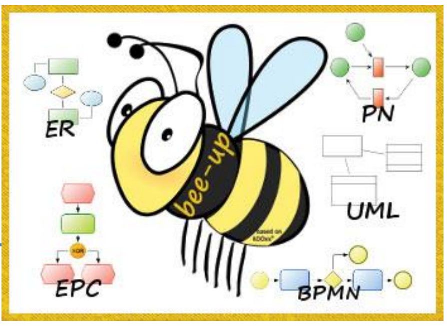
Bee-Up Handbook v1.7



Updated: 2023-10-16

Authors: Patrik Burzynski, Dimitris Karagiannis

General information

This Handbook is written for Bee-Up version 1.7 based on the ADOxx 1.8[1] platform. The Bee-Up tool enables modelling according to several languages and techniques, among others:

- Business Process Model and Notation 2.0 (BPMN),
- Event-driven Process Chains (EPC) and extended EPCs (eEPC),
- Entity-Relationship (ER), and
- Unified Modeling Language 2.0 (UML)
- Petri Nets (PN).

If you should encounter problems, have questions or feature requests which are not covered yet, you can contact the Bee-Up development team directly at bee-up@omilab.org .

[1] http://www.adoxx.org/

Installation

The Bee-Up tool works primarily on a Windows operating system (8, 8.1, 10 or 11). It can however also be used on systems that allow the execution of Windows applications, for example through Wine. Therefore, different installers are provided for each operating system: Windows, Linux and macOS.

To install the Bee-Up tool follow these steps:

1. Download the ZIP-File containing the installation package for your OS from OMiLAB. Make sure to download the correct version.
2. Extract the contents to a folder.
3. Run install_on_winodws.msi (Windows), install_on_linux.sh (Linux) or install_on_macos.sh (macOS) from the extracted folder[2].

Starting with Bee-Up 1.7, the import of the functionality relevant attribute profiles should be performed automatically by the installation. If this should have failed for some reason, then please check the After the installation section.

Bee-Up uses the English language and expects the system to use it or a similar language as well. If your system is using a different language, then check the next section for possible solutions.

The installation guides the user through the installation process. It checks the prerequisites and informs the user about any that must be installed (typically performed by the installer automatically). This includes required frameworks (like MSVC redistributables on Windows) and applications (like Wine on non-Windows). For more details about the Linux and macOS versions please check the corresponding README.md files.

By default Bee-Up 1.7 uses SQLite as the database system and creates and uses the database with the name ‘beeup17’ (without the ‘ ’). The database file is in the user's roaming AppData folder under ‘ADOxx\sqlitedbs\’ by default (e.g. ‘C:\Users\Patrik\AppData\Roaming\ADOxx\sqlitedbs’). When the environment variable ‘ADO_SQLITE_DBFOLDER’ is set, then the database files are located in the folder specified by the environment variable instead. If you had Bee-Up previously installed and require a fresh installation then uninstall Bee-Up, delete the database (see Uninstallation section) and install Bee-Up again.

