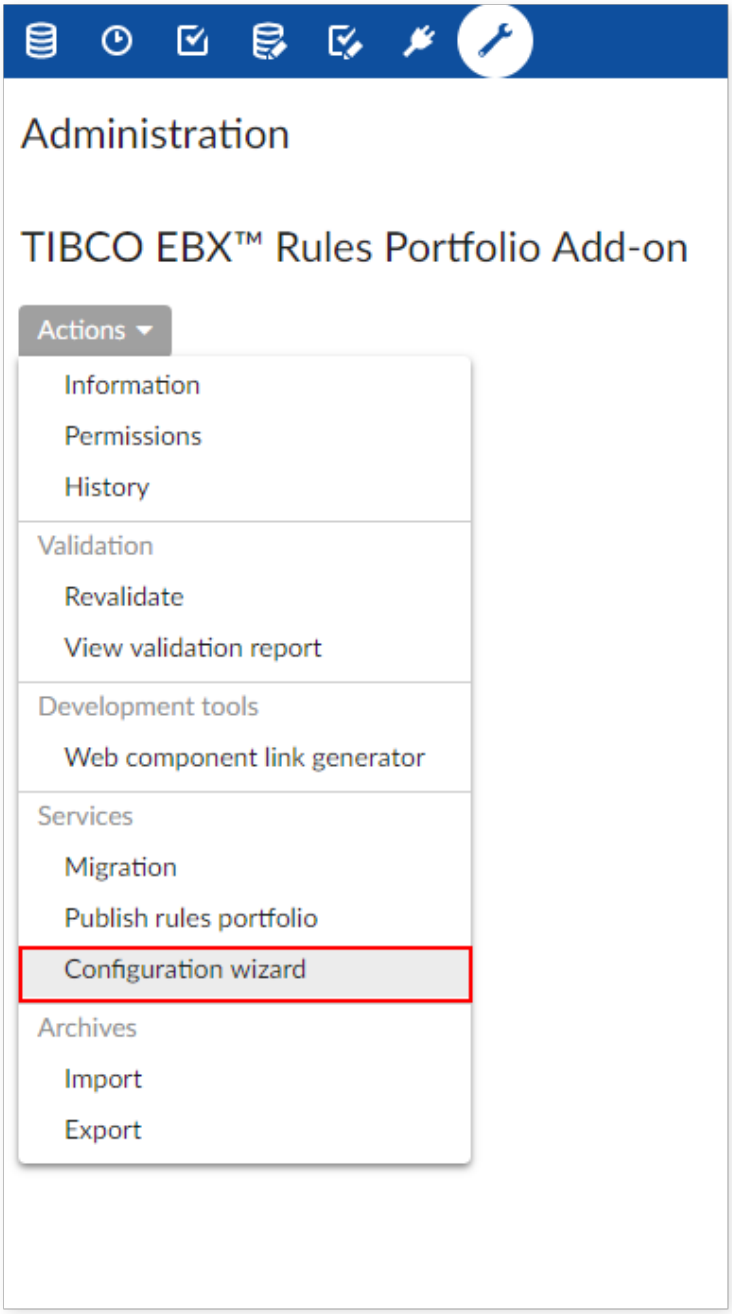


7.7 Use case: Using the wizard to configure a rule

The rule configuration wizard offers an alternative way for you to configure rules. This option is especially helpful for new users as it eliminates some of the complexity and manual configuration steps. If you have any questions about a particular option, just open the tooltip for additional information.

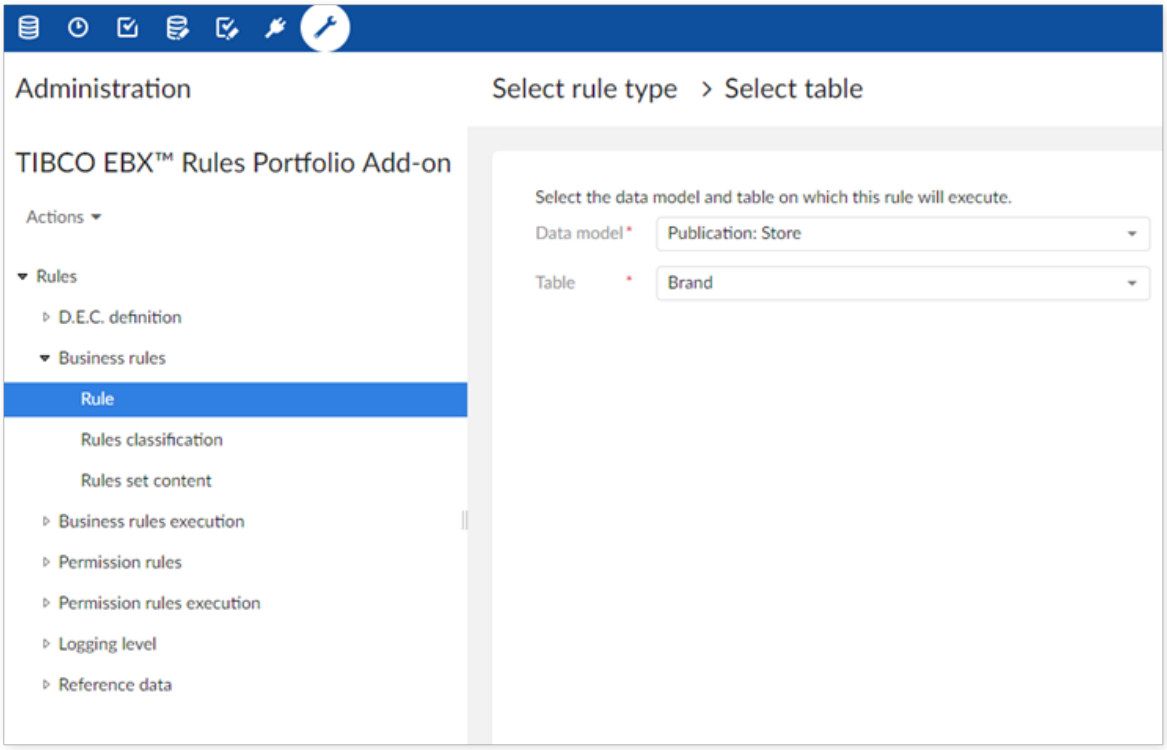
The following outlines the general flow of the wizard:

- Open the wizard from 'Administration' > 'Business rules' > 'TIBCO EBX® Rules Portfolio Add-on' > 'Actions' > 'Configuration wizard'.

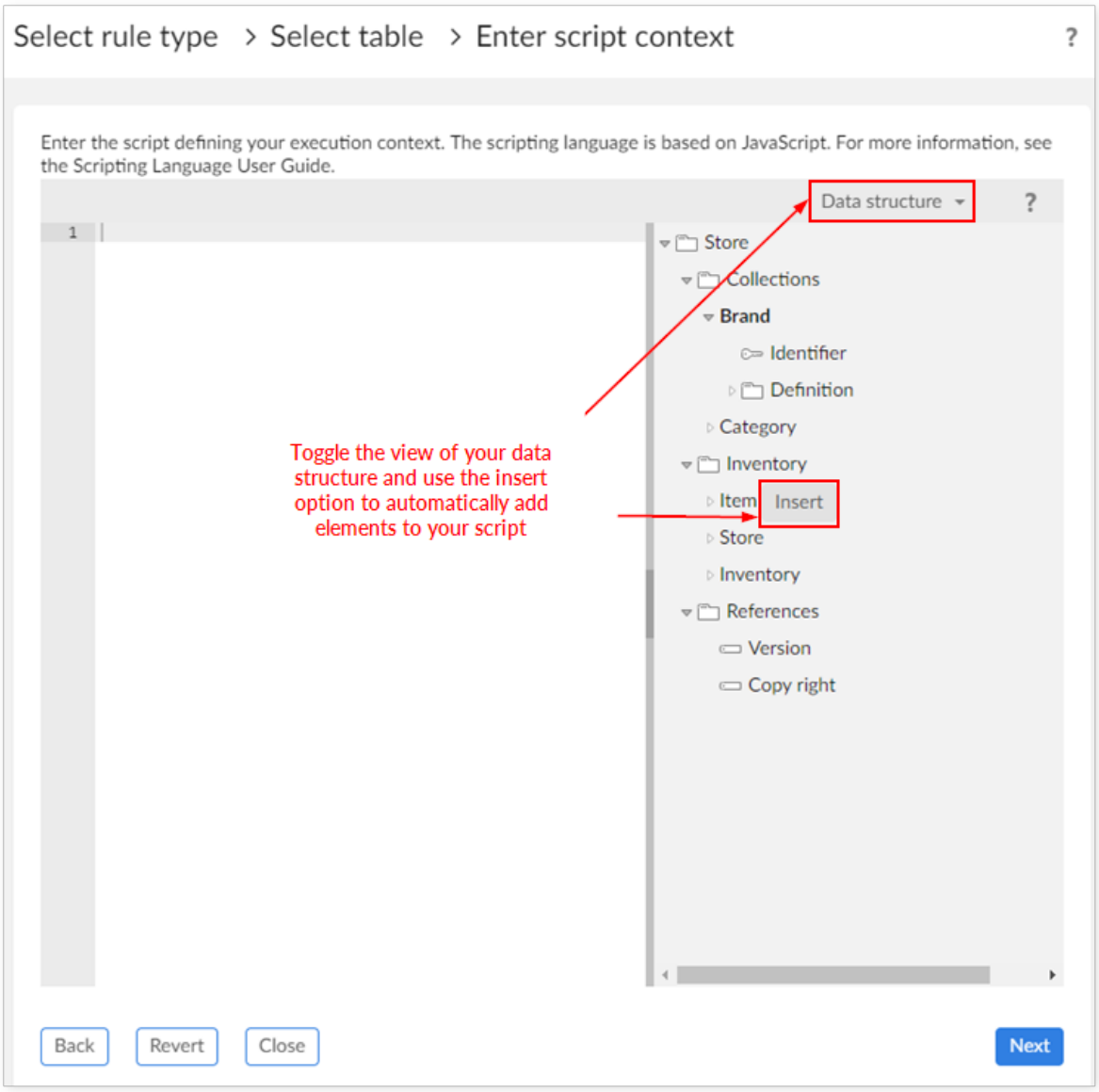


- Select the type of rule you would like to create. Note that as of this release the wizard only supports creation of a business rule.

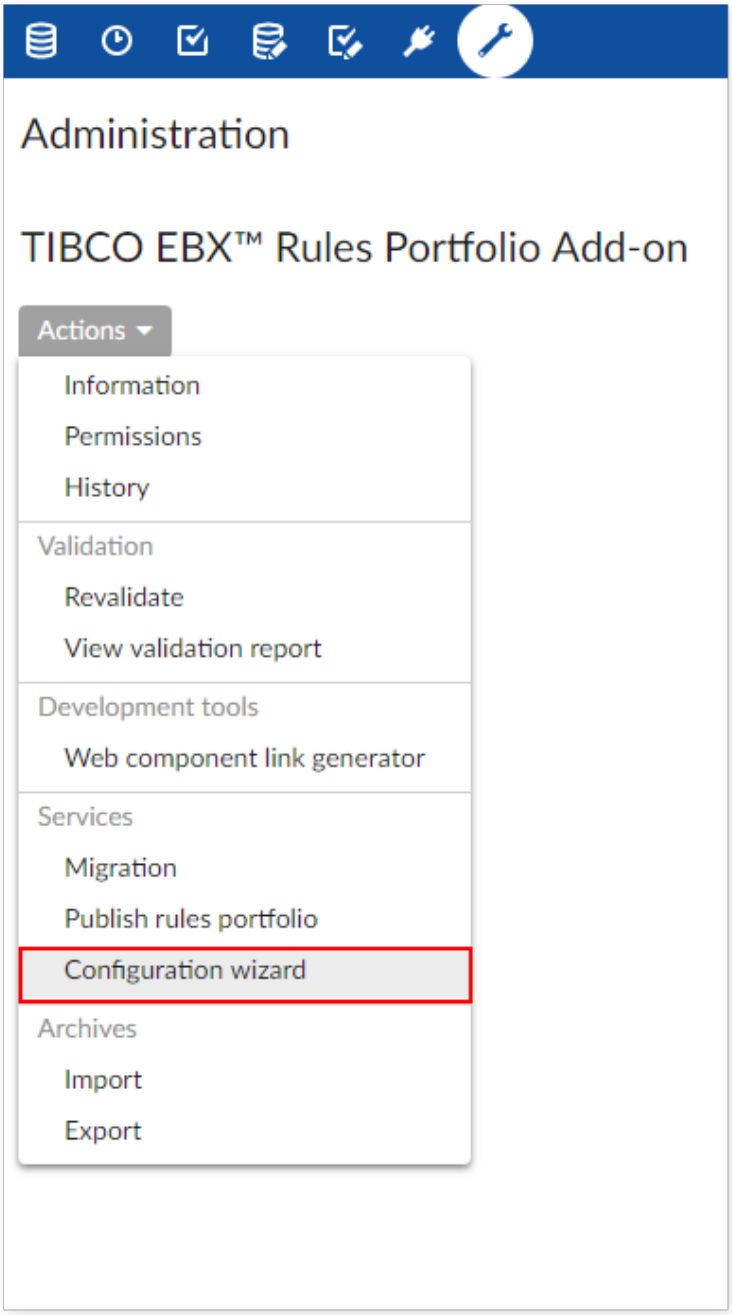
- Select the data model and table on which this rule will execute.



- Choose whether to use an execution context. An execution context defines any conditions for rule execution. If you choose to use a script to define a context, the following screen displays.



- Choose when you want the rule to execute.



- Enter the rule name, specify severity, and an optional message.
- Enter the script that specifies the business logic for the rule. See above the image associated with step four for tips on using the script.
- You have the following options after entering your script:
- - If you are finished, save the rule and close out of the wizard by clicking 'Finish'. The add-on will present you with the option to publish the portfolio.

- You can create a new set by entering the set's name in the 'Rule set name' box. You can then select 'Create more rule(s) for current table' to add additional rules to this set, or click 'Finish' to exit and close. The add-on presents you with the option to publish the portfolio.
- Select 'Create more rule(s) for current table' to create an additional rule for this table. You can continue to add rules to a table using this option and just click 'Finish' when done.
- Select 'Create rule for another table' to save the existing rule and restart the wizard.

CHAPTER 8

Quick rules configuration

This chapter contains the following topics:

8.1 Rules configuration overview

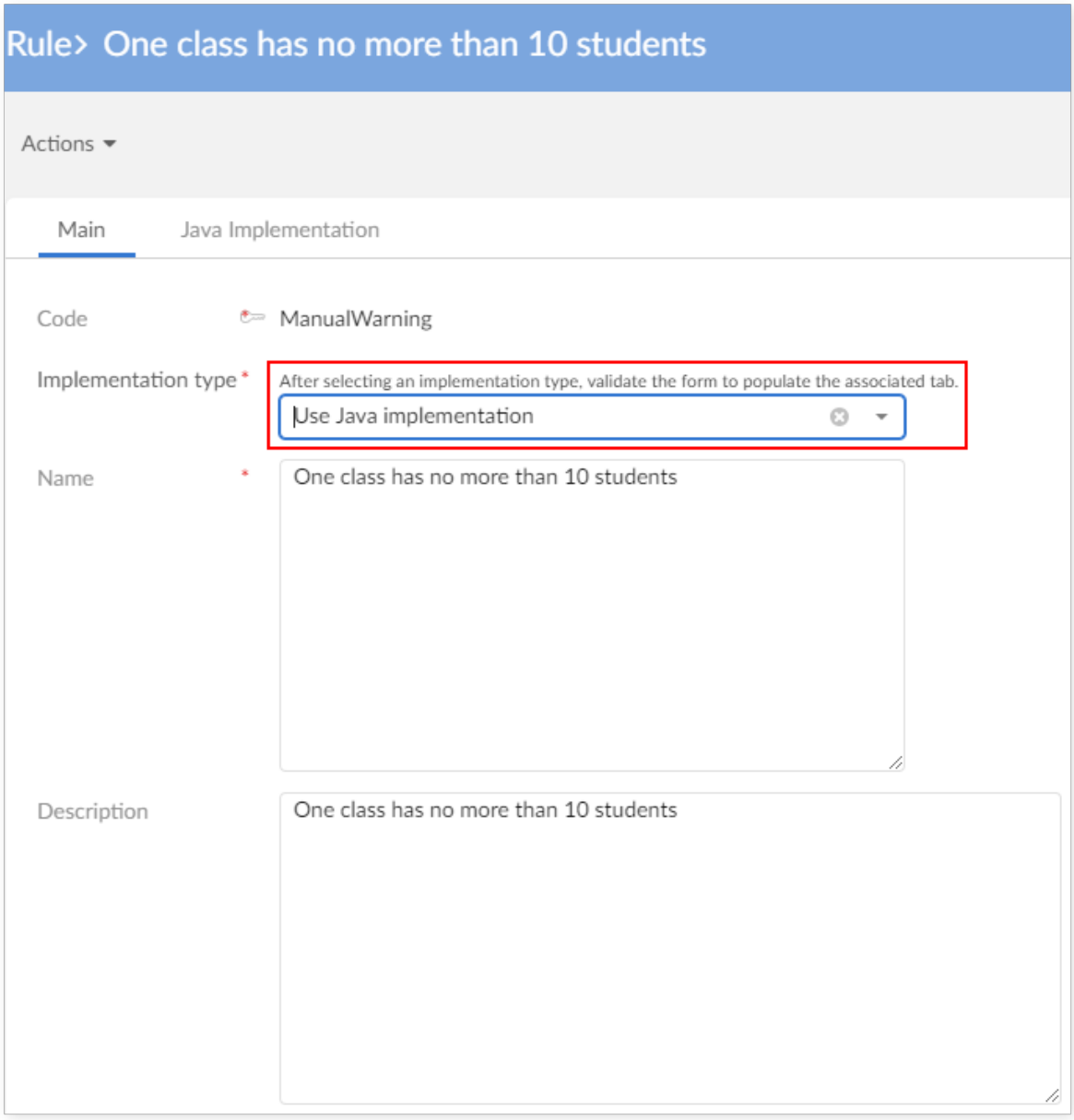
This section describes how to configure the most common types of rules as highlighted in the table below. These configurations do not use all properties available to adapt rule behavior just the most common ones.

Use case	Context	Data set execution	Result
Rule implemented using Java without conditions of execution	No	No	The Java-implemented rule executes on any data set without any conditions.
Rule implemented using a script without conditions of execution	No	No	The script-based rule executes on any data set without any conditions.
Rule executed on a data set	No	Yes	The rule is executed on a data set only without any additional conditions.
Rule executed on a data set with an execution condition implemented using Java	Yes	Yes	The rule is executed on a data set only with a condition of execution. The condition is defined by a rule which is implemented by Java.
Rule executed on a data set with an execution condition implemented using a script	Yes	Yes	The rule is executed on a data set only with a condition of execution. The condition is defined by a rule which is implemented by Script.

8.2 Java implementation without execution conditions

This type of rule executes on any data set without any execution conditions. The rule cannot execute unless its 'Active' property is set to 'Yes'. The following rule example basically says "One class has no more than 10 students".

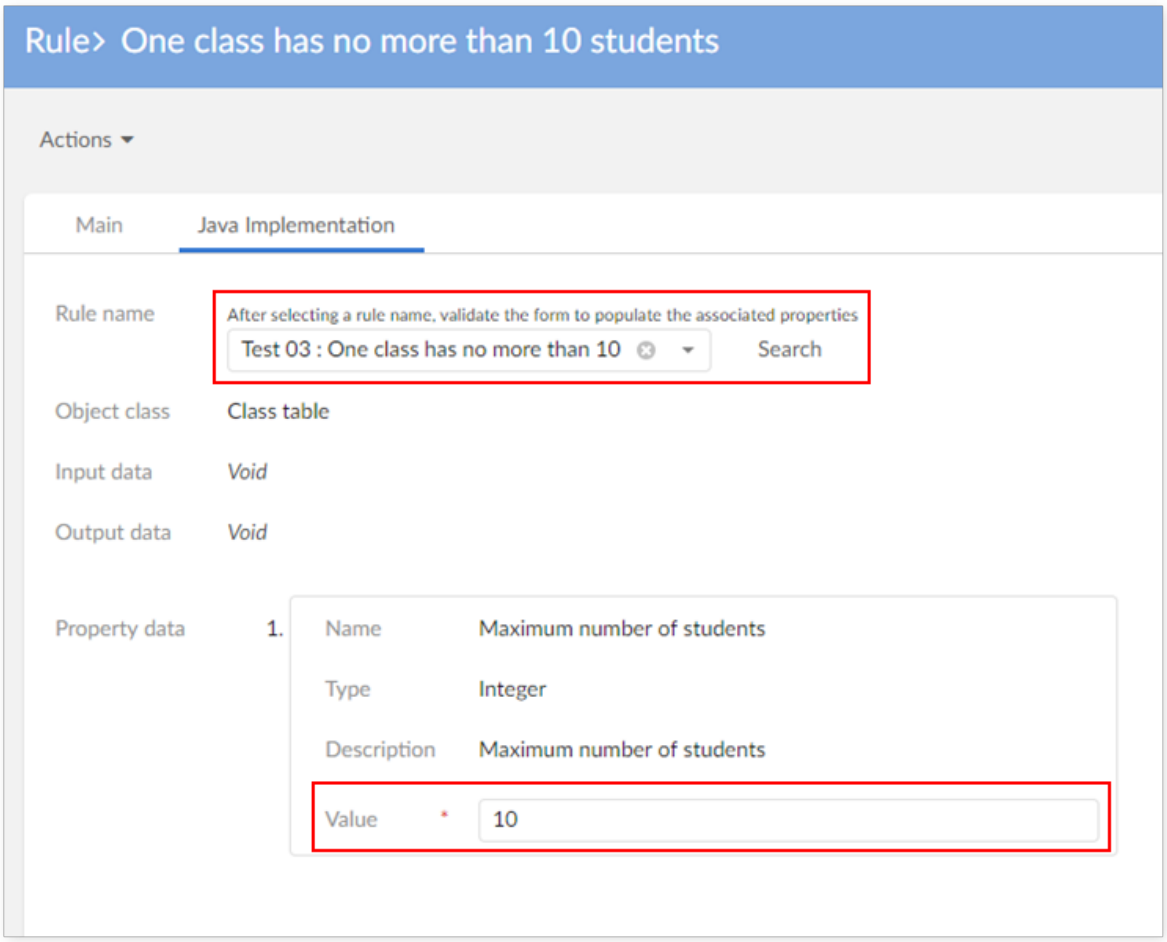
Step 1: Creation of the "Rule definition" (see screen-shot below).



The 'Code', 'Name' and 'Description' properties describe the rule from a business point of view. While the 'Implementation type' property specifies how you define the rule. If it is set to 'Use Script', the 'Script' tab displays and you can implement the rule using Scripting Language. If you set the property to 'Use Java implementation' the 'Java Implementation' tab displays and you can use a predefined rule. The 'Active' property allows you to deactivate the rule if necessary.

For a Java-implemented rule, you can define an unlimited number of rule definitions with different configurations. The "Property data" parameter published by the implemented rule makes its adaptation possible.

The configuration of a Java implementation is shown below and uses the 'Test 03: One class has no more than 10 students' rule.

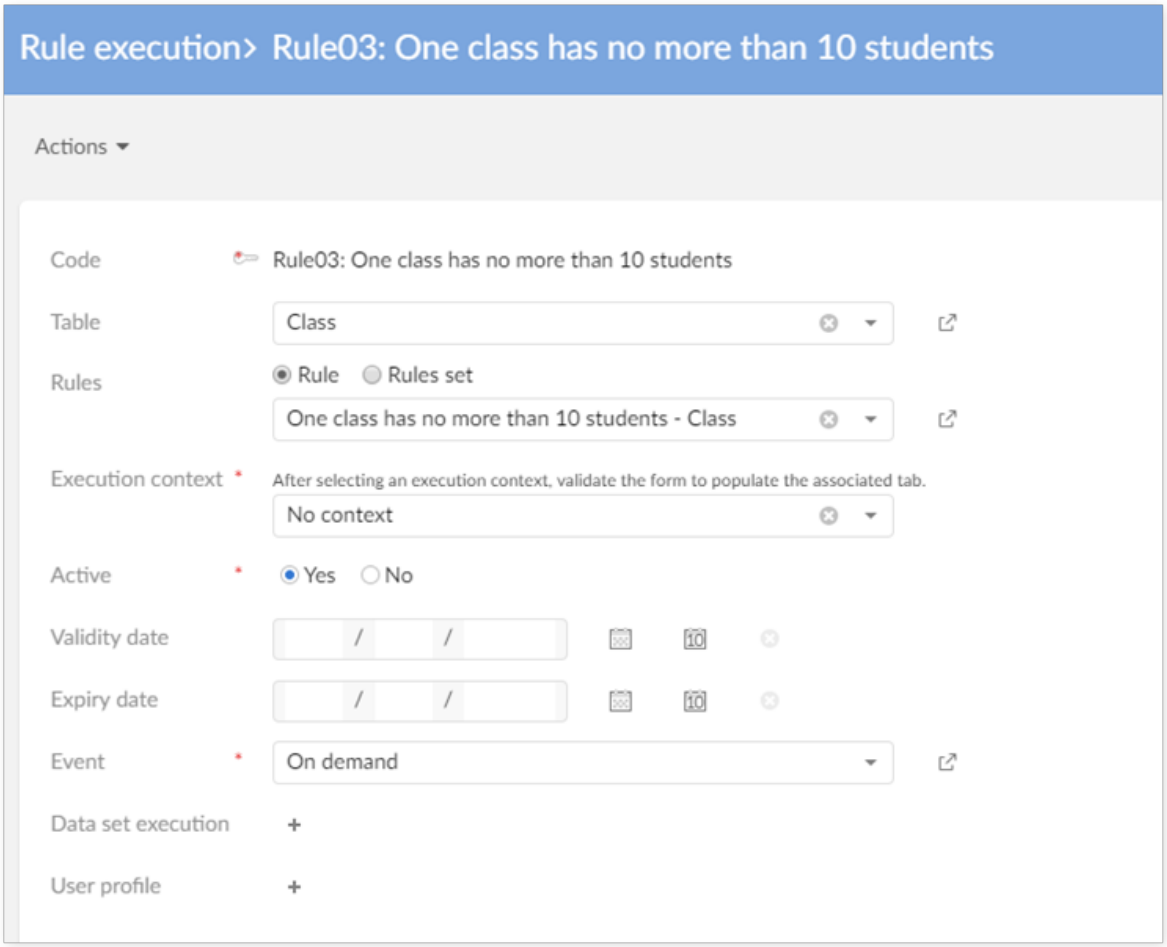


Step 2: Creation of the "Rule execution"

A "Rule execution" declares the conditions that permit a defined rule or set of rules to execute. To create a "Rule execution":

- Using the 'Table' property, declare the table (or, 'D.E.C. Attachment' for permission rule execution) this configuration applies to.
- Use the 'Rule' or 'Rules set' properties to specify which rule to use. Only one of these properties can be set per configuration. These fields are filtered to show only rules applied to the D.E.C. specified in the 'Table' field.
- From the 'Event' property specify when this configuration executes (after creation, modification or deletion).
- The 'Execution context' field allows you to choose how you want to construct conditions of execution ('Use script', 'Use Java implementation' or 'No context'). In the example shown below no context is needed and the 'Execution context' property is set accordingly. Based on this

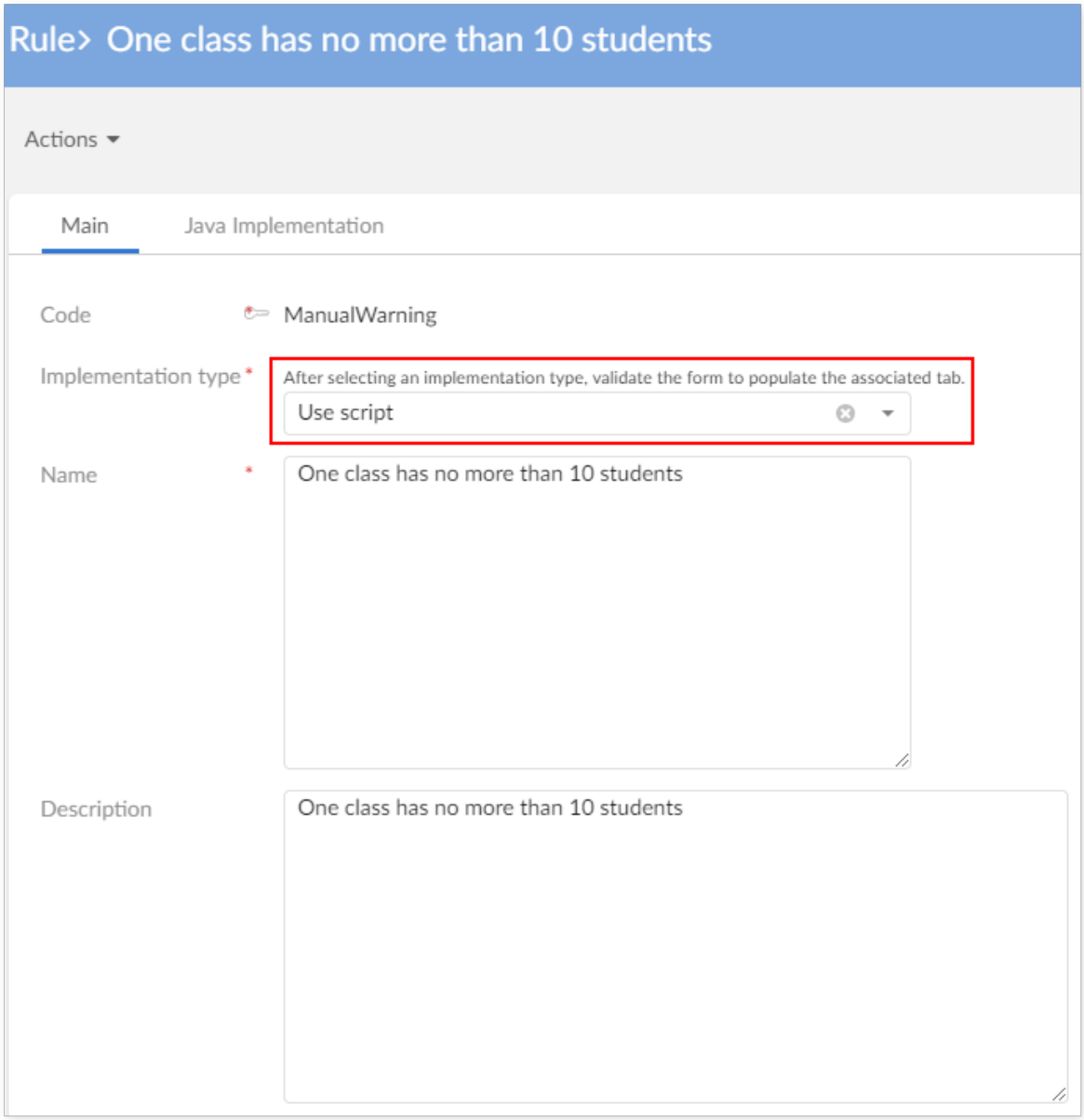
configuration, the rule executes on every data set's 'Class' table upon initiation of the 'On demand' event.



