Mendix Assist

1 Introduction

Mendix Assist refers to Mendix platform capabilities that leverage Artificial Intelligence (AI) and Machine Learning (ML) to assist developers in application development, the so-called AI-Assisted Development (AIAD). The purpose of Mendix Assist is to help development teams in modeling and delivering Mendix applications faster, more consistently, and with higher quality.

Mendix Assist consists of different capabilities that act as a virtual co-developer bot, each specialized in a certain domain or stage of the application lifecycle development. Currently, Mendix Assist consists of the following virtual co-developer bots:

- **MxAssist Logic Bot** – helps you model and configure microflows in Mendix Studio Pro. It gives you contextualized recommendations on the next best activity in your microflow based on the activities and parameters that are already configured in the specific microflow. It is built using machine learning analysis of over twelve million anonymized application logics (microflows) built with Mendix. It uses deep learning to detect and learn the best practice patterns in microflows. For more information, see MxAssist Logic Bot.
- **MxAssist Performance Bot** – helps you inspect your app against Mendix development best practice detecting and pinpointing development anti-patterns and, in many cases, automatically fixing them. The bot is built using statistical analysis of thousands of anonymized Mendix apps to learn common anti-patterns as well as Mendix Expert Services best practices in the development of microflows, domain models, pages, security, etc. For more information, see MxAssist Performance Bot.

MxAssist Logic Bot

1 Introduction

MxAssist Logic Bot is an AI-powered virtual co-developer bot that helps you to model and configure your application logic (microflows) in Mendix Studio Pro. It gives you contextualized recommendations on the next best activity in your microflow based

on the already designed activities, parameters, and other context-related information.

MxAssist Logic Bot is built using machine learning analysis of over twelve million anonymized application logics (microflows)*—*built with Mendix*—*to detect and learn the best practice patterns in microflows.

The key features of MxAssist Logic Bot are the following:

- **Next best action suggestion** – it recommends the top five next best activities out of more than 40 different options with accuracy of 95%.
- **Auto-configuration** – it does not only provide next best action, but automates the development further by pre-populating the parameters for such action.
- **Contextual suggestions** – it derives context in different ways, including by 'looking' left and right in a microflow when the developer inserts a new activity or decision mid-flow; and by inferring the context using the page where it is called from.
- **High accuracy** – continuous improvement and training of the model has elevated the accuracy level from 95%.

2 MxAssist Logic Bot Settings

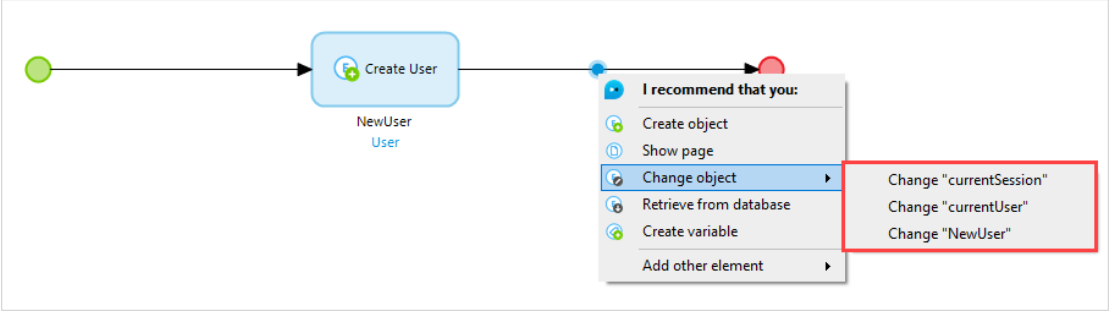
You can switch MxAssist Logic Bot on and off in the top right corner of the microflow editor.

To access settings of MxAssist Logic Bot,

open **Edit** > **Preferences >**the **General** tab >the **MxAssist Logic Bot** tab. For more information, see Preferences.

In the **MxAssist Logic Bot** tab, you can set the following:

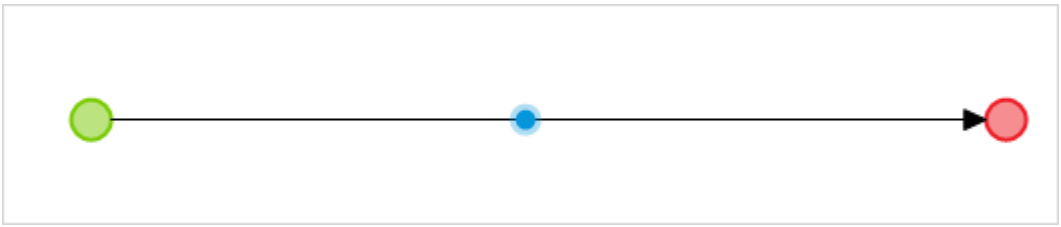
- **Enable MxAssist Logic Bot** – switches MxAssist Logic Bot on and off
- **Show suggestions for system variables** – when enabled, MxAssist Logic Bot will make suggestions for system objects (for example, it can suggest that you change such objects as **currentUser** or **currentSession**):



For more information on preferences, see Preferences.

3 Using MxAssist Logic Bot to Build Microflows

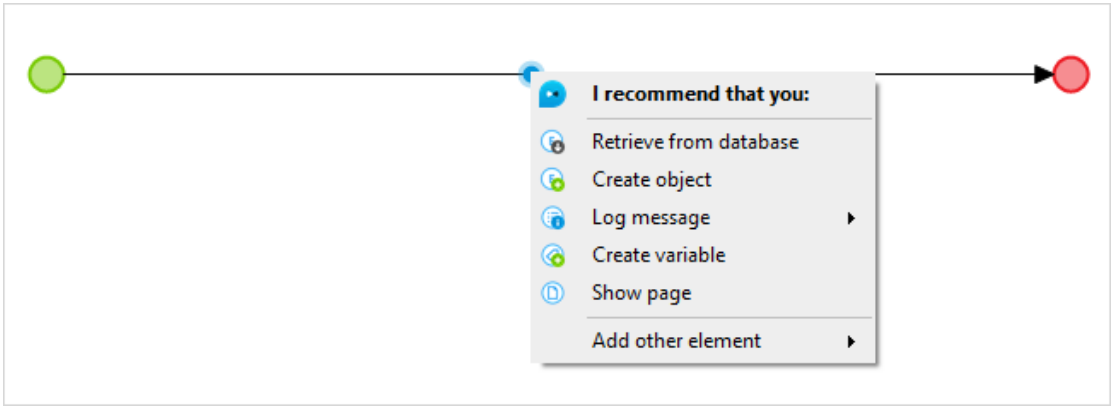
MxAssist Logic Bot is enabled by default and is displayed as a blue dot in the flow of a microflow:



It is possible to add elements to the microflow in a regular way without using MxAssist Logic Bot, however, MxAssist Logic Bot helps you add elements to the microflow faster as it suggests a short list of the most relevant activities.