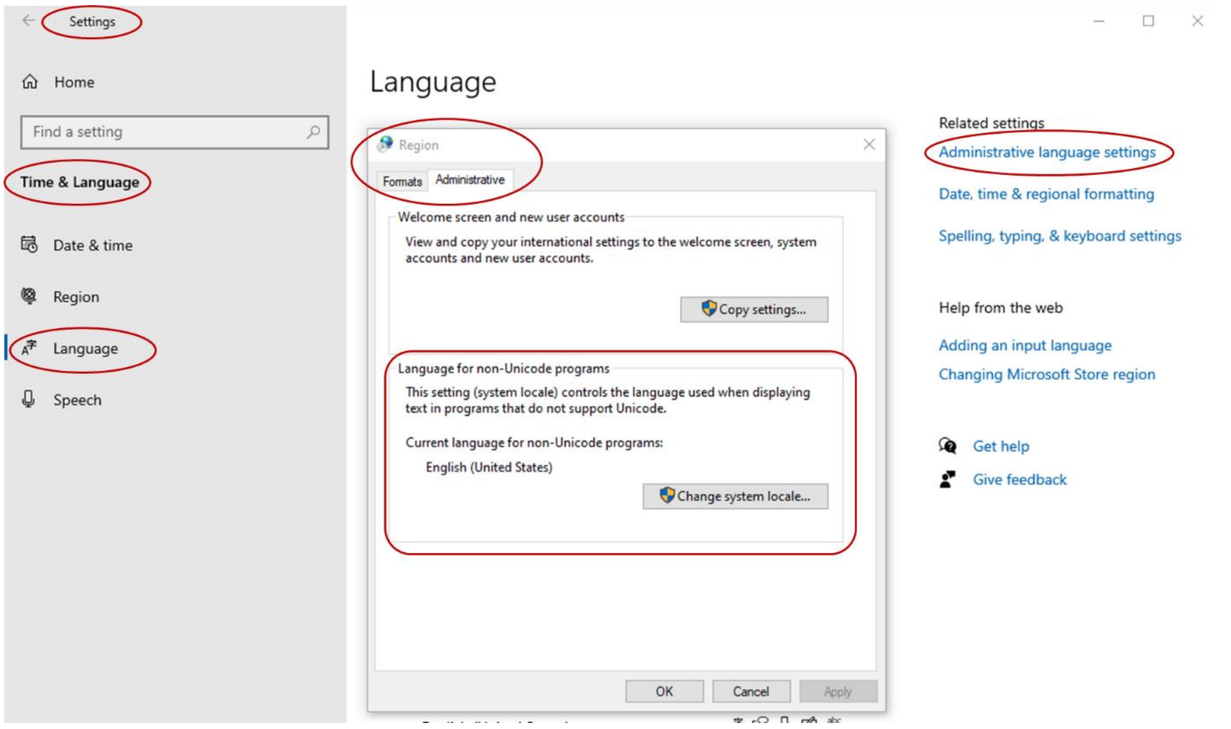
An alternative to SQLite is the use of a Microsoft SQL Server Express instance. The details for the installation and their use are shortly described in the Manual installation of Microsoft SQL Server instance section.

Before the installation

1. **Requirements for Linux / macOS**: The installations for Linux and macOS systems work very similar and share most of the requirements, which are: Homebrew (for macOS, necessary to install other dependencies) and Wine (version 3.21 or newer recommended, to run the main tool). The provided installation script should take care of installing those requirements. Details for their manual installation can be found in the README.md provided with the installer.
2. **How to actually start the Linux / macOS installation**: The installation on Linux and macOS works through a bash script. The simplest way to execute it is through a Terminal by navigating to the extracted folder and typing "./install_linux.sh" or "./install_mac.sh" respectively. Ensure that the script has execution permissions (using "chmod +x ./install_mac.sh" in the terminal beforehand).
3. **Language setting**: Bee-Up uses the English language and can cause problems on Windows systems that use a language with a vastly different alphabet (English and German work fine, languages like Greek, Persian, Chinese etc. are known to cause problems). The Linux and macOS installers do not face this problem, since they set the environment to English where the installer/tool is executed.
   On Windows it is possible to use Bee-Up without having to change the system language by changing the "Language for non-Unicode programs" of the operating system to "English". On Windows 7 this setting can be found in the "Control Panel" under "Region and Language" in the "Administrative" tab. On Windows 10 the setting can be found navigating to the "Settings" of

[2] Please note that the Linux/macOS installation scripts are provided to help you with the installation AS IS. While we tried our best to make them as robust as possible, we can’t guarantee that they will work in every case. The scripts have been tested with Ubuntu (18.04 LTS/18.10, 22.04 LTS), Fedora (28/29) and macOS (10.14.3, 13.5).

Windows, going to "Time & Language" → "Language" → "Administrative language settings" and then selecting the "Administrative" tab, as shown in the picture below.
If an error saying "The selected database does not exist or has not been catalogued yet." is encountered during the installation or an error like "Database … does not exist!" or "A character set error has been detected…" pops up when starting the tool after the installation, then it's most likely due to the language setting. In this case please uninstall the tool, delete the database (see Uninstallation section) and install Bee-Up again.



Welcome screen and new user accounts

View and copy your international settings to the welcome screen, system accounts and new user accounts.

Language for non-Unicode programs

This setting (system locale) controls the language used when displaying text in programs that do not support Unicode.

