Modeling Manual

OMNIA PLATFORM

Version 2.2
April 2017

Table of Contents

Introduction

The objective of this document is to be a user guide for the entire modeling component of the platform. For this, it covers all sections of the component, explaining them in detail and providing descriptive examples for each one.

Platform

The platform consists of a cloud-hosted application whose purpose is to allow anyone to model business-oriented computer applications without the need to write code.

It allows you to quickly create prototypes of solutions to solve several problems, as well as an agile delivery of different versions of a solution to address the changes in requirements that may arise during the project development.

Development in the platform is done using a specific domain language, based on the REA framework principles (described below).

By using this type of language (based on the economics and accounting theories), the platform allows modeling agents to describe solutions that are closer to the problem's domain and further from implementation issues. This ensures that it is easy to design an application if the user has limited computer science knowledge, but is highly specialized in the current problem's domain.

Technological requirements

An Azure subscription is required for the installation of the platform itself. The only requirement for end-users is that their browser is compatible – we support the current and previous versions of all major desktop browsers (Chrome, Edge, Firefox, Safari), the current desktop version of Opera, and Internet Explorer from version 10 onwards. For mobile, we also support the Android stock browser and Safari on iOS 7 and later versions.

Any tool developed for the platform (connector, etc.) should be assumed to require .NET 4.6, unless explicitly stated otherwise.

REA

The **REA** (**Resources, Events, Agents**) framework is focused on the development of accounting systems, centered on reporting the different economic events of an organization. It is based on the assumption of the existence of a limited number of concepts in all accounting software and that it is possible to design an easily adaptable application, without compromising the fulfillment of business rules.

The REA concepts are applied with the following interpretation in the platform:

- **Economic Resources**: system entities representing assets that can be quantified and with commercial value. These assets are held by the Economic Agents, who exchange them in financial transactions between agents. Examples: Products, Money and Services;

- **Economic Agents**: system entities representing individuals or groups of individuals. These entities hold economic resources and exchange them in financial transactions. Examples: Customers, Companies and Employees;

- **Commitments**: represent the commitment to trade resources made between two agents. A Commitment is made because the transaction often only happens after the moment when it is set. There are two types of commitments: increase or decrease, depending on whether it represents a reduction or an increase of a company's resources. Examples: Paying for purchased items, often with deferred payments;

- **Economic events**: represent the transaction of resources between two agents, one in the role of supplier and the other as recipient. This way, the transaction represents an increment for the receiver agent and a decrement for the supplier agent. Except for some cases such as the provision of services, an Event occurs at the moment it is created.

  All commitments are fulfilled by one or more economic events in an action referred to as *fulfillment*, triggered when resources are exchanged. Fulfillment can be total or partial.

These REA framework concepts are joined by some **additional concepts** brought by the implementation of the platform:

- **User-defined entities**: system entities required for financial transactions but which cannot be considered resources or agents. An example of such an entity used to categorize an agent would be his Department or Role in the company;

- **Interaction**: allows you to represent all the different resource exchanged as a document. An interaction is composed of three different areas: Header, Details and Summary;

- **Process**: allows you to group several interactions into business processes. This way, it is possible to create a logical organization of interactions of similar nature.

In the **Problem description** section of this document, there is an example of a problem that can be modeled in the platform to explain the concepts described in this section.

Development Process

The platform stands out against other development solutions due to its application generation structure. The produced application is directly interpreted based on the model and all changes are seen as adjustments to the model, using a *no code* structure (does not require writing code).

The **modeling user** can design his entire application testing the effects of his/her changes as the application is modeled without the need for time-consuming *builds* to obtain results.

The entire modeling is done in the context of a *tenant*. A *tenant* is a platform account with a specific model for solving an individual problem.

Templates

A *template* is a type of *tenant* with a model designed to answer a specific problem. The purpose is for this to be the 'basis' of a solution with no application data, and for this basis to be used in the future for creating new *tenants* which work as instances of a model for the end-customer.

Best practices

The type of account used for development should be a template. This makes it so that when the modeler wants to create an account for a specific client, they can specifically create a new tenant directly.

Using a template tenant will also allow the modeler to easily delete test data because, unlike entities, interactions are not copied when creating a tenant from a template (see How to delete test data? to understand how this deletion is performed).

As the application is modeled, changes are directly reflected on the final application. Using two browsers or two browser tabs - one to model and one to see the results - is helpful to see how the changes you are modeling will look for customers.

Entities and business processes

As explained in the REA section, the platform's logic is based on economic resources, agents and events. The first two, along with several other components described in that section, are represented as **entities**. An entity is therefore one of the basic components of an application in the platform.

As it was previously described, there are several types of platform entities that share modeling features:

- **Agents:** represent economic agents;
- **Resources:** represent economic resources;
- **User-defined entities:** support entities that help model a problem but do not fit the two previous types;
- **Key-value pairs:** entities used to limit the value of something to a set of values.

Economic events are also represented as entities, in this case, **transactional entities**:

- **Events:** transaction of resources between two agents;
- **Commitments:** commitment to make a resource transaction made between two agents.

How to create an entity?

Access **Menu | Entities | Agents / Resources**

All entities are created in their respective modeling **menu**. However, transactional entities cannot be created here since they only make sense in the context of an interaction. Agents and Resources share the same structure, therefore they are created in the same way.

When you access the menu, a **list of all existing same-type entities** that can edited is displayed, along with a button to create a new one. When you choose to create a new entity, a form with the following tabs is displayed:

- **General:** setting the entity's basic parameters:
  - **Code**: a unique identifier;
  - **Name**: the entity's name, localizable into several languages;
  - **Attributes:** setting all of the document's header attributes and behaviors, as well as the screen organization in groups. Attributes, UI elements and groups are described in more detail in the Attributes and How to manage groups? sections. Base attributes are detailed in the Base attributes attachment;
- **Entity Items**: items which exist in the context of an entity and come up as a table in the application. When you create an item of this type, a new configuration tab is displayed. For this, see the Configuration entities section;
- **View:** setting different views for the interaction being created. To configure it, see the How to configure a View? section;
- **Other configurations:** description of several other behaviors associated with entities:
  - **Hidden:** see How to hide an entity/interaction?;

- **Allow Users:** see How to allow users association? (only applicable to Agent-type entities);
- **After Submit:** see How to configure the behavior when saving?;
- **Form Operations:** see How to configure the behavior when saving?;
- **Tabbed Groups**: see How to manage groups?;
- **Allow to Revert:** see How to allow entity/interaction reversal?;
- **Is Contact:** see How to set an entity as a contact?.

How to allow users association?

Access **Menu | Entities | Agents**

You can allow associating a platform user with **Agent**-type entities. For this, access the Agent entity editing/creating option and select the **Allow Users** box in the **Other configurations** tab.

Example: Associating a user with an Employee entity in the platform allows you to know which user is associated with the user when he/she logs in, automatically identifying him/her when an order is placed.

How to create a process?

Access **Menu | Processes and Interactions | Add New**

A process is an organization unit in the platform which allows you to group several interactions of similar nature in a single aggregating element. Only two parameters are required to configure a Process:

- **Name**: the process name, localizable into several languages;
- **Code**: unique code which identifies the process.

Example: In an order application (see Problem description), create a "**Order Management**" process. At a later point, this will contain interactions that represent both the placing and the report of receiving of orders.

How to create interactions?

Access **Menu | Processes and Interactions**

In the platform, interactions are the way in which we define which operations are executed in the application, representing an interaction between several agents and the exchange of several resources.

There are three areas to an interaction: the **header** (**Document**), the **grid rows** (**Details**, optional) and a **summary** (**Summary**, optional). The header incorporates the attributes set in the context of the interaction, the details may have one or more transactional entities and the summary contains one transactional entity.

To add a new interaction, select the **Add Interaction** button in the context of a process. This form, just like the Entities form, contains a set of tabs where you can configure several parameters:

- **General**: setting the interaction's basic parameters:
  - **Code**: a unique identifier;

    - **Name**: the interaction's name, localizable into several languages;
    - **Attributes:** setting all of the entity's attributes and behaviors, as well as the screen organization in groups. Attributes, UI Elements and groups are described in more detail in the Attributes and How to manage groups? sections. Base attributes are detailed in the Base attributes attachment;
- **Entity Items**: items which exist in the context of an interaction and come up as a table in the application. When you create an item of this type, a new configuration tab is displayed (see Configuration entities);
- **Details**: the document body (optional). It may contain one or more transactional entities corresponding to grid rows (see How to configure a commitment/event?);
- **Summary**: the document summary (optional). It can contain a transactional entity (see How to configure a commitment/event?);
- **Views:** setting different views for the interaction being created (see How to configure a View?);
- **Other configurations**: other behaviors associated with interactions:
    - **Hidden**: see How to hide an entity/interaction?;
    - **After Submit:** see How to configure the behavior when saving?;
    - **Form Operations**: see How to configure the behavior when saving?;
    - **Tabbed Groups**: see How to manage groups?;
    - **Allow to Revert:** see How to allow entity/interaction reversal?.

How to create a commitment/event?

Access **Menu | Processes and Interactions | Create or edit an interaction**

When creating an interaction, it is often required that you create a commitment or event, whether for details or for the summary. For this, it is important to identify first whether a **commitment** is required (the row will stand for the promise of something) or an **event** (in which case it stands for the fulfillment of a previous promise). With this information, we can reuse a transactional entity or select a new one by using the **Add new** option.

When you select the **Add new** option, a new tab is created, where you can set the details for the commitment/event to be added:

- **Code:** transactional entity identifier;
- **Name:** the entity's name, localizable into several languages;
- **Kind:** indication of whether the transactional entity represents incoming (Increment) or outgoing (Decrement) resources;
- **Provider Agent:** type of agent responsible for providing resources;
- **Receiver Agent:** type of agent responsible for receiving resources;
- **Resource:** type of resource being exchanged.

Example: In an order application (see Problem description), create an interaction that represents the order request. There, set an entity that corresponds to the supplier request (**Provider Agent**) the item delivery (**Resource**) and the company (**Receiver Agent**) in the details. Since the date in which this delivery will take place is not set yet, the type of transactional entity will be a commitment.

How to configure a commitment/event?

Access **Menu | Processes and Interactions | Create or edit an interaction**

You can access a set of configuration parameters after creating a commitment or event, or when you edit an interaction that already has this data:

- **Min/Max:** minimum and maximum number of transactional entities allowed;
- **Allow Comments:** allows the adding of lines with comments to the grid;
- **Discard of records:** specify whether the commitment must be saved along with the interaction. The **Condition** allows you to set conditions for this field;
- **Automatic Creation:** allows for automatic creation of a transactional entities based on other transactional entities. **Field** and **Target Field** specify the fields used for this creation and **Distinct** specifies whether rows will be created for each unique **Field** value or for each row of the original entity;
- **Expand Details:** if there are attributes set as visible in a row's **Details** section, this option specifies that that section expands automatically;
- **Lock condition:** specifies if a whole row is read-only;
- **Conditions for each possible line operation:** set of conditions that specify, for each line operation, if it is available;
- **Calendar:** Specifies if the transactional entity can be displayed as a calendar (see How to configure a transactional entity to be displayed as a calendar?).

How to configure a transactional entity to be displayed as a calendar?

Access **Menu | Processes and Interactions | Create or edit an interaction**

You can set an event or commitment to be displayed as a calendar when you edit it in an interaction. For this, select the **Calendar** checkbox, which will make a set of configuration parameters appear:

- **Default:** if the calendar view is the default view;
- **Hidden Grid:** if you want to hide the grid view;
- **Calendar name:** name of the exported calendar file;
- **Available views:** which views are available from all possible options (Yearly, monthly, monthly (timetable), weekly or daily);
- **Default view:** calendar view to be displayed by default;
- **Invalid Week Days:** include days not valid for scheduling in the calendar;
- **Calendar Period Begin/End:** calendar start and end date;
- **Default date:** information of the date where you wish the calendar to be placed;
- **Allow to select ranges:** set if a user can schedule events or commitments for several days at once using the calendar;
- **Blocked days:** information of locked calendar days (holidays);
- **External events:** other events (different from events or commitments) that may be relevant for the calendar;

- **Title (event):** field with information to be used as the title of the event displayed;
- **Category (event):** field used to categorize the displayed events;
- **Begin date (event):** set the start date for a selection of days in the calendar;
- **End date (event):** set the end date for a selection of days in the calendar;
- **Group by (event):** attribute per which events are grouped in the timetable view;
- **Custom Icons (event):** specifies if custom icons can be used in the calendar;
    - **Property:** attribute of the **Value Pair** type whose values are used to calculate the icon;
    - **Icons:** mapping between target attribute values and the icon to be used – see What icons are available in the platform?.