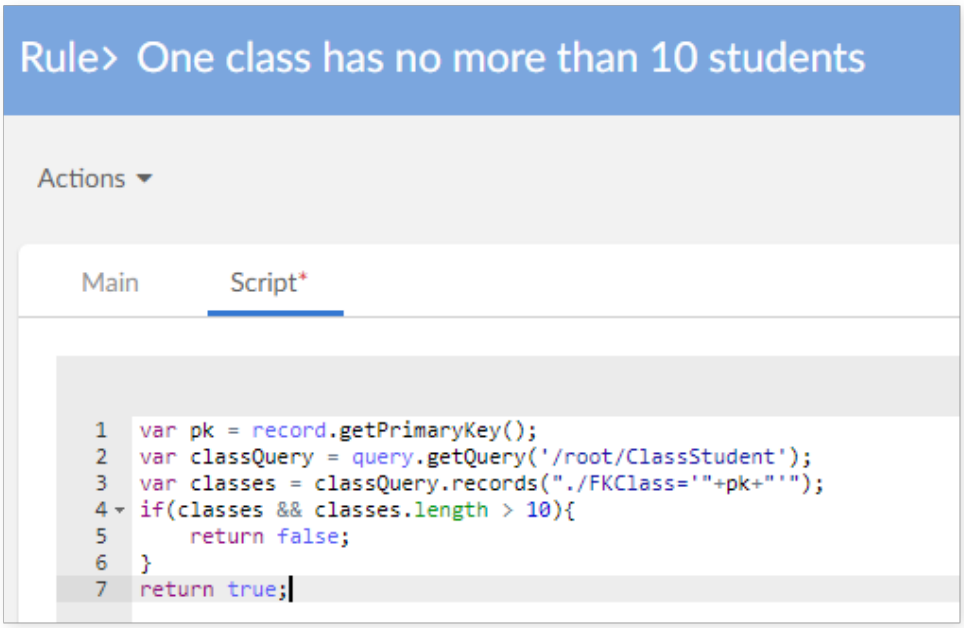
8.3 **Script implementation without execution**

This type of rule executes on any data set without any execution condition. The rule cannot execute unless you set its 'Active' property to 'Yes'. This rule example basically says "One class has no more than 10 students".

Step 1: Creation of the "Rule definition" (see screen-shot below).



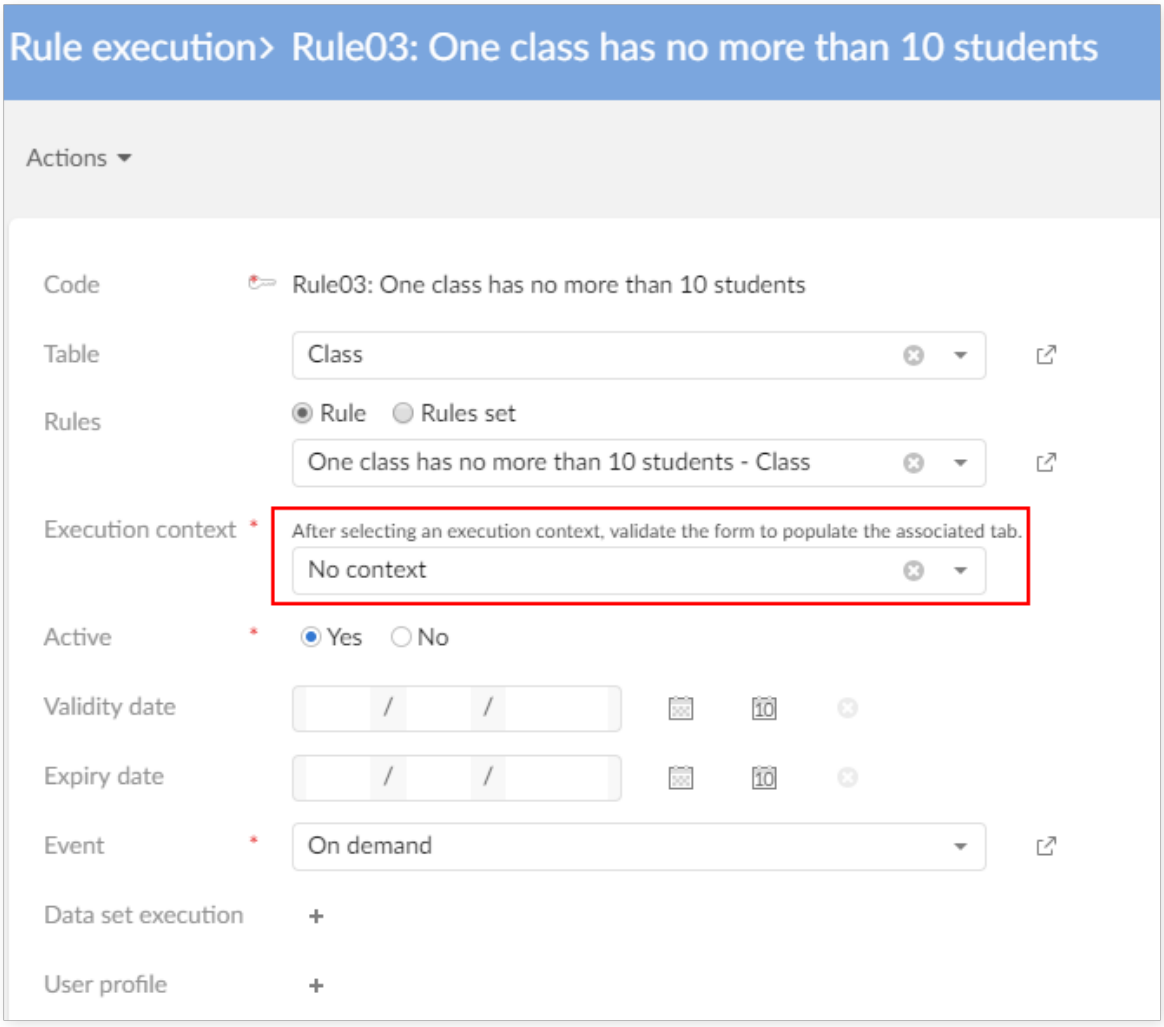
The content of script code is shown below:



The Rule definition is based on the script that you enter in the 'Script' tab. In this example, the rule statement "One class has no more than 10 students" is represented by the scripting code shown above.

Other properties in this rule are the same as properties of first example except the 'Use script' field is set to 'Yes'.

Step 2: Creation of the "Rule execution"



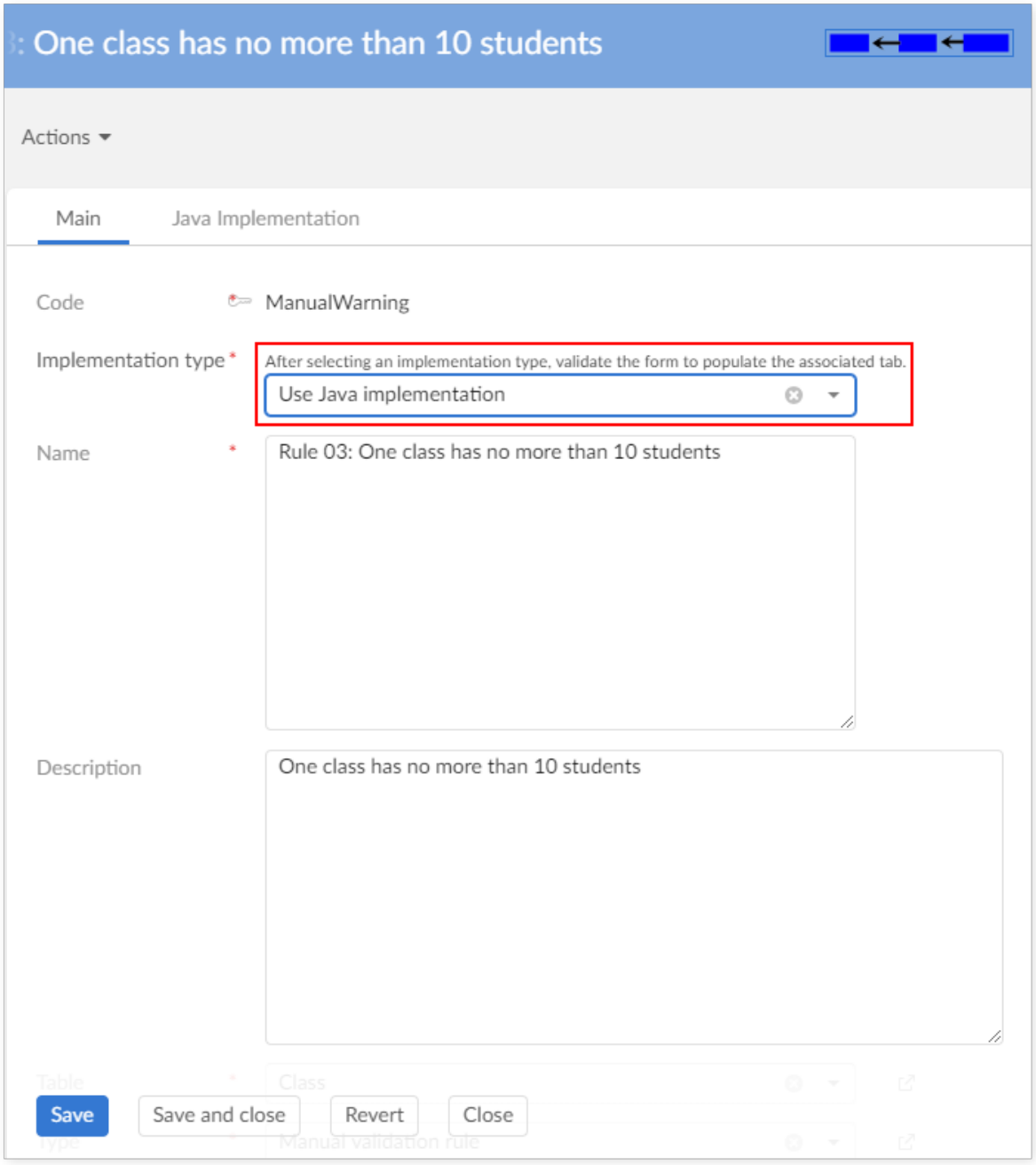
Refer to the first example for information on the "Rule execution" creation.

8.4 **Rule executed on a data set**

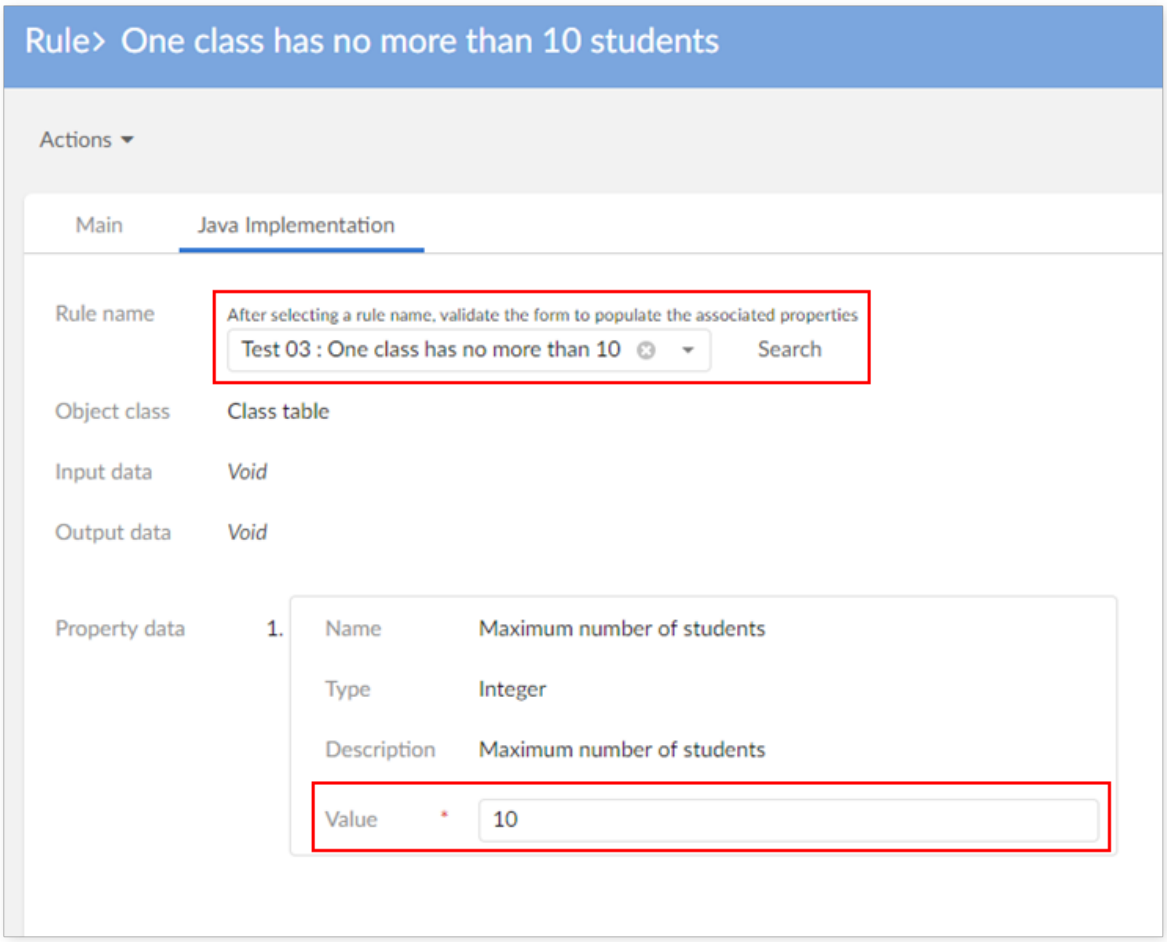
This type of rule is executed on a data set only and without any other execution conditions. The 'Active' property must be set to 'Yes' otherwise it cannot be executed.

The following example states "One class has no more than 10 students".

Step 1: Creation of the "Rule definition"

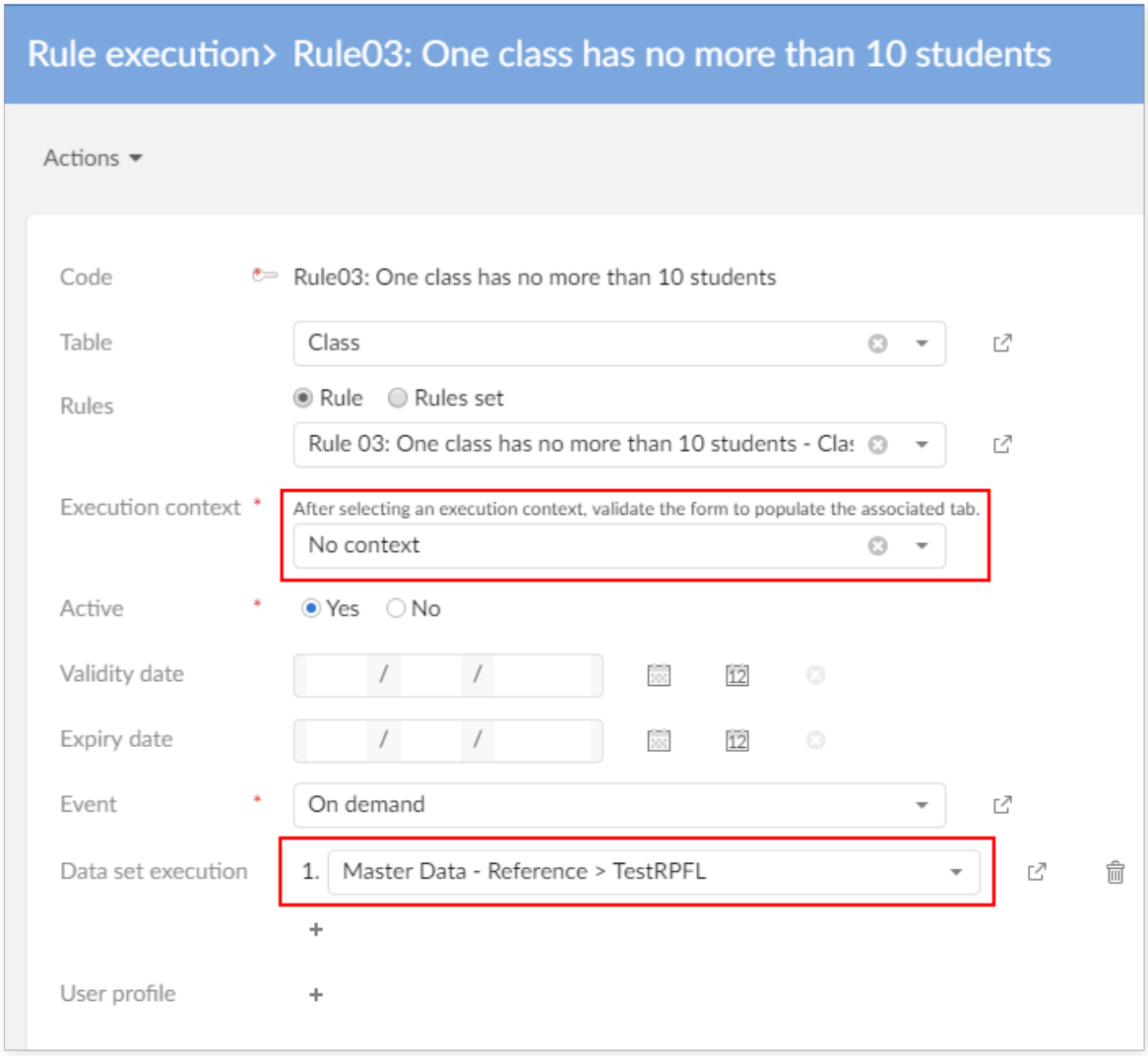


The configuration of Java implementation is shown below:



Refer to the first example for information on creating the "Rule definition".

Step 2: Creation of the "Rule execution"



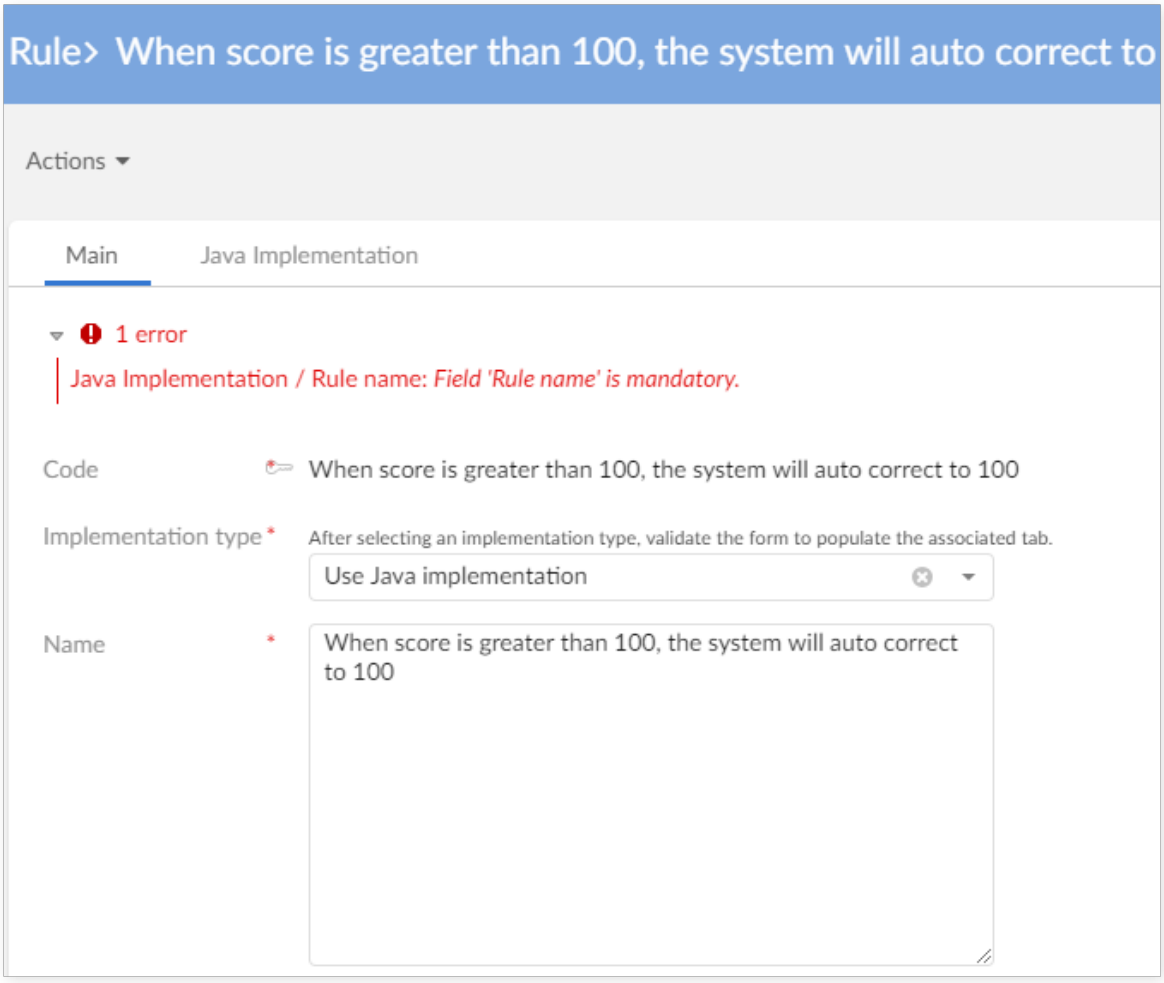
A "Rule execution" is created to declare the conditions that allow the defined rule's execution. In this example, only the 'Data set execution' property is used to limit the rule execution. The 'Execution context' property is set to 'No context' because there is no condition of execution.

8.5 Java implementation requiring execution conditions

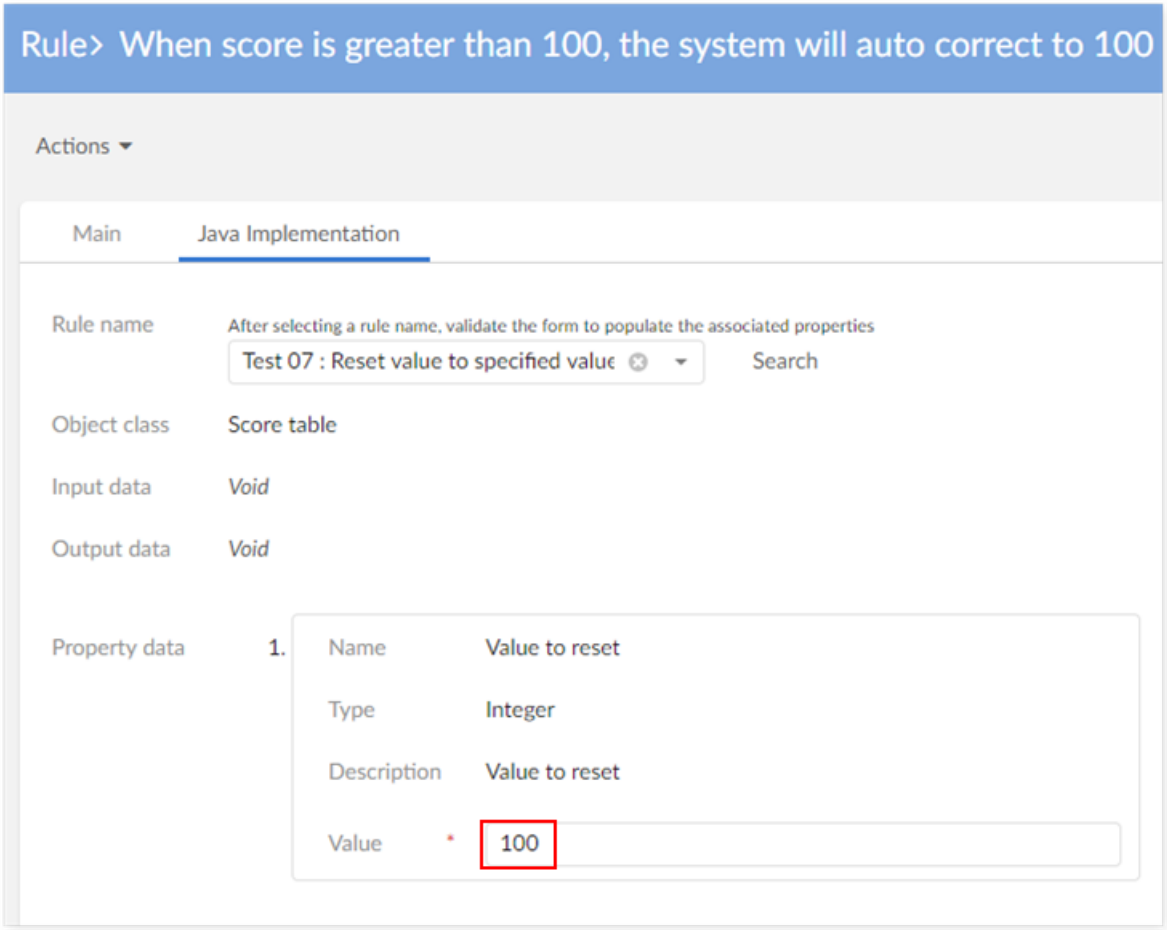
This type of rule executes on any data set if certain execution conditions are valid. The rule's 'Active' property must be set to 'Yes' otherwise its execution is not permitted.

The following rule example states: "When the score is greater than 100, the system will auto correct to 100".