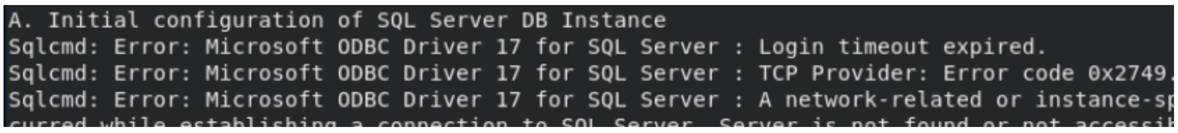
Current language for non-Unicode programs:

English (United States)

4. **Anti-Virus and Firewall Software**: Some anti-virus or firewall software can lead to an error during installation ("adbinst-18"), preventing the creation of the database and its population with the necessary data. If you want to avoid this error or should you have already encountered it, then retry the installation with your anti-virus/firewall software disabled for its duration (uninstall Bee-Up beforehand if it is installed). One known case where this error occurred was with the "Avast" anti-virus/firewall.

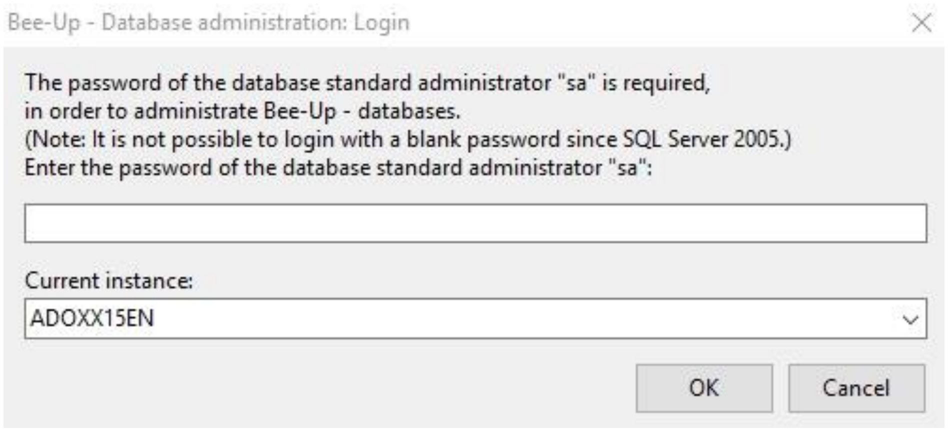
During the installation

1. **Installation of MS SQL Server instance fails on Windows**: (MS SQL Server specific) It is possible that the installation of the MS SQL Server instance fails. One of the reasons is that the MS SQL Server installer performs a check and the system doesn't meet the necessary requirements. One of the requirements is that a system restart is possible. Sometimes a different application can block the system restart, leading to the problem. So one possible solution is to close all other applications, restart the computer and then perform the installation.
2. **Errors occur during the Linux / macOS installation script**: When running the script for the first time, errors shouldn't occur during its execution. Unfortunately, there can be many causes for errors: a server is not available to download a necessary prerequisite, the script tries to reinstall an already available prerequisite, a wrong version of the prerequisite is already installed, there is not enough space available on the hard drive, an executed command went wrong etc. Please react accordingly. Something that often helped us was to abort the script (CTRL+C, might be necessary to press a few times) and try it again. One known case is during the configuration of the MS SQL Server database, which so far has always been solved by re-running the script. Also consider using the uninstallation script if the error occurs multiple times and/or to manually install the prerequisites.



```
A. Initial configuration of SQL Server DB Instance
Sqlcmd: Error: Microsoft ODBC Driver 17 for SQL Server : Login timeout expired.
Sqlcmd: Error: Microsoft ODBC Driver 17 for SQL Server : TCP Provider: Error code 0x2749.
Sqlcmd: Error: Microsoft ODBC Driver 17 for SQL Server : A network-related or instance-sp
curred while establishing a connection to SQL Server. Server is not found or not accessib
```

3. **Installation asks for standard administrator "sa" password:** (MS SQL Server specific) Sometimes during the installation a popup can ask for the database standard administrator password (see picture below). Some users reported, that after aborting the installation and restarting the computer the popup no longer appeared. Alternatively, if the MS SQL Server instance has been created automatically by the installation, then it has set the initial 'sa' password to '12+*ADOxx*+34' (without the ' '). Help on how to reset the 'sa' password can be found at the ADOxx.org community (in the FAQ: "SA Password during ADOxx Installation").