To use MxAssist Logic Bot, do the following:

1. Click the icon to see the next best action recommendations:



2. Click one of the recommended activities to insert it into a microflow.
3. In the **Properties** dialog box, configure the selected activity/event.

The activity/event is added to your microflow.
If you do not see the desired activity or element in the top-five recommendation list, you can click **Add other element** and choose an activity, loop, decision, merge, or object type decision.

Implement Mendix Best Practices for Development

1 Introduction

This document can be used as a reference for adopting consistent naming and modeling conventions while developing your Mendix applications. This will help to improve your application's maintainability and performance, and make it easier to read and understand.

Even with the powerful navigation and search support in Mendix Studio Pro, adhering to naming and modeling conventions is a wise thing because:

- After finishing an application, it is usually handed over to different people for maintenance, so even years later, they will need to understand what you did and how to improve on it
- Anybody should be able to quickly understand an existing app in terms of what is located where and how the different parts are related
- A clear structure helps in identifying reusable code

2 App Setup

2.1 The Application Development Language

The language that will be used to develop the application should be determined upfront. This way you have one language for modules, entities, microflows, pages, etc. The preferred language for development is English.

There are some reasons why certain parts of an application may use another language. The main reason to make an exception would be within the domain model of an integration module. For example, when the source data model is in another language already.
For more information, see How to Translate Your App Content.

2.2 App Name

Every app is named when it is created. Make sure you use a logical name that allows you to easily identify the application. You will probably create more apps in the

future, and will want to be able to recognize this app. We recommend leaving out dates or Mendix version numbers in the app name, since that information can be captured and extracted in a different way.

2.3 Configurations

Every app has at least one configuration, but it may have many. Every app starts with a single configuration called **default**. When you work with multiple people on an application it is beneficial to create multiple configurations. When doing so, we recommend using relevant names for those configurations, like the name of the developer or the app's purpose, like **Test** or **Acceptance**. Beware that the database passwords defined in the configuration will be visible to other team members, so be careful with using personal passwords you'd like to keep secret.

2.4 User Roles

The user roles should have logical names that reflect the different types of users that will use the application. The user roles are singular and use an UpperCamelCase notation, like **FunctionalAdministrator**. User roles are mostly defined in English, but there is an option to name these in a different language, since the user role is visible in the front-end.

Each user role should correspond to only one module role per module. In other words, a user role should not map to multiple module roles within the same module. This helps to keep the number of applicable module roles for a user to a minimum, which reduces complexity in understanding the security model and reduces the performance impact of complex security rules.

3 Naming Conventions

3.1 Modules

3.1.1 Module Names

Modules should be treated like stand-alone replaceable services; for example, the customer module should function as a stand-alone customer management system as much as possible, replaceable by a different customer management system. Module names should have UpperCamelCase names that identify the responsibility of the module, for example, **CustomerManagement** or **SharePointIntegration**.

3.1.2 Module Roles

The module roles should have logical names that reflect the access they should have within a module. In contrast to the user role, the module role should always be in English, for instance **Administrator** or **Employee**.

3.2 Domain Model

3.2.1 Entity Names

Most of the time, an entity reflects a real-world object that people can relate to. Therefore, the entity name should also reflect that object and identify its purpose. There are sometimes app-specific exceptions that lead to creating other types of entity, but that is up to you. The name of an entity is singular since an object is a single instance of the entity. A good example is using **Customer** and not **Customers**. Furthermore, we advise avoiding abbreviations, underscores, mathematical characters or any other special characters in the names of entities. Entity names also use UpperCamelCase, for example, **HousekeepingRecord** or **LogEntry**.

Following these entity naming conventions will prevent issues with naming conflicts between modules and entities. For example, if a module named **Customer** contains an entity named **customer** (note the lower-case “c”), there will be a Java compilation error and the app will not run. Renaming the entity to **Customer** will solve the problem.

3.2.2 Entity Attributes

The entity attribute should reflect a property of a real-world object that people can relate to and fits the purpose of that property. We advise avoiding abbreviations, underscores (except in the case described in the next paragraph), mathematical characters or any other special characters in the names. Entity attributes should use UpperCamelCase, for example, **FirstName** or **TelephoneNumber**.
Attributes that do not reflect business-related data, but are only necessary for technical reasons, should start with an underscore (`_`).

A strong indicator for determining whether or not an attribute is business-related is whether you would still capture it if you were using a paper-only process. If you would, it is likely that the attribute will deliver business value.

3.2.3 Associations

3.2.3.1 Naming Multiple Associations Between Entities

Association names in the domain model are automatically generated by Mendix. The auto-generated names follow the best practice and should be used by default.
If you have multiple associations between the same entities we recommend extending the association name. Extending this name with a recognizable purpose clarifies where you should use the association. For example, you can have a relationship between **Person** and **Address**. A person can have multiple addresses but you want to specify what their postal address is and what their delivery address is. An implementation choice could be that you create two associations for that purpose and adjust the names of the associations accordingly. For example, **Person_Address_Delivery**.

3.2.3.2 Renaming Entities

When an association already exists between entities and you change the name on one or both of the entities, Mendix will rename the association automatically.

With models built in lower versions of Mendix, however, you will need to manually rename the assocation to keep your model consistent and up-to-date.

3.3 Folders

The structure for your documents starts with a clear separation of folders. By using a good folder structure you will improve the maintainability of your application; you will be able to find required documents faster and therefore will be able to develop and fix faster.