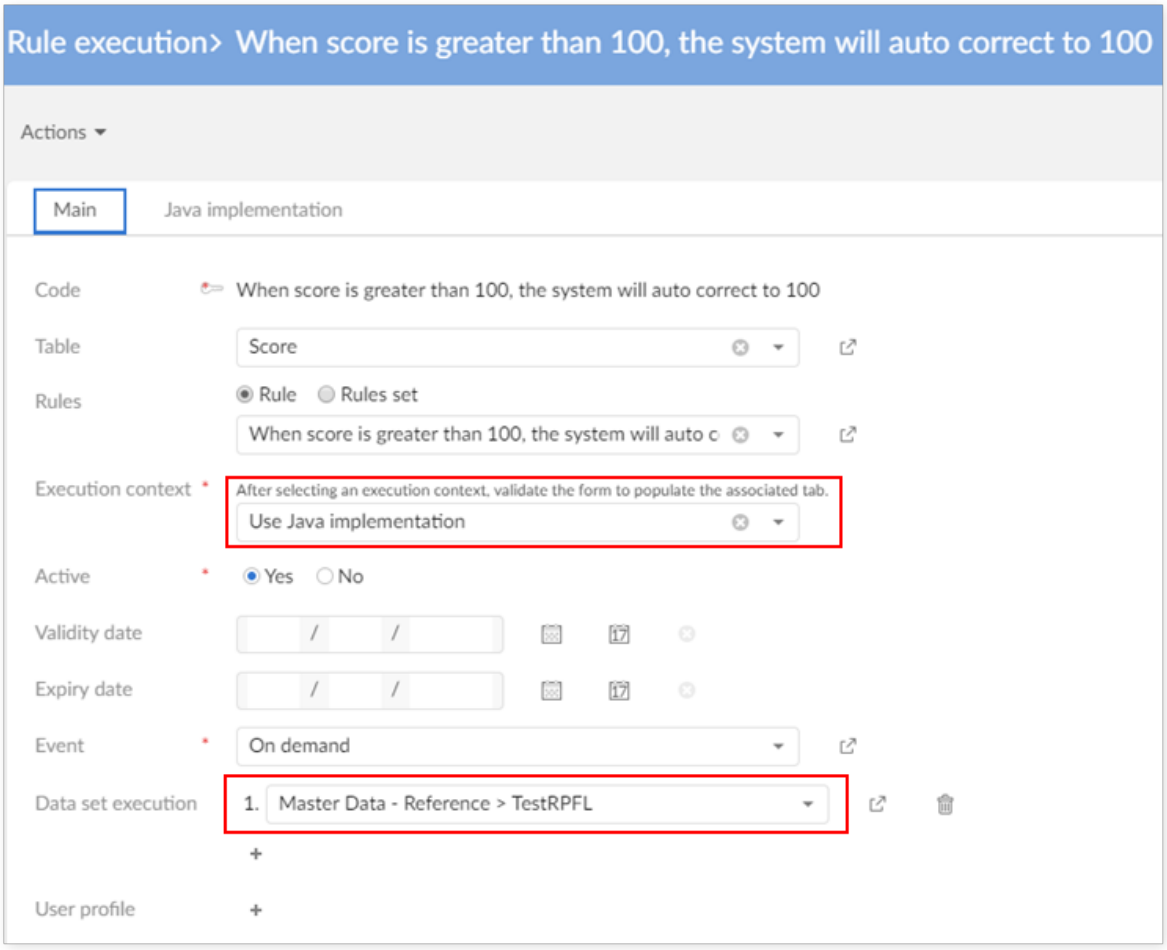
Step 1: Creation of the "Rule definition"



The configuration of Java implementation is shown below:

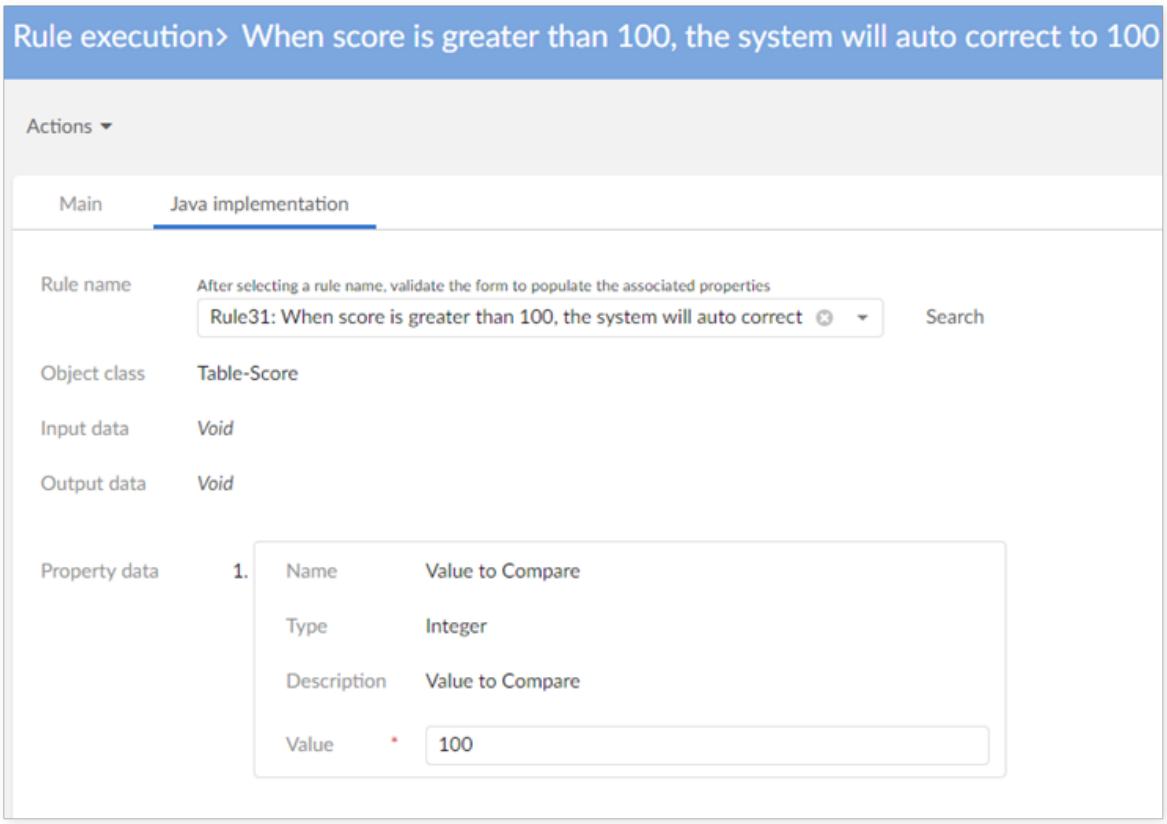


Step 2: Creation of the "Rule execution"



A "Rule execution" declares the conditions that allow the defined rule's execution. In this example, an execution context and data set execution are defined together to limit the rule execution. Additionally,

the condition of execution is implemented using Java so the 'Execution context' property is set to 'Use java implementation'. The image below shows the configuration set in the 'Java implementation' tab.

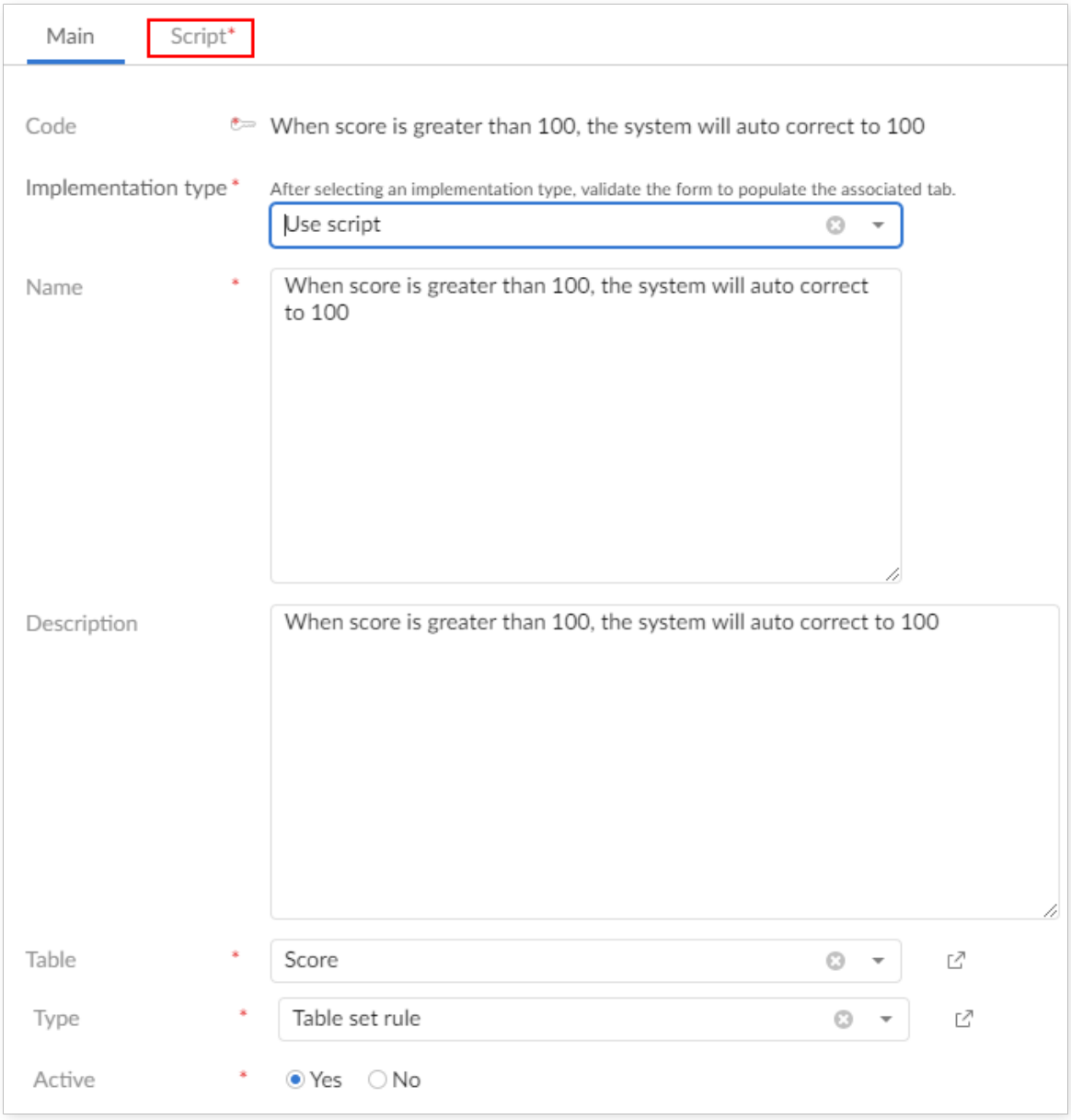


8.6 **Script implementation requiring execution conditions**

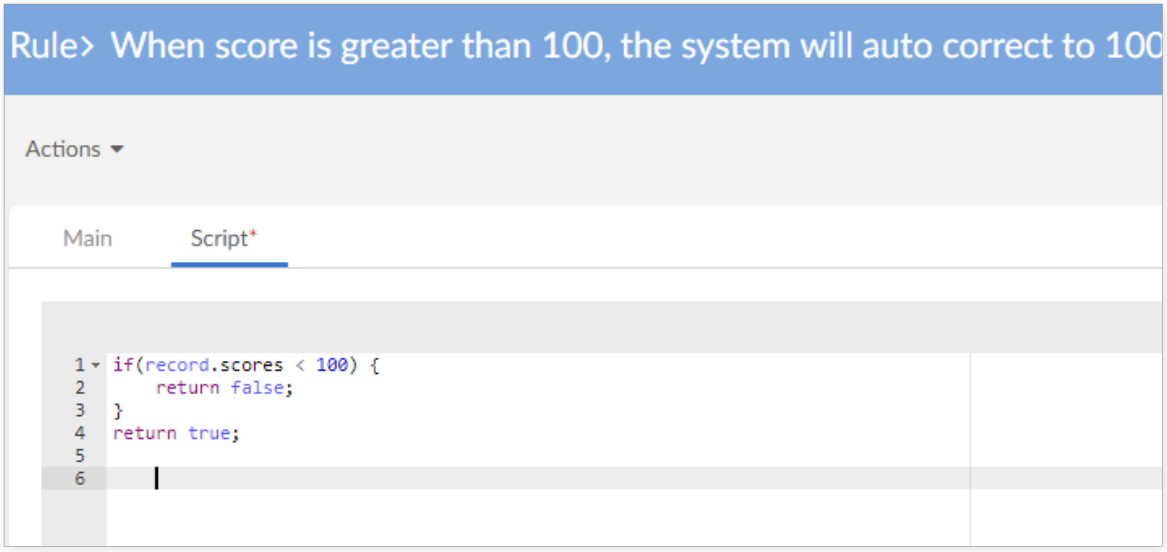
This type of rule is executed for a specific data set if certain execution conditions are valid. The rule's 'Active' property must be set to 'Yes' otherwise execution is not permitted.

The rule example states "When score is greater than 100, the system will auto correct to 100".

Step 1: Creation of the "Rule definition"

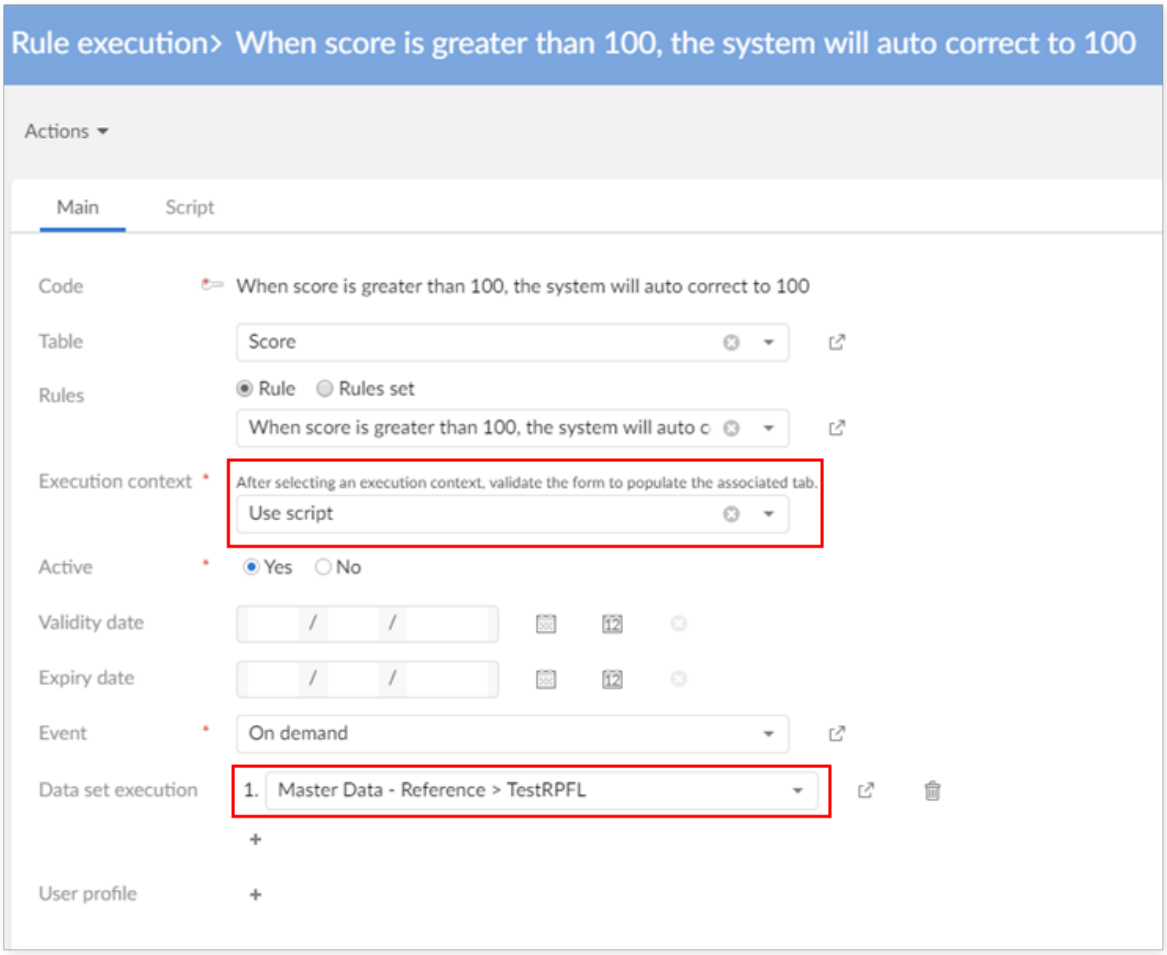


The content of Script code is shown below:

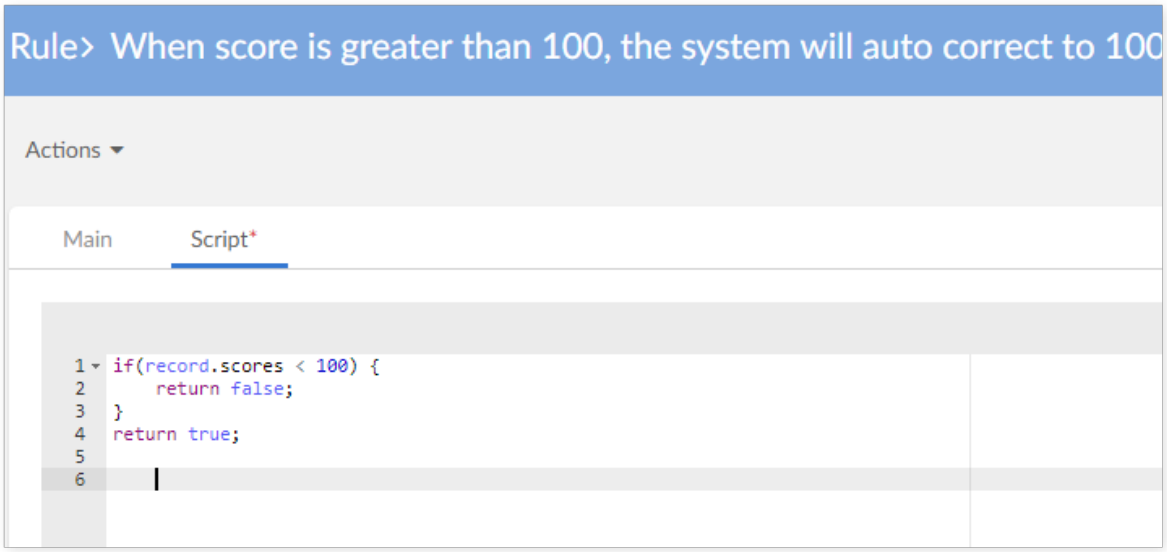


Refer to the first example for information on the "Rule definition" creation.

Step 2: Create the "Rule execution" to declare conditions that allow the rule's execution. In the example shown below an execution context and data set execution limit rule execution. Additionally, notice the 'Execution context' property is set to 'Use script'.



The image below shows the configuration in the 'Script tab.



8.7 Creating new rules with predefined assertions

Certain situations can lend themselves to using pre-built assertion rules to create new rules. These pre-built rules are script-based and alleviate the need to hand-code Java. The following lists the available pre-built assertion rules:

- [ON] Assertion true (Manual validation rule, Validation rule and Table set rule).
- [ON] Assertion false (Manual validation rule, Validation rule and Table set rule).

For example, to check if field (F) in a table (T) is empty, configure the following rule execution:

- Create a rule definition on (T) with the pre-built assertion rule '[ON] Assertion false (Manual validation)'. This rule systematically returns a false value.
- Configure a 'Rule execution' with scripting code to check that (F) is empty.

During execution, when the condition of execution is validated (the table field is empty), the rule executes and returns a value of 'False' which logs an error in the validation report. If the field is not empty, then the rule is not executed and no error is logged.

You can extend this type of configuration using a script to implement any execution context.

CHAPTER 9

Rules execution traceability

This chapter contains the following topics:

9.1 Traceability overview

'Rules Portfolio - Logging' allows you to keep track of executed rules. An automatic archiving process records log files that can be loaded on demand.

To apply the archiving process based on the scheduler, the task '[built-in] Rules logging clean-up' must be used.

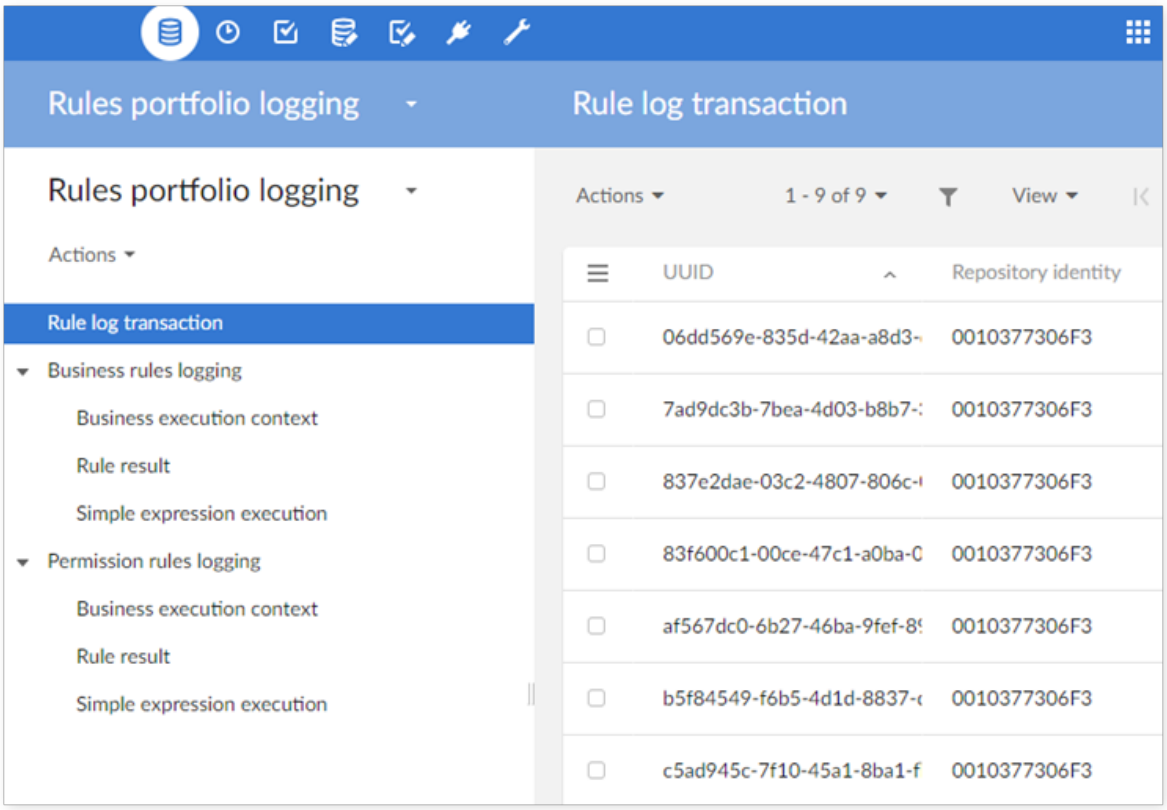
Rules execution logging may be required in the case of a business audit, but it is also useful when you need to debug executed rules.

Special Notation	
✔	The execution of the "Rules Portfolio - Logging" is done in asynchronous. A memory queue is used to detach the process of writing into the logging tables. To configure the size of the memory queue, please refer to 'Customization of the logging queue size'.

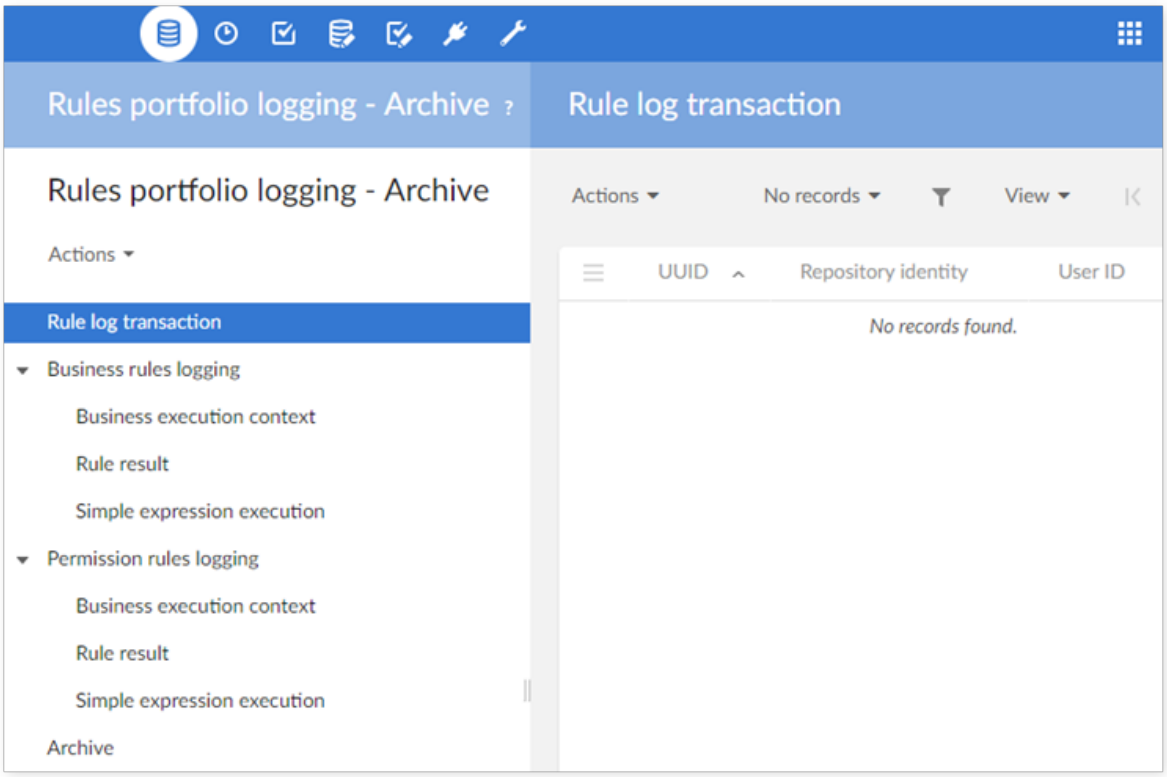
9.2 **Rules Portfolio - Logging**

You can configure rules logging using the 'Logging level' group in the 'TIBCO EBX® Rules Portfolio Add-on' data set, located under the Administration tab (see the 'Logging level' section).

Business rules logging is separated from permission rules logging which allows you to activate logging differently based on a rule's nature.



A second data set 'Rules Portfolio - Logging archive' allows you to collect and manage logging archive files. This data set relies on the same data structure as 'Rules Portfolio - Logging'.



9.3 **Root log transaction**

Every time a rule execution starts a "Root log transaction" record is created. Rule execution can begin when initiated by a table trigger or 'on demand' by initiating the validation report, or the add-on's 'Execute rules' service.

Property	Definition
UUID	The UUID must be a unique identifier.
Repository identity	Identifies the EBX® repository.
User ID	Identifies the user who executed the rules. Note: When rules are executed from the EBX® "Validation report", the user ID is unknown.
Execution time	Shows when the rule execution started. Rule execution can be initiated by a table trigger or on demand by running the validation report, or the add-on's 'Execute rules' service.
Date space	Data space in which the rules have been executed.
Data set	Data set in which the rules have been executed.
Rule publication	Saves the information to identify which snapshot is the running rule environment at execution time.

9.4 **Business rules logging**

This group of fields contains the properties to configure business rules logging.

Business execution context

This table records the logging data for every business "rule execution".

Property	Definition
Root log transaction	A link to the related 'Root log transaction'.
Rules execution	A link to the related rules execution.
Event	A link to the related event in the rules portfolio.
Table	A link to the table.
Record	A link to the record.
Field	A link to the field on which the rules have been executed.
Rule set	A link to the rule set in the rules portfolio.
Context result	If 'Accepted': the context has been validated and rules have been executed. If 'Not accepted': the context has not been validated and rules have not been executed.
Rules execution result	If 'Yes': all rules have been executed and there are no errors. If 'No': errors have occurred during rule execution.

Rule result

This table records the logging data for every business rule that has been executed.

Property	Definition
Business execution context	A link to the related 'Business execution context'.
Rule	A link to the rule.
Rule execution result	The result of the rule execution: If 'True': the rule has been executed without error. If 'False': an error has occurred during rule execution.
Error message	The error message that displays if the 'Rule execution result' is 'False'.
Error type	A link to the error type.

Simple expression execution

This table records the logging data for every 'simple expression' execution.

Property	Definition
Business execution context	A link to the related 'Business execution context'.
Simple expression	A link to the simple expression.
Context result	The simple expression's result: If 'Accepted': the simple expression was accepted. If 'Not accepted': the simple expression was not accepted.
Logical operator	A link to a logical operator if the execution of many simple expressions is required to compute the context.

9.5 **Permission rules logging**

This group of fields contains the properties to configure permission rules logging.

Business execution context

This table records the logging data for every permission "rule execution".

Property	Definition
Rule log transaction	A link to the related 'Root log transaction'.
Rules execution	A link to the related rules execution.
Service	A link to the service on which the rules have been executed.
Table	A link to the table on which the rules have been executed.
Field	A link to the field on which the rules have been executed.
Record	A link to the record on which the rules have been executed.
Rule set	A link to the rule set in the rules portfolio.
Restriction mode	The permission rules set restriction mode: If 'Most restricted': the most restricted permissions are returned. If 'Most unrestricted': the most unrestricted permissions are returned.
Context result	If 'Accepted': the context has been validated and the rules were executed. If 'No accepted': the context has not been validated and the rules have not executed.
Rules execution result	The result of the permission rule execution. For access permission: • 'Hidden': the resource is not displayed. • 'Read-only': the resource is displayed and can not be modified. • 'Read-write': the resource is displayed and can be modified. For action permission: • 'Hidden': the action is not displayed. • 'Forbidden': the action is displayed but can not be executed. • 'Allowed': the action is displayed and can be executed.

Rule result

This table records the logging data for every permission rule that is executed.

Property	Definition
Business execution context	Link to the related 'Business execution context'.
Rule	Link to the rule.
Rule execution result	The result of the permission rule: For access permission: • 'Hidden': the resource is not displayed. • 'Read-only': the resource is displayed and can not be modified. • 'Read-write': the resource is displayed and can be modified. For action permission: • 'Hidden': the action is not displayed. • 'Forbidden': the action is displayed but can not be executed. • 'Allowed': the action is displayed and can be executed.

Simple expression execution

This table records the logging data for every 'simple expression" execution.

Property	Definition
Business execution context	Link to the related 'Business execution context'.
Simple expression	Link to the simple expression.
Context result	Result of the simple expression: If 'Accepted': the simple expression has been accepted. If 'No accepted': the simple expression has not been accepted.
Logical operator	Link to a logical operator if execution of many simple expressions is required to compute the context.

9.6 Archive

This table records the archive files that are created automatically.

Property	Definition
Creation time	Date time of the archive creation.
Archive path	Path to the archive. The path is hidden by the button 'Import' to allow the import operation directly.