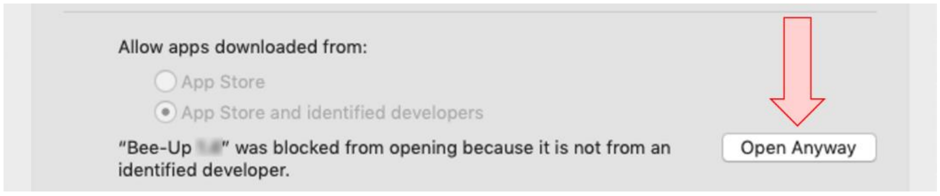
Bee-Up - Database administration: Login

The password of the database standard administrator "sa" is required,
in order to administrate Bee-Up - databases.
(Note: It is not possible to login with a blank password since SQL Server 2005.)
Enter the password of the database standard administrator "sa":

Current instance:
ADOXX15EN

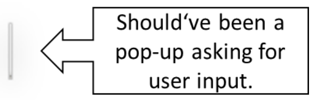
After the installation

1. **Can't start Bee-Up using the shortcuts**: Make sure that you have execution permissions for the shortcuts (especially on Linux and macOS). On macOS a Bee-up shortcut is provided in the Applications folder. However, macOS often doesn't allow starting programs from unknown developers and instead shows a warning. If this happens it is still possible to force macOS to open the program through the "Open anyway" option under "System Preferences > Security & Privacy".



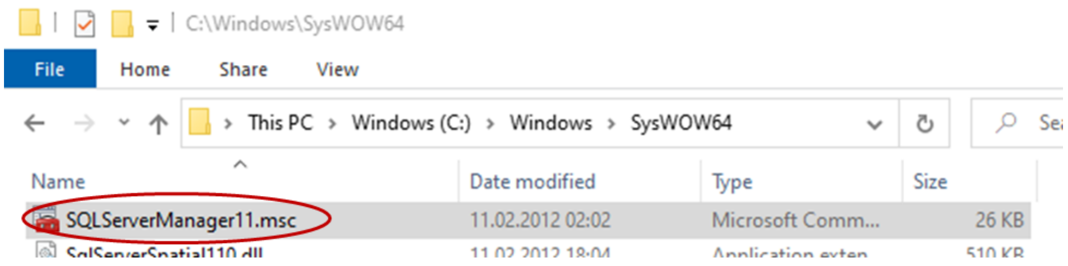
Allow apps downloaded from:
App Store
App Store and identified developers
"Bee-Up" was blocked from opening because it is not from an identified developer.

2. **Database connection failed**: (MS SQL Server specific) It can happen when starting the tool that an error is encountered with the message "ADOxx could not connect to the database …! Please try again." This can happen when the database service is not running. Please make sure that the proper MS SQL Server service is running ("SQL Server (ADOXX_SQL2019)" in the services panel of Windows or the corresponding Docker container on Linux / macOS) and try starting Bee-Up again.
3. **The tool is stuck / a pop-up isn't showing up**: When running Bee-Up through Wine it sometimes happens that a pop-up used by the functionality don't show up properly, which blocks the entire application until it is dismissed. In such a case it is recommended to press ESC to cancel the pop-up and re-run the functionality.

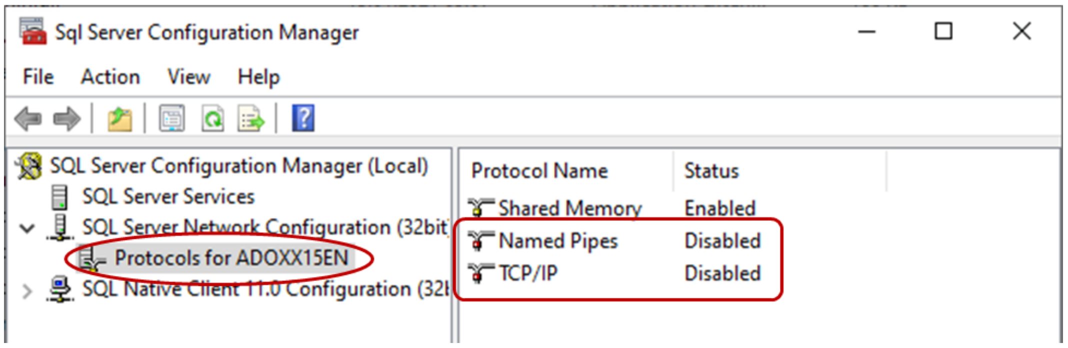


4. **Loosing connection to database while tool is running**: (MS SQL Server specific) On some Windows systems it can happen that the connection from Bee-Up to the database is lost, usually with an error message like "Due to a database exception the connection has been closed …" This often happens due to a longer inactivity which can lead to a connection time out. Unfortunately a consequence of this is a loss of all the changes that have been made since the last save. It can however often be prevented from happening by opening the "SQL Server Configuration Manager"