How to configure a view?

Access **Menu | Processes and Interactions | Create or edit an interaction**

In the **View** tab you can set different views for the entity/interaction being created. These views specify different forms for the same entity or interaction and can have different contents and/or be displayed to different employees. It has the following attributes:

- **Code:** view identifier;
- **Context:** setting the context in which the view is available. It can be available when creating, editing new entities/interactions, or both;
- **Name:** the view's name (can be translated into the application's different languages).

To create new views and change between existing ones, click the button on the upper right corner and choose which **view you wish to manage** or select the "**Create**" option.

How to delete test data?

Access **Menu | Processes and Interactions**

It is often necessary to batch delete data for every interaction or entity (e.g. demo or development data).

The platform provides a mechanism to address this by allowing you to delete data for each entity or interaction at a time. The interaction and entity context menus have an option named **Remove all data**, which will remove it after confirmation from the modeler.

Data removal can only be performed on Demonstration or Template tenants, and is not recursive / does not solve dependencies – if there is a document of another type referencing any of the created entities or interactions, it will fail to delete the data.

Interface

How to manage groups?

You can divide attributes into groups both for entities and interactions. To create a group, create or edit an interaction and press the **create group** button from the **Attributes** tab. This creates a new group whose name can be changed with the **Group Name** option. The **Visibility condition** option allows you to set conditions in which the group becomes visible.

After creating the groups, you can also reorganize the order in which they are displayed in the system in the **Attributes** tab.

By default, groups are displayed as tabs in the application. However, there is the **Tabbed Groups** option in the **Others** section which allows you to change this behavior and display groups vertically one after the other in the header.

Other behaviors

How to hide an entity/interaction?

To hide an entity or interaction, access the **Others** tab and select the **Hidden** option. This will hide them on the end application, but they can still be used in other entities or interactions.

How to configure the behavior when saving?

The **After Submit** parameter specifies the behavior followed by the application after saving an instance of that entity or interaction. You can select between **returning to the list** of entities/interactions, **returning to the main menu** or **staying in the creation screen** of the entity or interaction.

You can also change which options are available when saving. The **Form Operations** parameter allows you to set which of the platform's base operations are available. There are two possible modes:

- **Standard:** all options are displayed: Clear, Save and, if applicable, Submit for Approval;
- **Custom:** allows you to select each operation individually.

How to allow entity/interaction reversal?

When you access the **Others** tab, there is an **Allow Revert** option which allows you to set if the operation to reverse approval is available for this entity or interaction.

How to set an entity as a contact?

The **Allow Contacts** property is available when you access the **Other configurations** tab. It allows you to set if the type of entity can be exported as a contact file (in vCard format with the .vcf extension). If so, you must specify which of the entity's attributes will be used to create the contact file's information.

Attributes and UI Elements

Attributes are a general concept used across the platform, and these are common in the definition of the different types of entities and interactions. Their management is the same in all the different locations, following the same tab structure:

UI Elements are a special case of the attributes, of elements that show up in an interaction but have no expression in terms of the REA framework, and represent visual/auxiliary elements only. They have a more limited option set compared to Attributes, but follow the same structure.

- **General:** setting the attribute's basic parameters:
  - **Short Code:** attribute identifier, unique within the entity/interaction;
  - **Name:** the attribute's name displayed on the form, localizable into several languages;
  - **Description:** full description of the attribute to be displayed in the form, localizable into several languages;
  - **Persisted:** indication of whether the field is persistent or not, which is used when the field is only auxiliary and not intended to hold a value;
  - **Type:** Type of field (see Data types);
- **Behavior:** setting several properties related to the attribute's behavior in specific contexts:
  - **Default Value**: suggests a value for the attributes (see How to set a default value?);
  - **Required**: indication of whether the attribute is required (see How to set a field as required?);
  - **Editable**: indication of whether the attribute can be changed when editing (see How to set an attribute as non-editable?);
  - **Existence Condition**: formula that sets a condition for the existence of the attribute (see How to set formulas?);
  - **Required Condition:** formula that sets an attribute as required (see How to set formulas?);
  - **Formula**: formula that calculates the attribute's value (see How to set formulas?);
  - **Validation Formula**: formula that calculates if the attribute's value is valid;
- **Actions:** setting a set of actions to be executed under specific conditions (see Actions);
- **Interface**: setting properties directly related to displaying the attribute in the interface;
  - **Group**: group in which the attribute is placed;
  - **Visible**: whether the attribute is visible (see How to change the visibility?);
  - **Visible from Screen Sizes**: starting screen sizes from which the attribute is visible (see How to change the visibility?);
  - **Visible Condition:** formula which sets conditions for the attribute's visibility (see How to set formulas?);
  - **Read Only**: whether the attribute is only available for reading;
  - **Read Only Condition**: formula which sets conditions for the attribute's availability (see How to set formulas?);
  - **Row**: form row where the attribute is placed. Only applicable if the attribute is not on a grid (see How to change positioning?);
  - **Column**: form column where the attribute is placed (see How to change positioning?);

- **Order**: the attribute's position in a row. Only applicable if the attribute is on a grid (see How to change positioning?);
- **Grid Size**: the attribute's percent size in a grid (see How to change positioning?);
- **Size**: the attribute's size (see How to change positioning?).

Data types

The platform supports a set of **data types,** both for attributes and UI elements**,** which set conditions for the **appearance**, **content** or **feature**s used by the attributes:

Attribute data types

Data types	Description
Text	Text field.
Text area	Multiline text field.
Protected text	Protected text field (e.g. for passwords).
Numeric	Number field, with or without decimal places.
Percent	Percent value field.
Money	Number field that corresponds to a value of currency.
Integer	Field with a positive whole value.
Date	Date field.
Date Time	Date and time field.
True/False	Field with a true/false checkbox.
Value pair	Reference to a platform key/value entity.

File	Field to upload files. See How to restrict file types? and What file types cannot be uploaded?
Other Entity	Reference to a different platform entity.

UI Element data types

Data types	Description
Button	Button used to trigger platform scripts/actions.
Hyperlink	URL link that can be clicked.
Web Component	Reference to a web component.

After setting the type of field, there are additional fields which allow you to configure/set limits for the values to be entered, according to the type of attribute:

Property	Description	Types
Min Value	The field's minimum value.	Numeric, Percent, Money, Integer
Max Value	The field's maximum value.	Numeric, Percent, Money, Integer
Scale	The number of decimal places in the field.	Numeric, Percent, Money
Max Length	The field's maximum length.	Text, Short Text
Value Pair Type	Type of value pair that the field represents.	Value Pair
File Types	Types of files that can be *uploaded*.	File

Multiple Upload	Specifies whether it is allowed to upload several files in the context of an attribute.	File
Style	Setting the button style.	Button
Entity Kind	The category of the type of entity represented in this field.	Other Entity
Entity Type	Type of entity represented in this field.	Other Entity
Cardinality	Number of entities that can be referenced in this field. The application will use this configuration to set the maximum limit of entities that the user can choose from. This property takes values from 1 up to "n", where "n" means there is no limit.	Other Entity
Max Value	The field's maximum value.	Numeric, Percent, Money, Integer

Lastly, if the attribute's type is **Other Entity**, there is also a feature to filter displayed entities following the logic described in the How to filter entities? section.

How to restrict file types?

It is possible to restrict the kinds of files that will be uploaded after defining an attribute of the File type. This is done by specifying a set of categories:

- **All file types:** Applies no restrictions, overrides the other options;
- **Images:** Image file types - .jpg, .png, .jpeg, .bmp, .psd, .gif, .tga, .tif, .tiff, .raw;
- **Documents:** Document file types - .doc, .docx, .xls, .xlsx, .ppt, .pptx, .odp, .xps, .txt, .rtf, .csv, .xml, .odt, .ods, .pdf;
- **Compressed files:** .zip, .rar, .7z, .tar, .bz2, .gz.

What file types cannot be uploaded?

Independently of the restrictions set, there is a set of files considered potentially dangerous and that are not allowed to be uploaded onto the platform:

- **Executables:** .exe, .msi, .msp, .hta, .jar, .com, .pif;
- **Scripts:** .vb, .vbe, .vbs, .bas, .bat, .jse, .ps1, .cmd, .scf, .sct, .wsf, .wsh;
- **Macros:** .docm, .xlsm, .pptm;
- **Others:** .reg, .scf, .lnk, .inf, .pst, .dll;

Note: .js and .cs files are also blocked unless they are being uploaded in the context of a script.

Formulas

In the context of attributes, formulas allow users to perform a set of operations. This section provides a short description of the different types of operations available. You can find a more detailed description in the Formula References attachment.

- **References**: ability to make reference to the attributes of the current entity or other entities and combine them with other formulas;
- **Mathematical operations:** formulas support basic mathematical operations (+, -, *, /);
- **Boolean operations:** ability to use Boolean algebra logic (AND, NOT, …);
- **Comparison operations**: ability to compare any type of value with these operations (<, !=, ==…);
- **Date handling operations:** the platform supports a set of operations with dates, such as difference calculations or retrieving the time from a date/time value;
- **Mathematical operations**: besides basic operations, there is a set of more complex operations such as conditional counts or statistical operations;
- **Text handling operations**: the platform provides a set of operations that allow you to manipulate text *strings*, such as converting case or concatenating text.

The syntax to use an attribute in formulas depends on its location:

- In the current entity/interaction:
  - [AttributeCode]
- In the header of an interaction or entity:
  - [Document.AttributeCode]
- In the values of a grid:
  - [Details.TransactionalEntityCode.AttributeCode]
  - [Summary.TransactionEntityCode.AttributeCode]
- In entity items:
  - [Items.EntityItemCode.AttributeCode]
- For related transactional entities:
  - [Parent.AttributeCode]
  - [Childs.TransactionEntityCode.AttributeCode]

Places that use formulas have attribute selectors to help with the process.
For operations, the syntax to be used is always FUNCTION(parameter).

Actions

Actions can be specified in the context of an attribute and consist of an execution moment, along with a set of actions to be executed under that condition.

Moments

Event	Description
Change	Actions executed when the attribute's value is changed.
Create	Actions executed when the attribute is created – for example, when entering a form.
Check	Actions executed when a true/false field is changed.
Click	Actions executed when there is a mouse click.

Types of Action

Type	Description	Events
Get Entity	Retrieve the entity's additional information.	Create, Change
Get Entity Item	Retrieve the entity's Entity Items information.	Change
Get Entity Message	Retrieve support messages configured in the entity.	Change
Get Number	Retrieve the valid series and number for the interaction.	Create, Change

Execute Formula	Execute a formula that allows you to handle the value of other entity/interaction fields.	Create, Change
Execute Blocking script	Execute an extensibility *script* that locks the interface.	Click, Change
Execute non-blocking script	Execute an extensibility *script* that does not lock the interface.	Click, Change
Get User Agent	Retrieve information on the agent associated with the user currently logged in.	Create
Show List	Display an entity's list.	Click
Show Dashboard	The dashboard is displayed.	Click

How to set an action?

First, you need to set the execution moment and then set a condition to limit the execution (see How to set formulas?).

Next, set the type of action to be executed. After choosing the type of action, you can set the conditions according to which the action will be executed.

In the context of an entity:

- **Entity Kind** and **Entity Type** identify it;
- You can **set attributes to retrieve** from an entity and apply them to the entity where the action is set. For this, is requires the list of attributes **To Get** and the attributes **To Set**;

You can **filter attributes** using the **Conditions** option (see How to filter entities?).