9.7 Localization of the archive files

The archive files are stored on the web server (within Tomcat server: %CATALINA_TMPDIR%, in the specified path for the add-on's : ebx-addon-rpfl\auto_archive\ ). For example: <temp_dir>\ebx-addon-rpfl\auto_archive\logRule-archives-1394513868906.zip

The naming convention of the archive file is logRule-archives-<current time milies>

9.8 Customization of the logging queue size

The asynchronous mode used for recording rule execution is based on a memory queue. This queue is issued by an asynchronous process to persist the data in the logging tables.

By default, the size of this queue is set to 100,000 records. Once the maximum size is reached, then rules execution has to wait until new space is available in the queue.

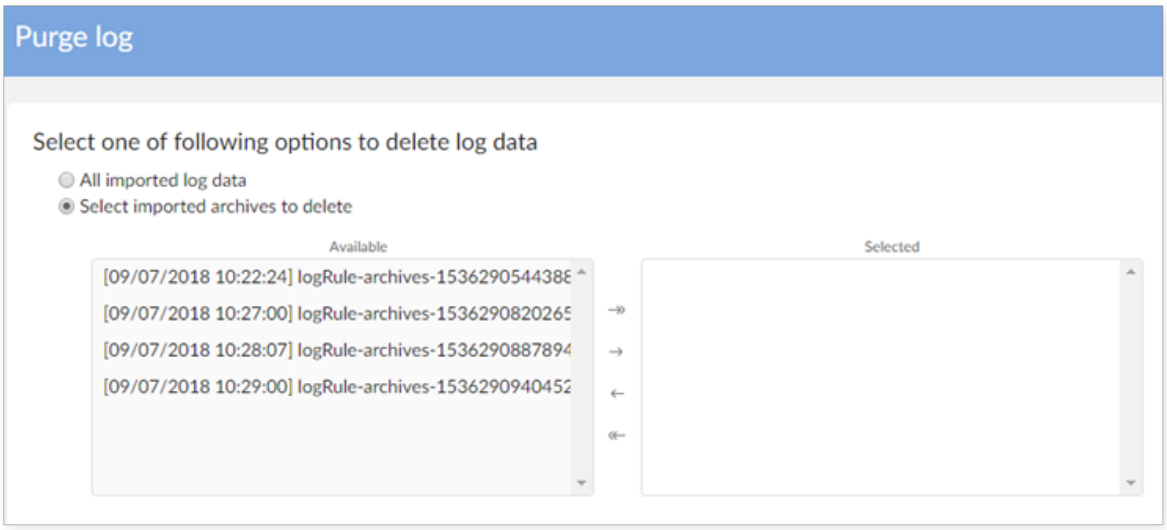
You can change this value by configuring a parameter in the java environment variable (refer to the appendix 'Environment configuration').

9.9 "Purge root log" service

The purge service is located on the 'Rule log transaction' table and allows you to delete all log data related to a selected root log record.

9.10 **"Purge log" service**

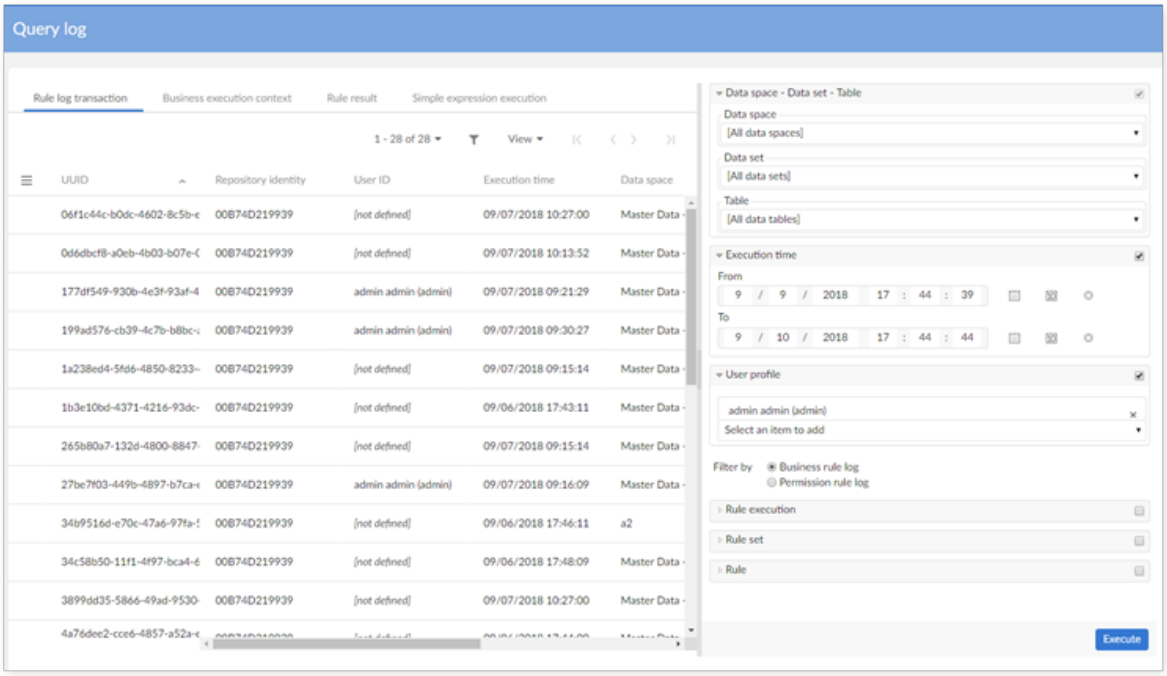
This service is available at data set level of the 'Rules Portfolio - Logging archive' data set. This service allows you to delete all imported log data from archive files or imported log data from selected archive files.



9.11 **"Query log" service**

This service can be run from the 'Rules Portfolio - Logging' and 'Rules Portfolio - Logging archive' data sets. This service provides the ability to search log data based on given criteria.

The image below shows the 'Query log' service configuration page. You can use multiple query options such as data space, data set, table, execution time, type of rule log, etc. The 'Query log' service uses this input a the log data search criteria.



CHAPTER 10

API for declaring rules

A set of Java APIs allow you to add and manage rules from the add-on. See the Javadoc provided with the add-on and the next appendixes of this user guide.

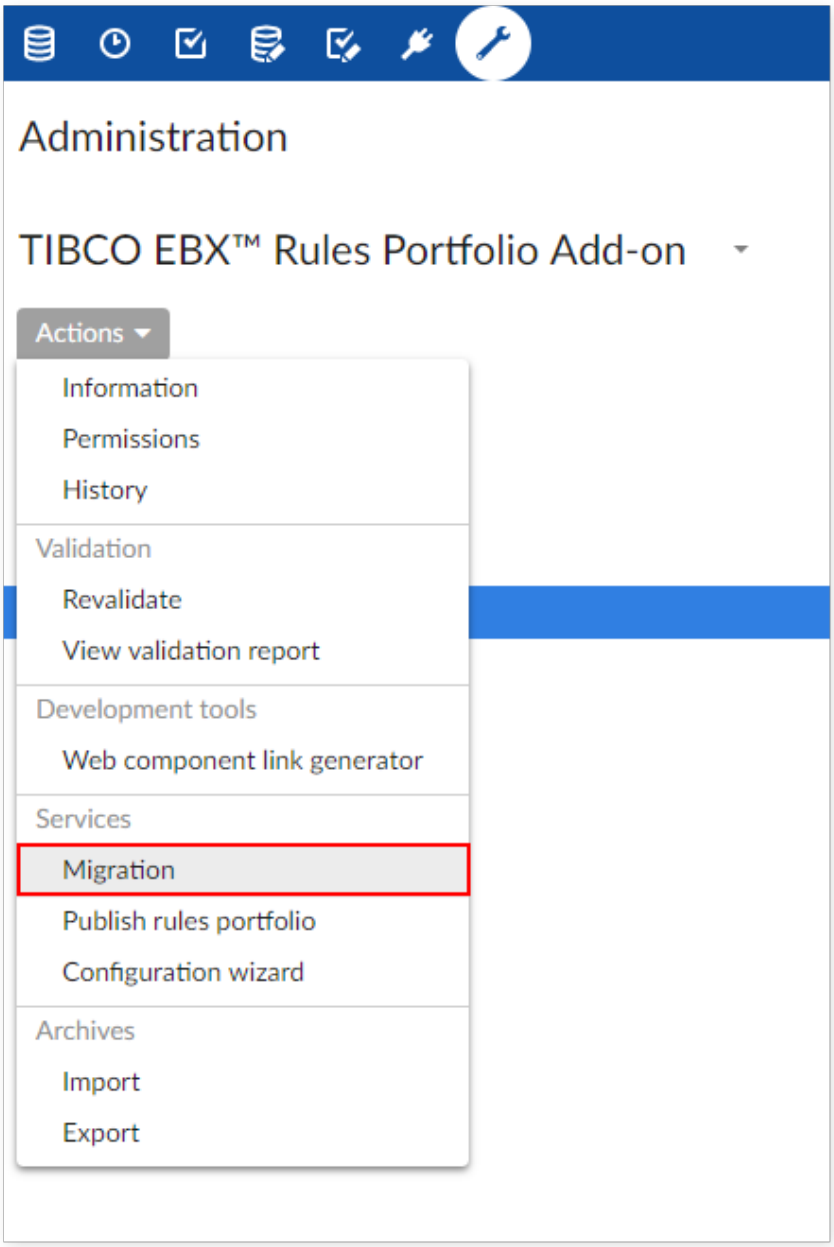
CHAPTER 11

Migration

Version 1.5.0

This version requires you to perform two steps to complete migration. In the first step, a migration procedure automatically executes when registering the add-on. The second step, requires you to run the 'Migration' service, located in the 'TIBCO EBX® Rules Portfolio Add-on' data set. Each time you upgrade to new version, after starting the server, you must run that migration service manually to ensure that all the rules that are defined by other modules are migrated. If you import an old archive

file (.ebx) into the 'TIBCO EBX® Rules Portfolio Add-on' data set, you must run the migration service in order to execute the migration procedure.



CHAPTER 12

Appendix

This chapter contains the following topics:

12.1 Business rule declaration

The declaration process is the same for the 'Rule' and the 'Rule context' that is used at the level of a simple expression context.

Step 1: Creation of the definition class.

```
public class RuleTest_Definition implements BusinessRuleDefinition
{
   public List<InputData> getInputData()
   {
      List<InputData> inputData = new ArrayList<InputData>();
      inputData.add(InputData.RECORD);
      return inputData;
   }
   public UserMessage getLabel()
   {
      return UserMessage.createInfo("Rule test");
   }
   public List<OutputData> getOutputData()
   {
      List<OutputData> outputData = new ArrayList<OutputData>();
      outputData.add(OutputData.BOOLEAN);
      return outputData;
   }
   public List<PropertyData> getPropertyData()
   {
      return null;
   }
   public UserMessage getBusinessConcept()
   {
      return UserMessage.createInfo("Any table");
   }
   public Rule getRuleInstance()
   {
      return new RuleTest();
   }
   public RuleType getRuleType()
   {
      return RuleType.ValidationRule;
   }
   public boolean isContext()
   {
      return false;
   }
   public UserMessage getDefaultMessage()
   {
      return UserMessage.createError("Default error message");
   }
}
```

Step 2: Creation of the rule class that implements the execution part of the rule.

```
/**
 */
public class RuleTest implements Rule
{
   private UserMessage errorMessage;
   private List<PropertyData> propertyData;

   public RuleTest()
   {
   }

   public RuleExecutionResult execute(RuleExecutionContext executionContext) throws RpfIException
   {
      RuleExecutionResult result = new RuleExecutionResult();
      result.setErrorMessage(this.errorMessage);

      // do something ...
      result.setResult(false);
      return result;
   }

   public void setup(RuleConfigurationContext configurationContext)
   {
      this.errorMessage = configurationContext.getErrorMessage();
      this.propertyData = configurationContext.getPropertyData();
   }
}
```

Step 3: Registration of the rule.

```
try
{
   RulesCatalog.add(new RuleTest_Definition());
}
catch (OperationException ex)
{
   throw new RuntimeException(ex);
}
```

12.2 **Permission rule declaration**

Step 1: Creation of the definition class.

- For access permission rule:

```
public final class AccessPermissionRuleTest_Definition
   implements AccessPermissionRuleDefinition
{
   public PermissionScope getPermissionScope()
   {
      return PermissionScope.TABLE;
   }
   public UserMessage getLabel()
   {
      return UserMessage.createInfo("Access permission rule test");
   }
   public UserMessage getBusinessConcept()
   {
      return UserMessage.createInfo("Any table");
   }
   public AccessPermissionRule getRuleInstance()
   {
      return new AccessPermissionRuleTest();
   }
   public List<InputData> getInputData()
   {
      return null;
   }
   public List<OutputData> getOutputData()
   {
      return null;
   }
   public List<PropertyData> getPropertyData()
   {
      return null;
   }
}
```

- For action permission rule:

```
public final class ActionPermissionRuleTest_Definition implements ActionPermissionRuleDefinition
{
   public List<InputData> getInputData()
   {
      return null;
   }

   public UserMessage getLabel()
   {
      return UserMessage.createInfo("Action permission rule test");
   }

   public List<OutputData> getOutputData()
   {
      return null;
   }

   public List<PropertyData> getPropertyData()
   {
      return null;
   }

   public UserMessage getBusinessConcept()
   {
      return UserMessage.createInfo("Any service");
   }

   public ActionPermissionRule getRuleInstance()
   {
      return new ActionPermissionRuleTest();
   }
```

Step 2: Creation of the rule class that implements the execution part of the rule.

- For access permission rule:

```
public final class AccessPermissionRuleTest implements AccessPermissionRule
{
   public AccessPermission getPermission(PermissionExecutionContext context)
   {
      // do something
      // ...
      return AccessPermission.getHidden();
   }

   public void setup(PermissionRuleConfigurationContext configurationContext)
   {
   }
}
```

- For action permission rule:

```
public class ActionPermissionRuleTest implements ActionPermissionRule
{
   public ActionPermission getPermission(PermissionExecutionContext context)
   {
      // do something
      // ...
      return ActionPermission.getDisabled();
   }

   public void setup(PermissionRuleConfigurationContext configurationContext)
   {
   }
}
```

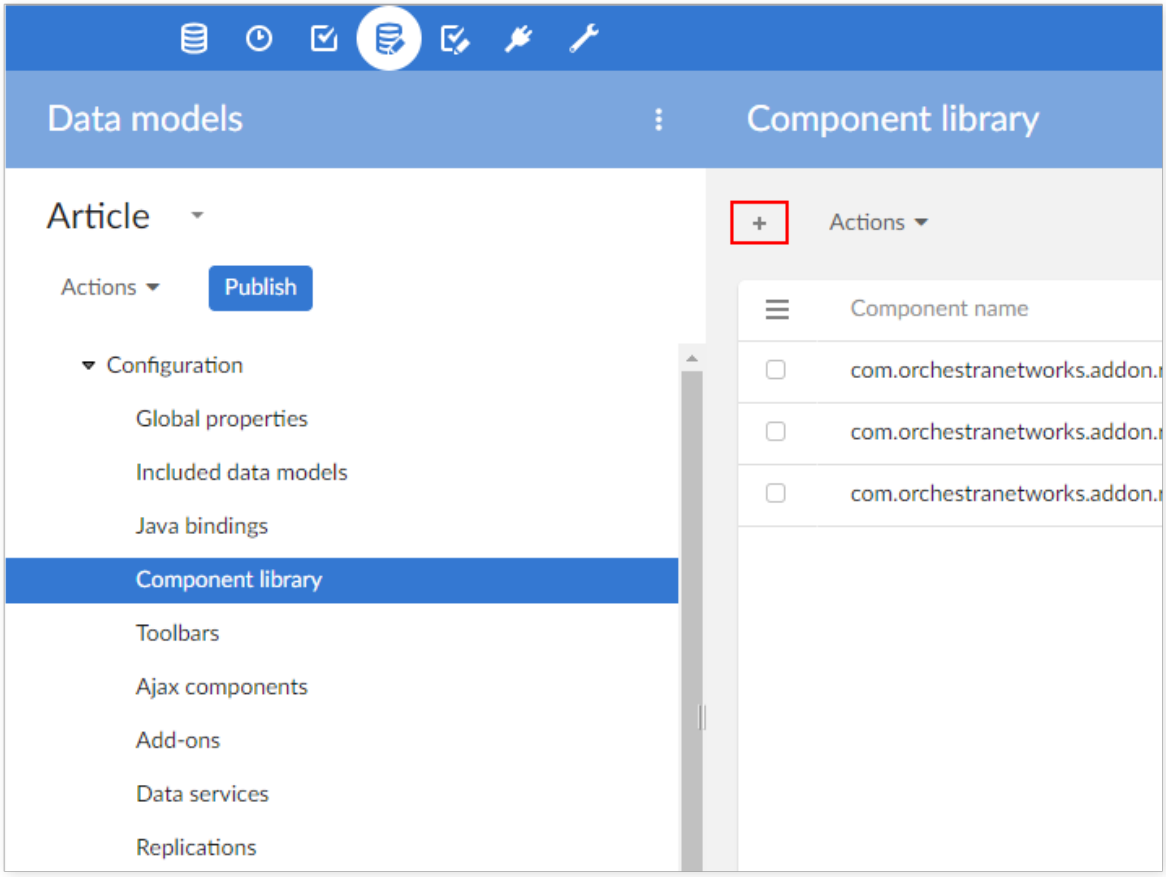
Step 3: Registration of the rule.

```
try
{
   RulesCatalog.add(new AccessPermissionRuleTest_Definition());
   RulesCatalog.add(new ActionPermissionRuleTest_Definition());
}
catch (OperationException ex)
{
   throw new RuntimeException(ex);
}
```

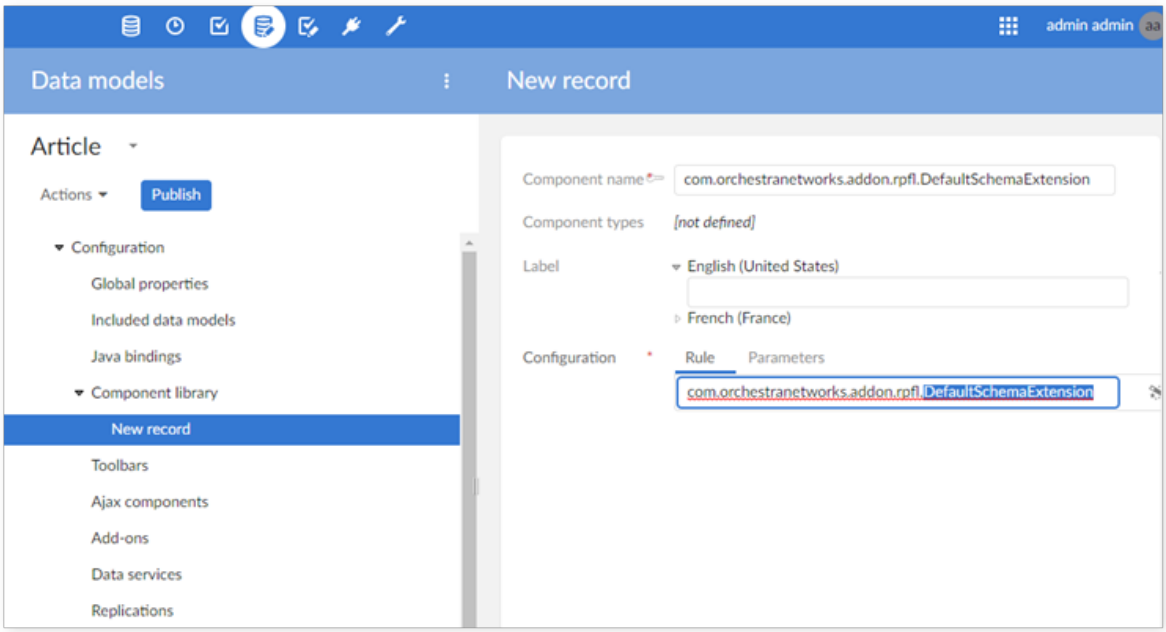
12.3 **Access permission on a data model**

The following steps describe how to control the access permission on a data model:

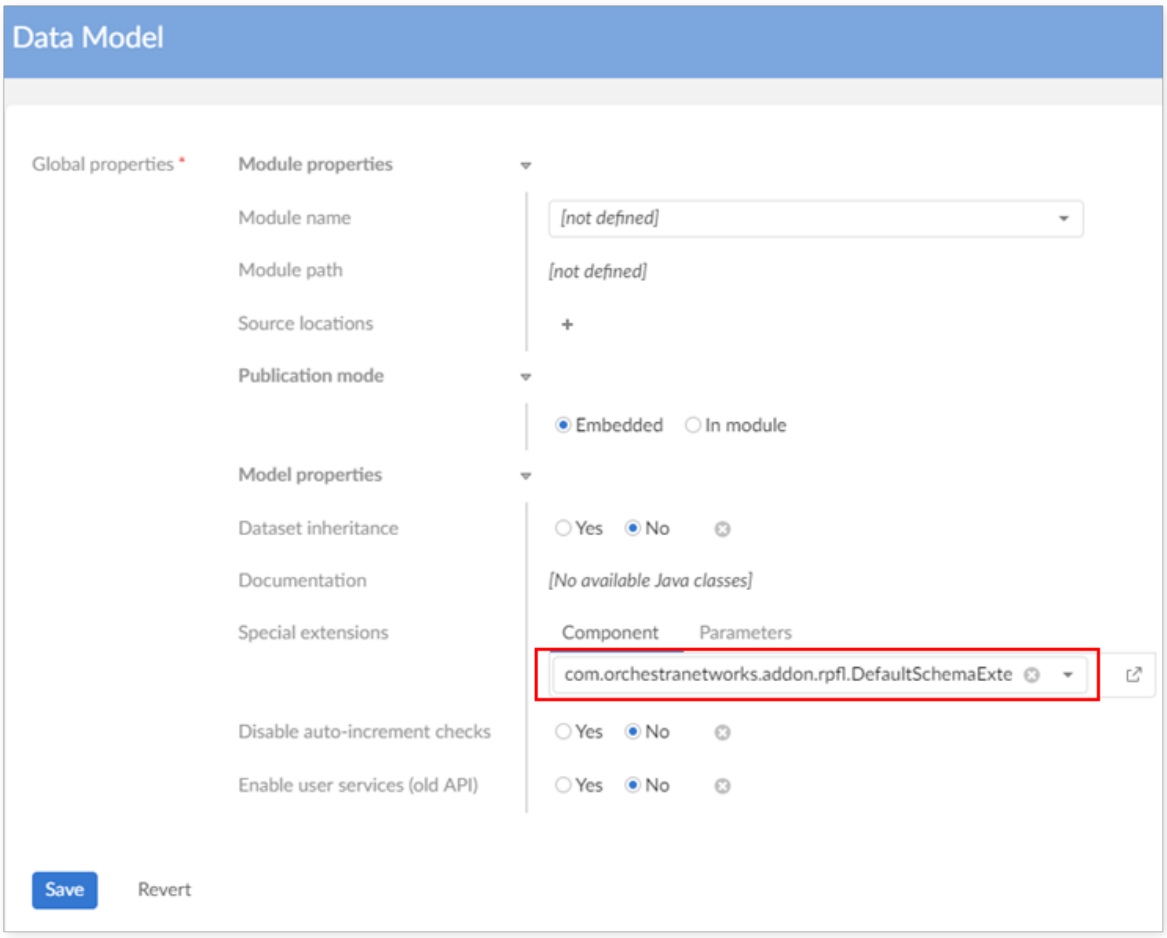
Step 1: Go to Modeling > Data models > Configuration > Component library and create new record:



Step 2: Paste the following class path to Configuration > Rule:
com.orchestranetworks.addon.rpfl.DefaultSchemaExtension



Step 3: Go to Configuration > Data model properties > Model properties > Special extensions and select the new component created in step 2.



12.4 Classic access rule declaration

This section describes how to reuse your programmatic access rules.

Step 1: Separate all access rules from your schema extension. Note that your classic access rules have to satisfy 2 conditions:

- Implement the interface AccessRule.
- Contain a no-argument constructor.

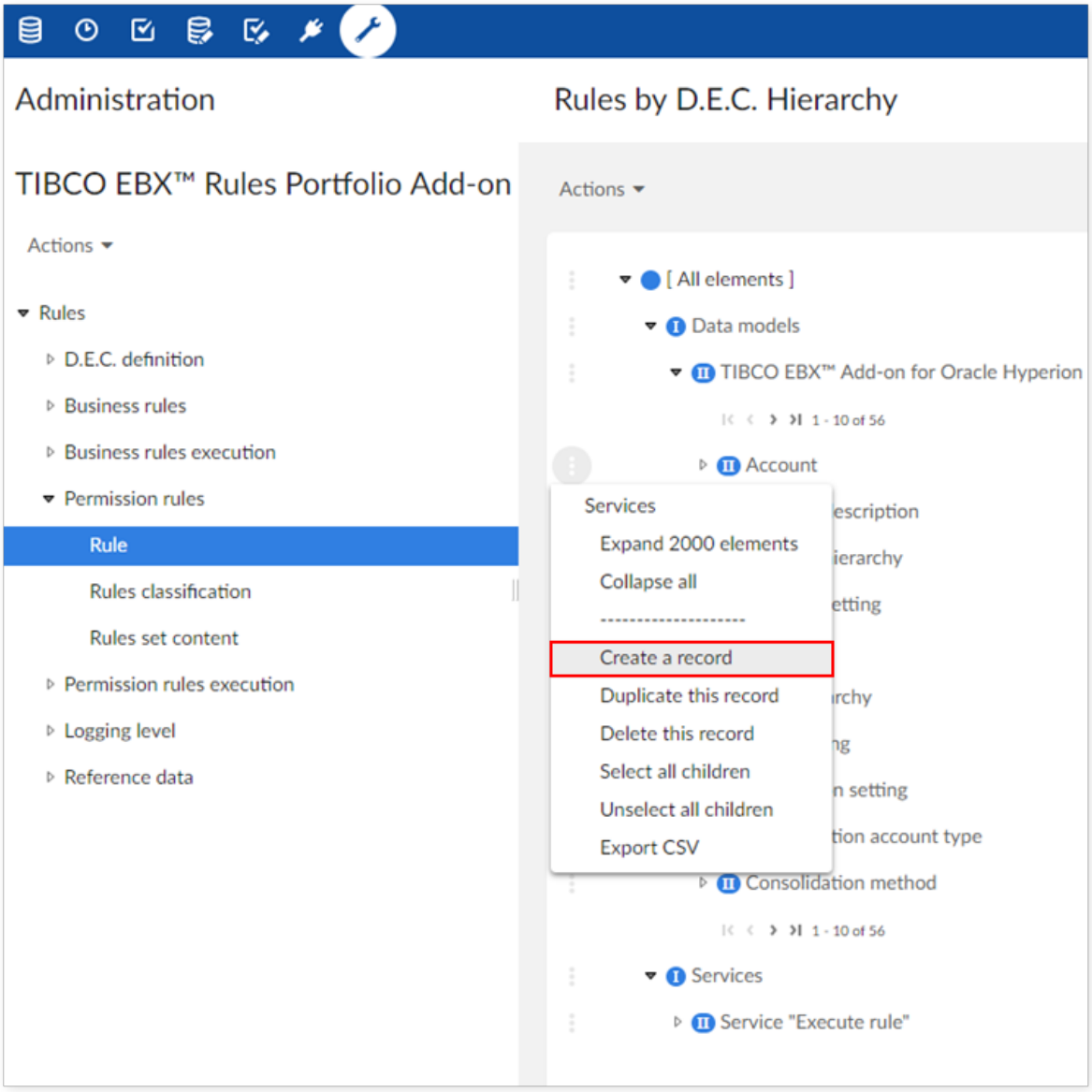
Step 2: If your data model is contained in a module, you only need to put your classic access rule classes in that module. If your data model is not contained in any module, you have to export all classic access rules to a .jar file (feel free to use any naming convention) and put in the folder '../_ebx-eclipse/catalina.base/shared/lib' (that contains "ebx.jar" file). Without this step the add-on cannot load your classic access rules.

Step 3: Restart the server.