The optimal grouping of your documents into folders depends on the circumstances and on the functionality of your application. We recommend combining the guidelines below in a way that fits your app.

3.3.1 Process-Related Sources

Every app consists of processes. Structure your documents for these processes into folders that reflect individual processes and their steps.

3.3.2 Entity-Related Sources

Every app has documents that are needed for specific entities. Think of overview pages for maintenance, validation microflows that prevent commits, or other event triggers. These types of document should be structured into one folder that is named after the entity. Optionally, sub-folders could be used to organize, for example, **events** and **pages**.

3.4 Microflows

Generally, microflow names should include the type of event which triggers them, the name of the main entity being processed, and the operation being performed: **{PREFIX}_{Entity}_{Operation}**. For example, **ACT_Vendor_StartWorkflow**.

There are exceptions, such as where there is no main entity, or there is another reason to use a different name to improve understandability. The important thing is to make sure the name of the microflow clearly indicates its purpose.

To easily find and recognize the purpose of a microflow, you can use standard prefixes. Common purposes or events and their standard prefixes are listed below. If a microflow is triggered by several events, consider using more than one prefix. If a microflow does not comply to any of the patterns listed below, it should not have a prefix.

3.4.1 Entity Event Microflows

For some entities you use entity events that are always triggered when a specific operation is executed on the entity.
For example, an attribute **TotalOrderAmount** is automatically filled based on the amount values of the order-related order lines. You can define an after-commit event that ensures that **TotalOrderAmount** is updated when a related order line is saved: *ACO_Order_CalculateTotalOrderAmount*.

The microflows related to such an event handler should have the following prefixes:

Event Type	Prefix
Before commit	BCO_{Entity name}
After commit	ACO_{Entity name}

Event Type	Prefix
Before create	BCR_{Entity name}
After create	ACR_{Entity name}
Before delete	BDE_{Entity name}
After delete	ADE_{Entity name}
Before rollback	BRO_{Entity name}
After rollback	ARO_{Entity name}

3.4.2 Calculated Attribute Microflows

For attributes, you can choose to store the value in the database or to calculate the value based on a microflow. For the microflow that does the calculation, you should use **CAL_** as a prefix and refer to the entity and attribute which is being calculated. The calculation is triggered when you show the entity on a page or use it in a microflow. On a page, the object's calculation attribute refreshes if you navigate away from the object and back to it in any way (via pagination buttons or tabs or by re-entering the page).

Event Type	Prefix
Calculation	CAL_{Entity name}_{Attribute name}

3.4.3 Page-Based Microflows

Pages have a number of events that can trigger a microflow. See the following list for the examples and prefixes:

Event Type	Prefix	Used In
On enter event	OEN_{Purpose}	Input widgets
On change event	OCH_{Purpose}	Input widgets
On leave event	OLE_{Purpose}	Input widgets
Data source	DS_{Purpose}	Data view, list view, data grid, template grid

Event Type	Prefix	Used In
Action button	ACT_{Purpose}	Menu item, navigation item, microflow and action button, drop-down button ("IVK_" is used historically)

3.4.4 Workflow Microflows

You can call a microflow from a workflow. See the list of examples and prefixes in the table below:

Event Type	Prefix	Description
User assignment	WFA_	Returns a list of users who can perform the workflow task.
System action	WFS_	Accepts a workflow object and returns a workflow task result.
On Created Event	WFC_	Starts when a user task is created, accepts a workflow object.

3.4.5 Validation Microflows

Microflows that are used for data validation use the prefix **VAL_**.

Event Type	Prefix
Validation	VAL_

3.4.6 Scheduled Event Microflows

For the microflow that you use in your scheduled events, use the prefix **SCE_**. The event itself should have a descriptive name since it will be shown in the cloud configuration portal. The scheduled event and the microflow should have the same name.

Event Type	Prefix
Scheduled Event	SCE_

3.4.7 App Microflows

Your app settings provide three events that can trigger a microflow. In these cases we advise writing out the purpose as a microflow name. These microflows are defined only once per app and should preferably call sub-microflows to do the actual processing. These sub-microflows should have a prefix indicated below:

Event Type	Microflow Name	Sub-Microflow Prefix
After startup	AfterStartUp	ASU_
Before shutdown	BeforeShutDown	BSD_
Health check	HealthCheck	HCH_

3.4.8 Unit Test Microflows

Microflows containing unit tests should have the prefix **TEST_**.

Event Type	Prefix
Unit Test	TEST_

3.4.9 Integration Microflows

For integrations, you have the following types of microflow:

Event Type	Prefix
Consumed web service operation microflow	CWS_
Published web service operation microflow	PWS_
Published REST service operation microflow	PRS_

3.5 Other Document Types

3.5.1 Layouts & Snippets

Layouts and snippets should be identified with prefixes.

Document Type	Prefix
Layout	LAY_
Snippet	SNIP_

3.5.2 Enumerations

Enumerations should be identified with a prefix.

Document Type	Prefix
Enumeration	ENUM_

3.5.3 Pages

Pages use a **suffix** to indicate their use.
Pages that show an overview of a single entity should have a suffix of **_Overview**.
Pages that are to create, edit, or view entity data, and that are not part of a process, should have the suffix **_New**, **_Edit**, **_NewEdit**, or **_View**.
Pages that are used to make a selection of one object have a suffix of **_Select** where the multi-object selection pages should have the suffix **_MultiSelect**.
Pages that are used as a tooltip page should have the suffix **_Tooltip**.
Pages that are called when a user task in a workflow is executed, have suffix **_Workflow**. There is one task page per user task. These pages always have a WorkflowUserTask data view and are specific to performing workflow tasks.

Page Purpose	Suffix
List objects of a single entity type	_Overview
Create an object	_New
Update an object	_Edit
Create *or* Update an object	_NewEdit
View an object (read-only)	_View
Select a single object	_Select
Select multiple objects	_MultiSelect
Tooltip	_Tooltip
Interact with a user task	_Workflow

3.5.4 Integration Documents

Documents used to support integration should have the prefixes listed below.