tool (typically found under "C:\Windows\SysWOW64\SQLServerManager11.msc"), selecting "Protocols for ADOXX_SQL2019" and disabling both "Named Pipes" and "TCP/IP" as shown in the picture below.

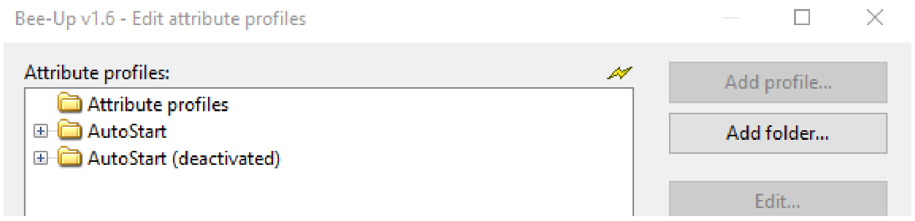


Name	Date modified	Type	Size
SQLServerManager11.msc	11.02.2012 02:02	Microsoft Comm...	26 KB


Sql Server Configuration Manager
File
Action
View
Help
SQL Server Configuration Manager (Local)
SQL Server Services
SQL Server Network Configuration (32bit
Protocols for ADOXX15EN
SQL Native Client 11.0 Configuration (32

Protocol Name	Status
Shared Memory	Enabled
Named Pipes	Disabled
TCP/IP	Disabled

5. **Some of the Bee-Up functionalities are missing:** Starting with version 1.6 of Bee-Up some of the functionalities have been externalised into "AutoStart" Attribute Profiles. If they are missing, then it might be that they have not been properly imported or that the directory structure of the Attribute Profiles differs from what is expected. The Bee-Up installation package comes with a file named "Standard Bee-Up attribute profiles.adl", which contains the necessary Attribute Profiles and their groups (directories). For details on how to import Attribute Profiles see section Exporting and importing Attribute Profiles. The relevant directories to add the functionalities in Bee-Up are shown in the picture below, accessible through "Edit → Attribute profiles..." in the "Modelling" component. Note that both "AutoStart" and "AutoStart (deactivated)" must be on the same level (siblings) as the default "Attribute profiles" directory and NOT inside another group. Should that not be the case then you can move the directories using the "Move" option in the dialogue available through the menu "Edit → Attribute profiles..." in the "Modelling" component.



Manual installation of Microsoft SQL Server instance

An alternative to using Bee-Up with SQLite is to use Bee-Up with a MS SQL Server instance with the proper configuration. To use an MS SQL Server, it is necessary to manually install it, create the Bee-Up database and then adapt the shortcuts to use SQLSERVER instead of SQLITE. On Linux and macOS you can use a Docker image instead, which comes with the proper configuration, but will not be described here.

An installer for MS SQL Server 2019 Express can be downloaded from the Microsoft Homepage[3]. Detailed descriptions for the manual installation of the SQL Server instance on Windows can be found in the "dbinfo" folder (the PDF-files with "install" like "BOC-Product_sqlserver_2008_express_install_en.pdf" are relevant, not "createdb"). An alternative is available online on the ADOxx homepage (under "Download" → "Windows Installation Guide" → "Installation of different collation database").

[3] https://www.microsoft.com/en-us/download/details.aspx?id=101064 Microsoft® SQL Server® 2019 Express.

IMPORTANT: Independent of what these guides say, the "Instance ID" of the SQL Server instance SHOULD be ADOXX_SQL2019 and the collation should be set to Latin1_General_CS_AS (CS: case-sensitive and AS: accent-sensitive)!