Step 4: Configure your permission rules to use classic access rules. Please follow all steps below:

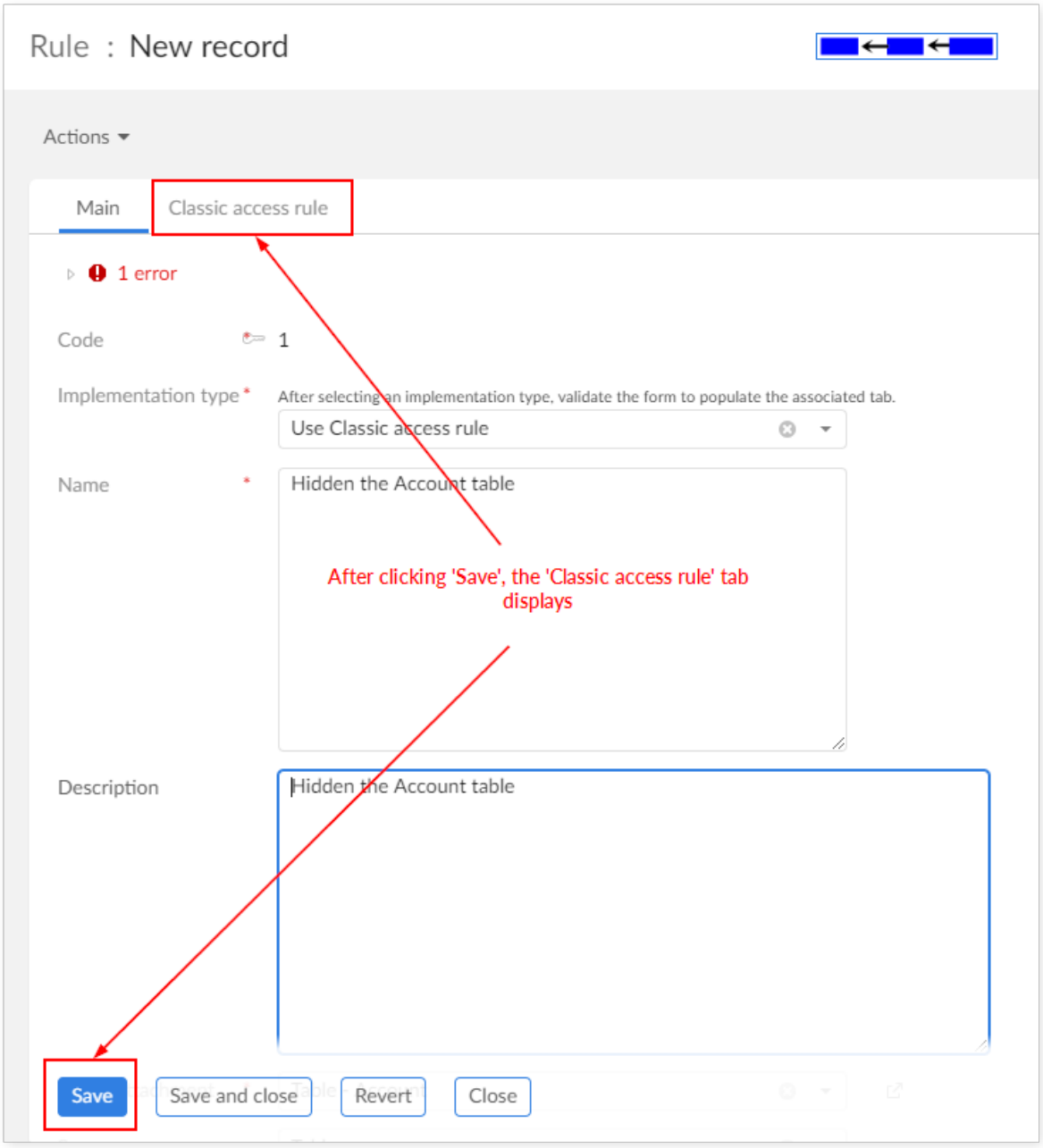
- Navigate to the 'Administration' tab → 'Business rules' → 'TIBCO EBX® Rules Portfolio Add-on' → 'Permission rules' → 'Rule' table and 'Create a record' on the appropriate D.E.C. This table's

view defaults to 'Rules by D.E.C.' and allows you to easily select the D.E.C. to which you want to add the rule.

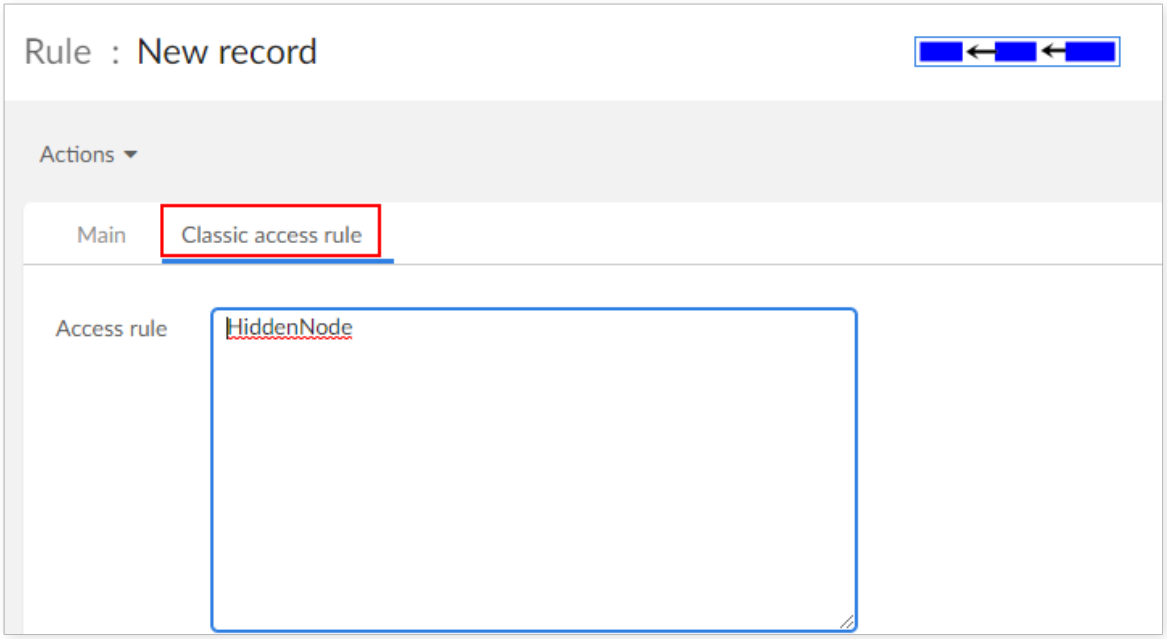


- Enter the required information and click 'Save'. The 'Classic access rule' tab displays.

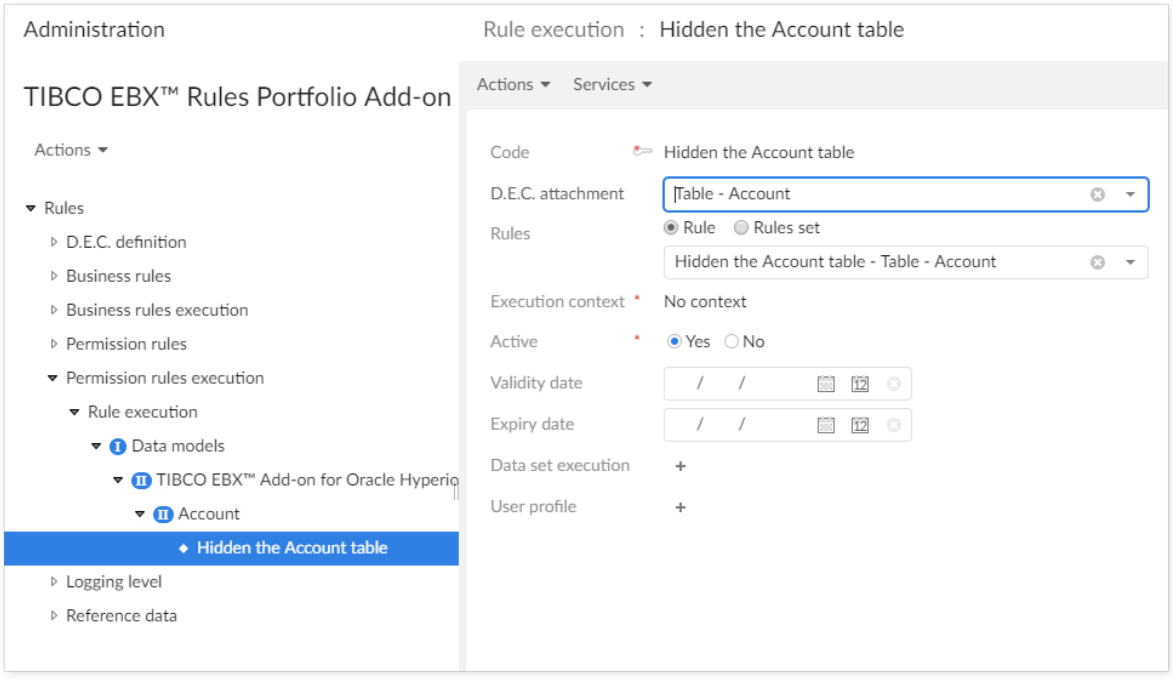
- Set the 'Implementation type' property to 'Use Classic access rule'.



- Click on the 'Classic access rule' tab and add the class path of your classic access rule to the 'Access rule' field.



- To create a rule execution, navigate to 'Permission rules execution' → 'Rule execution' and add a record to the same D.E.C. used for the rule definition. A rule execution determines the status of the associated rule (active or inactive) and a rule validity time period. In this case, the 'Rule' field is set to the permission rule that links to your classic access rule.



12.5 **Business rule on a table**

The rules are launched by different methods depending on their type:

- "Automated rule" is called by triggers on a table.
- "Validation rule" is called by triggers on a table or a constraint on field.
- "Manual validation rule" is called by the validation service.
- "Table set rule" is called by the "Execution rules" service.

Step 1: Create a trigger that calls the class RuleController in order to run the "Automated rule" and "Validation rule" (see more detail in the Java API documentation). Or you can use the class com.orchestranetworks.addon.rpfl.DefaultTableTrigger which is provided by the add-on.

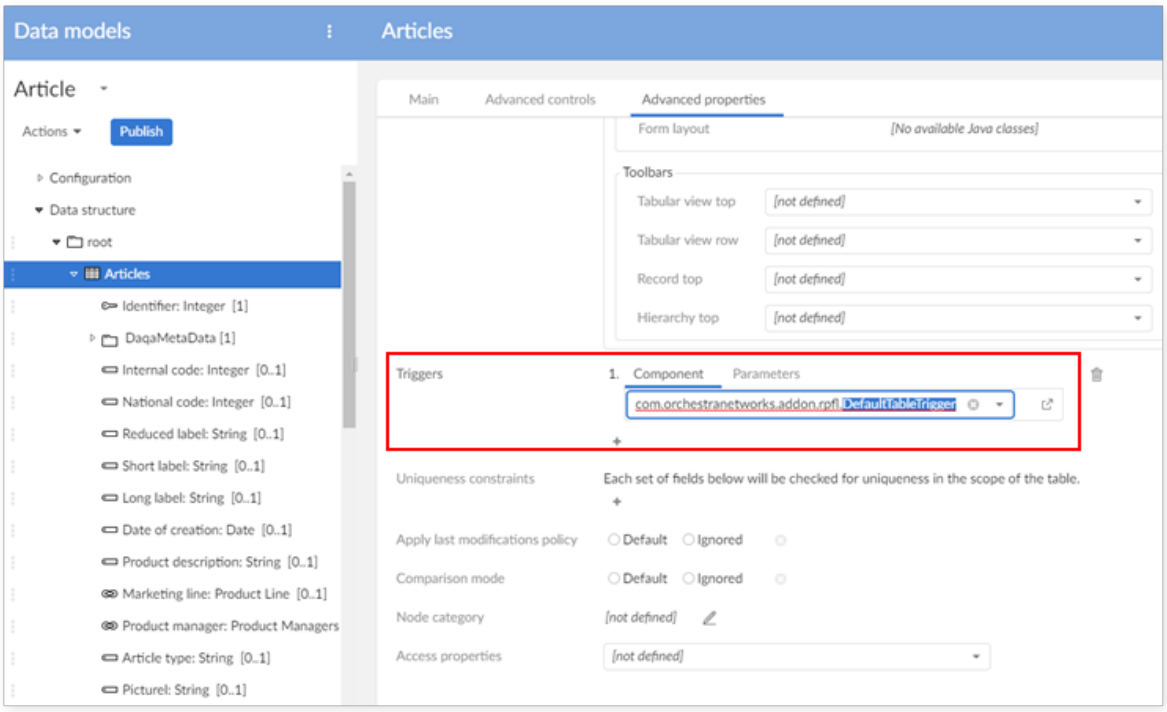
```
public class BusinessRuleTrigger extends TableTrigger
{
   public void setup(TriggerSetupContext arg0)
   {
   }

   public void handleAfterCreate(AfterCreateOccurrenceContext arg0) throws OperationException
   {
      ExecutionContextOnTrigger context = new ExecutionContextOnTrigger();
      context.setWhen(Hook.AfterDelete);
      context.setValueContextForUpdate(arg0.getProcedureContext().getContextForNewOccurrence(
         arg0.getTable()));
      context.setProcedureContext(arg0.getProcedureContext());
      context.setValueContext(arg0.getOccurrenceContext());

      RuleControllerFactory.getController().executeOnTrigger(context);
   }

}
```

Step 2: Add this class to the new table in data model as shown in this example:



Step 3: Create a class that implements the interface ConstraintOnTable and call the class RuleController in order to run the "Manual validation rule"(see more detail in the Java API documentation). Or you can use the class com.orchestranetworks.addon.rpfl.DefaultConstraintOnTable which is provided by the add-on.

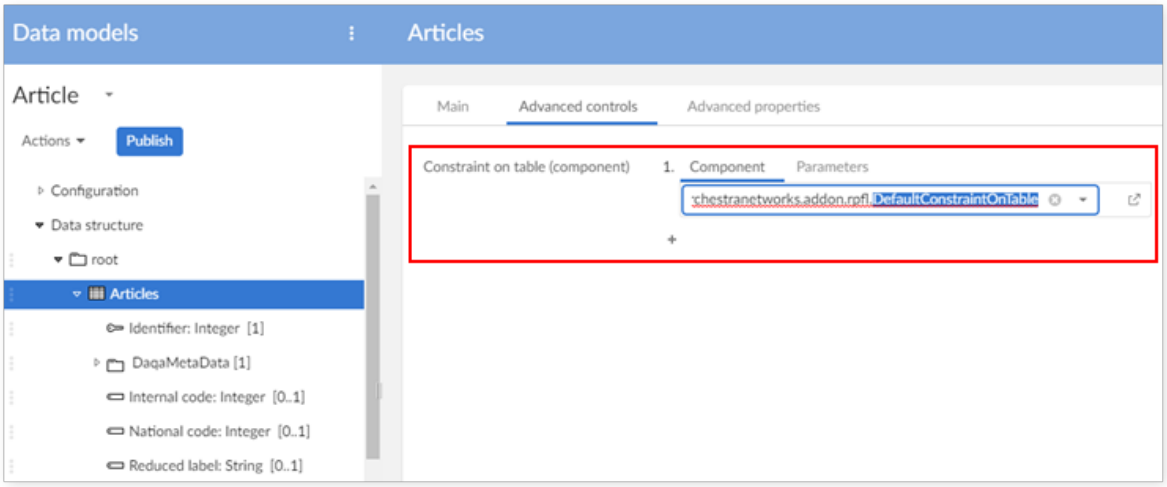
```
public class ManualValidationContrain implements ConstraintOnTable
{
    public void checkTable(ValueContextForValidationOnTable context)
    {
        ExecutionContextOnDemand executeOnDemandContext = new ExecutionContextOnDemand();
        executeOnDemandContext.setValueContextForValidationOnTable(context);
        try {
            RuleControllerFactory.getController().executeOnDemand(executeOnDemandContext);
        } catch (OperationException e) {
            e.printStackTrace();
        }

    }

    public void setup(ConstraintContextOnTable context)
    {
    }

    public String toUserDocumentation(Locale arg0, ValueContext arg1) throws InvalidSchemaException
    {
        return null;
    }
}
```

Step 4: Add this class to new table as shown in the following example:



The "**Execution rules**" service is available on the table if at least one configuration for this table exists in the "**TIBCO EBX® Rules Portfolio Add-on"** data set.

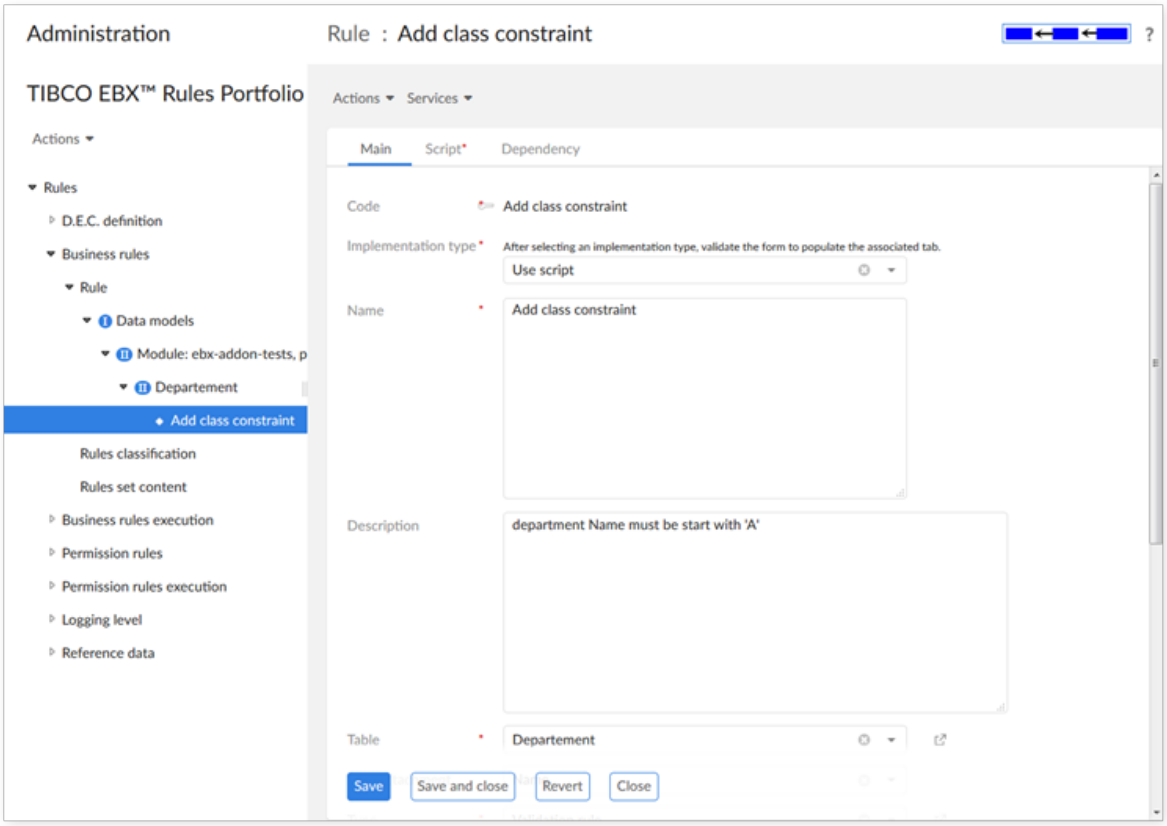
12.6 Dependency configuration

The add-on allows you to set dependencies on a configured validation rule that applies to a field. This section focuses on how you include these dependencies in a rule configuration. For more information on dependencies, see the standard EBX® documentation.

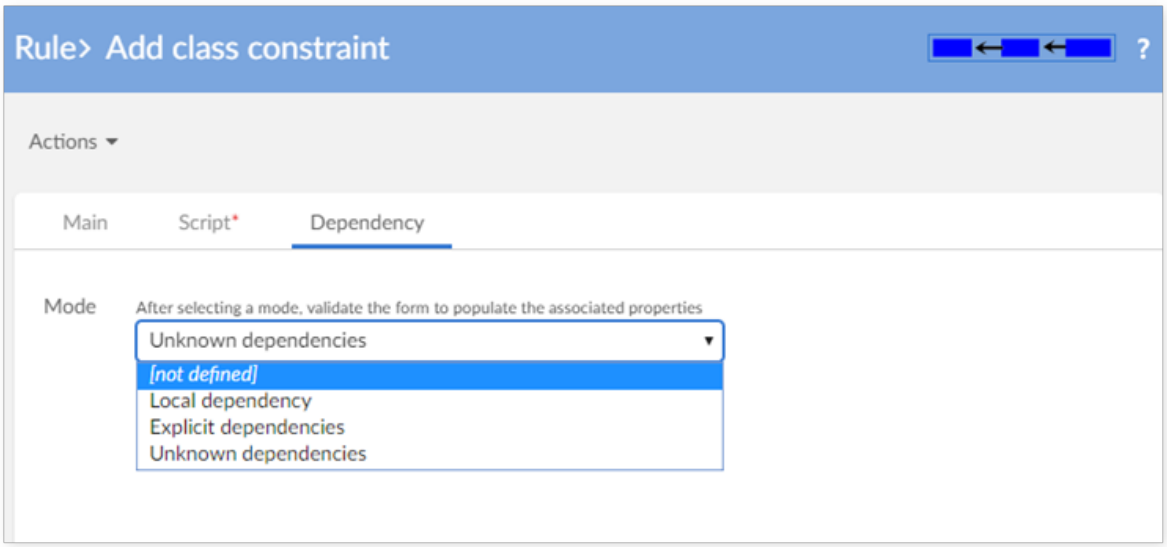
Special Notation	
✔	Dependencies on computed values are not allowed. The add-on automatically sets constraints applied on elements defining a value function to the 'Unknown dependencies' mode. Consequently, if a constraint attempts to specify an explicit dependency, an error displays. To avoid this error, you must keep your D.E.C.'s current. If you want to change one field on your data model to computed value or vice versa, you have to follow the steps below: • Make change on your data model. • Go to the 'D.E.C. (Data Element Concept)' table in the TIBCO EBX® Rules Portfolio Add-on administration area. • Find the D.E.C. 'Field' that corresponds to the field changed on your data model. • Update 'Computed value' field of the corresponding D.E.C. • Come back to your data model and publish it.

To set a dependency:

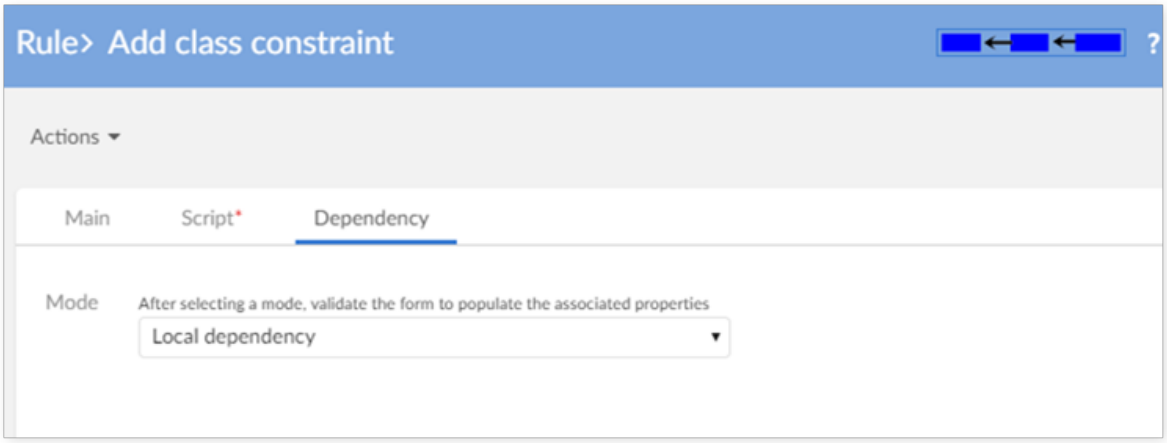
- Create a validation rule that applies to a field (and executes by an 'On constraint' event). After creating and saving the rule, the 'Dependency' tab displays. You can now configure a dependency based on your needs.

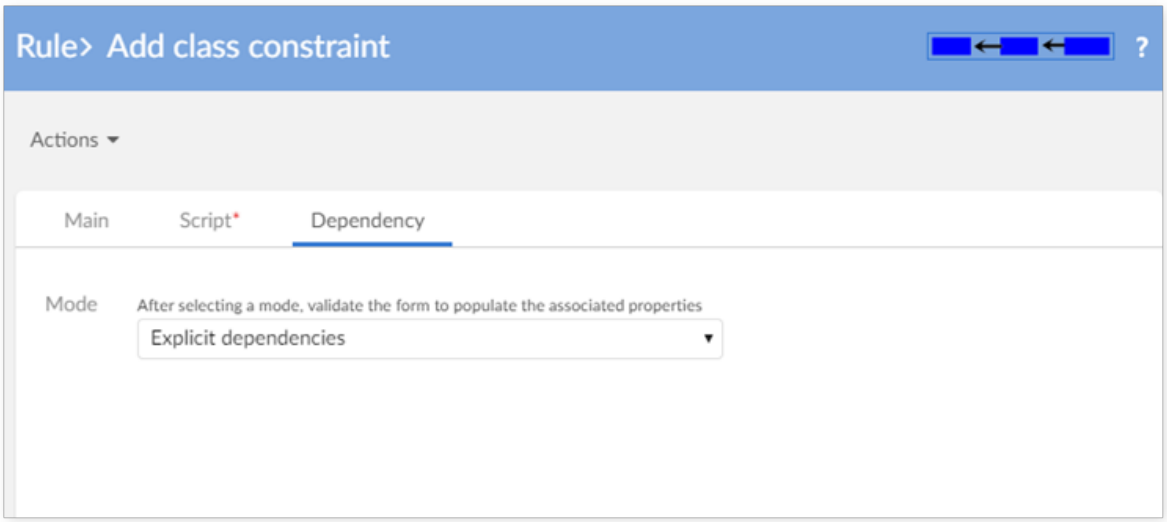


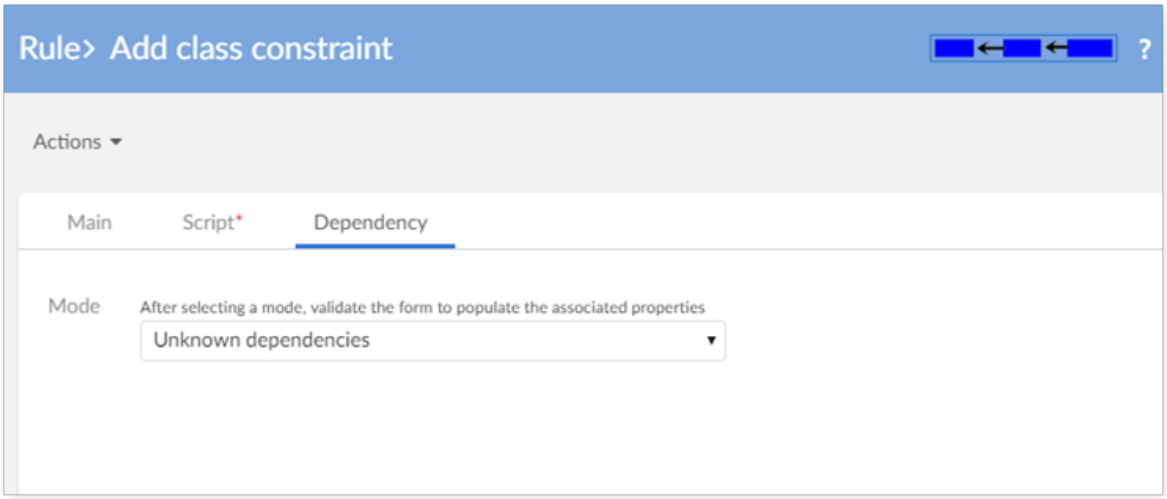
- Choose the dependency mode that best suits your requirements. Refer to the 'configuring a business rule' section above for more information on each dependency mode.



- As shown below, if you use either the 'Local dependency', or 'Unknown dependencies' modes, configuration completes after you click 'Save'. However, if you select 'Explicit dependencies', refer to the following steps to complete configuration.