Document Type	Prefix
Import mapping	IMM_
Export mapping	EXM_
XML schema definition	XSD_
JSON structure	JSON_
Deeplink	DL_

3.6 Home Pages

You can define the home pages per device and role in your navigation. The recommended page names are listed below:

Event Type	Device	Page Name
Default home page	Desktop	Home_Desktop_Default
Default home page	Tablet	Home_Tablet_Default
Default home page	Mobile	Home_Phone_Default
Role based home page	Desktop	Home_Desktop_{Userrole}
Role based home page	Tablet	Home_Tablet_{Userrole}
Role based home page	Mobile	Home_Phone_{Userrole}

4 General Guidelines & Best Practices

4.1 Domain Models

4.1.1 Attributes

Using calculated (virtual) attributes is discouraged. These introduce a performance risk since they need to be calculated every time the object is used, regardless of whether the attribute itself is used.

4.1.2 Inheritance

When using inheritance (specialization/generalization), it is recommended to use no more than two levels for performance reasons.

4.1.3 Delete Behavior

Delete behavior must be specified where possible. Delete behavior must, however, never be relied upon when deleting large amounts of data. For performance reasons it is better to explicitly delete dependent objects when doing batch deletes.

4.1.4 Event Handlers

Event handlers on domain entities must be used with a lot of caution. They can quickly result in complex and possibly unexpected behavior when several of them are applied to a single entity. It is often best to make the execution of microflows more explicit by using sub-microflows that are called manually, for example, just before committing an object.

4.2 Microflows

4.2.1 Size

The size of a microflow should not exceed 25 elements. An element is any block that Studio Pro allows in a microflow (loops, action activities, decisions, etc.). In some cases exceeding this limit is acceptable; this can occur, for instance, for validation or data copying flows.

Split microflows up into logical, functional elements. If a microflow has more than twenty-five elements, split the microflow up by creating a sub-microflow for a part of it. For example, by separating presentation logic from business logic.

Certain cases (such as validation checks) may require this rule to be ignored to produce an understandable result.

4.2.2 Documentation & Annotations

All complex microflows (more than ten activities or more than two decisions) should have an annotation describing the purpose of the microflow, expected parameters, and return values. This annotation should be placed at the start, so it is visible when the microflow is opened. This will assist other developers in quickly understanding the general purpose of a microflow, without having to read through it entirely.

Complex, non-standard or integration-related sections in microflows should also have an accompanying annotation. Examples of these are web service calls, custom loops, and Java calls.

4.2.3 Readability

The normal flow in a microflow should be aligned from left to right to ensure readability. Exceptions to the normal flow may branch out vertically: downwards is preferred, upwards if the downwards direction is already used.

Avoid crossing of lines of the links between the microflow elements.

If you decide to color code the different activities in your app, be sure to align within your team on their meaning.

4.2.4 Complexity

Nested `IF` statements in a single microflow expression are not recommended. If multiple checks depend on one another, this should be represented by multiple decisions in the microflow, so that the complexity is not hidden away in the expressions. You can use `AND` and `OR` operators to produce complex expressions if necessary.

Event triggers on input fields must be kept as simple as possible, since they are potentially executed very often, depending on user behavior. Complex operations here will reduce performance.

The number of parameters for a microflow should be kept to a minimum to facilitate reusability. The more parameters a microflow has, the more difficult it is to determine what should be put into the parameters to make the microflow run correctly.

4.2.5 Error Handling & Logging

Use microflow error handling for all integration and Java calls. Make sure to determine the correct rollback behavior. Always log the error that occurred, even if the process can continue, this is essential for later analysis of the error.
Complex processes and important business logic (like workflow processing or validations) must include debug and trace logging. Logging actions must write the current state and progress of the process and must include a request ID or other

identifying information. The log node should be the name of the module. This will greatly assist error analysis.

4.3 Workflows

Guidelines below can help you choose a short yet meaningful name for your workflow:

- The name matches the context entity name
- The name consists of a noun + verb (e.g. *EmployeeOnboarding*)
- The name reflects what the process is about, what the goal of the process is

4.4 Warnings & Studio Pro Feedback

No warnings should be visible in Studio Pro, unless explicitly documented with a reason. Warnings can indicate many issues, including maintainability and security risks, which must be resolved.

Unused and excluded items should be removed from the model when they are no longer needed. When a version of the application is prepared for a release, all these items should be cleaned up. Make sure to check whether items that appear unused are not actually called from a Java action before removing them. Studio Pro provides the possibility to mark such items as used to override warnings about this.

4.5 XPath

XPath constraints in any part of the model should be kept as simple as possible. As a general rule, XPaths must not appear when the **Find advanced > XPath** option in Studio Pro is used with all options enabled.

4.6 Security

The security overview in Studio Pro must not show any incomplete (yellow) parts. All entity, microflow, and page access must be configured completely.

Assigning default rights to new members when defining entity access is NOT recommended. This will ensure that access is only granted after a conscious decision.

4.7 Mendix Version

Apps should keep up with new Mendix releases as much as possible.

4.8 Marketplace Content

When introducing a new Mendix Marketplace component to an app, carefully consider the support level of the component. Using components that are community supported introduces a maintainability and upgrade risk.

Marketplace modules should NOT be modified. If an ApMarketplace module is modified, updating to a new version becomes much harder, because the changes will be overwritten when a new version is downloaded from the Marketplace. If changing an Marketplace module is unavoidable, you have two options:

- Mark any changes you make explicitly and clearly, and perform them again when the module is updated
- Copy the contents of the Marketplace module to another module in your app and use that module instead (remember that your app will no longer reflect updates to the original Marketplace module)

To minimize the number of changes in the actual Marketplace module, it is advisable to combine them in a separate extension module wherever possible.