A simplified installation procedure:

1. Run the SQL Server installer and perform a "Custom" installation.
2. Create a new installation / instance, follow the instructions and ensuring the following:
   a. During "Instance Configuration" a "Named instance" must be created and the "Instance ID" SHOULD be set to "ADOXX_SQL2019" and preferably the name too.
   b. During "Server Configuration" specify the collation to be "Latin1_General" (Windows collation designator) with "Case-sensitive" and "Accent-sensitive" both on or try "SQL_Latin1_General_CP1_CS_AS" (SQL collation, not tested).
   c. During "Database Engine Configuration" specify "Mixed Mode" as the authentication mode and preferably use the password "12+*ADOxx*+34" (without the double quotes). It is also recommended to add the current user under "Specify SQL Server administrators".

After the installation the asqladm.exe tool from the installation folder can be used to set up the database.

Uninstallation

Yes, it is also possible to uninstall Bee-Up if so wished or necessary. Please note that this will of course also delete all the created models that are stored in the database. Therefore, it is advised to first back up everything from Bee-Up that should be saved using the Export functionality (see section "Exporting and importing models").

To completely uninstall Bee-Up using SQLite on windows, two things should be performed:

1. Uninstall the Bee-Up tool – This can simply be achieved through running the uninstaller. This can be done from the systems control panel / settings.
2. Delete the database file (also deletes all models) – With SQLite the database is stored into a simple file ending with ".sqlite3". By default, the database file is located in the user's roaming AppData folder under ‘ADOxx\sqlitedbs\’ (e.g. ‘C:\Users\Patrik\AppData\Roaming\ADOxx\sqlitedbs’). When the environment variable ‘ADO_SQLITE_DBFOLDER’ is set, then the database files are located in the folder specified by the environment variable instead. Simply delete the file corresponding to the Bee-Up database, e.g. "beeup17.sqlite3" in most cases.

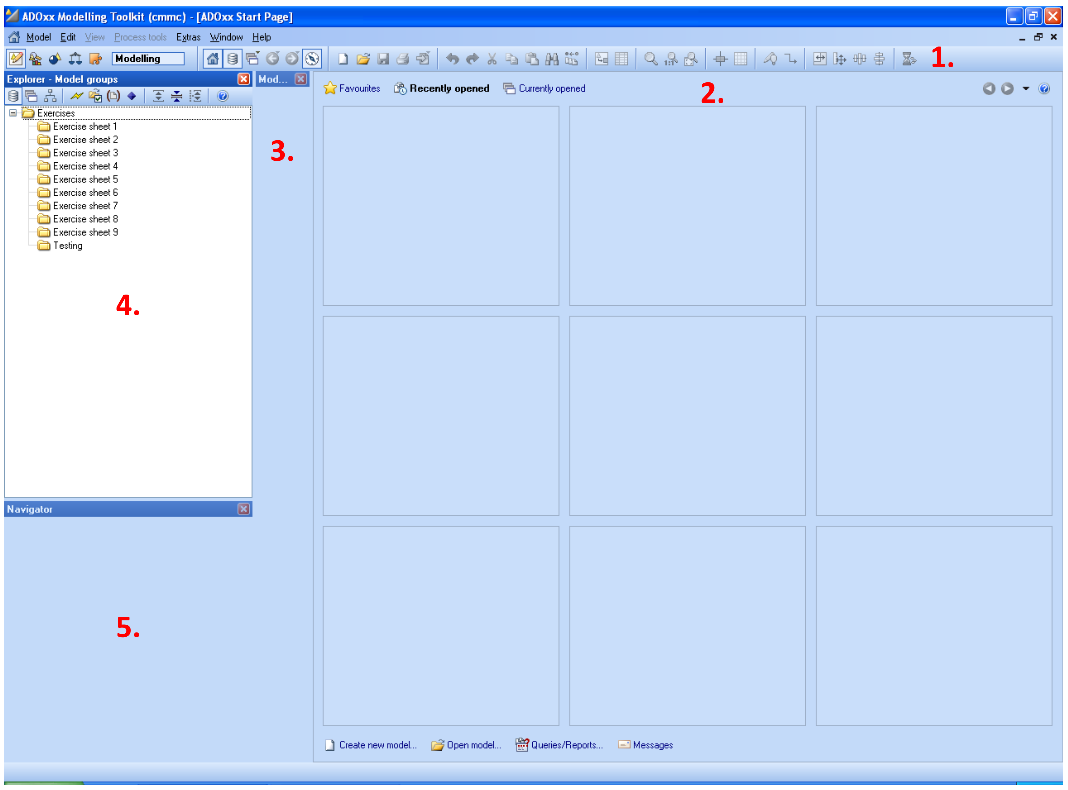
For Linux and macOS please use the provided uninstallation script (uninstall_on_linux.sh / uninstall_on_macos.sh). Note that the scripts do not remove the prerequisite applications (Homebrew, Wine). These have to be removed manually.