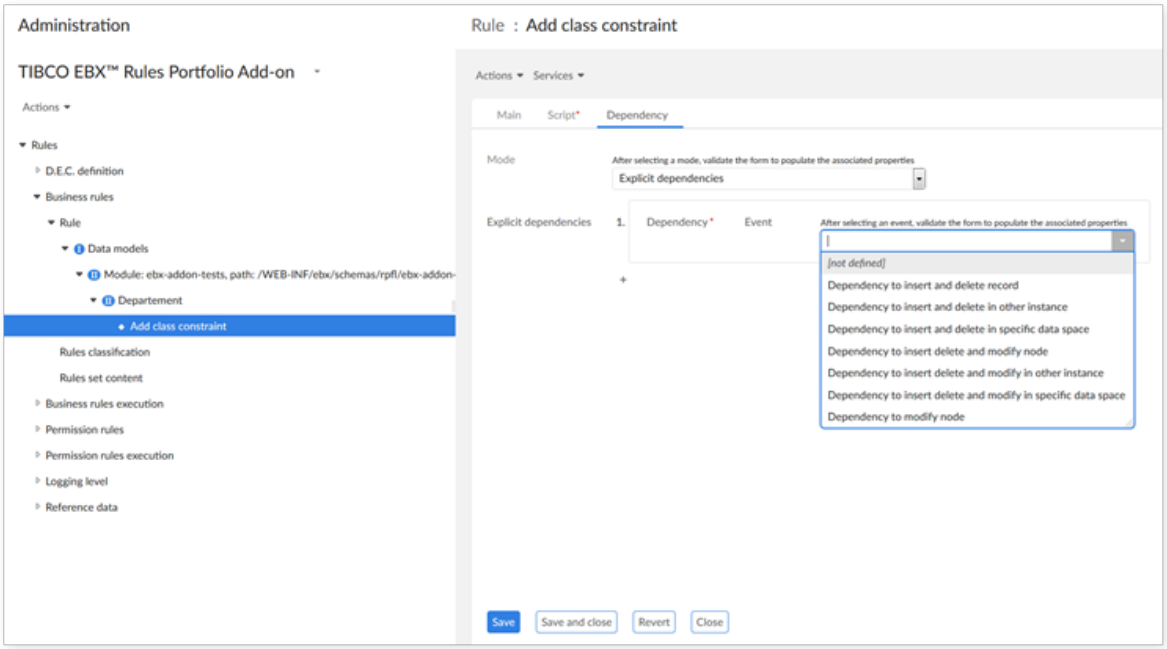




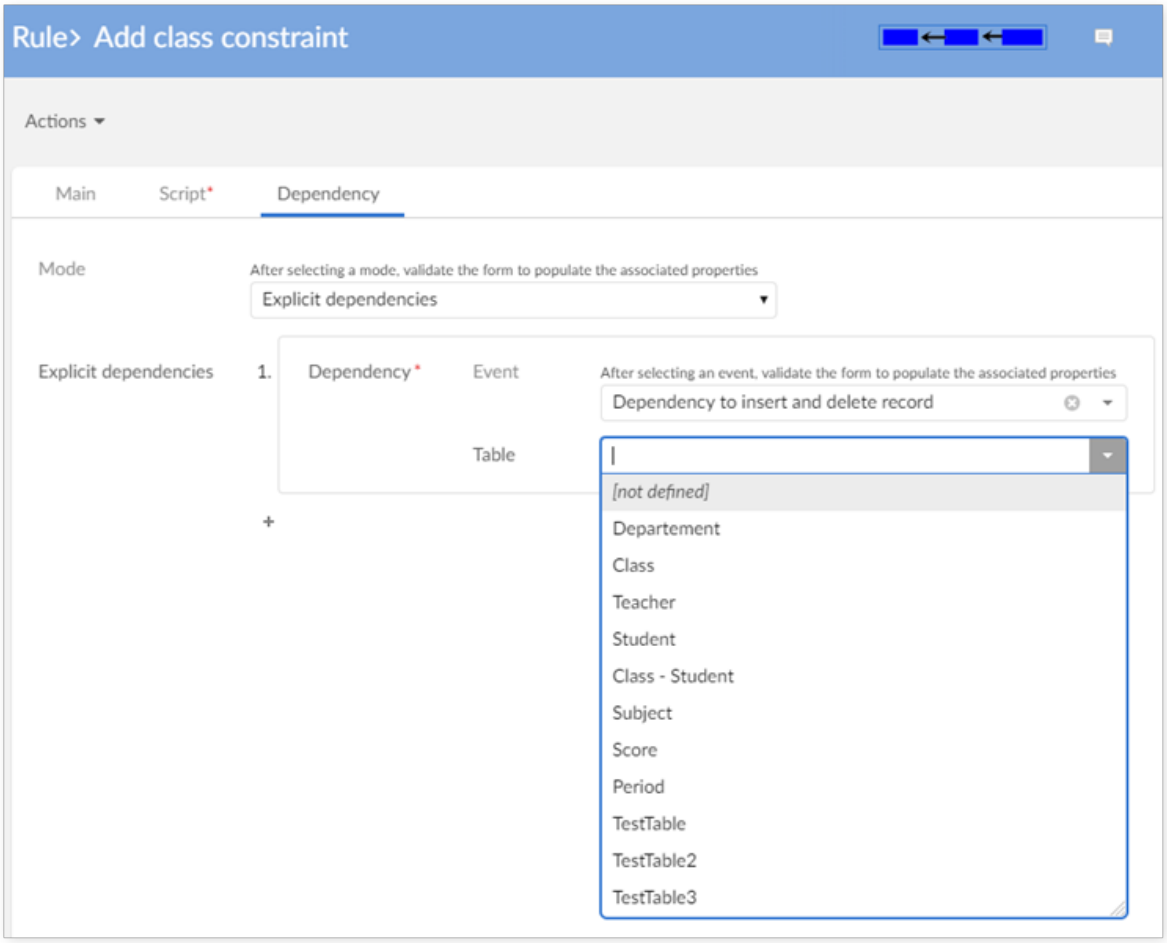


- In 'Explicit dependencies' mode, you can declare one or more dependencies that suit your needs. Based on the dependency you choose, fields display that allow you to indicate the node your

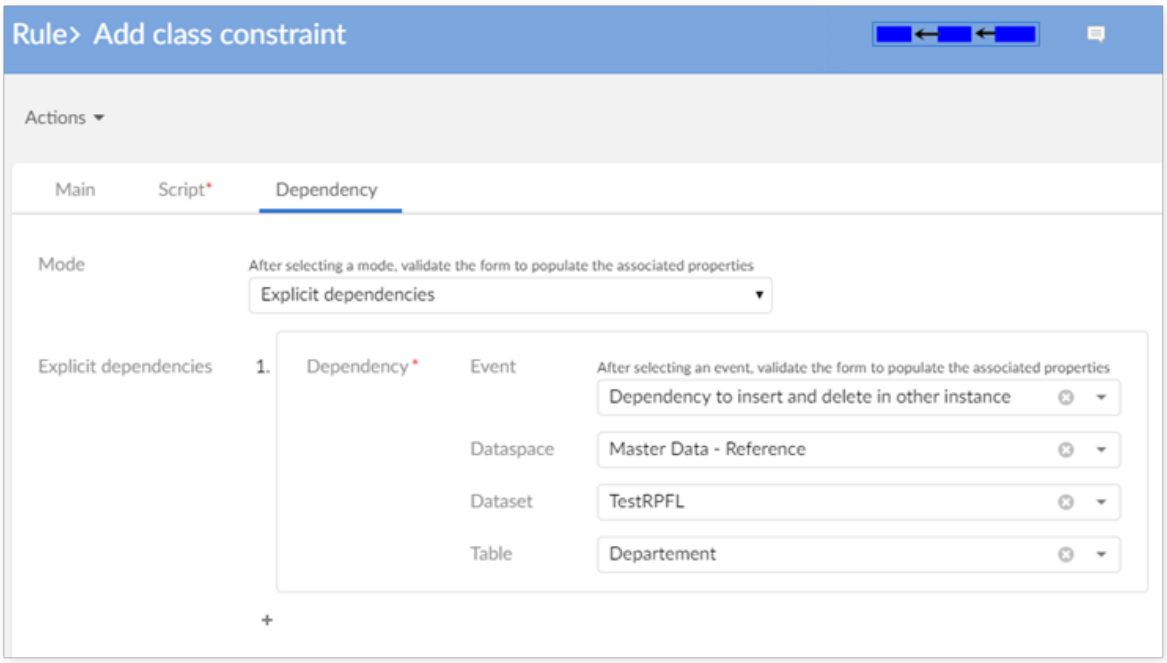
constraint depends on. The remainder of this section describes these options. To add multiple dependencies, click the '+' icon and repeat the process.

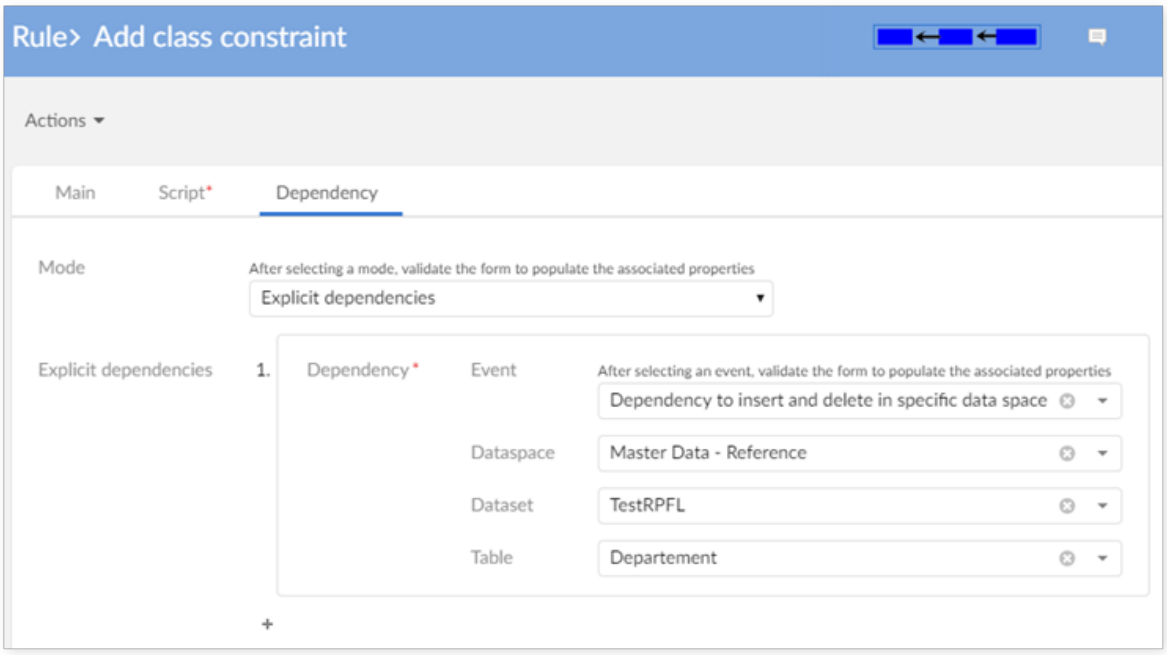


With the 'Dependency to insert and delete record' event, you only need to choose the correct table node.

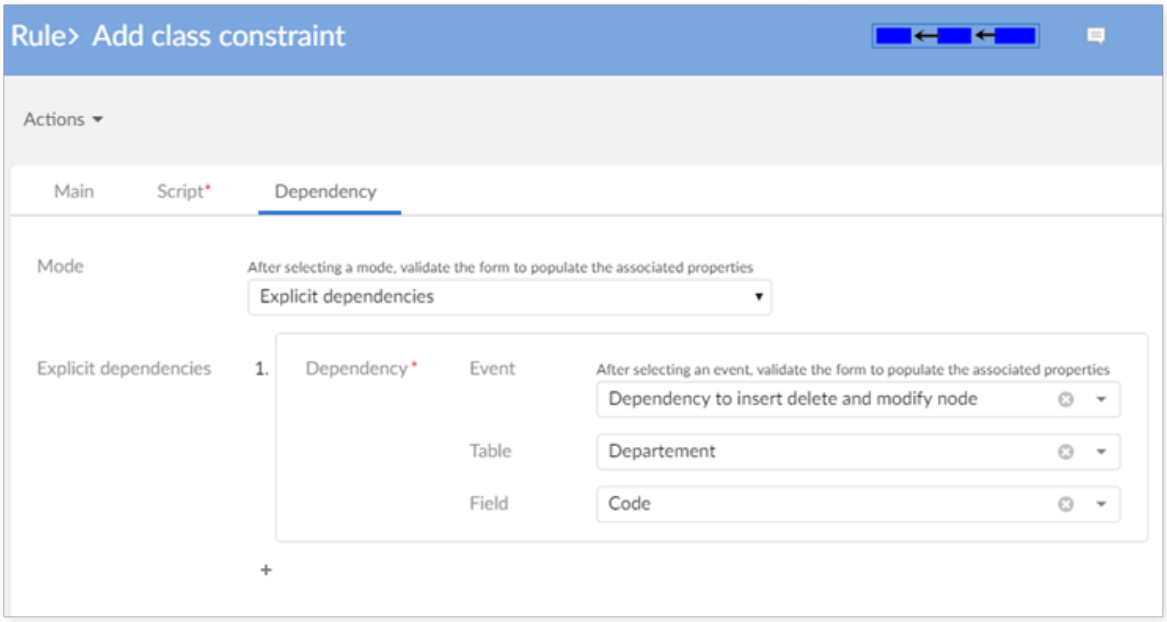


With the 'Dependency to insert and delete in other instance' and 'Dependency to insert and delete in specific data space' events, you have to select-in order: Data space → Data set → Table.

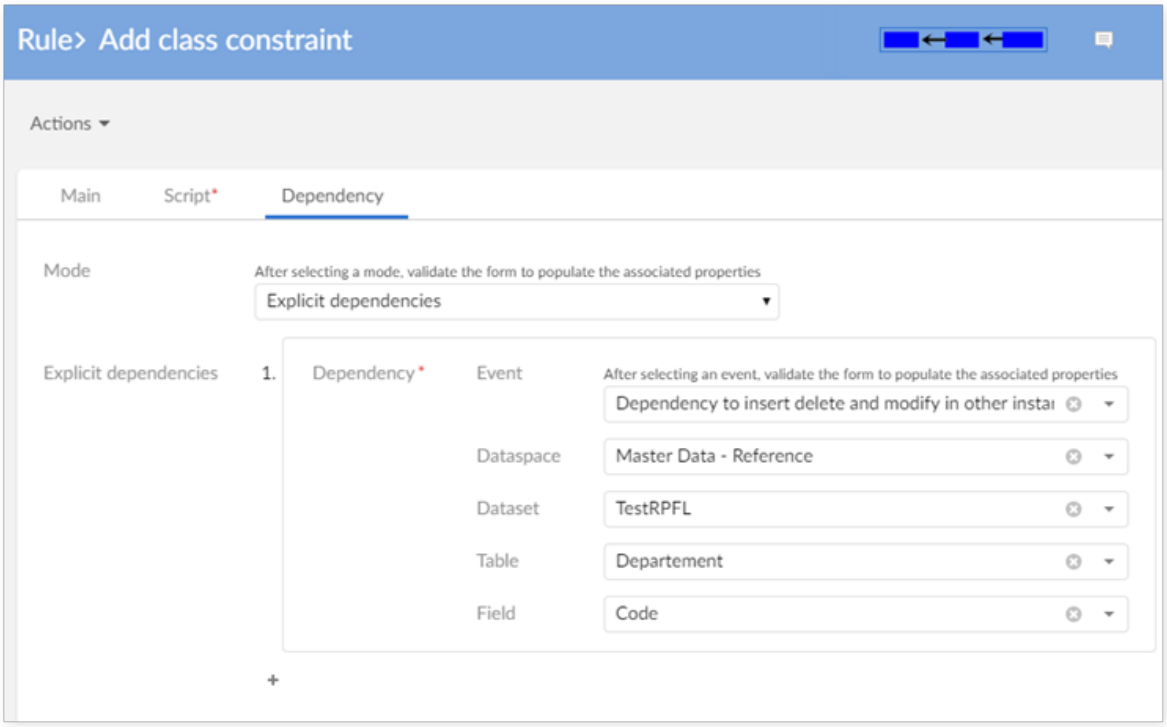


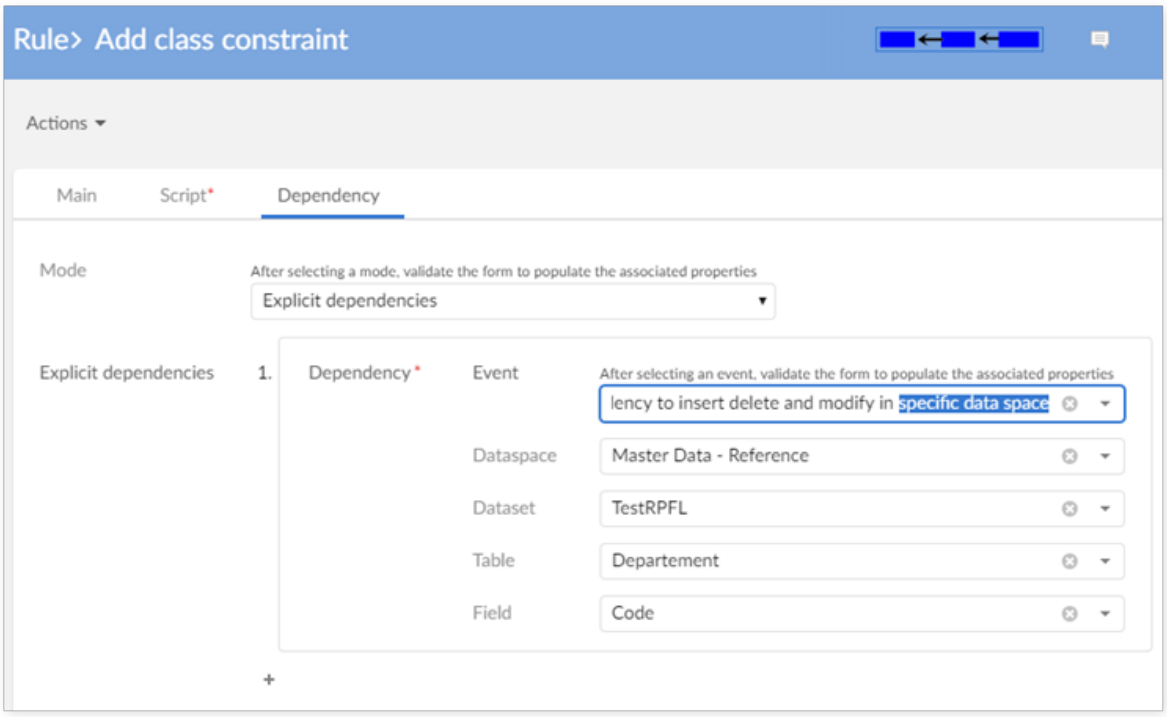


With the 'Dependency to insert delete and modify node' event, you have to select-in the following order: Table → Field.

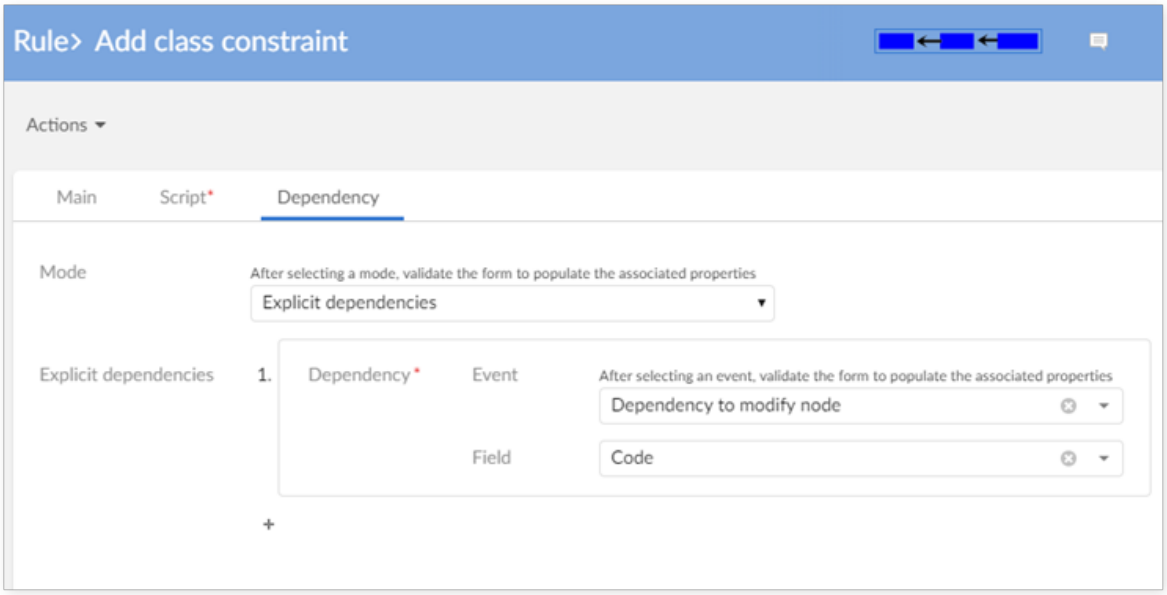


With the 'Dependency to insert delete and modify in other instance' and 'Dependency to insert delete and modify in specific data space' events, you have to select-in the following order: Data space → Data set → Table → Field.





With the 'Dependency to modify node' event, you have to choose one field node.

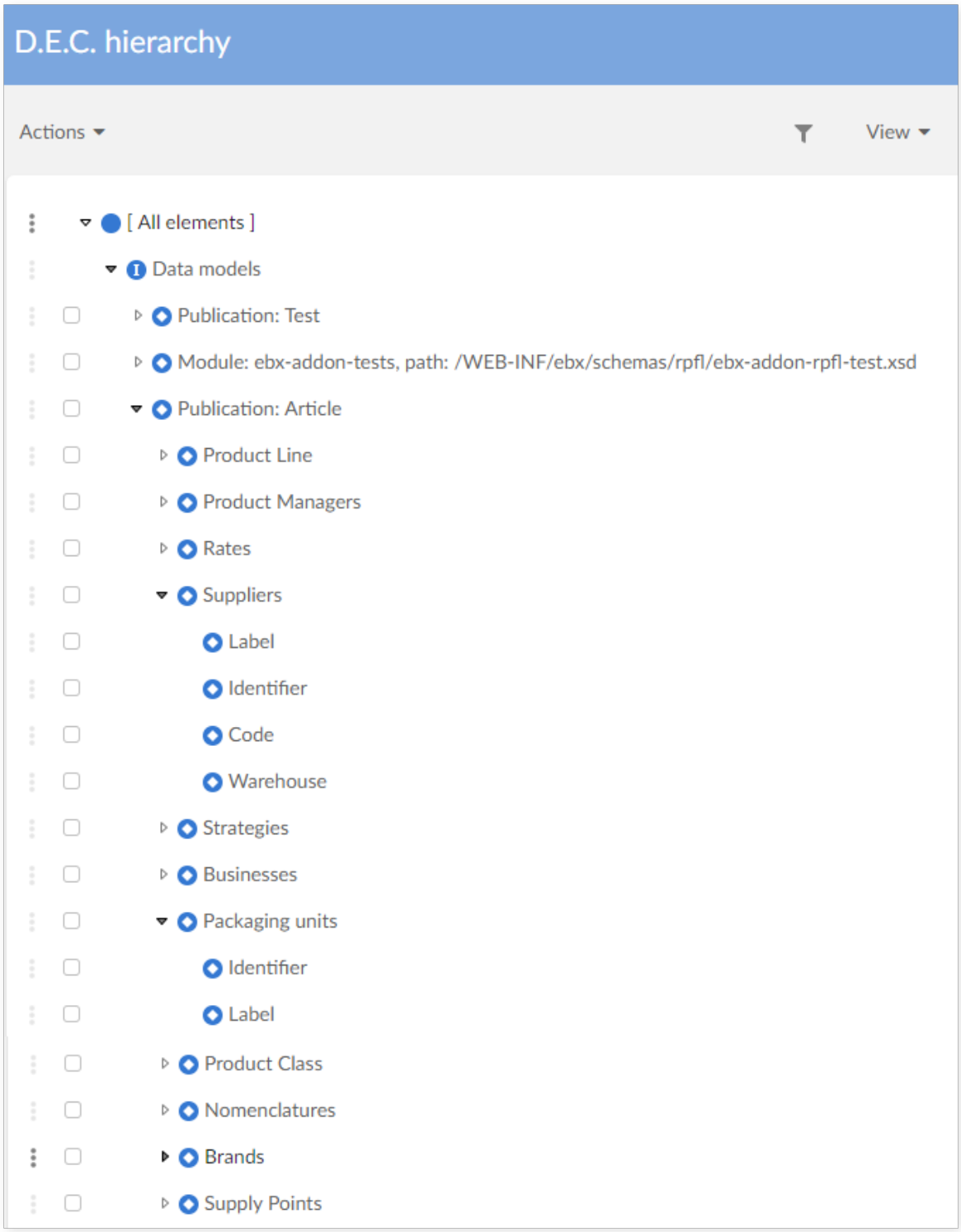


12.7 **Hierarchy view related to D.E.C.**

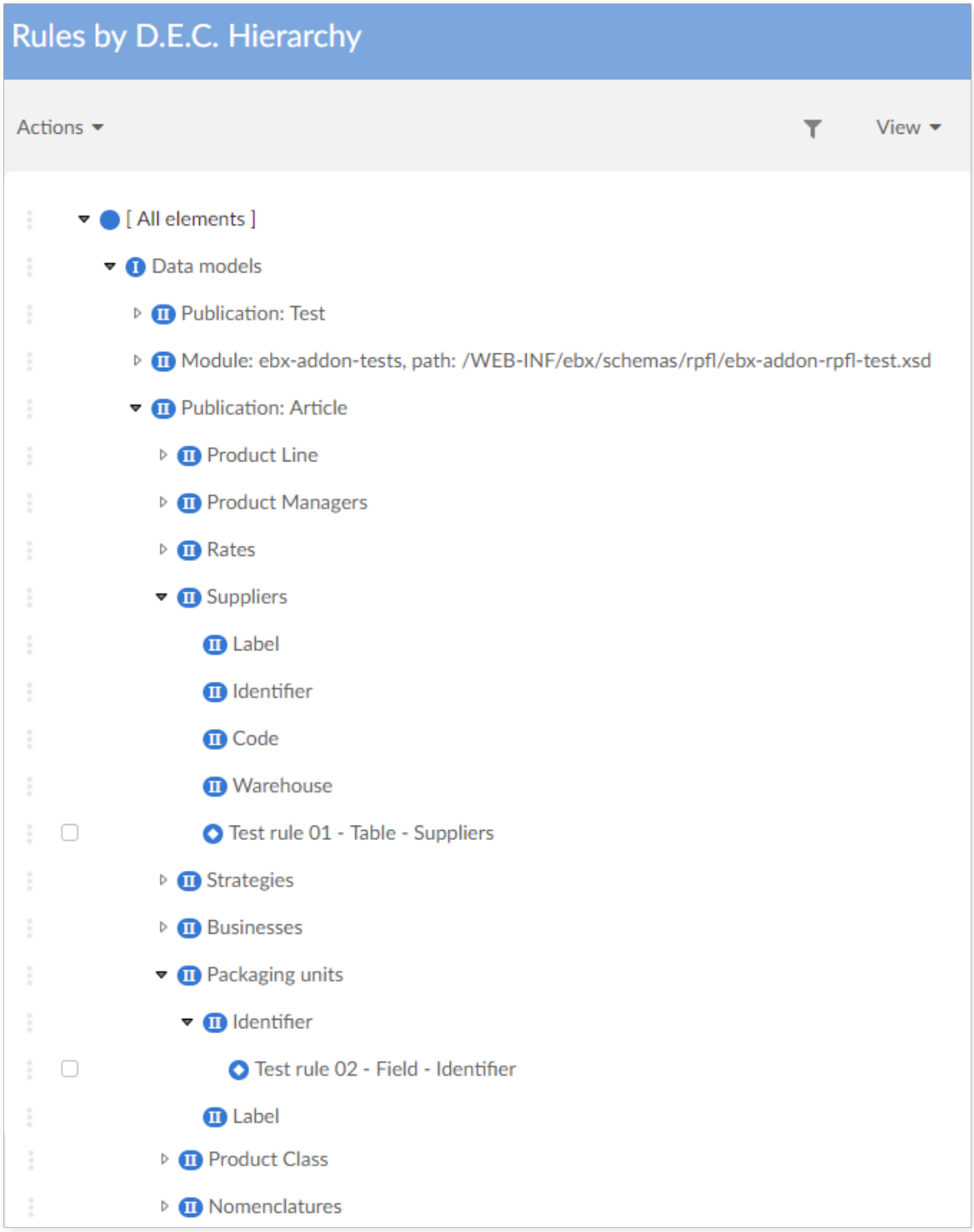
Beginning in release 1.5.0, the add-on provides hierarchy views that show D.E.C.s, rules and rule execution using the following structure: Data model → Table → Field.

'D.E.C. hierarchy' view

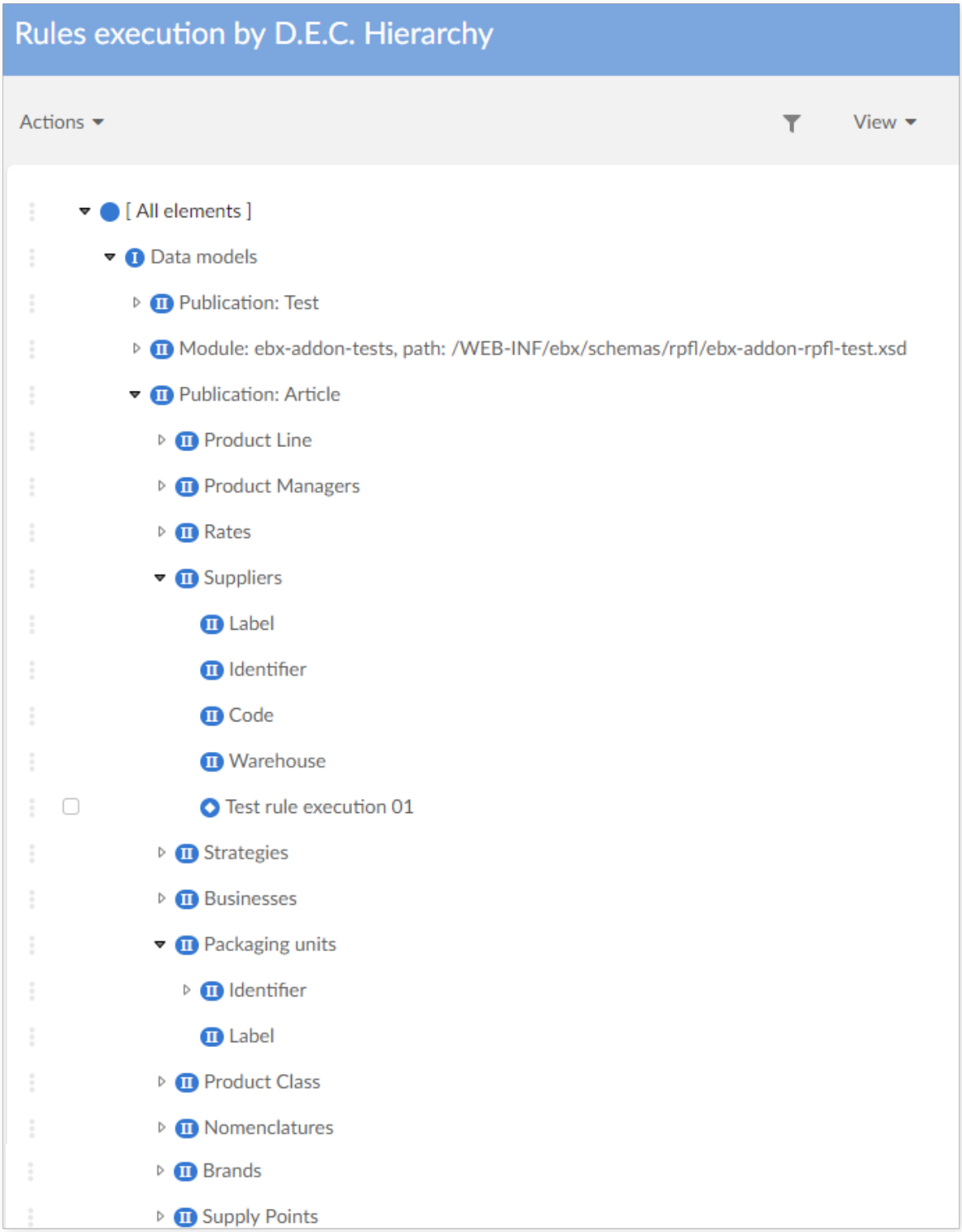
The 'D.E.C. hierarchy' view shows D.E.C.s based on the 'D.E.C. type' value. All D.E.C. types in this view are graded into a hierarchy view.