Mendix Assist

1 简介

Mendix Assist 是指利用人工智能 (AI) 和机器学习 (ML) 来协助开发人员进行应用程序开发的 Mendix 平台功能，即所谓的 AI 辅助开发 (AIAD)。Mendix Assist 的目的是帮助开发团队更快、更一致、更高质量地建模和交付 Mendix 应用程序。

Mendix Assist 由充当虚拟合作开发机器人的不同功能组成，每个功能都专门用于应用程序生命周期开发的某个领域或阶段。目前，Mendix Assist 包含以下虚拟合作开发者机器人：

- **MxAssist Logic Bot** – 帮助您在 Mendix Studio Pro 中建模和配置微流。它根据特定微流中已配置的活动和参数，为您提供有关微流中下一个最佳活动的情境化建议。它是使用 Mendix 构建的超过 1200 万个匿名应用程序逻辑（微流）的机器学习分析构建的。它使用深度学习来检测和学习微流中的最佳实践模式。有关更多信息，请参阅 MxAssist 逻辑机器人。
- **MxAssist Performance Bot** – 帮助您根据 Mendix 开发最佳实践检查您的应用，检测和查明开发反模式，并在许多情况下自动修复它们。该机器人使用对数千个匿名 Mendix 应用程序的统计分析来构建，以了解常见的反模式以及 Mendix 专家服务在微流、域模型、页面、安全性等开发方面的最佳实践。有关更多信息，请参阅 MxAssist Performance Bot .

MxAssist 逻辑机器人

1 简介

MxAssist Logic Bot 是一个 AI 驱动的虚拟合作开发机器人，可帮助您在 Mendix Studio Pro 中建模和配置应用程序逻辑（微流）。它根据已设计的活动、参数和其他上下文相关信息，为您提供有关微流中下一个最佳活动的上下文建议。

MxAssist Logic Bot 使用机器学习分析对超过 1200 万个匿名应用程序逻辑（微流）*——*使用 Mendix 构建*——*构建，以检测和学习微流中的最佳实践模式。

MxAssist Logic Bot 的主要功能如下：

- **下一个最佳行动建议**- 它以 95% 的准确率推荐 40 多个不同选项中的前五个最佳活动。
- **自动配置**——它不仅提供下一个最佳操作，而且通过预先填充此类操作的参数来进一步自动化开发。
- **上下文建议**——它以不同的方式推导出上下文，包括当开发人员在流程中插入新的活动或决策时，通过在微流中向左和向右"看"；并通过使用调用它的页面推断上下文。
- **高准确率**——模型的不断改进和训练使准确率水平从 95% 提高。

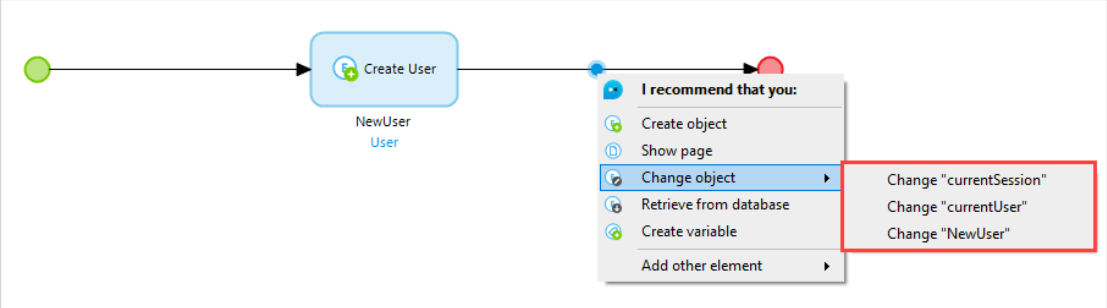
2 MxAssist Logic Bot 设置

您可以在微流编辑器的右上角打开和关闭 MxAssist Logic Bot。

要访问 MxAssist Logic Bot 的设置，请打开**"编辑"** > **"首选项"**> **"常规"**选项卡 >" **MxAssist Logic Bot"**选项卡。有关更多信息，请参阅首选项。

在 **MxAssist Logic Bot** 选项卡中，您可以设置以下内容：

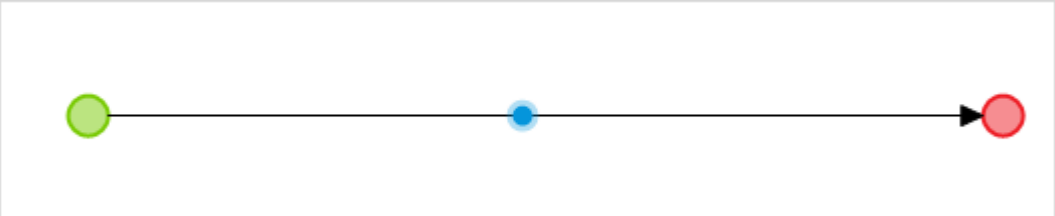
- **启用 MxAssist Logic Bot** – 打开和关闭 MxAssist Logic Bot
- **显示系统变量建议**——启用后，MxAssist Logic Bot 将为系统对象提供建议（例如，它可以建议您更改诸如 **currentUser** 或 **currentSession** 之类的对象）：



有关首选项的更多信息，请参阅首选项。

3 使用 MxAssist Logic Bot 构建微流

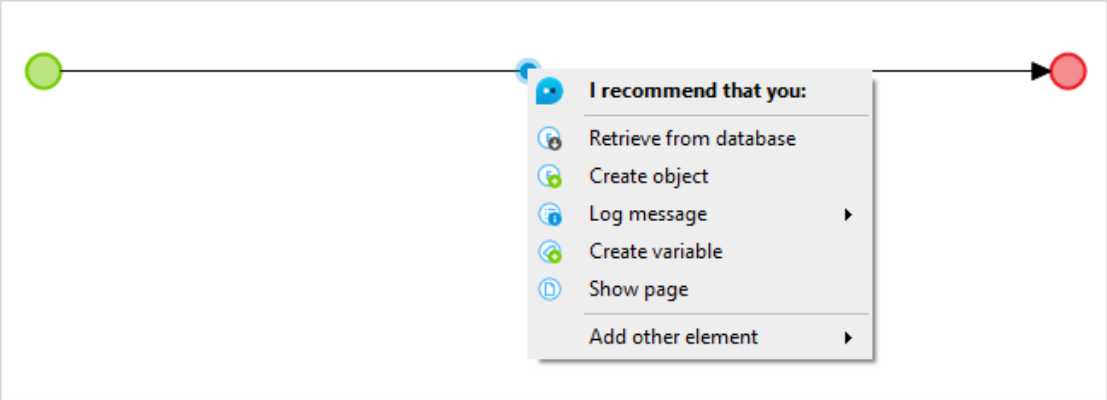
默认情况下启用 MxAssist Logic Bot，并在微流的流程中显示为蓝点：