Outside the context of an entity:

- For **Execute Scripts**, a *script* selection field will be displayed instead of settings related to entity types.

In both cases you can **temporarily disable the action** if you do not want it to be executed, but you do not want to delete it either. You can also set if you want an error to occur if no action results are retrieved when it is executed.

How to manage an attribute?

There are three attribute options available on every creating or editing screen:

- **Add:** by using the Add Attribute button, you can add a new attribute;
- **Change:** you can edit each attribute of the current entity or interaction. However, not all parameters can be edited, namely you cannot edit the code and attribute type and parameters associated with the type (such as cardinality);
- **Remove:** All attributes added by the user can be deleted except for base attributes.

Behavior

How to set a field as required?

There are two fields in the **Behavior** tab that set an attribute as required. First, there is the **Required** *checkbox* specifying whether or not it is required. If there is a formula set (see How to set formulas?) in the **Required Condition** field, its result will take precedence over the *checkbox*, meaning that the attribute will only be required if the condition is true.

How to set a default value?

The **Default Value** field allows you to set the default value that attributes assume. It always corresponds to a concrete value and not a reference to another attribute or formula. For example, you can set a **true/false** field to always start as active by setting the **Default Value** to *true*, or set the base quantity of orders to always be 10, by setting it to 10.

How to filter entities?

The platform allows you to filter an entity, an operation that can be executed both in the context of attributes and actions that make reference to other attributes.

For this, you need to set a **logical operation** that compares the value that you want to be used for filtering and the value of the entity being used for the comparison.

To access these properties you need to use the data access syntax.

Property	Description

Element	Attribute of the entity /interaction whose value will be used to filter the results.
Operator	Logical comparison operator
Value	Attribute of the original entity whose value will be used to filter the results.

How to set formulas?

Formulas used in the context of attributes share the same syntax. They must be written with **=formula** and obey a syntax similar to the one used by spreadsheet programs. However, some locations where formulas are used have additional requirements:

- **Conditional** formulas always have to return a *true* or *false* value;
- For **validation** formulas you can also set the message to be displayed in the platform's several languages when validation fails;
- There is no need for a specific returned value in the remaining formulas.

The Formula Examples attachment shows examples of some of these formulas.

Interface

How to change the visibility?

There are two fields in the **Interface** tab that set conditions for the visibility of attributes. First, the **Visible** *checkbox* specifies whether the attribute is visible. If there is a configured formula (see How to set formulas?) in the **Visibility Condition** field, its result will take precedence over the checkbox, meaning that the attribute will only be visible if the condition is true.

There is a third **Visible from Screen Sizes** field that specifies the screen sizes where the attribute is visible.

How to change positioning?

In the **Interface** tab you can set the way in which the attribute will be displayed in the application. There is a set of attributes that can be used for this setting:

- **Row** and **Column:** set the row and column where you want the attribute to be displayed in the form. You can have a maximum of four columns for each row, which means there is support for four consecutive attributes;
- **Order**: in the context of a grid, specifies the order in which the attribute will be displayed on the grid;
- **Size:** represents the current attribute's size, from 0 to 10. The sum of all attribute sizes in the same **Row** must be 12 and please note that 2 must be added to each **Size** value for the *label*;
- **Grid Size:** represents an attribute's percent size on a grid. The sum of attributes in the same grid row must be 100%.

How to set an attribute as non-editable?

The **Editable** field in the **Behavior** tab is a *checkbox* that specifies whether the current attribute may or may not be changed when editing a document.

How to format an attribute with a condition?

The **Conditional formatting** field allows modelers to determine a condition that will be used to format an attribute. It consists of a formula (see How to set formulas?) that must return either 'error', 'warning', or 'success'. Depending on this value, the attribute will be formatted accordingly.

How to set an attribute as read-only?

There are two fields in the **Interface** tab that make an attribute read-only. First, the **Read Only** *checkbox* specifies whether the attribute can be edited or only read. If there is a formula set (see How to set formulas?) in the **ReadOnly Condition** field, its result will take precedence over the *checkbox*, meaning that the attribute will only be read-only if the condition is true.

The options in this tab take precedence over the ability to edit the attribute, meaning that even if the attribute is set as **Editable**, if the **ReadOnly Condition** fails you cannot edit the attribute at the editing moment.

Data access

Writing formulas that are not executed in the *interface* (but are executed, for example, in actions, lists or security filters), uses a slightly different terminology to allow them access to all the necessary attributes from any context:

- **Difference between base and non-base attributes:** all attributes created by the user must start with the "#" character;
- **Difference between entity type attributes:** all attributes that represent some other system entity must be preceded by its type: Agent:Attribute, Resource:Attribute, UserDefinedEntity:Attribute, etc. If you want to access the attribute's property, it must come after setting the attribute – for example: Agent:Attribute.Property;
- **Difference between system attributes**: some attributes are not actually displayed in the model but are still available to allow validations, namely regarding security mechanisms. These not physical attributes are represented between "@"s.

The different conditions that a formula consists of can be concatenated with the "&" (and) and "|" (or) logical separators. The Data access formulas section describes some examples of this type of formulas.

Throughout the platform, there are helpers available that allow you to select these attributes.

Configuration entities

Not all entities used by the platform fit the basic principles of the REA framework. This chapter describes the points where some support structures were introduced to complement the basic features of this model.

How to create other entities?

Access **Menu | Entities | User-defined Entities**

User-Defined Entities are the platform's support entities that do not fit the REA model and describe other entities which are neither agents nor resources, but which are important for the created models because of the additional information the introduce in existing transactions.

These are configured just like agents and resources and therefore the logic described in How to create an entity? must be followed.

How to create configuration tables?

Entity Items, or configuration tables, exist in the context of a specific type of entity and allow you to set a new structure for a table, ensuring a 1 to N connection. After setting the new structure's **Code** and **Name** a new tab is added where you can manage its attributes.

These tables are configured like the remaining **Entities** with only small differences: **Entity Items** have only one group and their attributes are always set within the logic of a grid.

How to create a list with preset values?

Access **Menu | Entities | Value Pairs**

Value Pair type entities are a support element that must be used to limit user *input* to a small set of options that rarely suffers changes. There are two points that must be configured to set an entity of this type:

- **Code:** a unique identifier;
- **Options:** a set of key/value pairs:
    - **Value:** value we want the entity to stand for;
    - **Text:** visible text which identifies this option, localizable into several languages.

Example: In an order application (see Problem description), we want to set the state of the order. Create a Value Pair with the 0/Proposed, 1/Accepted and 2/Closed. Add an attribute to the interaction of type Value Pair and identify the pair you created.

Commitment fulfillment

Commitment fulfillments, or *fulfillments*, are a mechanism that complement Commitments, allowing the exchange of resources.

Commitment fulfillment operations can be performed both by events and other commitments. It is possible for more than one type of event or commitment to fulfill the same commitment.

Fulfillments area always configured in the context of the interaction that contains the transactional entity (commitment or event) that will fulfill the commitment.

How to configure the fulfillment of a commitment?

Access **Create or edit an interaction | Configure Fulfillments**

There is a tab in the configuration screen for each commitment or interaction event that will perform the fulfillment. After accessing the desired tab, select the commitment to be fulfilled by specifying the following information:

- **Process**: process that incorporates the interaction that includes the commitment to be fulfilled;
- **Interaction**: interaction that incorporates the commitment to be fulfilled;
- **Commitment**: commitment to be fulfilled;
- **Kind:** type of *fulfillment*:
    - **Any**: any value allows for fulfillment;
    - **Equal:** the original quantity allows for fulfillment;
    - **Less than**: a quantity smaller than the original one allows for fulfillment;
    - **Greater than:** a quantity bigger than the original one allows for fulfillment;
    - **Equal only:** only the total quantity allows for fulfillment, no partial *fulfillments* are allowed.

Then, configure the fulfillment process details:

- **Attribute:** header attribute for the interaction that also corresponds to the "**Receiver Agent**" of the commitment to be fulfilled. You can then filter commitments returned;
- **List:** list of commitment to be fulfilled to display;
- **Type conditions:** set of conditions which allow you to filter the commitments returned (see How to configure mappings between entities?);
- **Mappings:** setting attribute mappings between the original commitment and the transactional entity that will perform the *fulfillment* (see How to configure mappings between entities?).

How to configure mappings between entities?

Access **Create or edit an interaction | Configure Fulfillments**

To configure mappings between entities you need to use two different locations. First, you need to set the **Type conditions** that allow you to filter commitments to be fulfilled. The syntax used is the same as the one described in How to filter entities?.

Next, you need to map the original commitment's attributes with the attributes of the commitment or event that will fulfill it. For this, fill in the following parameters:

- **From:** attributes of the commitment to be fulfilled. It uses the logic described for Data access with the following differences:
    - You do not need to reference the type of entity to access an entity attribute (for example, ProviderAgent.Code);
    - You do not need the '#' when accessing non base fields;
- **To:** attributes corresponding to the transactional entity, selectable from a *dropdown*.

By default, the platform creates a set of mappings between the base fields of both entities, which the modeling user can change or delete.

Approval flows

The platform provides an approval mechanism that can be applied to entities and interactions, allowing them to be validated by one or more individuals before reaching a final state.

How to create an approval stage?

Access **Create or edit an interaction | Configure Approval Stages**

To create an approval stage you need to set two parameters:

- **Code**: a unique identifier;
- **Name**: the interaction's name, localizable into several languages.

You can set several stages following this logic and it is necessary to configure each one:

- **Is a start stage:** specifies if the stage represents the start of the approval flow. There can only be one start stage;
- **Is an end stage:** specifies if the stage represents the end of the approval flow. There can be several end stages, and at least one is required;
- **To Approve:** specifies which transactional entity (commitment or event) is approved by this stage. To be specified only for middle stages;
- **Next stage:** stage to which the interaction goes after being approved (see How to configure the stage flow?);
- **When rejected:** stage to which the interaction goes after being rejected (see How to configure the stage flow?);
- **List:** the list to be used to show the records with pending approval for this approval stage
- **Approver definition:** determining the approver's choice (see How to configure the next approver?);
- **Conditional execution:** formula that determines if the current stage will be executed, based on information about the interaction (see How to configure the stage flow?);
- **Available Attributes:** attributes of the interaction and corresponding transactional entities that can be edited in the approval stage (see How to make an attribute editable in an approval stage?).

How to configure the next approver?

Access **Create or edit an interaction | Configure Approval Stages**

You need to set which users will have access to the approval when you configure an approval stage. The choice is always based on two attributes:

- **Condition:** formula to evaluate if the interaction matches the conditions set. If true, the approver set under **Assign To** for this row can approve the document;
- **Assign To:** attribute containing the information of which **Agent** is associated with the user who will have access to this approval.

Formulas use the Data Access logic, with the following differences:

- You do not need to reference the type of entity when you want to access an entity's attribute (for example, ProviderAgent.Code);
- You do not need the '#' when accessing non base fields;
- Comparisons use '=' and '<>' and to compare with an empty value simply leave the other part of the comparison blank, for example: "Employee.Manager.User<>" is a statement that declares that the Manager's user does not exist;
- Boolean variables are written TRUE and FALSE.

How to configure the stage flow?

Access **Create or edit an interaction | Configure Approval Stages**

The approval stage flow logic consists of a sort of directed graph with an initial state and a number of end states. This start state is not exactly an approval stage, but the value of entities or interactions before they enter the flow. For example when they are saved as a draft before being submitted for approval.

To set the flow, **each non final stage has two attributes** defining which state they will move on to: **Next Stage when Approved** and **Next Stage when Rejected**.

Apart from this setting, there is a **Conditional execution** parameter which determines whether the stage will be run or not. A formula is defined in that parameter. If the formula returns FALSE, interaction goes to the **Next Stage**, as if it had been approved. The formula uses the Data Access logic, with the following differences:

- You do not need to reference the type of entity when you want to access an entity's attribute (for example, ProviderAgent.Code);
- You do not need the '#' when accessing non base fields;
- Comparisons use '=' and '<>' and to compare with an empty value simply leave the other part of the comparison blank, for example: "Employee.Manager.User<>" is a statement that declares that the Manager's user does not exist;
- Boolean variables are written TRUE and FALSE.