'Rules by D.E.C. hierarchy' view



'Rules execution by D.E.C. hierarchy' view



Special notation	
✔	The 3 new hierarchy views 'D.E.C. hierarchy', 'Rule by D.E.C. hierarchy', and 'Rule execution by D.E.C. hierarchy' display are not available on old publication of the add-on. Please read the 'Known limitation' section for more information.

12.8 Service permission on a data model

The add-on sets permission rules on services using an 'Action permission rule'. You can declare a service on a data model and directly in the 'module.xml' file. You set service permissions based on how the service is declared.

Declare a service on a data model

Attach the com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission class to a service definition on schema, this class specifies in which conditions the service may be executed.

This class must be declared within a service declaration in schema, at attribute class:

```
<xs:complexType name="MyService">
    <xs:annotation>
        <xs:appinfo>
            <osd:service resourcePath="/services-dispatcher?action=MyService"
                class="com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission" >
                <serviceName>MyService</serviceName>
                ...
            </osd:service>
        </xs:appinfo>
        ...
    </xs:annotation>
</xs:complexType>
```

Declare a service on module.xml

Create a class that extends DefaultRulesServicePermission. Define the service name by implementing the abstract method getServiceName()

```
public final class MyServicePermission extends DefaultRulesServicePermission
{
   public String getServiceName()
   {
      return "MyService";
   }
}
```

Declare this class as service permission for the declared service in the module.xml.

```
<service name="MyService">
    <resourcePath>/services-dispatcher?service=MyService</resourcePath>
        <instanceActivation all="true">
            <scope availableOnChild="true">
                ...
                <permission class="com.foo.MyServicePermission"/>
            </scope>
    </instanceActivation>
    ...
</service>
```

12.9 Environment configuration

This section guides you about some settings of the environment configuration.

Setting logging queue size parameters

By default, the logging queue size is set to 100,000 records. Once the maximum size is reached, rule execution has to wait until new queue space is available.

You can change the maximum queue size by configuring its parameter in the JAVA_OPTS java environment variable. To set the logging queue size to '200000':

JAVA_OPTS="-Debx.properties=$EBX_HOME/ebx.properties -Debx.home=$EBX_HOME -Xmx512m **-Drpfl.logging.queuesize=200000**"

Setting parallel rules execution parameters

The default number of threads used to run the 'Manual validation rules' in parallel mode is set to up to '5'.

You can change this value by configuring a parameter in the java environment variable. For example, to set the number of threads to up to '2', change the variable's parameter as shown below:

JAVA_OPTS="-Debx.properties=$EBX_HOME/ebx.properties -Debx.home=$EBX_HOME -Xmx512m **-Drpfl.max.threadpool=2**"

Setting associated queue parameters:

By default, the associated queue is set to 10,000. It is possible to change this value by configuring a parameter in the java environment variable. For example, to set size of this queue to '100000':

JAVA_OPTS="-Debx.properties=$EBX_HOME/ebx.properties -Debx.home=$EBX_HOME -Xmx512m **-Drpfl.threadpool.queuesize=100000**"

Scripting Language

CHAPTER 13

JavaScript-based language

JavaScript is a lightweight, interpreted and object-oriented language. The Scripting Language is based on JavaScript and has the same syntax and library. Additionally, you can use predefined objects to perform basic EBX®-specific operations. For example, you can query data, create/update/delete records and write to a log. The following image shows a sample.

```
/**
 * Hello world!
 */

function add(x,y){
    var resultString = "Hello, The result of your math is: ";
    var result = x + y;
    return resultString + result;
}

var addResult = add(3,2);

```

This document focuses on the predefined objects provided by the Scripting Language. If you need help with JavaScript you can easily find reference information on the World Wide Web that discusses the syntax and other features.

You can use a script to directly return a result. The add-on uses this result as the rule result. For example, the result could be true/false for a business rule, or a specific permission for a permission rule.

```
var pk = record.getPrimaryKey();
var classQuery = query.getQuery('/root/ClassStudent');
var classes = classQuery.records("./FKClass='"+pk+"'");
if(classes && classes.length>10){
    return false;
}
return true;
```

For example, the result of the following script is 'false' if classes length >10, otherwise this returns 'true'.

CHAPTER 14

Predefined objects

This section describes the predefined objects used in the Scripting Language. The predefined objects are highlighted in the editor as default JavaScript reserved words.

This chapter contains the following topics:

14.1 Accessing the current record's property values

In order to access record data, use the 'record' object. The current record is converted to a script bean object and assigned to the 'record' object. The conversion does as follows:

- A normal field is converted to a property having the same name of 'record'.
- A multiple value field is converted to an array then assigned to a property having the same name of the 'record'.
- Groups convert to another script bean and are then assigned to a property having the same name as the 'record'.

- For a foreign key field, the system looks for the reference record first. If the reference record exists, it will be converted to another script bean before being assigned to the property of the 'record'.

For example, with the given structure of the 'Strategies' table, the script bean of one record on this table is presented as follow:

The following example shows how to access different record properties in the 'Strategies' table.

```
/* Get formatted primary key of record */
var pk = record.getPrimaryKey(); //1

/* Get 'Memento active' value */
var mementoActive = record.actifMemento;  //true

/* Get 'Pullable' value in 'Smartphone active' group */
var smartphoneActivePullable = record.actifSmartphone.enlevable; //true

/* Get short label of the strategy's article */
var mementoActive = record.article.libelleCourt;
```

The Scripting Language supports "navigation" through foreign keys. As shown in the following image, the 'Strategies' table's script bean presents all foreign keys as objects instead of strings. These objects contain all associated records and not just the associated records' primary keys.

```
record{
    Identifiant: 1,
    Memento active: true,
    E-commerce active{
        Pullable: false,
        Deliverable: false
    }
    Smartphone active{
        Pullable: true,
        Deliverable: true
    }
    Article: object,
    Depots: object
}
```

The following example shows how to access 'article.libelleCourt' object's data.

```
/* Get short label of the strategy's article */
var mementoActive = record.article.libelleCourt;
```

In order to access to a record attribute whose name contains a dot such as 'loc.code' field in 'Article' table, you must use the format record.["loc.code"]. The following example shows how to access 'loc.code' field's data.

```
/* Get value of 'loc.code' field on 'Article' table */
var loc_code = record.article["loc.code"];
```

Even though the foreign key is presented as an object, you only need to assign another associated primary key value to change its association. So, when you 'get' the foreign key value the script returns an object containing all of the associated records. When you 'set' the foreign key value, the parameter is only the primary key of the associated record.

```
var newArticleID = "newArticleID";
record.article = newArticleID;
```

Special Notation	
✕	You cannot add a new value to a multi-valued foreign key field. You can only update or delete the old field value.

14.2 Updating the current record

The script bean is just a copy of the record. Modifications made in the script bean do not have any affect on the corresponding record until you call the 'save()'method. However, this only works in certain contexts. For example, only an 'Automated rule', or a 'Table set rule' can modify data. Additionally, permissions may prevent a rule from executing. If you are using a different type rule, the 'save()' method returns 'false' and no modifications are committed.

Additionally, you can disable triggers by calling the 'save(isActivated)' method with a 'true' or 'false' parameter. These enable and disable the trigger, respectively. If you call this method without a parameter, triggers default to enabled.

```
/* Get article short name */
var shortName = record.libelleCourt; //old name

/* Update article short name */
record.libelleCourt = 'new name';

/* But the short name of corresponding record in table Article is still the old name */
/* Call save() mothod to commit this change to corresponding record in table Article */

record.save(); // now the name of corresponding record in table Article is changed to the new name
```

14.3 Deleting the current record

You can call the 'del()' method to delete a record. However, just as with the process of updating a record, you can only modify data in certain contexts. You must have sufficient permission and the rule type should be either an 'Automated rule' or a 'Table set rule'. If the correct conditions are not met, the method returns 'false'.

The following example shows how to delete the current employee record:

```
19
20  /* Delete the current record */
21  var deleteResult = record.del(); // 'true' if delete successfull, 'false' otherwise
22
```

The 'del()'method returns 'true' if it deletes corresponding record successfully, otherwise it returns 'false'.

14.4 Querying data in the current table

The Scripting Language uses the 'query' object to query data in the current table. It also supports the 'records(XPathPredicate)' function to get a list of records - which, are converted to object beans - in the current table based on the specified predicate. To get all records in the current table, set the predicate to 'null' (query.records(null)). If you want to set conditions replace 'null' with an XPath predicate. To get the number of rules use JavaScript's 'length' property.

The following example checks the number of records in the current table. If the number of records is smaller than 5, it returns 'true'. Otherwise, it returns 'false'.

```
var size = query.records(null).length;

if(size < 5){
    return true;
}

return false;

```

14.5 Querying data in the current data set

When defining a rule that involves querying data in a different table from the same data set you most likely "navigate" through the foreign keys. To accommodate this process, the Scripting Language contains the 'getQuery(tablePath)' method.

The script in the following example states, "One article cannot be the substitute article for more than 10 articles."

```
var pk = record.getPrimaryKey();
var articleQuery = query.getQuery('/root/Article');
var articles = articleQuery.records("./articleSubstitution='" + pk + "'");

if(articles && articles.length>10){
    return true;
}

return false;
```

The following bullet points breakdown the content of the above script:

- The 'pk' variable returns the current record's primary key.
- The 'articleQuery' variable uses the 'getQuerry()' method to query the 'Article' table. Notice that input parameter uses the table's path.
- The 'articles' variable is a query object that queries the 'Article' table records and uses the 'records(XPathPredicate)' function to return all articles that use the current article as the substitute article.
- The final statement counts the number of object beans contained by the 'articles' object. If the number is larger than 10, the rule returns 'true', otherwise it returns 'false'.

14.6 Querying data from a data set in the current data space

The 'query' object allows you to query data in a table from a specific data set in the current data space. To accomplish this, the Scripting Language contains the 'getQuery(String datasetName, String tablePath)' method. The query object returned from this method allows you to modify data on a given data set in the current data space.

The following shows an example query:

```
var otherDatasetQuery = query.getQuery('Chart-API','/root/DataType');
var dataType = otherDatasetQuery.records("./label='DateTime'");
if(dataType && dataType.length >= 1){
    return true;
}
var newDataType = otherDatasetQuery.newRecord();
newDataType.clazz = "java.util.DateTime";
newDataType.label = "DateTime";
newDataType.save();
```

The following bullet points breakdown the content of the above script:

- The 'otherDatasetQuery' variable uses the 'getQuerry(String datasetName, String tablePath)' method to query the 'DataType' table from the 'Chart-API' data set in the current data space. Notice that input parameter uses the data set's unique name and the table's full path.
- The 'dataType' variable is a query object that queries the 'DataType' table records and uses the 'records(XPathPredicate)' function to return all data types that have the 'DateTime' label.
- If the 'dataType' object is not null and contains one or more objects, the rule returns true. Otherwise, the 'newRecord()' method creates the 'newDataType' variable. After setting the value for each of the 'newDataType' variable's fields, the 'save()' method commits the new record to the 'DataType' table.

14.7 Querying data from a specified data set and data space

You can use the 'query' object to query data in a table from a specific data set and data space. To support this functionality, the Scripting Language contains the 'getQuery(String homeKey, boolean isBranch, String datasetName, String tablePath)' method. However, the query object returned by this method does not allow you to modify data.

The following examples will help you to understand this feature more deeply:

```
var otherDatasetQuery = query.getQuery("TestDataspace",true,"Chart-API","/root/DataType");
var dataType = otherDatasetQuery.records("./label='DateTime'");
if(dataType && dataType.length >= 1){
    return true;
}
return false;
```

The following bullet points breakdown the content of the above script:

- The 'otherDatasetQuery' variable uses the 'getQuerry(String homeKey, boolean isBranch, String datasetName, String tablePath)' method to query the 'DataType' table from the 'Chart-API' data set in the 'TestDataspace' data space. Notice that input parameter uses the data space's identifier, the data set's unique name and the table's full path.
- The 'dataType' variable is a query object that queries the 'DataType' table records and uses the 'records(XPathPredicate)' function to return all data types that have the 'DateTime' label.
- If the 'dataType' object is not null and contains one or more objects, the rule returns true. Otherwise, it returns false.

14.8 Creating new records in the current table

The 'newRecord()' method returns the object bean for a new record on the current table. You can then assign values to each of the record's fields. As shown below, you must use the 'save()' method to commit the new record to the current table.

```
var newRecord = query.newRecord();

newRecord.Name = "Project_1";
newRecord.Code = "123";
newRecord.save();

```

14.9 Writing to the log file

As the complexity of your code increases, keeping track of each line becomes more and more important. To assist you in this endeavor the Scripting Language provides the 'log' object to write information to the log file.

The following example shows how to check the current record's name. If the name is not 'null', this returns 'true'. However, anything that returns a value of 'false' gets added to the log file. Four methods - 'error(message)', 'info(message)', 'debug(message)' and 'warn(message)' - write messages to the 'ebx-addon-common.log' file with respective severity levels of 'ERROR', 'INFO', 'DEBUG' and 'WARNING'. You can find that log file in configured logs folder. See section 'Configuring the EBX® logs' in EBX® document for more detail.

```
var name = record.Name;

if(name !=null){
    return true;
}

log.error('the name is null');// write log message to log file

return false;

```

```
ure javaClass="com.orchestranetworks.addon.rpfl.service.rulepublication.RulePublicationProcedure" />

15:56:12,916 ICT INFO   log.fsm 0985:0012 [http-bio-9080-exec-3]     0:0:0:0:0:0:0:1 02FEC7C812AD73ACDB9C
15:56:12,916 ICT INFO   log.fsm 0985:0012 [http-bio-9080-exec-3]     0:0:0:0:0:0:0:1 02FEC7C812AD73ACDB9C
15:56:14,319 ICT INFO   log.fsm 0985:0012 [http-bio-9080-exec-3]     0:0:0:0:0:0:0:1 02FEC7C812AD73ACDB9C
15:56:16,686 ICT INFO   log.fsm 0985:0012 [http-bio-9080-exec-3]     0:0:0:0:0:0:0:1 02FEC7C812AD73ACDB9C
15:56:20,391 ICT ERROR  log.wbp.ebx-addon-common.main 0985:0012 [http-bio-9080-exec-5] The name is null!
15:56:20,392 ICT ERROR  log.kernel 0985:0012 [http-bio-9080-exec-5] Error when executing the following ac
est="Procedure" branch="Reference" date="2015-07-31T15:56:20.360" entryId="0322C080-3762-11E5-B982-00D716
WebUser login="Uadmin" ipAddress="0:0:0:0:0:0:0:1" sessionId="02FEC7C812AD73ACDB9C2B65D28C3316:0" />
ure javaClass="com.orchestranetworks.manager.core.context.CreateOccurrenceProcedure" executionInfo="Creat
e message="aaa" />
```

14.10 Setting a validation message

You can use the 'validation' object to change add-on error messages. This object contains one message that corresponds to the respective severity level.

The following example shows how to check the current record's name. If this returns a value of 'false', an error message displays. In this case, 'The name is null'.

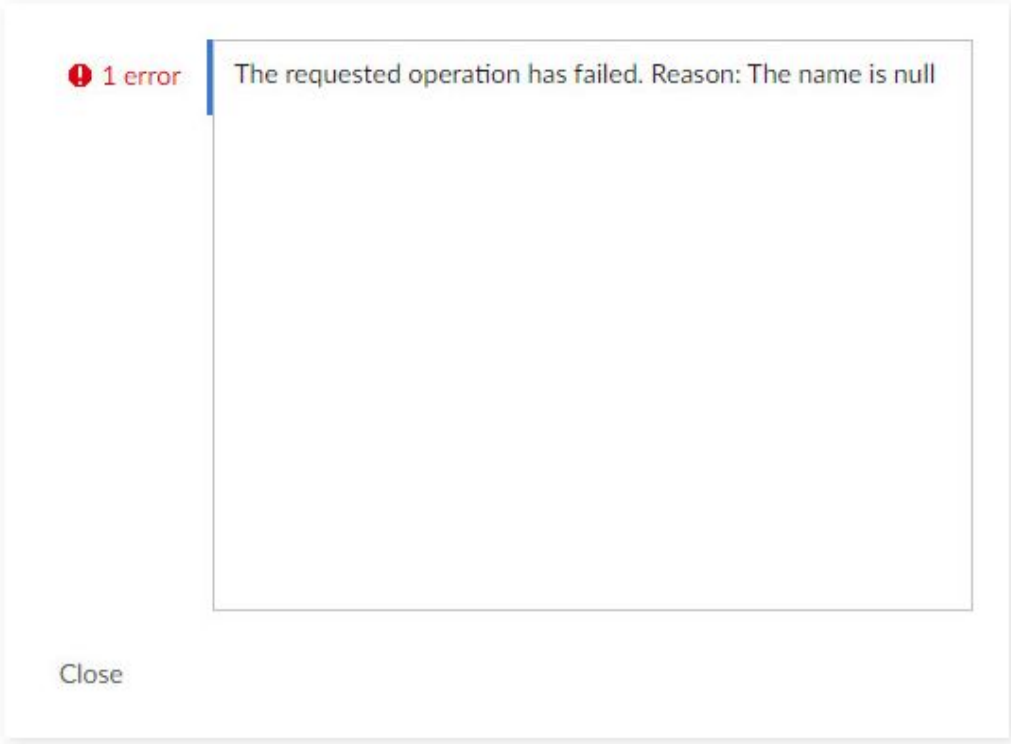
```
var name = record.Name;

if(name !=null){
    return true;
}

validation.error('the name is null');
return false;

```

The following image shows this error message. The 'validation' object supports other methods, see the appendix for more information.



14.11 Getting data set information

You can retrieve information about the current data set using the 'dataset' object. This object specifically applies to permission rules. The following table shows the relationship between the type of information and its related method:

Dataset Information	Script Method
Owner	'getOwner()'
Adaptation name	'getUniqueName()'
Module name	'getModuleName()'
Schema location	'getDataModel()'

The following example shows how to set permissions by checking the owner. If the owner is 'Beveryone' (B is a specific prefix), permission becomes read-only. Otherwise, it becomes read-write.

```
if(dataset.getOwner() == "Beveryone"){
    return READONLY;
}

return READWRITE;

```

14.12 Getting node information

Like 'dataset', the 'node' object only applies to permission rules. This object retrieves information about the current node and is very useful because nodes are frequently used when creating permission rules. Information retrieved can be a path in the schema, category and data type using the 'getPath()', 'getCategory()', 'getDataType()' pre-defined methods. Other methods tied to the 'node' object check whether the current node is a terminal node and whether history is disabled on the current node. These are 'isTerminalValue()' and 'isHistoryDisabled()', respectively. See the appendix for more information.

Example 1: In this example, the permission rule returns the read-only or read-write access permission depending on whether the current node is auto-incremented.

```
if(node.isAutoIncrement()){
    return READONLY;
}

return READWRITE;

```

Example 2: In this example, a line of code is added to show the node's path in the schema to the log file.

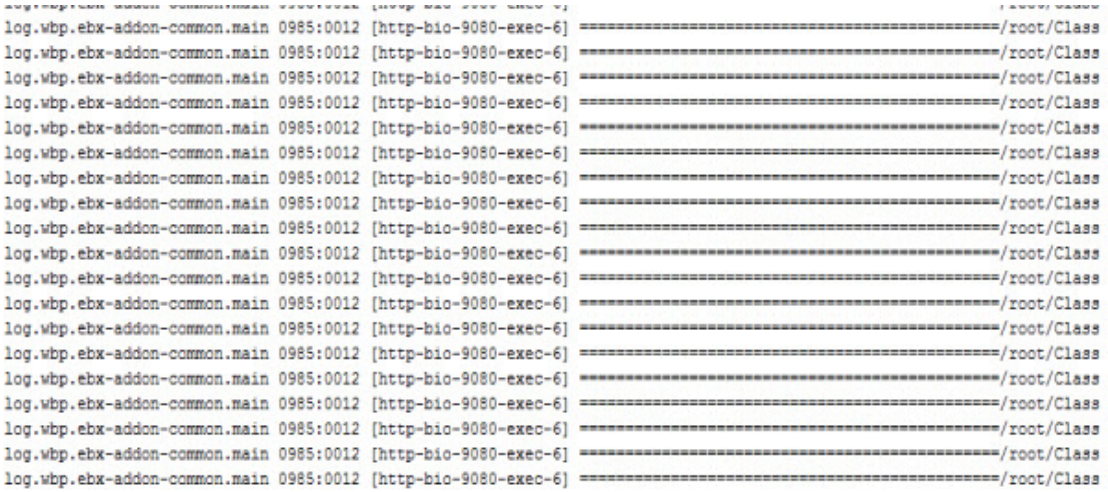
```
log.error("============================================" + node.getPath());

if(node.isAutoIncrement()){
    return READONLY;
}

return READWRITE;

```

The image below shows the result.



14.13 Getting session information

When you create a permission rule, a tool to retrieve current session information is extremely useful. The Scripting Language provides the 'session' object. By using the supported methods, this object can get the current user id, attribute values by name and check whether the current user is in a specific role.

For example, the following piece of scripting code expresses a permission rule. This permission rule controls access permission based on the specific role of the current user. If the role is 'developer', the access permission will be read-only. Otherwise, the access permission will be read-write.

```
1 if(session.isUserInSpecificRole('developer')){
2     return READONLY;
3 }
4 return READWRITE;
```

14.14 Predefined permission

The add-on provides some predefined keywords that represent access and action permissions. These keywords are used in the permission rule to return the permission as the result of the permission rules.

These keywords are listed in the following table:

Keyword	Access permission	Action permission
HIDDEN	Represents the hidden permission	Represents the hidden permission
READONLY	Represents the read-only permission	N/A
READWRITE	Represents the read-write permission	N/A
ENABLE	N/A	Represents the enable permission
DISABLE	N/A	Represents the disable permission

The following example shows how a permission rule sets read-only permission for the article record if the internal code is > 1000:

```
/* Article will be read-only if internal code > 1000 */
if(record.codeInterne > 1000){
    return READONLY;
}

/* Otherwise, it'll be read-write */
return READWRITE;
```

CHAPTER 15

Using the script editor

This chapter contains the following topics:

15.1 Overview of the script editor

To simplify script implementation, the script editor provides an array of useful features. The following section describes how the editor:

- Checks and highlights the syntax.
- Allows you to search and replace.
- Provides code suggestions.
- Shows the data model's structure.

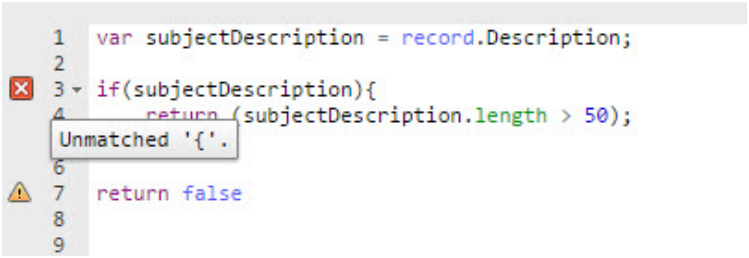
Checking and highlighting the syntax

When you write your code in the script editor, it displays in different colors based on the category of terms. This feature improves readability and allows you to distinguish between different contexts.

```
1  var subjectDescription = record.Description;
2
3  if(subjectDescription){
4      return (subjectDescription.length > 50);
5  }
6
7  return false;
```

If your code is syntactically incorrect, the script editor automatically raises alerts. In line three of the example below a red 'X' indicates a problem. By hovering over the icon, a message alerts you to the

missing element. The yellow caution icon indicates a warning that might not break your code, but really is not a best practice either.

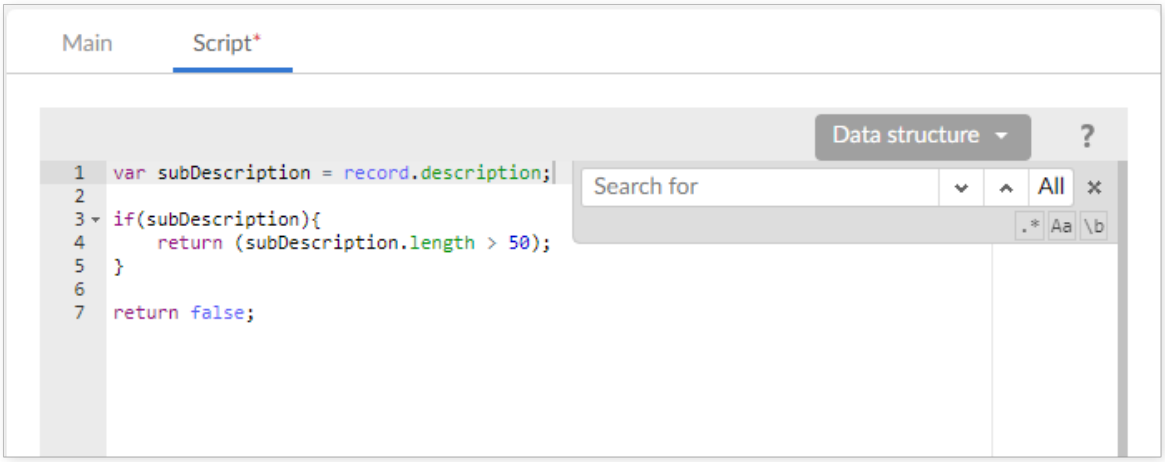


Search and replace.

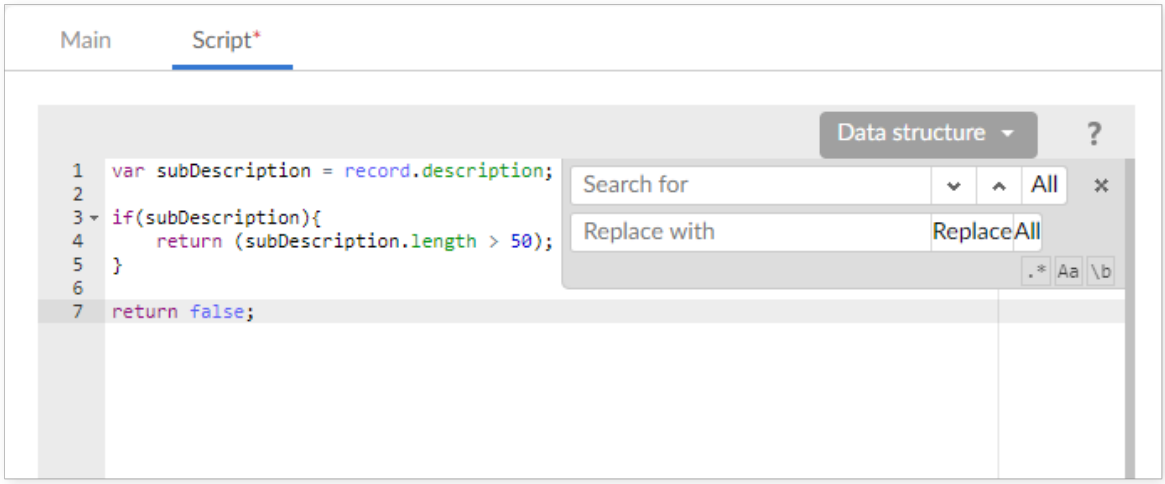
This basic - yet important function - can greatly assist you in keeping long and complicated code under control. The script editor supports both finding and replacing code.