可以在不使用 MxAssist Logic Bot 的情况下以常规方式向微流添加元素，但是，MxAssist Logic Bot 可帮助您更快地向微流添加元素，因为它会建议最相关活动的简短列表。

要使用 MxAssist Logic Bot，请执行以下操作：

1. 单击该图标可查看下一个最佳操作建议：



2. 单击推荐的活动之一将其插入到微流中。
3. 在“**属性**”对话框中，配置选定的活动/事件。

活动/事件被添加到您的微流中。

如果在前五个推荐列表中没有看到所需的活动或元素，您可以单击**添加其他元素**并选择活动、循环、决策、合并或对象类型决策。

实施 Mendix 最佳开发实践

1 简介

本文档可用作在开发 Mendix 应用程序时采用一致命名和建模约定的参考。这将有助于提高应用程序的可维护性和性能，并使其更易于阅读和理解。

即使使用 Mendix Studio Pro 中强大的导航和搜索支持，遵守命名和建模约定也是明智之举，因为：

- 完成申请后，通常会交给不同的人进行维护，因此即使多年后，他们也需要了解您做了什么以及如何改进
- 任何人都应该能够快速了解现有应用程序的位置以及不同部分的相关方式
- 清晰的结构有助于识别可重用的代码

2 应用程序设置

2.1 应用程序开发语言

应预先确定将用于开发应用程序的语言。这样你就有了一种用于模块、实体、微流、页面等的语言。开发的首选语言是英语。

应用程序的某些部分可能使用另一种语言有一些原因。例外的主要原因是在集成模块的域模型中。例如，当源数据模型已经是另一种语言时。
有关更多信息，请参阅如何翻译您的应用内容。

2.2 应用名称

每个应用程序在创建时都已命名。确保您使用的逻辑名称可以让您轻松识别应用程序。您将来可能会创建更多应用程序，并且希望能够识别此应用程序。我们建议在应用程序名称中省略日期或 Mendix 版本号，因为可以以不同的方式捕获和提取这些信息。

2.3 配置

每个应用程序至少有一种配置，但也可能有多种配置。每个应用程序都从一个名为 **default** 的配置开始。当您在一个应用程序上与多人合作时，创建多个配置是有益的。这样做时，我们建议为这些配置使用相关名称，例如开发人员的名称或应用程序的用途，例如 **Test** 或 **Acceptance**。请注意，配置中定义的数据库密码对其他团队成员可见，因此请谨慎使用您希望保密的个人密码。

2.4 用户角色

该用户角色应具有反映了不同类型，将使用应用程序的用户的逻辑名称。用户角色是单一的，并使用大写字母表示法，如 **FunctionalAdministrator**。用户角色大多用英语定义，但可以选择用不同的语言命名这些角色，因为用户角色在前端是可见的。
每个用户角色应该只对应一个模块的一个模块角色。换句话说，用户角色不应映射到同一模块内的多个模块角色。这有助于将用户适用的模块角色的数量保持在最低限度，从而降低理解安全模型的复杂性并降低复杂安全规则对性能的影响。

3 命名约定

3.1 模块

3.1.1 模块名称

模块应该被视为独立的可替换服务；例如，客户模块应尽可能作为一个独立的客户管理系统，可以由不同的客户管理系统替代。模块名称应具有标识模块职责的大驼峰名称，例如 **CustomerManagement** 或 **SharePointIntegration**。

3.1.2 模块角色

该模块的角色应该有反映他们应该有一个模块内的访问逻辑名称。与用户角色相反，模块角色应始终使用英语，例如 **Administrator** 或 **Employee**。

3.2 领域模型

3.2.1 实体名称

大多数情况下，实体反映了人们可以与之相关的现实世界对象。因此，实体名称还应反映该对象并标识其用途。有时，特定于应用程序的异常会导致创建其他类型的实体，但这取决于您。实体的名称是单数的，因为对象是实体的单个实例。一个很好的例子是使用 **Customer** 而不是 **Customers**。此外，我们建议避免在实体名称中使用缩写、下划线、数学字符或任何其他特殊字符。实体名称也使用 UpperCamelCase，例如 **HousekeepingRecord** 或 **LogEntry**。

遵循这些实体命名约定将防止模块和实体之间的命名冲突问题。例如，如果名为 **Customer** 的模块包含名为 **customer** 的实体（注意小写的“c”），则会出现 Java 编译错误并且应用程序将无法运行。将实体重命名为 **Customer** 将解决问题。

3.2.2 实体属性

实体属性应该反映现实世界对象的属性，人们可以与之相关并符合该属性的目的。我们建议避免在名称中使用缩写、下划线（下一段中描述的情况除外）、数学字符或任何其他特殊字符。实体属性应使用大写字母，例如 **FirstName** 或 **TelephoneNumber**。

不反映业务相关数据但仅出于技术原因才需要的属性应以下划线（`_`）开头。

确定一个属性是否与业务相关的一个强有力的指标是，如果您使用纯纸流程，您是否仍会捕获它。如果您愿意，该属性很可能会带来业务价值。

3.2.3 关联

3.2.3.1 命名实体间的多重关联

领域模型中的关联名称由 Mendix 自动生成。自动生成的名称遵循最佳实践，应默认使用。

如果您在同一实体之间有多个关联，我们建议扩展关联名称。使用可识别的目的扩展此名称可阐明您应该在何处使用关联。例如，您可以在 **Person** 和 **Address** 之间建立关系。一个人可以有多个地址，但您要指定他们的邮政地址和收货地址。一个实现选择可能是您为此目的创建两个关联并相应地调整关联的名称。例如，**Person_Address_Delivery**。