How to make an attribute editable in an approval stage?

Access **Create or edit an interaction | Configure Approval Stages**

The platform allows you to set cases in which the approver can perform small corrections to the document without having to reject it. For these cases, there is an **Available Attributes** section where you can individually select which attributes are editable from a *dropdown* field.

Viewing data

When it comes to viewing information, the platform allows you to build two different types of data views: **Lists** and **Charts**. These can be organized into control panels (**Dashboards**) or displayed individually (in the case of lists).

Lists

A list is a way to view a specific entity or interaction's data. Lists can be displayed in several ways: in the context of a dashboard, in the platform's default lists or other entity/interaction list contexts (for example, individual menu options).

How to create a list?

Access an **entity or interaction | Configure Lists**

A configurations screen with the following options is displayed when you create or edit a list:

- **Name:** the list's name, localizable into several languages;
- **Description:** a brief description of the list;
- **Only to dashboards**: specifies if the list can only be used in dashboards;
- **Default**: identifies which list will be shown as the default list for this entity or interaction. If no list has this parameter set, the first one created / shown in the list will be used.
- **Default date range:** default time period by which the list results will be filtered. The time range can be changed when viewing the data;
- **Fields:** the fields to be shown as columns in the lists. The How to configure list data? section describes how to configure these data sources:
- **Data filtering:** section to configure the filters to be applied in the data
    - **Condition:** filter to be applied on the information retrieved. It must be composed like a formula (optional);
    - **Top:** maximum number of elements retrieved (optional);

- **Date Field:** indication of which list field will be used to filter the list by date (optional);

- **Calendar:** specifies if the list should be displayed as a calendar (see How to view the list as a calendar?).

Lastly, in the **Operations** tab you can configure which options are available in this list. The How to configure which options are available in a list? sections explains how to do it in detail.

How to view the list as a calendar?

Access an **entity or interaction | Configure Lists**

There is an additional set of options to be configured if you want to configure a list as the base for a calendar:

- **Default view:** calendar view to be displayed by default: yearly, monthly, monthly (timetable), weekly or daily;
- **Calendar name:** name of the exported calendar file;
- **Title field:** contains information to be used as the title of the event displayed;
- **Category field:** the calendar's category;
- **Begin date field:** set the start date for a selection of days in the calendar;
- **End date field:** set the end date for a selection of days in the calendar;
- **Blocked days field:** information of locked calendar days (holidays);
- **Group by field:** group elements in the timetable view;

How to configure list data?

Access an **entity or interaction | Configure Lists**

You need to specify the entity/interaction attributes to be displayed on the list and the complementary information about each of one, as described in the following table:

Property	Description
Field	The entity or interaction attribute you wish to access.
Label	The attribute's caption, localized into several languages.
Type	The type of data this attribute represents, for display purposes. Very important for numeric values, which users may or may not want to display with decimals and thousands separators.
Decimal Places	Number of decimal places to format the attribute to (0 to 5). Only applicable to attributes of Numeric type.

Aggregate Type	Type of aggregation of numeric values. You can perform sums, calculate averages and count the number of records.
Sort Type	Type of sorting to be applied on the list. You can sort each attribute in an ascending or descending order.
Sort Order	The order by which attributes are sorted. Since it is possible to sort according to several attributes, you must configure the logic to be applied when sorting.
Group By	Specifies if information should be grouped by an attribute.
Visible on Sizes	Specification of the starting screen sizes from which the attribute is visible. Possible values: xs, sm, md, xl.

How to configure which options are available in a list?

Access an **entity or interaction | Configure Lists | Operations tab**

In this tab, you can decide whether you want to make each of the platform's individual options available (Edit, Approve, Revert, Print, Copy), as well as add new operations associated with the execution of extensibility *scripts*.

In this case, you need to provide the following information:

- **Action:** identifier of the action to be executed;
- **Name:** name of the interaction to be executed, localizable into several languages;
- **Is Available:** indication of whether the action is available;
- **Script:** indication of the *script* to be executed;
- **Icon Code:** icon to be displayed in the button that allows the current operation to be run (optional) -- see What icons are available in the platform?.

Dashboard

A control panel, or dashboard, is a platform structure that can contain a set of lists and/or charts. These can be displayed to the user in menus or on the homepage after logging in.

How to create a chart?

Access **Menu | Data Visualization | Dashboard Charts**

A chart is a graphical way to view a specific entity or interaction's data. The chart configuration logic depends on the following settings:

- **Code:** a unique identifier;
- **Data source configuration:** select an existing data source to be used as the base for the chart or create a new one:

The field choice logic is the same as the lists, described under How to configure list data?;

  - **Where:** filter to be applied on the information retrieved, written like a formula (optional);
  - **Top:** number of elements to be retrieved (optional);
  - **Date Field:** field that represents the date by which you want to filter the chart elements (optional);
  - **Company Code Field:** field that represents the company code that allows you to filter data by company (optional);
- **Data display configuration:** sets the chart's appearance:
  - **Chart Type:** the platform supports four types of chart: line charts, pie charts, vertical bar charts and area charts;
  - **X axis:** attributes that will be used as categories;
  - **Value:** attributes that will be used as values.

The formula to be used in **Where** uses the Data Access logic, with the following differences:

- Comparisons use '=', '>', '<', '<=', '>=' and '<>', and to compare with an empty value simply leave the other part of the comparison blank;
- Boolean variables are written TRUE and FALSE.

How to create a dashboard?

Access **Menu | Data Visualization | Dashboards**

The dashboard configuration is divided into two main sections: the header and the set of elements to be added to the dashboard. The following parameters are available with regards to the header:

- **Code**: a unique identifier;
- **Homepage dashboard**: indication of whether the dashboard is displayed on the homepage (see How to set a dashboard to appear on the homepage?);
- **Name:** the dashboard's name, localizable into several languages;
- **Description**: the dashboard's name, localizable into several languages;
- **Default date range**: default time period to be applied to the dashboard data.

Next, you can configure the elements to be displayed on the *dashboard* (see How to add elements?).

How to add elements?

Access **Menu | Data Visualization | Dashboards**

You can add both lists and charts to a *dashboard*. For this, you need to configure a set of parameters:

- **Title:** the element's title, localizable into several languages;
- **Visible from:** the starting screen size from which the element is visible;
- **Description:** the element's description, localizable into several languages;
- **Row:** the dashboard row where the element is located;
- **Column:** the dashboard column where the element is located;
- **Size:** the element's size, on a scale of 1 to 12. The sum of the **Size** value of all elements located on the same dashboard **Row** must match 12;
- **Chart / List:** chart or list to be displayed;
- **Dimensions:** the list/chart attributes that can be used as dimensions, allowing you to view information based on them;
- **Default dimensions:** which of the dimensions set in **Dimensions** must be selected by default, using a true/false parameter for each one;
- **Max number of elements:** maximum number of elements on the list/chart;
- **Compare with corresponding period:** within the context of a chart, it specifies whether there will be comparison with the corresponding previous year;
- **Hide until have data:** specifies if the element must be displayed in the dashboard even if it contains no data.

How to set a dashboard to appear on the homepage?

Access **Menu | Data Visualization | Dashboards**

Only one dashboard can be displayed on the homepage. If you want to control which information is displayed, for example, to display different charts to different users, you need to place all charts as dashboard elements and control access to them through roles and the **Hide until have data** option.

Printing reports

The platform allows you to retrieve interactions in the form of reports. These are generated as .pdf files and users of platform applications can download them to their device.

How to design a printing report?

The reporting technology used by the platform is called **Microsoft Reporting Services** (SSRS) and it supports ".rdlc" reporting files. Please note that, to be able to design them, you need to have "**Microsoft SQL Server Data Tools**" installed.

To start designing, you need to access the modeling area, select the "**Configure Printing Reports**" option in the context of an interaction of any type and download the **Visual Studio** solution that will help with testing and development.

In **Visual Studio** you can access a set of files:

- **Report.xml:** sample of an XML interaction holding the data necessary to test the *script*;
- **Report.xsd:** interaction structure in XSD that will be used to build the report structure;
- **ReportDevelopment.cs:** parameter settings to be used in tests, namely, the identification of the Report file to be used, the data file and the data structure;
- **Report.rdlc:** file where the report will be designed.

To obtain an interaction sample, access the modeling area again, select the list for the type of interaction to be designed and choose the **Download XML example** option.

To obtain the definition of the structure to be used in the report you must select the **Download XML Schema** option. Please note that you can only download these two elements after creating an interaction of the desired type.

After getting all the information, you will need to access the *reporting* file and the **View | Report Data** option to set the different **Datasets** by choosing the "**Add Dataset**" option. Then, after setting all the **Datasets** you can **change the report** by defining its several areas and elements using the Visual Studio designer.

How to test a report during its design?

To test the report locally you need to fulfill two requirements:

- Set which **report** and **data will be used** for the test (in the **ReportDevelopment.cs** file);
- Get an XML **data sample** from an interaction of the desired type (see How to design a printing report?).

After this, operation is similar to any other application and you can use *breakpoints*, print debugging information and other operations. To access the Visual Studio debugging tools we recommend you to design reports in the machine where the external system that will execute the *scripts* is installed.

How to add a report to the platform?

Access **Processes | Configure Printing Reports**

To add a new report to the platform you need to specify the value of the following parameters:

- **Code**: unique report configuration code;
- **Description**: optional description of the report definition in several languages;
- **Language**: language that the report applies to. Printing uses the language configurations for the user currently logged in;
- **Number of Copies**: number of copies to be generated in each print;
- **Configuration**: optional report configuration. If this is not specified, it will be automatically generated;

- **Default**: indication of which configuration will be used by default for the type of interaction;
- **File**: load the “rdlc” report file.

How to configure a report print?

Access **Other | XML Configurations**

Printing a report can be configured, allowing you to specify the following settings:

- **Remarks**: setting up the element where the comment rows will be sent:

```
<GridCommentsProperty>Resource</GridCommentsProperty>
```

- **Datasets**: definition of the organization of the different elements (interactions, commitments and events) in Datasets:

```
<Datasets>
        <DataSet>
            <Name>DataSetInteractionCompany</Name>
            <TableName>myCompany</TableName>
        </DataSet>
        <DataSet>
            <Name>DataSetInteractionHeader</Name>
            <TableName>ExpenseReport</TableName>
        </DataSet>
</DataSets>
```

- **Additional fields**: definition of fields that are not on the interaction but still belong to entities associated with it:

```
<ReportExtraFieldsConfiguration>
      </Entity Code="@Employee.Code" Type="Employee"
Kind"Agent">
        <Field Name="Name" />
  </Entity>
</ReportExtraFieldsConfiguration>
```

- **Decimal places:** optional definition of the number of decimal places for a specific attribute of an interaction:

```
<DecimalScaleConfig>
        <ExpenseReport>
              <ExpenseReport_Rate>5</ExpenseReport_Rate>
        </ExpenseReport>
  </DecimalScaleConfig>
```

The resulting XML must be added as a new XML configuration and it can be referenced later in the report configuration (see How to add a report to the platform?).

Reading data from other systems

The platform supports an extensibility mechanism based on a program - the connector - which allows communication with other systems by performing information reading and writing operations.

Setting the method of communication with other systems is done by way of external systems, and the reading of information can be modeled as external entities, as described in this chapter.

Writing information into other systems is ensured by way of extensibility *scripts*, described in chapter Scripting.

How to create an external system?

Access **Menu | External | Systems**

External systems types allow the platform to communicate with all sorts of systems. Since all systems have specific communication needs, this must be achieved with a different type of external system that allows you to set specific attributes.

These entities are configured in the same way as the remaining Entities, but **External Systems** only support the default view.

How to create an external entity?

Access **Menu | External | Entities**

External entities allow you to represent entities from other systems in the platform. The modeling of these entities is different from all other platform entities since it is done by way of SQL queries executed on the external system.

After setting them, external entities can be used in different platform contexts:

- **In other entities or interactions:** to avoid repeating information, you can view information from an external system;
- **In lists and charts:** view data directly from an external system.