IMPORTANT when using MS SQL Server: Removing the SQL Server instance will also remove all the data stored in all the databases of the instance. If other ADOxx based tools have been installed and use the "ADOXX_SQL2019" instance (which is the default), or other relevant data has been put there, then it is not advised to uninstall the SQL Server instance, since it could break the other applications!

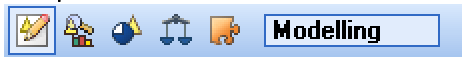
Modelling with the Bee-Up tool

Tool overview

When first starting the Bee-Up tool you see a screen similar to the following one (without the numbers):



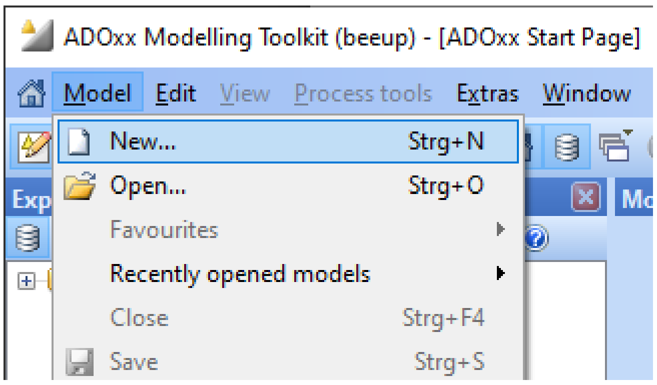
At the top is the menu bar with different menu items for direct access to some of the platforms functionality. The numbered elements of the above picture are:

1. The **Toolbar** with icons as shortcuts for different functions. On the left side of the toolbar is the component selection:
   

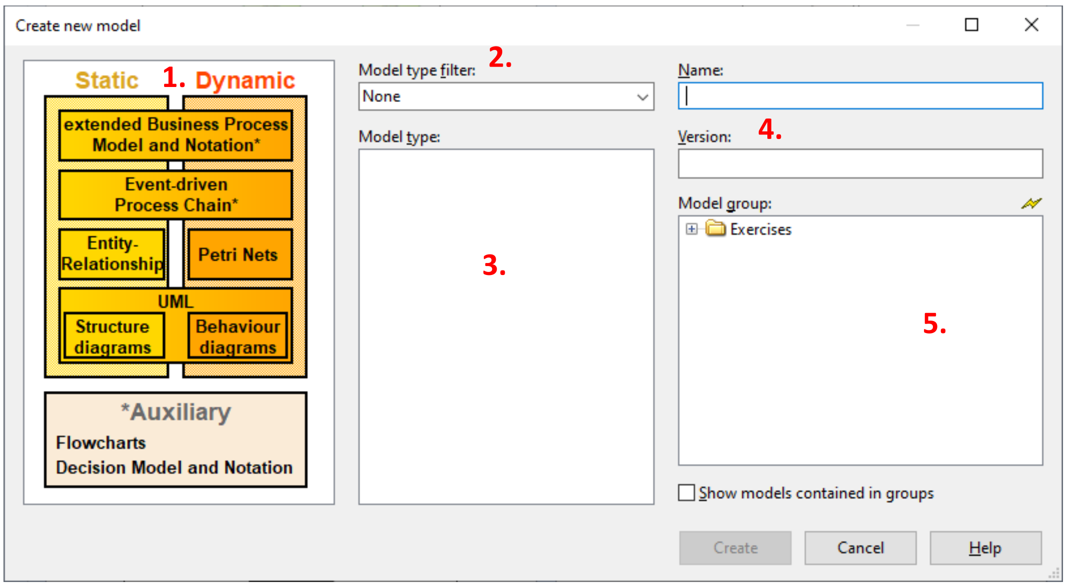
   This changes which menus, menu items and toolbar icons are available. The two important components are "**Modelling**" (left most icon) and "**Import/Export**" (right most icon). The current section of this document focuses on the "Modelling" component, while the next will describe some functionalities of the "Import/Export" component.
2. The **Start Page** is visible, showing recently opened models. The **Start Page** can be accessed through the house icon in the **Toolbar**. Once a model is opened it will be shown instead of the **Start Page**. This area is then referred to as the **Modelling area**.
3. The **Modelling window** shows the modelling objects and relations available for the currently opened model (in the figure above none, because no model is open).
4. The **Explorer window** shows all folders (called model groups) and the models stored in a model group. Initially, model groups for all exercise sheets are preconfigured, accompanied by a testing model group.
5. The **Navigator window** shows an overview of the currently opened model.