3.2.3.2 重命名实体

当实体之间已经存在关联并且您更改了一个或两个实体的名称时，Mendix 将自动重命名关联。

但是，对于在较低版本 Mendix 中构建的模型，您需要手动重命名关联以保持模型一致和最新。

3.3 文件夹

您的文档结构从文件夹的清晰分离开始。通过使用良好的文件夹结构，您将提高应用程序的可维护性；您将能够更快地找到所需的文档，从而能够更快地开发和修复。
将文档最佳分组到文件夹中取决于环境和应用程序的功能。我们建议以适合您的应用程序的方式组合以下指南。

3.3.1 过程相关来源

每个应用程序都由进程组成。将这些流程的文档组织到反映各个流程及其步骤的文件夹中。

3.3.2 实体相关来源

每个应用程序都有特定实体所需的文档。想想维护、验证微流以防止提交或其他事件触发器的概述页面。这些类型的文档应组织到一个以实体命名的文件夹中。可选地，子文件夹可用于组织，例如，**事件**和**页面**。

3.4 微流

通常，微流名称应包括触发它们的事件类型、正在处理的主要实体的名称以及正在执行的操作：**{PREFIX}_{Entity}_{Operation}**。例如， **ACT_Vendor_StartWorkflow**。
也有例外，例如没有主要实体，或者有另一个原因使用不同的名称来提高可理解性。重要的是确保微流的名称清楚地表明其目的。
要轻松查找和识别微流的用途，您可以使用标准前缀。下面列出了常见的目的或事件及其标准前缀。如果一个微流由多个事件触发，请考虑使用多个前缀。如果微流不符合下面列出的任何模式，则不应有前缀。

3.4.1 实体事件微流

对于某些实体，您使用在实体上执行特定操作时始终触发的实体事件。
例如，根据订单相关订单行的金额值自动填充属性 **TotalOrderAmount**。您可以定义一个提交后事件，以确保在保存相关订单行时更新 **TotalOrderAmount**：*ACO_Order_CalculateTotalOrderAmount*。
与此类事件处理程序相关的微流应具有以下前缀：

事件类型	字首
提交前	BCO_{实体名称}
提交后	ACO_{实体名称}
创建前	BCR_{实体名称}
创建后	ACR_{实体名称}

事件类型	字首
删除前	BDE_{实体名称}
删除后	ADE_{实体名称}
回滚前	BRO_{实体名称}
回滚后	ARO_{实体名称}

3.4.2 计算属性微流

对于属性，您可以选择将值存储在数据库中或基于微流计算值。对于进行计算的微流，您应该使用 **CAL_** 作为前缀并引用正在计算的实体和属性。当您在页面上显示实体或在微流中使用它时，会触发计算。在页面上，如果您以任何方式（通过分页按钮或选项卡或通过重新进入页面）从对象导航离开并返回到该对象，则该对象的计算属性会刷新。

事件类型	字首
计算	CAL_{实体名称}_{属性名称}

3.4.3 基于页面的微流

页面有许多可以触发微流的事件。有关示例和前缀，请参阅以下列表：

事件类型	字首	用于
进入事件	OEN_{目的}	输入小部件
在更改事件	OCH_{目的}	输入小部件
休假事件	OLE_{目的}	输入小部件
数据源	DS_{目的}	数据视图、列表视图、数据网格、模板网格
操作按钮	ACT_{目的}	菜单项、导航项、微流和动作按钮、下拉按钮 （“IVK_”是历史上使用的）

3.4.4 工作流微流

您可以从工作流中调用微流。请参阅下表中的示例和前缀列表：

事件类型	字首	描述
用户分配	WFA_	返回可以执行工作流任务的用户列表。

事件类型	字首	描述
系统动作	WFS_	接受工作流对象并返回工作流任务结果。
在创建的事件上	WFC_	在创建用户任务时启动，接受工作流对象。

3.4.5 验证微流

用于数据验证的微流使用前缀 **VAL_**。

事件类型	字首
验证	VAL_

3.4.6 预定事件微流

对于您在计划事件中使用的微流，请使用前缀 **SCE_**。事件本身应具有描述性名称，因为它将显示在云配置门户中。计划事件和微流应该具有相同的名称。

事件类型	字首
预定活动	SCE_

3.4.7 应用微流

您的应用设置提供了三个可以触发微流的事件。在这些情况下，我们建议将目的写成微流名称。这些微流每个应用程序只定义一次，最好调用子微流来进行实际处理。这些子微流应具有如下所示的前缀：

事件类型	微流名称	亚微流前缀
启动后	启动后	ASU_
关机前	关机前	BSD_
健康检查	健康检查	HCH_

3.4.8 单元测试微流

包含单元测试的微流应该有前缀 **TEST_**。

事件类型	字首
单元测试	测试_

3.4.9 集成微流

对于集成，您有以下类型的微流：

事件类型	字首
消费的 web 服务操作微流	CWS_
发布的 web 服务操作微流	PWS_
发布的 REST 服务操作微流程	PRS_

3.5 其他文件类型

3.5.1 布局和片段

布局和片段应该用前缀标识。

文件类型	字首
布局	LAY_
片段	片段_

3.5.2 枚举

枚举应该用前缀标识。

文件类型	字首
枚举	枚举_

3.5.3 页面

页面使用**后缀**来表示它们的用途。

显示单个实体概览的页面应具有**_Overview** 后缀 。

这是创建，编辑或查看实体数据，以及页面是不是一个过程的一部分，应该有后缀 _**新**， **_Edit**， **_NewEdit**，或 **_View**。

用于选择一个对象的页面的后缀为 **_Select** ，而多对象选择页面的后缀为 **_MultiSelect**。

用作工具提示页面的页面应具有后缀 **_Tooltip**。

执行工作流中的用户任务时调用的页面具有后缀**_Workflow**。每个用户任务有一个任务页面。这些页面始终具有 WorkflowUserTask 数据视图，并且特定于执行工作流任务。

页面目的	后缀
列出单个实体类型的对象	_概述

页面目的	后缀
创建对象	_新的
更新一个对象	_编辑
创建*或*更新对象	_NewEdit
查看对象（只读）	_看法
选择单个对象	_选择
选择多个对象	_多选
工具提示	_工具提示
与用户任务交互	_工作流程