To create an external entity you have to set the following information:

- **Code:** identifier of the new type of external entity. This identifier is unique and cannot be changed after being set;
- **Name:** the name of the external entity type, localizable into several languages;
- **Query:** setting the SQL query that allows access to an entity's information;
- **Query Key Parameter:** setting the query parameter to be used as key parameter;
- **External System:** type of external system where the query will be executed.
- **Script file:** a Remote Query script – see How to develop a script for obtaining data from external entities?;

Scripting

The platform's extensibility scripts allow writing into and reading from external systems, using the Connector (**connector-side** / **on-premise** scripts). These scripts run on the customers' machines, and can use any C# library.

The scripting mechanism also allows for scripts to be developed for the cloud (**cloud-side** scripts), extending the platform's default behaviors.

Scripts must be designed in C#, using this programming language's syntax.

How to install and configure the connector?

It is possible to setup a new connector with the provided installer. During the installation, will be required to enter some configuration data.

In the application, under **Administration**, is possible to configure the available connectors to the tenant. The setup process will be available after creating a connector, the code and license file required to finish.

After the installation, the configuration parameters entered in the installation process can only be changed directly in the connector's configuration file. The file has the following structure:

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
  <Server Endpoint="https://[ServerAddress]/api" />
  <Connector Code="[ConnectorCode]" />
  <References>
    <Reference Path="C:\PathToMyDlls\" />
    <Reference Path="C:\Dev\AnotherPathToMyDlls\" />
```

```
            </References>
        </Configuration>
```

- **Server Address:** The platform's server address;
- **Connector Code:** The connector's code, defined in the application;
- **References:** A set of records, each one referring a folder to load DLLs to be used in scripts.

How to execute the connector?

The connector can be executed in three ways:

- **Development**: through the executable, in the command line or by double clicking. A command line screen will display the information coming in to the connector;
- **Service:** accessing the Windows services screen, locating the connector service and starting it. Information is only displayed in the Windows **Event Viewer**;
- **Script test**: through the command line, where you can test a single script without need to communicate with the platform, by using the following command:

  MyMis.Connector.Service.exe run -script:ScriptFileLocation [-context:ContextFileLocation] [-entity:EntityFileLocation]

  ScriptFileLocation: The full path of the script to execute. The script must contain the Entity C# class;
  ContextFileLocation: The full path of the context data file in a JSON format (optional);
  EntityFileLocation: The full path of the entity sample data file in a JSON format (optional);
  Output: the exit code is 0 if the execution has success.

How to design a *script*?

First, you need to access the modeling area and download the development support tool. This tool is distributed as a .VSIX file that can be found in the **Scripting** section of any entity or interaction on the platform, under the "**Download development solution**" button.

This extension will install a new type of project into your Visual Studio installation (compatible with any edition, from 2012 onwards, but 2015 is recommended), from where you can create a project to begin testing.

Install this file, and create a new solution via Visual Studio's **New Project** menu option – use the project type **MyMis.Scripting**. A set of files becomes available by opening this solution:

- **Default.json:** interaction or entity sample in JSON format, containing the data necessary to test the *script*;
- **Entity.cs:** interaction or entity structure in C#, used for the solution to understand the data structure;
- **Program.cs:** define the context parameters to be passed to the test *script*;

- **Script.cs:** file where the *script* will be designed. The structure of this file is described in What is the structure of a script?.

To get the file to be used in **Entity.cs**, access the modeling area again and use the **Download C# Sample** method, selecting the list of entity or interaction type to be designed.

To get the **Default.json** file use **Download JSON example** on the same place. Data obtained using this method are retrieved from the application.

The structure of the **Entity** is automatically added to the scripts when they are executed on the platform, **except** for cloud-side scripts that execute on **List** and for scripts that execute on **External Entities**. These two cases will require the **Script.cs** file to have a `public class Entity` class containing the necessary structure.

What is the structure of a script?

The structure of scripts (namespace, class, method, parameters and returned value names) cannot be changed, except for adding references (see How to add references to a script?).

The Execute method needs to have the following signature:

```
public ScriptResponse Execute(ContextData context, Entity document, Dictionary<string, object> parameters)
```

This method will return a **ScriptResponse** object, which can contain several different types of responses:

- If you are developing a script that will be used to **query an external system** (in External Entities), the ScriptResponse will need to have a **QueryResult** object, which is composed of object[,] Data and string[] Headers.

  These scripts also need to handle many different cases (query paging, date filtering) that the platform allows, and should be developed based on a provided example.

- *Scripts* used on the **cloud** can return **Actions** to be performed. See How to add actions to a cloud-side script?

- The general use case of external system scripting is returning a **Message**, a string that will be shown on the platform.

You can use this to give feedback to the user about the script's results (e.g. integrated document numbers).

- You can also return a ScriptResponse without any information.

This can be used, for example, for scripts that only need to throw exceptions in abnormal cases, but do not need to inform the user if everything is normal.

How to add references to a *script*?

You can load *assemblies* before executing the script to allow the developer to reference any DLL library necessary for the implementation of connector-side scripts. For this, add a row for each DLL using one of the following formats:

- Full path
  `//#R C:\Program Files\DllName.dll`
- Full path, using environment variables
  `//#R %PROGRAMFILES%\DllName.dll`
- Relative path (using the paths configured in the connector's configuration file)
  `//#R DllName.dll`

On top of this, you need to add the *using* directives (just like any C# project):

```
using MyDll;
```

How to add actions to a cloud-side script?

There are two different kinds of actions that cloud-side scripts can execute: the **File** action, and the **URL** action:

- The **File** action is used for the UI to download a file, passed on the path of the added action;
- The **URL** action is used to redirect users to a new URL.

The example below is an excerpt that could be used to create a **Person Agent**, with the **Code** field being filled with the current document's **Customer** attribute.

```
var fields = $"Code:{document.Customer}";

var urlAction = new Dictionary<string, string>();
urlAction.Add("Location", $"../../Entity/Create/?EntityTypeCode=Person&
MisEntityKind=Agent&View=Default&Attributes=
{HttpUtility.UrlEncode(HttpUtility.UrlEncode(fields))}");

urlAction.Add("Target", "CURRENT");
response.Actions.Add("URL", urlAction);
```

How to access external system parameters?

In the *scripts*, you can use the attributes configured in the application's company / external system mapping. A common example will be having credentials to log into services running on the external system configured on the platform.

The script will receive information about the external systems in the ContextData context object. Here, you have a dictionary called **ExternalSystems** which has all the information for these external systems. Obtaining the value for the external system you want will allow you to get these parameters. For example:

```
var externalSystem = context.ExternalSystems.FirstOrDefault().Value;


var connectionString = new SqlConnectionStringBuilder();
connectionString.DataSource = (string)externalSystem.Parameters["SqlSer
ver"];
```

How to make a cloud-side script communicate with the connector?

Cloud-side scripts have some limitations that connector-side scripts do not. However, you can have a script that is triggered in the cloud access the connector.

A common example is obtaining data from an existing database from another system.

The excerpt below shows what you need to do to invoke the connector from a cloud-side script. After receiving the result, the script can perform any necessary operation on the UI.

```
var fileToExecute = context.Files.Where(f => f.ExecutionType == "Remote").Fir
stOrDefault();

if (fileToExecute == null){
   throw new Exception("File not found");
}

ConnectorClient client = new ConnectorClient(context.Api.Endpoint,
     externalSystem.ConnectorCode);

var connectorResponse = client.Execute(context.Tenant,
new MyMis.Scripting.Core.Contracts.Script()
            {
                Code = fileToExecute.Name,
                Version = fileToExecute.Version
            }, context, document, parameters
            );

 var connectorResult = connectorResponse.Result;
```

Within the connector, it is possible to receive information on what section of the UI called this script. The information is received in the Parameters object, and it may be necessary to parse it (for example, to understand which line of the document needs to be updated).

How to communicate with the platform's API?

You can use a set of support methods to allow *scripts* to access data that only exist in the platform.

The following code sample displays an API client initialization, followed by a platform authentication using the user credentials configured in the external system. After this authentication you can access data using the API.

```
ApiClient apiClient = new ApiClient(context.ServerEndpoint, "TENANT CODE");
apiClient.Authenticate(context.ApiUsername, context.ApiPassword);
apiClient.Entity.GetByCode("PROJ001", "Project", EntityKind.Agent);
```

How to apply platform changes to documents?

Any changes made by the *script* to the entity or interaction can be reproduced in the corresponding platform object. For this, you need to use:

```
document.MarkToApplyChanges();
```

Executing this line of code causes any changes previously made to be reproduced in the platform.

How to test a *script* during development?

Two requirements must be met to be able to locally test your development:

- Configure context data for **access to an external system** (in **Program.cs**);
- Retrieve a **JSON data sample** from the entity/interaction under development (see How to design a *script*?).

After this, development is similar to any other application and you can use *breakpoints*, print debugging information and other operations. To access the Visual Studio debugging tools we recommend you to test scripts in the machine where the external system that will execute the *scripts* is installed.

How to place a *script* in the platform?

Access any **Entity** or **Interaction | Configure Scripts | Add New**

There is a set of parameters that must be configured every time you want to add a *script* to the platform:

- **Name:** a name that identifies the *script*'s operation;
- **External System:** type of external system it is intended for;
- **Execution Moment:** action and moment when the script is executed (see How to set the moment when a script is executed?);
- **On Error:** how the *script* handles an error:
    - **Fail:** does not allow the operation that triggered it to proceed;
    - **Continue:** continues with the operation that triggered it;
    - **Remove:** removes the interaction or entity that triggered the error;
- **Files:** where the file will be *uploaded*:
    - **Execution Type:** location where the *script* is executed: in the platform (Cloud) or in the external system (On-Premises);
    - **File:** *upload* of the .cs file corresponding to the *script* you want to execute.

How to set the moment when a *script* is executed?

Access any **Entity** or **Interaction | Configure Scripts | Create/Edit**

There are several actions from where the configured *scripts* can be invoked:

- **Create:** when creating the entity/interaction;
- **Update:** when updating an entity or interaction;
- **State transition:** when changing approval stages;
- **Revert:** when reversing an entity or interaction;
- **Revert state:** when reversing an approval stage;
- **List Operation:** in the context of an operation on a platform list;
- **Attribute Action:** triggered by something in the UI (for example, a **Button** attribute);
- **Account setup:** when creating a new account based on the current one – used, for example, to load demo data;
- **Remote Query:** script to be executed to perform remote queries to an external system (only in External Entities).

These actions, combined with an execution **moment**, determine which *script*s are invoked:

- **Before:** before the current action;
- **After:** after the current action;
- **Request:** executed upon request.

How to develop a *script* for obtaining data from external entities?

The scripts that are used when querying external entities are mostly the same as any other scripts used in the platform. However, there are a few points to take into account when writing them to guarantee the best user experience.

Remote query scripts must return a **QueryResult** object, which contains the headers, the data and the number of records returned by the query.