You can use the 'Search' function by pressing the 'Ctrl + F' key combination. The search box contains the following features:

- The up/down arrows highlight the next, or previous occurrence of the search term found.
- The 'All' option closes the search box and highlights every occurrence found with a flashing cursor.
- The three icons in the lower right of the search box allow you to toggle between using: regular expression search, case sensitive search and whole word search.

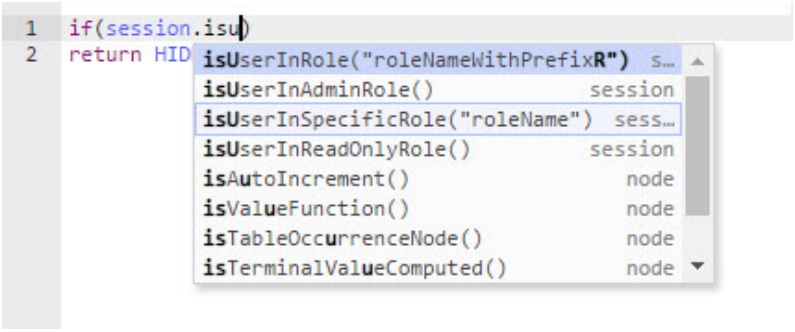


Use the key combination of 'Ctrl+H', to display the 'Replace' feature. Enter your replacement text in the box and click 'Replace' to update individual occurrences, or click 'All' to update all found occurrences.



Code auto-suggest

The script editor's auto-suggestion feature helps speed up the process of coding by reducing typos and other common mistakes. A main function of this feature is that it provides suggestions about methods associated to predefined objects.

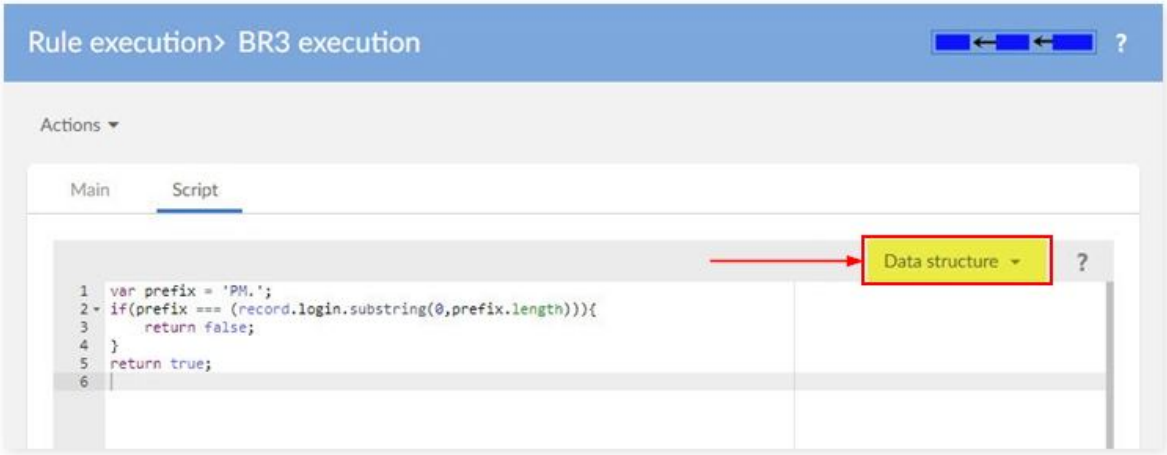


'Data structure' panel

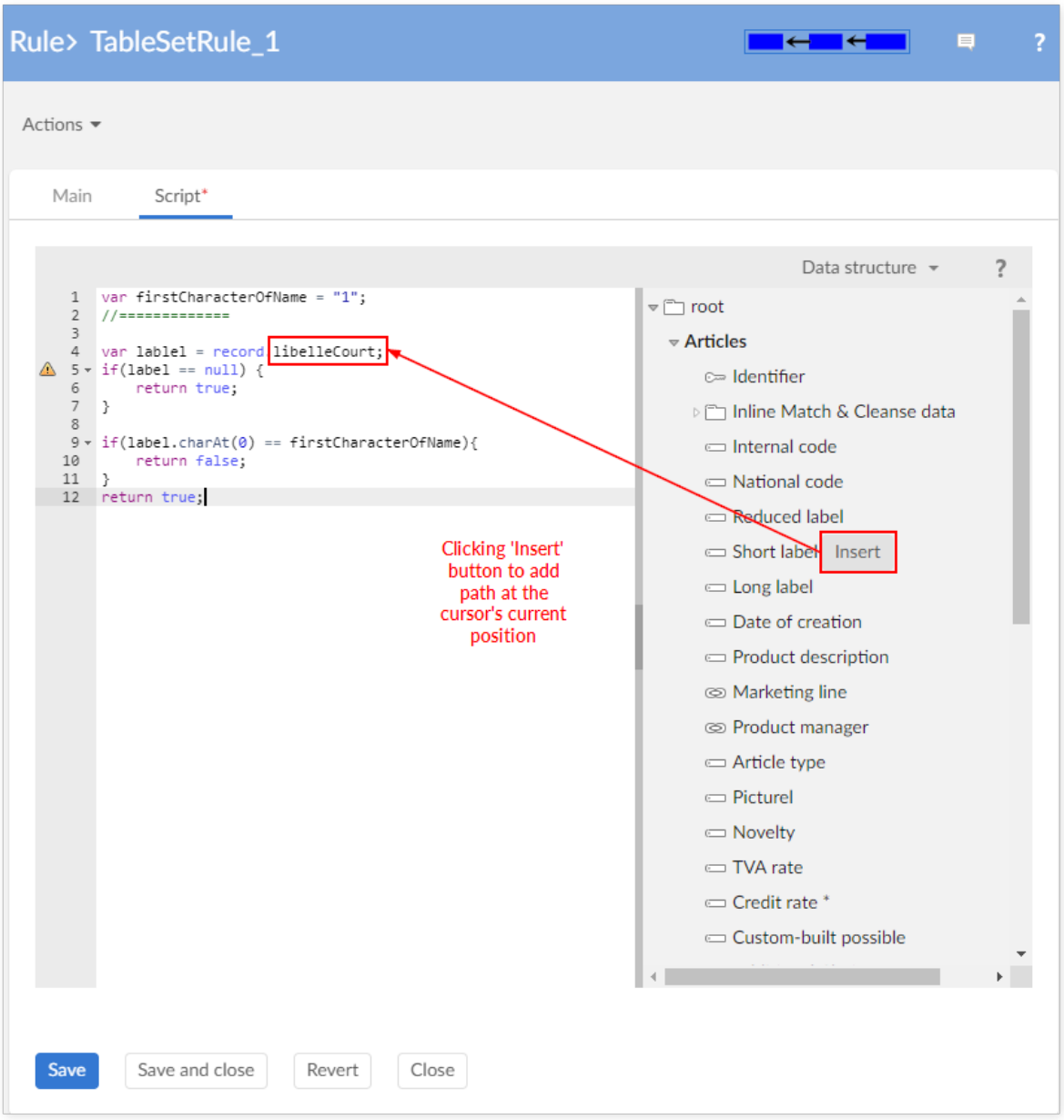
When using a script implementation, most tasks require you to frequently enter node paths. Correctly remembering each path in the data model can be very difficult. And, if you type paths incorrectly, the script will not work. To alleviate this burden, the script editor can display your data structure, allow you to select a node, and insert its path.

To use the 'Data structure' panel:

- While working in the 'Script' tab, click 'Data structure' to open the panel and display a structural view of your current data model.



- To add a node's path to the script, hover your mouse over the node you want to add and click insert. As shown in the following image, the path inserts at your cursor's current position.



Note that the 'Data structure' panel only displays the model of the data set on which the rule executes.

CHAPTER 16

Appendix

This chapter contains the following topics:

16.1 **List of all predefined objects**

Object	Method	Description
record		An object bean representing the current record.
	getPrimaryKey()	Returns the record's formatted primary key.
	save()	This method commits any changes to the current record. Keep in mind only an 'Automated rule' and a 'Table set rule' can modify data. If you call this method with any other type of rule, or have insufficient permissions, it does nothing. You can call this method using 'save()' and 'save(isTriggerActive)'. In the first example, the parameter is not specified and triggers are enabled. However, in the second example, the trigger depends on the 'isTriggerActive' boolean value. Setting the input value to 'true' activates the trigger. Otherwise, the trigger is inactive.
	del()	This method deletes the current record. Keep in mind only an 'Automated rule' and a 'Table set rule' can modify data. If you call this method with any other type of rule, or have insufficient permissions, it does nothing.
query		Queries the current table's data.
	getQuery(String tablePath)	Returns the query object for the specified table using tablePath.
	getQuery(String datasetName, String tablePath)	Returns the query object for a table from a specific data set in the current data space.
	getQuery(String homeKey, boolean isBranch, String datasetName, String tablePath)	Returns the query object for a table from a specific data set and data space (isBranch = true) or snapshot (isBranch = false).
	newRecord()	Returns the object bean for the new record on the current table.
	records(xPathPredicate)	Returns a list of object beans based on the specified xPathPredicate.
log		Writes information to the log file.
	error(message)	Writes a message to the log file with the severity level of ERROR.
	info(message)	Writes a message to the log file with the severity level of INFO.

Object	Method	Description
	debug(message)	Writes a message to the log file with the severity level of DEBUG.
	warn(message)	Writes a message to the log file with the severity level of WARNING.
validation		Sets the validation message.
	error()	Sets to error.
	error(message)	Sets to error with message.
	info()	Sets to info.
	info(message)	Sets to info with message.
	warning()	Sets to warning.
	warning(message)	Sets to warning with message.
	setMessage(message)	Sets the message.
	getMessage()	Returns the message.
	hasInfo()	Returns true if it has info.
	hasError()	Returns true if it has error.
	hasWarning()	Returns true if it has warning.
dataset		Gets information about the current data set.
	getDataModel()	Returns the formatted schema location of the current data set.
	getModuleName()	Returns the module name of the current data set.
	getOwner()	Returns the owner of the current data set.
	getUniqueName()	Returns the adaptation name of the current data set.
node		Gets information about the current node.
	getPath()	Returns the path in the schema of node.
	getCategory()	Returns the current node's category.

Object	Method	Description
	getDataType()	Returns the current node's data type.
	isAssociationNode()	Returns true if the current node is an association node.
	isAutoIncrement()	Returns true if the current node is an auto increment.
	isComplex()	Returns true if the current node is a complex node.
	isHistoryDisabled()	Returns true if history is disabled on this node.
	isSelectNode()	Returns true if the current node is a selection node.
	isTableNode()	Returns true if the current node is a table node.
	isTableOccurrenceNode()	Returns true if the current node is a table occurrence node.
	isTerminalValue()	Returns true if the current node is a terminal node.
	isTerminalValueComputed()	Returns true if the current node is a terminal computed node.
	isTerminalValueDescendant()	Returns true if the current node is a terminal descendant node.
	isValueFunction()	Returns true if the current node is a value function node.
session		Gets information for the current session.
	getUserID()	Returns the current user id.
	getAttribute(String name)	Returns the attribute value by name.
	isUserInRole(String roleName)	Returns true if the current user is in the specified role. The roleName must start with prefix 'R' This method is deprecated. Use the isUserInSpecificRole(String roleName) method instead.
	isUserInSpecificRole(String roleName)	Returns true if the current user is in the specified role.
	isUserInAdminRole()	Returns true if the current user is in the admin role.
	isUserInReadOnlyRole()	Returns true if the current user is in the read-only role.
HIDDEN		Represents the hidden permission for both Access and Action permission.

Object	Method	Description
READONLY		Represents the read-only access permission.
READWRITE		Represents the read-write access permission.
DISABLE		Represents the disable action permission.
ENABLE		Represents the enable action permission.

Release Notes

CHAPTER 17

Version 1.7.9

Released: October 2022

This chapter contains the following topics:

17.1 New features

This release contains no new features.

17.2 Changes in Functionality

This release contains no functionality changes.

17.3 Changes to third-party libraries

This release contains no changes to third-party libraries.

17.4 Closed issues

[RPFL-598] A vulnerability needs to be fixed.

17.5 Known issues

This release contains the following known issues:

- Schema compilation does not support all rule types. If you want to use action permission rules to set service permissions, you have to add the 'com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission' class to those services manually.

- You must keep your D.E.C.'s current. This is especially true if you add or remove the mandatory attribute, the computed value attribute or maximum number of values attribute from a data model field.
- Default classes are those available to be added automatically-including the default schema extension. However, one data model can only have one schema extension. So, if you have your own schema extension in the data model, you have to remove it from the data model before adding a default class. To reuse your programmatic access rules, invoke them in the add-on using the method described in the appendix.
- From version 1.5.0, the add-on provides the following views: 'D.E.C. hierarchy' on D.E.C. table, 'Rule by D.E.C. hierarchy' on Rule table, and 'Rule execution by D.E.C. hierarchy' on the Rule execution table. However, on the old snapshot of the add-on, new views do not display any data. In this case, you have to use the default view or the old view.

This release contains the following known issues related to the scripting language:

- The script does not support to 'query' data by the association node.
- The script does not support to modify data on other data spaces. You can modify data on the current data space only.
- The wizard panel only shows the model of the data set that the rule will execute.

CHAPTER 18

All release notes

This chapter contains the following topics:

18.1 Version 1.7.9

Released: October 2022

New features

This release contains no new features.

Changes in Functionality

This release contains no functionality changes.

Changes to third-party libraries

This release contains no changes to third-party libraries.

Closed issues

[RPFL-598] A vulnerability needs to be fixed.

Known issues

This release contains the following known issues:

- Schema compilation does not support all rule types. If you want to use action permission rules to set service permissions, you have to add the 'com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission' class to those services manually.
- You must keep your D.E.C.'s current. This is especially true if you add or remove the mandatory attribute, the computed value attribute or maximum number of values attribute from a data model field.
- Default classes are those available to be added automatically-including the default schema extension. However, one data model can only have one schema extension. So, if you have your own schema extension in the data model, you have to remove it from the data model before adding a default class. To reuse your programmatic access rules, invoke them in the add-on using the method described in the appendix.
- From version 1.5.0, the add-on provides the following views: 'D.E.C. hierarchy' on D.E.C. table, 'Rule by D.E.C. hierarchy' on Rule table, and 'Rule execution by D.E.C. hierarchy' on the Rule execution table. However, on the old snapshot of the add-on, new views do not display any data. In this case, you have to use the default view or the old view.

This release contains the following known issues related to the scripting language:

- The script does not support to 'query' data by the association node.
- The script does not support to modify data on other data spaces. You can modify data on the current data space only.
- The wizard panel only shows the model of the data set that the rule will execute.

18.2 Version 1.7.8

Released: August 2022

New features

This release contains no new features.

Changes in Functionality

This release contains no functionality changes.

Changes to third-party libraries

The jQuery UI library was updated to version 1.13.2.

Closed issues

This release contains no closed issues.

Known issues

This release contains the following known issues:

- Schema compilation does not support all rule types. If you want to use action permission rules to set service permissions, you have to add the 'com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission' class to those services manually.
- You must keep your D.E.C.'s current. This is especially true if you add or remove the mandatory attribute, the computed value attribute or maximum number of values attribute from a data model field.
- Default classes are those available to be added automatically-including the default schema extension. However, one data model can only have one schema extension. So, if you have your own schema extension in the data model, you have to remove it from the data model before adding a default class. To reuse your programmatic access rules, invoke them in the add-on using the method described in the appendix.
- From version 1.5.0, the add-on provides the following views: 'D.E.C. hierarchy' on D.E.C. table, 'Rule by D.E.C. hierarchy' on Rule table, and 'Rule execution by D.E.C. hierarchy' on the Rule execution table. However, on the old snapshot of the add-on, new views do not display any data. In this case, you have to use the default view or the old view.

This release contains the following known issues related to the scripting language:

- The script does not support to 'query' data by the association node.
- The script does not support to modify data on other data spaces. You can modify data on the current data space only.
- The wizard panel only shows the model of the data set that the rule will execute.

18.3 **Version 1.7.7**

Released: March 2022

New features

This release contains no new features.

Changes in Functionality

This release contains no functionality changes.

Changes to third-party libraries

This release contains the following changes to third-party libraries:

- The jQuery library was updated to version 3.6.0.
- The jQuery UI library was updated to version 1.13.1.

Closed issues

This release contains no closed issues.

Known issues

This release contains the following known issues:

- Schema compilation does not support all rule types. If you want to use action permission rules to set service permissions, you have to add the 'com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission' class to those services manually.
- You must keep your D.E.C.'s current. This is especially true if you add or remove the mandatory attribute, the computed value attribute or maximum number of values attribute from a data model field.
- Default classes are those available to be added automatically-including the default schema extension. However, one data model can only have one schema extension. So, if you have your own schema extension in the data model, you have to remove it from the data model before adding a default class. To reuse your programmatic access rules, invoke them in the add-on using the method described in the appendix.
- From version 1.5.0, the add-on provides the following views: 'D.E.C. hierarchy' on D.E.C. table, 'Rule by D.E.C. hierarchy' on Rule table, and 'Rule execution by D.E.C. hierarchy' on the Rule execution table. However, on the old snapshot of the add-on, new views do not display any data. In this case, you have to use the default view or the old view.

This release contains the following known issues related to the scripting language:

- The script does not support to 'query' data by the association node.
- The script does not support to modify data on other data spaces. You can modify data on the current data space only.
- The wizard panel only shows the model of the data set that the rule will execute.

18.4 **Version 1.7.6**

Released: December 2021

New features

This release contains no new features.

Changes in Functionality

This release contains no functionality changes.

Changes to third-party libraries

The jQuery library was updated to version 1.13.0.

Closed issues

This release contains no closed issues.

Known issues

This release contains the following known issues:

- Schema compilation does not support all rule types. If you want to use action permission rules to set service permissions, you have to add the 'com.orchestranetworks.addon.rpfl.DefaultRulesSchemaServicePermission' class to those services manually.
- You must keep your D.E.C.'s current. This is especially true if you add or remove the mandatory attribute, the computed value attribute or maximum number of values attribute from a data model field.
- Default classes are those available to be added automatically-including the default schema extension. However, one data model can only have one schema extension. So, if you have your own schema extension in the data model, you have to remove it from the data model before adding a default class. To reuse your programmatic access rules, invoke them in the add-on using the method described in the appendix.
- From version 1.5.0, the add-on provides the following views: 'D.E.C. hierarchy' on D.E.C. table, 'Rule by D.E.C. hierarchy' on Rule table, and 'Rule execution by D.E.C. hierarchy' on the Rule execution table. However, on the old snapshot of the add-on, new views do not display any data. In this case, you have to use the default view or the old view.

This release contains the following known issues related to the scripting language:

- The script does not support to 'query' data by the association node.
- The script does not support to modify data on other data spaces. You can modify data on the current data space only.
- The wizard panel only shows the model of the data set that the rule will execute.

18.5 **Version 1.7.5**

Released: January 2021

Product updates

The add-on no longer supports the Document Type Definition declaration in XML files. If this causes an exception, the add-on will inform you that this declaration is not allowed.

18.6 Release Note 1.7.4

Release Date: September 18, 2020

Product update

- The add-on has been updated to support the OpenJDK8 and OpenJDK11 libraries.
- Libraries were updated to fix some potential issues.

18.7 Release Note 1.7.3

Release Date: June 23, 2020

Product update

The jQuery library has been updated to version 3.4.0.

Bug fixes

[RPFL-563] An add-on description in French is not translated.

18.8 Release Note 1.7.2

Release Date: June 20, 2019

Featured update

The add-on has been updated to ensure compatibility with the TIBCO EBX® 5.9.4 release.

18.9 Release Note 1.7.1

Release Date: February 15, 2019

Bug fixes

[RPFL-550] D.E.C. does not show all tables and fields in the hierarchy view.

18.10 Release Note 1.7.0

Release Date: October 26, 2018

Featured updates

The EBX® Rules Portfolio Add-on has undergone significant updates to ensure compatibility with the EBX® 5.9.0 GA release.

18.11 Release Note 1.6.1

Release Date: May 19, 2017

New features

- **[26010]** A new button is available on the 'Java Implementation' tab to clear your selection when configuring the 'Property data' type as Boolean.

Bug fixes

- **[26011]** An exception occurs when deploying on an application server that strictly validates web.xml.

18.12 **Release Note 1.6.0**

Release Date: May 9, 2017

New features

- **[08337]** It is now possible to search for a specific rule implementation when creating new rules.
- **[10356]** A new UI is available to select which rules execute after launching the 'Execute rules' service.
- **[20490]** The 'Publish rules portfolio' service now can be used to override an old publication.
- **[19505]** [Documentation] The user guide for scripting language needs to be updated.
- **[15487]** A wizard has been added to facilitate business rule configuration.

API

- BusinessRuleDefinition's API Javadoc has been updated to clearer information.
- It is now possible to get the label of hook in the Hook's API.

18.13 **Release Note 1.5.1**

Release Date: January 23, 2017

New features

- **[23916]** The 'Business/Permission rules logging' group has been changed to record in the Logging Level table.

Optimization

- Permission and validation rules have been optimized so that the overload against a pure Java EBX® API implementation is insignificant when implementing a Java based rule.

Bug fixes

- **[23454]** There are always error messages in the log file when starting EBX® Rules Portfolio Add-on in an empty repository.
- **[23788]** Portfolio - import XML error.

18.14 Release Note 1.5.0

Release Date: October 12, 2016

New features

- D.E.C. creation is based on the data model instead of data space and data set.
- The Java implementation post-fix is changed to (A), (V), (M), and (T).

UI improvement

- In the EBX® Rules Portfolio Add-on UI, services have been improved and are more user-friendly.
- The D.E.C. records will now display using the correct structure based on their types and relationships in the data model.
- On the rule execution table, only the 'Rules' field is displayed instead of the two fields 'Rule' and 'Rules set'. An option is available that allows you to select one or the other.

Service

- You can now apply the existing rule publication to other data spaces instead of creating a new publication with the same content.

API

- The API's Javadoc has been updated to provide more in-depth information.

Bug fixes

- **[21555]** Unexpected exception occurs when importing an XML file.
- **[21707]** User should not be allowed to delete the predefined configuration provided by the add-on.

18.15 Release Note 1.4.1

Release Date: May 19, 2016

New features

Script editor

- With the new wizard displaying on the right of script editor, you can easily insert a record object property or table path to the script. This wizard will display the full structure of the data model on which the script will be executed.
- Now all predefined objects and their published methods are available in the script editor's contextual help.

Script engine

- New methods in the 'session' object check whether the user is in administrator or read-only role.
- New methods in the 'query' object allow users to query data from another data set, or data space.

18.16 Release Note 1.4.0

Release Date: April 13, 2016

This version is only compatible with EBX® 5.7.0 fix C or higher.

New features

Automatically add EBX® Rules Portfolio Add-on classes to data models

- Previously, to enable the EBX® Rules Portfolio Add-on, you had to manually add default classes to your data model. Now, schema compilation automatically adds any required classes.

Dependency

- Dependencies limit validation execution to the specific events of interest, such as on node creation and deletion. It is now possible to set a dependency on a validation rule applied to a field using the 'Dependencies' tab in the UI. Previously, this functionality was only available via an API in the context of a constraint and its containing data model.

Ability to execute classic access rules

- It is possible to migrate your old data and improve backward compatibility using classic access rules.

18.17 Release Note 1.3.5

Release Date: December 11, 2015

Bug fixes

- **[18519]** Ajax validation using a scripting rule does not take into account a record's updated value.

18.18 Release Note 1.3.4

Release Date: December 2, 2015

Bug fixes

- **[18471]** A value of 0 is reported as invalid input for the 'Value' field, which is located in the 'Java implementation' tab's 'Property data' group.

18.19 Release Note 1.3.3

Release Date: November 19, 2015

Bug fixes

- **[18129]** An exception occurs when executing a scripting rule on a field.

18.20 Release Note 1.3.2

Release Date: November 3, 2015

Optimization

- Script execution has been optimized.

18.21 Release Note 1.3.1

Release Date: October 9, 2015

Bug fixes

- **[17710]** An exception occurs when executing a scripting rule on a field.

18.22 Release Note 1.3.0

Release Date: September 16, 2015

New features

Scripting language

- Scripting language is a newly available and significant feature that allows you to define a rule by writing a script in scripting language. See the Scripting Language for more information.
- You can now define conditions for rule execution by using a script or Java implementation instead of using 'Context' and 'Simple expression'.

Rule publication

- Rules now execute on publications of rule configurations and no longer execute via the EBX® Rules Portfolio Add-on configuration. A new data set, 'TIBCO EBX™ Rules Portfolio Add-on - Production', contains the 'Rule publication' and 'Rule repository' tables. Respectively, these tables store:
  - Published snapshot information.
  - Mappings between data spaces and rule publication.

New services

- A new 'Publish rule portfolio' service is available on the 'TIBCO EBX™ Rules Portfolio Add-on' data set. This service creates the publication of rule configuration and saves all information in the 'Rule publication' and 'Rule repository' tables, which are located in the 'TIBCO EBX™ Rules Portfolio Add-on - Production' data set.
- New services are added into 'Context', 'Simple expression', 'Rule publication' and 'Rule repository' tables which allow users to delete records from these tables.

18.23 Release Note 1.2.0

Release Date: January 26, 2015

New features

- You can now execute a 'Validation rule' using a constraint set on a field.
- The logging data set is now divided into two data sets in relational mode: 'Rules Portfolio - Logging' and 'Rules Portfolio - Logging archive'. Logging configuration has been moved to the 'Logging level' group in the 'TIBCO EBX™ Rules Portfolio Add-on' data set.
- A new EBX® task can execute the archive log procedure using the EBX® scheduler.
- A new "Purge log" service is available on the "Rules Portfolio - Logging archive" data set. This service deletes the log data from imported archives.
- A new "Query log" service searches log data using criteria, such as, User ID, data space, data set, table, execution time, etc.

API

- New classes DefaultConstraint and DefaultConstraintOnNull execute validation rules via constraints.

18.24 Release Note 1.1.1

Release Date: June 10, 2014

Parallel rule execution

- In order to improve the rule execution response time, 'Manual validation rules' now execute in a multi-threaded environment.

Rules configuration

- The rules execution context is extended with a 'User-Profile' property that activates or deactivates a rule depending on the user/profile combination.

Predefined rules

- The following pre-built rule properties are now configurable:
  - 'Is active'
  - 'Data set execution'
  - 'User profile'

18.25 Release Note 1.1.0

Release Date: April 14, 2014

New features

The EBX® Rules Portfolio Add-on is extended with a new traceability function to track rules execution. An automatic archiving process is applied to keep log over time. A purge service is used to clean up the logs on demand.

18.26 **Release Note 1.0.0**

Release Date: January 24, 2014

New features

The EBX® Rules Portfolio Add-on manages two categories of rules:

- The 'Business rules' for the data validation that are executed from triggers or through the EBX® data validation service. They are classified into four types as follows:
  - 'Automated rule' can modify the user data. It is executed through triggers on tables and raises an error in case of failure.
  - 'Validation rule' cannot modify the user data. It is executed through triggers on tables and raises an error in case of failure.
  - 'Manual rule' cannot modify the user data. It is executed by a user through the EBX® validation service, and raises either an error or a warning.
  - 'Table set rule' can modify the user data. It is executed by a user through the 'Execute rules' service.
- The 'Permission rules' for the rights management that is implemented in compliance with the EBX® permission scheme applied to data and services. They are classified into two types as follows:
  - 'Access permission rule' defines permission for data set, table, field or record.
  - 'Action permission rule' defines permission for service.

Every rule is configured through a set of metadata such as:

- 'Property data': Parameters that can be modified at the level of the configuration, and used as input data for the execution of the rule. These properties are defined through the Java Interface that allows to publish a rule into the EBX® Rules Portfolio Add-on.
- 'Is active': a rule can be active or inactive (the rule is deactivated). It is not possible to execute the rule even if the other conditions of the execution are valid.
- 'Validity date': date from which the “Rule execution” can be executed.
- 'Expiry date': date from which the “Rule execution” is impossible to execute even though it is configured to be active.
- 'Data set execution(s)': one or more data sets can be selected as a context of execution. This means that the rule execution happens only if the execution is requested in the defined data sets. It is possible to use the special tokens '[All data spaces]' and '[All data sets]'.
- 'Simple expression': used to define the conditions under which the rules can be executed.

A set of predefined business rules are available to get 'True' or 'False' results directly. By combining these assertion rules with the 'Simple expression' configurations, it becomes possible to create new assertion rules based on any field of a table:

- [ON] Assertion true (Manual validation)
- [ON] Assertion true (Validation)
- [ON] Assertion true (Table set)
- [ON] Assertion false (Manual validation)
- [ON] Assertion false (Validation)
- [ON] Assertion false (Table set)

A set of predefined permission rules are available:

- [ON] Hidden service
- [ON] Disable service
- [ON] Enable service
- [ON] Hidden the data set
- [ON] Permission read-only for the data set
- [ON] Permission read-write for the data set
- [ON] Hidden table
- [ON] Permission read-only for the table
- [ON] Permission read-write for the table
- [ON] Hidden field
- [ON] Permission read-only for the field
- [ON] Permission read-write for the field
- [ON] Hidden record
- [ON] Permission read-only for the record
- [ON] Permission read-write for the record

API

- Java API for declaring rules into the EBX® Rules Portfolio Add-on.