3.5.4 集成文档

用于支持集成的文档应具有下面列出的前缀。

文件类型	字首
导入映射	IMM_
导出映射	EXM_
XML 模式定义	XSD_
JSON 结构	JSON_
深层链接	DL_

3.6 主页

您可以在导航中定义每个设备和角色的主页。下面列出了推荐的页面名称：

事件类型	设备	页面名称
默认主页	桌面	首页_桌面_默认
默认主页	药片	首页_平板电脑_默认
默认主页	移动的	首页_电话_默认

事件类型	设备	页面名称
基于角色的主页	桌面	Home_Desktop_{用户角色}
基于角色的主页	药片	Home_Tablet_{用户角色}
基于角色的主页	移动的	Home_Phone_{用户角色}

4 通用指南和最佳实践

4.1 领域模型

4.1.1 属性

不鼓励使用计算的（虚拟）属性。这些会带来性能风险，因为每次使用对象时都需要计算它们，而不管是否使用属性本身。

4.1.2 继承

使用继承（特化/泛化）时，出于性能原因，建议使用不超过两个级别。

4.1.3 删除行为

必须尽可能指定删除行为。但是，在删除大量数据时绝不能依赖删除行为。出于性能原因，最好在执行批量删除时显式删除相关对象。

4.1.4 事件处理程序

必须非常谨慎地使用域实体上的事件处理程序。当它们中的几个应用于单个实体时，它们会迅速导致复杂且可能出乎意料的行为。通常最好通过使用手动调用的子微流来使微流的执行更加明确，例如，在提交对象之前。

4.2 微流

4.2.1 尺寸

一个微流的大小不应超过 25 个元素。元素是 Studio Pro 允许在微流（循环、动作活动、决策等）中使用的任何块。在某些情况下，超过此限制是可以接受的；例如，这可能发生在验证或数据复制流程中。

将微流拆分为逻辑的、功能性的元素。如果一个微流有超过 25 个元素，通过为它的一部分创建一个亚微流来拆分微流。例如，通过将表示逻辑与业务逻辑分开。

某些情况（例如验证检查）可能需要忽略此规则以产生可理解的结果。

4.2.2 文档和注释

所有复杂的微流（十个以上的活动或两个以上的决策）都应该有一个注释来描述微流的目的、预期参数和返回值。这个注解应该放在开头，这样在微流打开的时候就可以看到了。这将有助于其他开发人员快速理解微流的一般用途，而无需通读全文。
微流中复杂的、非标准的或与集成相关的部分也应该有一个伴随的注释。这些示例包括 Web 服务调用、自定义循环和 Java 调用。

4.2.3 可读性

微流中的正常流应该从左到右对齐，以确保可读性。正常流动的例外可能垂直分支：向下是首选，如果向下方向已经使用，则向上。
避免交叉微流元件之间的链接线。
如果您决定对应用程序中的不同活动进行颜色编码，请务必在您的团队中根据其含义进行调整。

4.2.4 复杂性

`IF` 不建议在单个微流表达式中嵌套语句。如果多个检查相互依赖，这应该由微流中的多个决策来表示，这样复杂性就不会隐藏在表达式中。如有必要，您可以使用 `AND` 和 `OR` 运算符来生成复杂的表达式。
输入字段上的事件触发器必须尽可能简单，因为它们可能会经常执行，具体取决于用户行为。这里的复杂操作会降低性能。
微流的参数数量应保持最少，以促进可重用性。一个微流的参数越多，就越难确定应该在参数中放入什么才能使微流正确运行。

4.2.5 错误处理和日志记录

对所有集成和 Java 调用使用微流错误处理。确保确定正确的回滚行为。始终记录发生的错误，即使过程可以继续，这对于稍后分析错误至关重要。
复杂的流程和重要的业务逻辑（如工作流处理或验证）必须包括调试和跟踪日志记录。日志记录操作必须写入流程的当前状态和进度，并且必须包含请求 ID 或其他识别信息。日志节点应该是模块的名称。这将极大地帮助错误分析。

4.3 工作流程

以下指南可帮助您为工作流程选择一个简短而有意义的名称：

- 名称与上下文实体名称匹配
- 名称由名词 + 动词组成（例如 *EmployeeOnboarding*）
- 名称反映了流程的内容，流程的目标是什么

4.4 警告和 Studio Pro 反馈

Studio Pro 中不应出现任何警告，除非明确说明原因。警告可以指示许多问题，包括必须解决的可维护性和安全风险。

当不再需要未使用和排除的项目时，应将其从模型中删除。当应用程序的一个版本准备发布时，所有这些项目都应该清理干净。在删除它们之前，请确保检查看起来未使用的项目是否实际上没有从 Java 操作中调用。Studio Pro 提供了将此类项目标记为用于覆盖有关此警告的可能性。

4.5 XPath

模型任何部分中的 XPath 约束都应尽可能简单。作为一般规则，在启用所有选项的情况下使用 Studio Pro 中的**查找高级 > XPath** 选项时，不得出现 **XPath**。

4.6 安全

Studio Pro 中的安全概览不得显示任何不完整（黄色）的部分。所有实体、微流和页面访问都必须完整配置。

不建议在定义实体访问权限时为新成员分配默认权限。这将确保仅在有意识的决定后才授予访问权限。

4.7 Mendix 版本

应用程序应尽可能跟上新的 Mendix 版本。

4.8 市场内容

向应用程序引入新的 Mendix Marketplace 组件时，请仔细考虑该组件的支持级别。使用社区支持的组件会带来可维护性和升级风险。

不应修改市场模块。如果修改了 ApMarketplace 模块，更新到新版本会变得更加困难，因为从 Marketplace 下载新版本时更改将被覆盖。如果无法避免更改 Marketplace 模块，您有两种选择：

- 明确并清楚地标记您所做的任何更改，并在模块更新时再次执行它们
- 将 Marketplace 模块的内容复制到您的应用程序中的另一个模块并使用该模块（请记住，您的应用程序将不再反映对原始 Marketplace 模块的更新）

为了尽量减少实际 Marketplace 模块中的更改次数，建议尽可能将它们组合在单独的扩展模块中。