```
QueryResult response = new QueryResult()
    {
        Headers = headers,
        Data = data,
        NumberOfRecords = numberOfLines
    };


    return new ScriptResponse
    {
        Result = response
    };
```

There is a series of filtering parameters that are possible with the data the connector provides to the scripts. If you are implementing your own remote query script, we recommend starting your work with a sample remote query script that implements these features:

- Date filtering;
- Security filtering;
- Paging;
- Take (e.g. number of records desired).

Best practices for scripting

There is a set of practices that save development time and guarantee more quality when developing scripts for the platform.

1. While testing/developing your *script*, use the tool described in How to test a script during development?. This avoids the complexity of having to repeat the operations on the platform that will trigger the *script*, and you can still test with real data;
2. After it is developed, however, it is important to ensure that it will also work under real (platform) conditions – some things may not be the same between the two scenarios, such as the references described in How to add references to a script?;
3. When interacting with external systems, ensure that if a method returns a ***dynamic*** object, you cast it to the correct type before proceeding, to avoid obtaining "*Cannot convert object to string*"-type errors;
4. Your *scripts* should be designed considering the possible failure scenarios. See the following case: you integrate a document into an external system, but the external system crashes before returning the response to the platform that it was successfully integrated. This will lead to an inconsistency when you integrate again, potentially causing a duplicate document;

- To avoid this, ensure that there is always a field in the external system identifying the platform's document and vice-versa, and that you perform checks on these fields when running the *script*.

5. Ensure your scripts are developed in the UTF-8 encoding (no BOM);
6. Ensure you always close any connections that you open, even in failure scenarios (e.g. an exception should go into a catch that closes any connections that were still open)
7. Be pessimistic when dealing with responses from external systems – if the connection fails or the response is not correct, your script should fail gracefully.

Web Components

Besides scripting, there is another way to enhance the platform's behavior by programming, through **web components**. This mechanism allows users to write HTML and JavaScript, which will be integrated with the platform's UI to extend it.

Common examples would be displaying a map, integrating data from a Business Intelligence tool, or enhancing the UI with more visual components.

When a user designs a web component, they should evaluate whether it can be **reused** – a single component such as a map may apply to many different screens of the application, and it is not necessary to customize it for each of those screens, as the components can be reused within a single tenant.

How to add a web component to the platform?

Access **Other | Web Components | Add New**

When uploading a web component to the platform, it is necessary to define its **Code**. The Code must be the same as what is defined in the .js file; e.g. if the JavaScript object is **WebComponents.Map** = function () { … }, the Code must be **Map**.

It is also necessary to upload a .js file containing the component itself. This structure is described in How to design a web component?

How to design a web component?

Web components in the platform have a fixed structure.

First, the JavaScript file must contain an object named **WebComponents.<componentcode>.** As explained in How to add a web component to the platform?, this component code will have to be the same as the Code used in the platform.

Inside that JavaScript object, there must be a function that returns the HTML that the component is associated with, in a string. This function must be named **this.html.**

```
this.html = function () {
    return '<iframe id="ShippingAddressMap" width="100%" height="450" frameborder="0" style="border:0" allowfullscreen src="https://www.google.com/maps/embed/v1/place?key=@@APIKEY@@&zoom=5&q=Portugal" ></iframe>';
};
```

Optionally, there is a function that is called when the web component is added to the page, **this.initialize.** The function is where any custom behaviors may be added – examples would be adding listeners so that other functions may be defined (setting functions that can be called when users click parts of your component or when a field is changed), or calling any auxiliary functions.

```
this.initialize = function (domElement) {
    document.getElementById("IBAN").addEventListener('change', validateIBAN);

};

var validateIBAN = function (domElement) {
    ...
};
```

Many common usage cases will not require usage of the **initialize** function, as you will only need to embed an iframe – these components are simpler to design and maintain, as they do not require writing any JavaScript.

How to use a web component in the platform?

There are three separate usage scenarios for Web Components:

- **In a list:** As a column of any list of the platform. Common usage scenarios: showing an image of a product; Having a audio/video sample of the list entry.

  To model – When creating or editing a list, define the attribute's type as **Web Component** and identify the desired Component.

- **In a dashboard:** As a dashboard element like a list or a chart. Common usage scenarios: displaying external data from a BI platform.

  To model – When creating or editing a dashboard, use the context buttons at the bottom to **Add a Web Component**.

- **In an entity or interaction:** As an UI Element of an entity or interaction. Common usage scenarios: Custom input forms; Visual enhancements to the UI of the platform.

  To model – When creating or editing an interaction or entity, access the UI Elements section, and add a new UI element, setting it to Web Component type and identifying the Web Component.

Best practices for developing Web Components

There is a set of practices that save development time and guarantee more quality when developing scripts for the platform.

1. Ensure that you are comfortable with any information in your web component being, potentially, public, as they are stored in a publicly-accessible storage. This means any API keys in your component may be seen by others.
2. Use, as much as possible, native JavaScript, as any imported libraries cannot be guaranteed to work.

Notifications

The platform allows you to **send email notifications** for several operations regarding entities and interactions. To manage notifications for a specific type of entity or interaction you need to access that type and select the "**Configure Notifications**" option.

Email notifications can be sent for the following entity or interaction operations:

- **Create:** creation of a new entity or interaction;
- **Update:** updating an existing entity or interaction;
- **Go to Approve:** changing an entity or interaction's approval stage;
- **Revert:** reversing an entity or interaction's approval stage.

You can configure several notifications to be sent for each of these operations. To configure them, see How to set an email notification?.

The platform automatically creates a set of notifications to be sent for the most common cases (see the Examples of notifications attachment).

How to set an email notification?

Access **Create or edit an interaction | Configure Notifications**

To setup a notification, you need to complete a set of attributes:

- **Condition:** formula with a condition that, when evaluated, determines whether the notification is sent (optional). See How to set a condition on a notification?;
- **Approval Stage:** the notification is sent whenever an entity/interaction enters the configured stage. It only applies to **Go to Approve** operations;
- **To:** contains the email address where the notification will be sent;
- **CC:** contains the email address where the notification will be sent as CC;
- **BCC:** contains the email address where the notification will be sent as BCC;

- **From and From Display Name:** It is possible to define, per notification, optional parameters for the email's From field. Users can model a **From** address, using the same syntax as To, CC and BCC, and also set a **Display Name** for that email address. If the option to use a custom From address is not set, all notifications will be sent from the email configured in that installation of the platform.
- **Subject:** the email subject, localizable into several languages. It can be composed based on parameters (see How to set notification parameters?);
- **Parameters:** list of interaction attributes (separated by commas) that will be used to compose the email subject (see How to set notification parameters?);
- **Body:** email body, translated into the application's several languages. Allows for HTML syntax and can be composed based on parameters (see How to set notification parameters?);
- **Parameters:** attribute list, separated by commas, which will be used to compose the email body (see How to set notification parameters?).

How to set notification parameters?

Access **Create or edit an interaction | Configure Notifications**

You can introduce text samples based on attributes through the use of parameters, both in the subject and in the body of emails sent by the notification logic. The logic for the notification parameters consists on passing a set of attributes, separated by commas and between brackets, in the **Parameters** field of the notification screen.

The attribute access formula uses the Data Access logic, with the difference that **you do not need to reference the type of entity** when you want to access an entity's attributes (for example, ProviderAgent.Code).

Then, you can reference them in the **Subject** and **Body** by using a syntax similar to the *string* formatting method: for example, {0} will be replaced by the first element in parameters.

This section also supports a set of specific attributes:

- **@URL-PROCESSINTERACTION-APPROVE@:** generates a link to approve a specific interaction in the email;
- **@URL-PROCESSINTERACTION-EDIT@:** generates a link to edit a specific interaction in the email
- **@URL-ENTITY-APPROVE@:** generates a link to approve a specific entity in the email
- **@URL-ENTITY-EDIT@:** generates a link to edit a specific entity in the email.

How to set a condition on a notification?

Access **Create or edit an interaction | Configure Notifications**

The **Condition** field determines when the notification will or will not be sent. This condition uses the Data Access logic, with the following differences:

- You do not need to reference the type of entity when you want to access an entity's attribute (for example, *ProviderAgent.Code*);

- You do not need the '#' when accessing non base fields;
- Comparisons use '=' and '<>' and to compare with an empty value simply leave the other part of the comparison blank, for example: "*AlternativeEmail*=" is a statement that declares that the *AlternativeEmail* field has a value;
- Boolean variables are written **TRUE** and **FALSE**.

Menu

The platform automatically generates the menu displayed to the user as the application is modeled. However, since the organization of the different elements may not be the best for specific applications, you can develop a custom menu tailored to individual needs.

The application's default menu cannot be removed and is automatically updated when a new element is added to the model.

How to create a custom menu?

Access **Menu | Navigation | Application menus**

To simplify the process of adding a custom menu, the platform suggests a structure similar to the automatically generated menu.

If all custom menus are inactivated, the default menu is activated, guaranteeing that the users always have the option to navigate the application.

The different elements that make up a menu must be configured and connected to each other as tree connections with a maximum depth of five levels. There are three types of elements that you can use:

- **Container:** element that aggregates other elements;
- **Application Link**: link inside the same application;
- **External Link**: link to an external element.

How to add a link to another application?

Access **Menu | Navigation | Application menus**

The platform provides a menu element that allows you to create a hyper link to any webpage. To set an **External Link** type element you need:

- **Text:** text displayed on the menu (localizable into different languages);
- **Icon:** icon to be displayed – see What icons are available in the platform?;
- **URL:** hyper link to be accessed.

How to add a link to options inside the same application?

Access **Menu | Navigation | Application menus**

The **Application link** element allows access to a platform feature. To set the access to a feature you need:

- **Text:** text displayed on the menu (localizable into different languages);
- **Icon:** icon to be displayed – see What icons are available in the platform?;
- **Link To:** type of platform feature to which you want to point (Agent, Interaction, etc.) and indicate the specific element;
- **Operation:** if applicable, you can set the operation to which the menu item provides access (List, create or approve);
- **View:** if applicable, you can set which view will be displayed;
- **Approval Stage:** if applicable, you can set which approval stage to which the menu item provides access.

How to add a link container?

Access **Menu | Navigation | Application menus**

A **Container**, or link container is a menu element that works like an aggregating element. No action can be performed from this container, it only provides access to other menu elements.

To set a "**Container**" type element, you need to specify the icon and the text to be displayed in the menu (localized into several languages).

What icons are available in the platform?

The platform uses FontAwesome to provide its icons. You can use the selection boxes in the menu to select them, or check out http://fontawesome.io/icons/ for the organized list. Icons are identified without the "fa-" prefix in the platform.

Security

All applications modeled on the platform have access to an **administration area**. In this area, you can manage security configurations and, although not exactly under the same domain as the modeling, it is important to introduce the different configurations available in this section.

How to create a role?

Access **Application | Administration | Security Configurations | Roles**

Configuring application roles allows you to attribute different users privileges, ensuring control of the access to the different operations. Configuration is very detailed:

- **Interactions:** interaction access privileges are managed in the context of companies, allowing users different interaction access privileges according to the platform company and/or document series;

  There is an option available to copy the privileges of the interaction using a company's privileges as template.

  Interactions will not be shown in the menu if there are no company, as you cannot have privileges for a company/series that do not exist.

- **Entities:** privileges set for each type of entity;
- **Operations:** privileges are applied to entities/interactions individually for each operations (Data access, create new, edit, remove, among others). Security filters can be applied to some operations (see How to set a security filter?);
- **Views:** access privileges take into account the different entity/interaction views. You can provide users with limited access privileges for them to only have access to a specific set of views;
- **Approval stages:** if there are any approval stages, privileges take them into account. You can ensure that users can only access the necessary stages (see How to set a security filter?);
- **Dashboards:** each role may or may not have access to each dashboard;
- **Others:** remaining privileges are individually attributed and not relevant from the modeling point of view.

How to set a security filter?

Access **Application | Administration | Security Configurations | Roles**

In role creation, security filters can be applied to two different locations: in **operation access** and **approval stage access**. The syntax to be used in any of these cases is described in the Data access section.

How to add privileges to access a List?

1. Access **Application | Administration | Security Configurations | Roles;**
2. Select the Role to Edit or Create a new one;
3. Select the type of the List;
4. Select the **List** checkbox;
5. Additionally, a filter can be provided to restrict the data that can be listed;
6. In the **Views** section, select the checkbox related to the given List;

How to add privileges to create or edit an entity?

1. Access **Application | Administration | Security Configurations | Roles;**
2. Select the Role to Edit or Create a new one;
3. Select the type of the Create or Edit;
4. Select the **Create** or the **Edit** checkbox;
5. In the **Views** section, select the checkbox related to the View of Create or Edit;

How to create a user?

Access **Application | Administration | Security Configurations | Users**

Configuration is required for a set of fields when you access the user creation screen from an application:

- **Name:** name to be displayed in the application;
- **Email:** email to be used to *login*;
- **Email contact:** email to be used for contacts (notifications, etc.);
- **Password:** password field. See What are the platform's password rules?;
- **Associate User to Agents:** assignment of agent instances in the platform to users;
- **Associate Privileges:** assignment of roles.

What are the platform's password rules?

Access **Application | Administration | Security Configurations | Users**

The following password rules for the platform were designed to comply with the NIST Special Publication 800-63, namely section SP800-63B:

Minimum Password Length:

- 8 characters

Passwords must meet at least 3 out of the following 4 complexity rules:

- at least 1 uppercase character (A-Z)
- at least 1 lowercase character (a-z)
- at least 1 digit (0-9)
- at least 1 special character

There is no requirement of password expiry.

How to inactivate a user?

Access **Application | Administration | Security Configurations | Users**

To inactivate a user, you need to edit it. In the document header you can see the **Inactive** *checkbox* that you must select. After saving, the user will become inactive.

Errors

There is a set of situations in which the platform returns errors when the modeling user is developing the application, which may cause the modeling environment to terminate. This section describes some common situations and how to overcome them:

Error	Solution
Your user has no AGENT related. Please check user configuration.	Make sure that an agent of the desired type was created and assigned to the current user in user editing.
An error has occurred: Model error (…) Unable to process binding (…)	An error has occurred in one of the configured formulas. Validate if the attribute formulas are correct.
External Entity created. An error has occurred while creating the default list.	When you create the external entity the platform tries to generate a default list, but it will not be created if there is an error connecting to the external system. A list is created containing only the Query Key Parameter. To solve the issue, edit the list manually to match what you want.

Help Links

The platform has a functionality that allows you to define URLs that will show up as 'help' links for some of its elements.

These can be defined for **Dashboards**, **Numerators**, **Security Configurations** (Users and Roles), and **Tools** (e.g. the File Inbox).

Model comparison tool

The platform includes a tool to compare two different models and ask for user input to apply (part of or all) the differences between them to the target model. This tool can be downloaded from the platform, and it only requires an account with modeling privileges in both the source and destination tenant.

How to execute the tool?

The tool is distributed as an interactive console application that can be used in three different execution modes:

- **Download**, which downloads all the information of the model for two *tenants*;
- **Compare**, which obtains the difference between two downloaded models;
- **Apply**, which uses the file with the list of changes from Compare to apply them on the target *tenant*.

For each of these execution modes, each *tenant* is either:

- **Source:** Basis for the comparison. You will be prompted to apply on the target any changes performed on this *tenant*.
- **Target:** Destination for the comparison. You will be prompted to discard any changes performed on this *tenant*.

After changes are identified in the **Compare** step, a **Changes.json** file with the identification of each change is produced. You can edit this file, determining which changes you want to apply (see How to manually apply changes?).

The execution is performed via the command line, calling the executable file with the following parameters:

- **Label:** Path (full or relative) to the folder to be created to save the files in.
- **Execution Mode:** Text that identifies the desired execution mode: Download, Compare, Apply, or DownloadAndCompare (executes both steps consecutively).
- **Config:** (Optional) Path (full or relative) that points towards the configuration file to use. If not present, Config.json is used.

After execution, two files with a summary of the information that the application returned during its execution are produced:

- Log.html: contains the information of the execution in a readable format. Automatically opened in the browser;
- Log.txt: contains only raw information.

How to configure the tool?

The tool retrieves its configuration from a json file that the user should modify before execution. This file identifies the two tenants we want to compare, each with the following fields:

- **Endpoint:** API URL of the platform subscription the *tenant* is on, e.g. https://www.mymis.biz/api;
- **Tenant:** Code (GUID) representing the *tenant*. Can be obtained from the modeling main page's tenants list;
- **Username:** Platform login with modeling access to this *tenant*;
- **Password:** Password for this login.

The default configuration file used, as described in How to execute the tool?, is **Config.json**. For organization purposes, you can pass a different file as an argument.

How to manually apply changes?

A **Changes.json** file is produced in the *Result* folder after the **Compare** step, along with the files matching the file changes identified.

In case the default process was not enough, these files may be manually edited to obtain the desired result.

The changes file has a parameter for each of these changes that indicates whether they will be applied in the **Apply** step.

Any manual changes may lead to inconsistencies in the process (e.g. not creating an entity that is referenced elsewhere).

How to handle platform errors?

Sometimes platform errors may occur when you apply the changes. If this happens, there are different ways the tool can handle them:

- **Continue:** Ignores the error and applies the remaining changes. **Continue All** does the same, but for all errors;
- **Retry:** Attempts to apply the change again. Useful for scenarios such as transitory network or database failures;
- **End:** Terminates the apply process **(does not revert already executed commands)**.

Errors obtained in the download step and some critical errors from other steps will interrupt the process (e.g. login failures on the Apply step).

How to control the order of operations?

The **Changes.json** file includes an order for the operations – the order by which the files show up in the JSON. These objects may be moved inside the file to guarantee that steps are executed in the desired order.

Limitations

- In some scenarios, the application of changes depends on other changes in ways that the tool does not automatically solve. To deal with these cases, see How to control the order of operations?;
- The platform does not have the concept of 'removing' notifications or approval stages. These cases are treated as an update which removes all the entries;
- Applying differences between entities with multiple lists may cause the order of the lists to be different from the original order. In cases where there is no list set as Default, the platform will assume the first list as the default list, which may cause issues (for example, with approvals). To guarantee this does not happen, every entity with multiple lists should have one marked as default;
- The tool does not compare roles, as they are not part of the model.

Attachments

Attachment 1: Problem description

Many examples described throughout this guide are based on a concrete case – modeling a supplier order management application for a company.

The objective of this implementation is for Employees to have the freedom to introduce supplier item order requests and report receiving them at a later point.

The following behaviors are desired for these two operations:

- **Order Request:**
  - Placed by the employees;
  - Identification of the different items in a document;
  - Request approval by the manager;
- **Item Delivery:**
  - Placed by the employees;
  - The report must identify the items in the order requests;
  - It must also allow for items not expected in the initial request;
  - It must be possible to close item entries when they are received, not allowing any more entries for a specific order request.

The platform entities that address this problem are:

- Agents:
  - **Employee:** responsible for introducing documents;
  - **Company:** entity with which the Employee is associated. Receives the ordered items;
  - **Supplier:** entity with whom orders are placed. Supplies the ordered items;
- Resources:
  - **Item:** entity that represents items to be ordered;
- Processes:
  - **Order management:** element that aggregates operations related with order management;
- Interactions:
  - **Order request:** interaction that represents a new supplier order request;
  - **Item order:** transactional entity of the commitment type that represents a promise of delivery. Modeled as an item increment (resource) between a supplier (supplier agent) and the company (receiving agent);
  - **Item delivery:** interaction that represents a new item delivery for the company;
    - **Item entry:** transactional entity of the event type that represents the actual entry of the item in the company. Modeled as an item increment between the supplier and the company. Completes the order commitment.
- Integrations:

- The order request will generate a document of the Supplier Order (ECF) type in the Primavera ERP.

Attachment 2: Formula references

This section contains a reference to all features tested in the platform. Examples for common use cases can be found in Formula examples.

Mathematical operations

Operation	Description
+	Sum of two values.
-	Subtraction of two values.
*	Multiplication of two values.
/	Division of two values.

Boolean operations

Operation	Description
AND(set)	Given a set of conditions, the AND operation returns true if all conditions are true.
OR(set)	Given a set of conditions, the OR operation returns true if at least one condition is true.
NOT(cond)	Given a "cond" condition, the NOT operation returns the opposite logical value of the condition - true if the condition is false and false if the condition is true.
XOR(set)	Given a "cond" condition, the NOT operation returns the opposite logical value of the condition - true if the condition is false and false if the condition is true.

IF(cond,trueAction, falseAction)	Given a "cond" condition, the IF operation returns the trueAction value if the condition is true and the falseAction value if it is false.

Comparison operations

Operation	Description
>	A value greater than another value.
>=	A value greater than or equal to another value.
==	A value that is the same as another value.
!=	A value that is different from another value.
<=	A value lesser than another value.
<	A value lesser than or equal to another value.

Mathematical operations

Operation	Description
SUM(set)	Given a set of attributes, the SUM operation returns the sum of the values of these attributes.
PRODUCT(set)	Given a set of attributes, the PRODUCT operation returns the product of the values of these attributes.
SUMIF(set,cond)	Given a set of attributes, the SUMIF operation returns the sum of the values of the attributes that meet the "cond" condition.
SUMIFS(set1, cond1, set2, cond2…)	Given several setN - condN attribute - condition pairs, the SUMIFS operations returns the sum of values of the attributes that meet the associated condition.

SIGN(n)	Given a "n" number, it returns 1 for positive, 0 for 0, or -1 for negative.
ABS(n)	Given a "n" number, the ABS operation returns its absolute value.
ISEVEN(n), ISODD(n)	Given a "n" number, ISEVEN(n) returns true if the value is even, false for odd; while ISODD(n) returns false for even and true for odd.
AVERAGE(set)	Given a set of attributes, the AVERAGE operation returns the average value of these attributes.
AVERAGEIF(set,cond)	Given a set of attributes, the AVERAGEIF operation returns the average value for the attributes that meet the "cond" condition.
MEDIAN(set), STDEV(set), DEVSQ(set)	Given a set of attributes, the MEDIAN operation returns the median value of these attributes, the STDEV operation returns their standard deviations, and the DEVSQ operation returns the standard square standard deviation (variance).
COUNT(set)	Given a set of attributes, the COUNT operation returns the number of occurrences of these attributes.
COUNTIF(set,cond)	Given a set of attributes, the COUNTIF operation counts the number of occurrences of the attributes that meet the "cond" condition.
COUNTBLANK(set)	Given a set of attributes, the COUNTBLANK operation counts the number of occurrences of empty or null attributes.
COUNTUNIQUE(set)	Given a set of attributes, the COUNTUNIQUE operation counts the number of occurrences of attributes different from all the others.
MAX(set), MIN(set)	Given a set of attributes, the MAX operation returns the attribute with the highest value; while the MIN operation returns the one with the lowest.

Date handling operations

Operation	**Description**

DAY(d), MONTH(d), YEAR(d)	Returns the day, month and year, respectively, which corresponds to a complete "d" date.
NOW()	Returns the current time and date. If applied to date fields, only the date value is used.
HOUR(h), MINUTE(h), SECOND(h)	Given an "h" time (or time and date), returns the value that corresponds to the hours, minutes and seconds.
NETWORKDAYS(d1, d2)	Returns the net work days between two dates.

Text handling operations

Operation	Description
UPPER(txt), LOWER(txt)	Given a "txt" text, the UPPER operation converts the entire text to upper case; while the LOWER operation converts it to lower case.
CONCATENATE(txt1, txt2…)	Given a set of "txt1"..."txtN" texts, the CONCATENATE operation joins them into a single text.
FIND(txt1, txt2, [optional] position)	FIND verifies if txt1 is contained within txt2 and returns its index within that text. Position is the start position. DOES NOT IGNORE CASING!
SEARCH(txt1, txt2, [optional] position)	FIND verifies if txt1 is contained within txt2 and returns its index within that text. Position is the start position. Ignores casing.
SUBSTITUTE(text, oldt, newt)	SUBSTITUTE replaces all occurences of *oldt* within *text* with *newt*.

Platform context references

These parameters can be used in formulas on the platform, where applicable.

Parameter	Description
[Context.viewMode]	Retrieves the value of the current view mode of the document – CREATE, EDIT or APPROVE.

[Context.ApprovalStage]	Retrieves the current approval stage.
[Context.User.Email], [Context.User.Name], [Context.User.Language]	Retrieves information on the current user.
@ROWNUMBER	Returns the current row number in a Details or Summary element.

Attachment 3: Formula examples

Mathematical operations

=[UnitPrice]*[Quantity]

This formula allows you to assign the result of multiplying "UnitPrice" by "Quantity" to an attribute.

=([UnitPrice]*[Quantity])/12

Based on the previous formula, calculates the total price divided by 12.

Boolean operations

=IF([Selected]==true&[RowNumber]==[@1],true,false)

This formula returns true only when the **Selected** *checkbox* is checked in the first row of the selected entity.

=IF([Context.viewMode]==[@CREATE],TODAY(),[DateCreated])

This formula sets the DateCreated to TODAY() if we are in a Create; else, it maintains the date that was already there.

Mathematical operations

=SUM([Details.GoodsPurchaseRequest.Amount])

This formula allows you to assign the sum of the "**Amount**" attribute in all transactional entities (rows) of the "**GoodsPurchaseRequest**" type to an attribute.

=PRODUCT([Details.GoodsPurchaseRequest.Amount])

This formula allows you to assign the product of the "**Amount**" attribute in all transactional entities (rows) of the "**GoodsPurchaseRequest**" type to an attribute.

=PRODUCT(1,2,15)

Obtained value: 30

=SUMIF([Details.GoodsPurchaseRequest.Quantity], '>10')

This formula allows you to assign the sum of all "Quantity" attributes for all transactional entities (rows) of the "GoodsPurchaseRequest" type if the order was made with a quantity higher than 10 to an attribute.

=SUMIF([2,4,8,16], '>5')

Obtained value: 24

=SUMIFS([Details.GoodsReception.VATIncidence], [Details.GoodsReception.VatRate], '==' + [VatRate])

This formula adds all the values in VATIncidence of the Details.GoodsReception, if their VatRate matches this element's VatRate.

=ABS(-5)

Obtained value: 5

=SIGN(-35)

Obtained value: -1

=ISEVEN(2)

Obtained value: true

=AVERAGE([Details.GoodsPurchaseRequest.Amount])

This formula allows you to assign the average value of "**Amount**" attributes in all transactional entities (rows) of the "**GoodsPurchaseRequest**" type to an attribute.

=AVERAGE([1,2,3])

Obtained value: 2

=AVERAGEIF([Details.GoodsPurchaseRequest.Amount], '>10')

This formula allows you to assign the average value of "**Amount**" attributes in all transactional entities (rows) of the "**GoodsPurchaseRequest**" type whose value is higher than 10 to an attribute.

=MEDIAN([Details.GoodsPurchaseRequest.Amount])

This formula allows you to assign the median value of the "**Amount**" attribute in all transactional entities (rows) of the "**GoodsPurchaseRequest**" type to an attribute.

=COUNT([Details.GoodsPurchaseRequest.ProviderAgent])

This formula allows you to assign the number of different suppliers set in transactional entities (rows) of the "**GoodsPurchaseRequest**" type to an attribute.

=COUNT([1,2],[3,4])

Obtained value: 4

=COUNTIF([Details.GoodsPurchaseRequest.Amount] , '>10')

This formula allows you to assign the number of all transactional entities (rows) of the "**GoodsPurchaseRequest**" type whose "**Amount**" value is higher than 10 to an attribute.

=COUNTBLANK([Details.GoodsPurchaseRequest.Amount])

This formula allows you to assign the number of all transactional entities (rows) of the "**GoodsPurchaseRequest**" type whose "**Amount**" value is empty or null to an attribute.

=COUNTBLANK([1,'',0,2,null])

Obtained value: 3

=COUNTUNIQUE([1,1,2,2,3,3])

Obtained value: 3

=MAX([Details.GoodsPurchaseRequest.Amount])

This formula allows you to assign the highest "**Amount**" value of all transactional entities (rows) of the "**GoodsPurchaseRequest**" type to an attribute.

=MIN([1,2,3,4])

Obtained value: 1

Date handling operations

=MONTH(DateOccurred)

Retrieve the month value from a DateOccurred field (by default) containing the date in which a specific event occurred (for example, an order was placed). You can also retrieve the day (DAY) or year (YEAR) values in the same way.

=NOW()

Assign the current date value to an attribute.

```
=DAY([Today]) + '/' + MONTH([Today])
```

Based on the previous example and using it to fill a new field with this sequence, model a field with the name **Today**, which will be displayed in the following format “15/4”.

```
=HOUR(NOW())
```

Use the NOW() operation to retrieve the current date/time value and then retrieve the current hour (UTC) from that date.

```
=MINUTE("1/1/2016 15:05:01")
```

Obtained value: 5

```
=DATE(YEAR([Today]),MONTH([Today]),DAY([Today])+5)
```

Add five days to the current day. This can also be used for years or months.

Text handling operations

```
=UPPER(“Winter”)
```

Obtained value: WINTER

```
=LOWER(“Autumn”)
```

Obtained value: autumn

```
=CONCATENATE(“John”, “ ”, “Smith”)
```

Obtained value: John Smith

```
=MID(“John de Sousa”,5,7)
```

Obtained value: de

```
=LEN(“Autumn”)
```

Obtained value: 6

```
=LEFT([BudgetItemYear],SEARCH(':',[BudgetItemYear])-1)
```

Takes the value in BudgetItemYear and obtains the first part of it, split by ‘:’.

```
=SUBSTITUTE([HyperlinkHelper],[@{0}],[Document.CompanyCode])
```

Replaces {0} in the attribute HyperlinkHelper with the document’s Company.

Attachment 4: Data access formulas

Formula	Description
#Supplier	Access the non-base “Supplier” attribute.
Agent:Company.Name	Access the “Company” attribute, whose base type is Agent. The “Name” attribute is selected from all the information available in the “Company”.
Resource:#Item.#Price	Access the “Item” attribute, whose base value is Resource. Next, the “Name” attribute of the “Item” is accessed.
UserDefinedEntity:#Supplier.Code	Access the “Supplier” attribute, whose base type is User Defined. Next, the “Code” attribute of the “Supplier” is accessed.
Interaction:Interaction.DateCreated	Access the “DateCreated” attribute, available in the context of an interaction. The prefix is only necessary in the context of another element (for example. a commitment or event).
@USER.Email@	Access the “Email” attribute of the “USER” system variable. This formula provides access to the email of the user currently logged in.
@USER.Employee.Code@	Access the “Code” attribute of the "Employee" from the “USER” system variable. This formula provides access to the employee code associated with the user currently logged in.
@LANGUAGE@	Obtain the current user language. It can be used in external entities, for example, to show things differently depending on language.
Employee.User	Access the user assigned to the “Employee” agent in the entity or interaction being approved.
Employee.Manager.OutOfOffice =TRUE	Verify whether the user assigned to the agent represented by the “Manager” attribute in the

	"Employee" agent has its OutOfOffice attribute set to TRUE.
Employee.Manager.User<>	Verify if the user assigned to the agent represented by the "Manager" attribute in the "Employee" agent exists.
Company.HRApprover<>& **Company.HRApprover<>Employee**	Verify whether the company has an agent represented by the "HRApprover" attribute. If it does, verify if it is the current Employee.

Attachment 5: Base attributes

Entity base attributes

Attribute	Description	Required
ID	Entity identifier automatically generated by the platform.	Yes
Code	The entity's code. This code is managed by the user and must be unique for each entity type.	Yes
Name	The entity's name.	Yes
Description	Other entity descriptions.	No
Inactive	Indication of whether the entity is inactive. If so, the entity will not be visible and cannot be used in interactions.	No
ExternalSystemCode	The external system's code. For those cases in which the entity is connected to an external system.	No
ExternalCode	The external entity's code. For those cases in which the entity is connected to an external system.	No

For Entity Items, the **Name** and **Description** are removed and the following is added:

Attribute	Description	Required
RowNumber	Number of the row where the Entity Item is located.	No

For External Systems the **ExternalSystemCode** and the **ExternalCode** are removed and the following set of attributes is added:

Attribute	Description	Required
Company	Number of the row where the Entity Item is located.	Yes
Endpoint	The connector's identifier.	Yes
Username	User name used to login into the external system.	No
Password	Password to login into the external system.	No
SystemVersion	Version of the external system.	No
Channel	The channel for communication with the connector.	Yes
ClientCertificate, ClientCertificateThumbprint, ClientCertificateExpirationDate	Client certificate information (for SSL connections).	No
ServiceCertificate, ServiceCertificateThumbprint, ServiceCertificateExpirationDate	Service certificate information (for SSL connections).	No

Transactional entities base attributes

Attribute	Description	Required
ID	Entity identifier automatically generated by the platform.	No
Code	The entity's code. This code is managed by the user and must be unique for each entity type.	No
RowNumber	Number of the row where the transactional entity is located.	Yes
Resource	The resource that undergoes transaction.	Yes
Quantity	The quantity of the resource undergoing transaction.	Yes
Amount	The value of the resource undergoing transaction.	Yes
DateOccurred	The date when the transaction occurred.	Yes
WhenNoticed	The date when the transaction was validated.	Yes
Inactive	Indication of whether the transactional entity is inactive. If so, the entity will not be visible and cannot be used in interactions.	Yes
ProviderAgent	The agent that provides the resource undergoing transaction.	Yes
ReceiverAgent	The agent that receives the resource.	Yes
TransactionKind	Event or commitment.	Yes
EntityType	Type of event or commitment.	Yes

CommitmentToFulfill, AmountToFulfill, CloseCommitment	Configurations associated with the *fulfillments* mechanism.	No
TransactionalEntityToRectify, AmountToRectify	Options associated with the rectifications mechanism.	No
ExternalSystemCode	The external system's code. For those cases in which the transactional entity is connected to an external system.	No
ExternalCode	The external entity's code. For those cases in which the transactional entity is connected to an external system.	No
Duration	The duration of an event.	Yes
ScheduledDate	Date when a commitment fulfillment is expected.	Yes
QuantityFulfilled	Quantity fulfilled by the commitment.	No

Interaction base attributes

Attribute	Description	Required
NumberSerieCode	Interaction document series in the platform.	Yes
NumberSerieID	Identifier of the interaction document series in the platform (associated with NumberSerieCode).	Yes
Number	Document number within the platform series.	Yes
DateCreated	Date when the document was created.	Yes
CompanyCode	Code of the company with which the interaction is associated.	Yes

ExternalSystemCode	The external system's code. For those cases in which the interaction is connected to an external system.	No
ExternalCode	The external entity's code. For those cases in which the interaction is connected to an external system.	No

Attachment 6: Examples of notifications

Default notifications are added when you add approval stages to an interaction or entity.

Default approval notifications

Initial Stage:

Field	**Value**
To:	[CreatedBy.ContactEmail]
Subject:	{0} rejected
Parameters:	[MisEntityType.Code]
Body:	Your {0} {1}/{2} has been rejected by {3}
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number],[PreviousApprover.Email]

Intermediate Stage(s):

Field	Value
To:	[ApproverContactEmail]
Subject:	{0} {1}/{2} is waiting for your approval
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number]
Body:	{0} {1}/{2} is waiting for your approval Stage: IntermediateStage Created by: {3}
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number],[CreatedBy.ContactEmail]

Final Stage(s):

Field	Value
To:	[CreatedBy.ContactEmail]
Subject:	{0} approved
Parameters:	[MisEntityType.Code]
Body:	Your {0} {1}/{2} has been approved by {3}
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number],[PreviousApprover.Email]

Default reversal notifications

All stages:

Field	Value
To:	[PreviousApproverEmail]
Subject:	{0} {1}/{2} has been reverted
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number],[Approver.Email]
Body:	{0} {1}/{2} was reverted and is no longer waiting for your approval. Please ignore previous e-mail Reverted by: {3} Reversion date: {4}
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number],[Approver.Email],[ModifiedDate]

Other examples

Informing a user (Employee agent) that their document was created and sent for approval, including the URL:

Field	Value
To:	[Employee.User.ContactEmail]
Subject:	Your {0} {1}/{2} has been successfully inserted
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number]
Body:	{0} {1}/{2} is now waiting for approval <b>Designated approver:

	{3} Approval stage: {4} Value: {5} Date: {6} Click here to see the document
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number],[Approver.Email],[ApprovalStage.Name.DefaultText],[Amount],[CreatedDate],@URL-PROCESSINTERACTION-EDIT@

Informing an approver that they have a document to approve, submitted by a specific person (Employee agent) including the URL:

Field	**Value**
To:	[ApproverContactEmail]
Subject:	{0} by {1} is waiting for your approval
Parameters:	[MisEntityType.Code],[Employee.Code]
Body:	{0} {1}/{2} is waiting for your approval Stage: Approval Designated approver: {3} Created by: {4}-{8} Value: {5} Date: {6} Click here to see the document
Parameters:	[MisEntityType.Code],[NumberSerieCode],[Number],[Approver.Email],[Employee.Code],[Amount],[CreatedDate],@URL-PROCESSINTERACTION-APPROVE@,[Employee.Name]