Creating a new model

In order to create a new model select the menu item "Model → New..." while in the "Modelling" component (left most icon in the **Toolbar**).

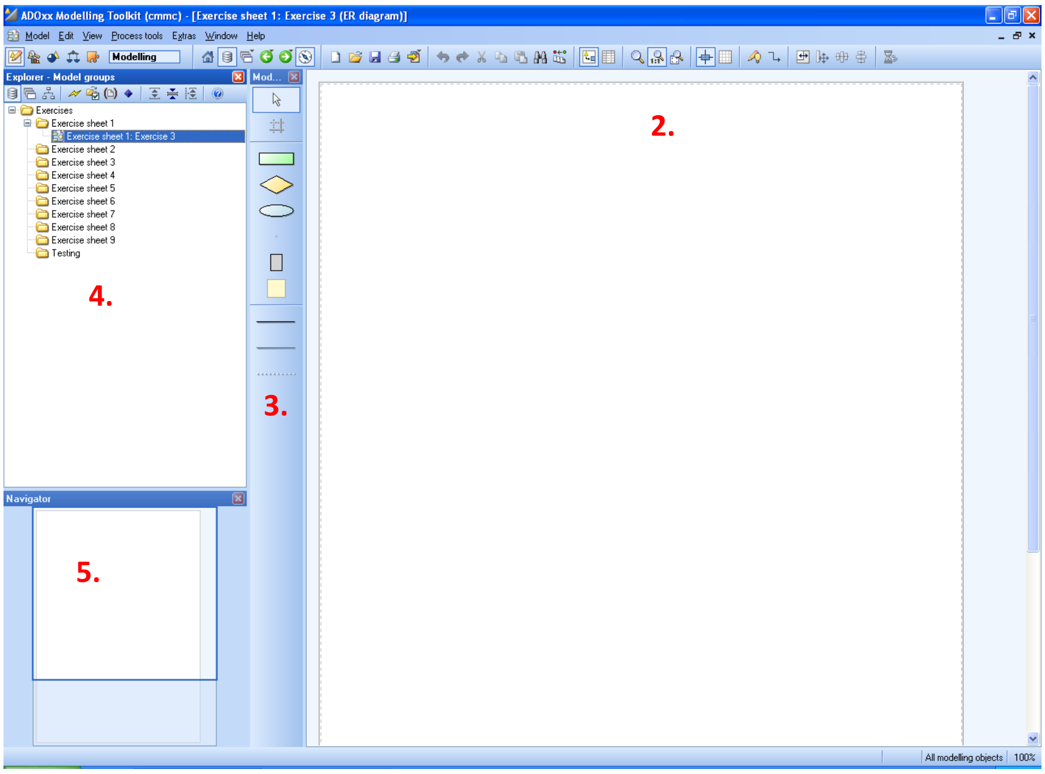


This will open the dialog shown below (without the numbers):



In this dialog first select the appropriate filter for the model types (e.g. Entity-Relationship for ER-models), either by using the graphic on the left side (**1.**) or the dropdown-list in the middle (**2.**). Afterwards, select the desired model type from the list in the middle (**3.**). Then enter a name (mandatory) and a version (optional) on the right side (**4.**). Finally, select to which model group the model should be assigned (**5.**). Use the model groups to organize your models in a clean manner.

Selecting "Create" will create an empty model of the selected model type, store it in the selected model group and open it, ready to be edited. Model groups can be managed (created, moved, deleted) through the context menu at (**5.**), the context menu in the Explorer window or through the "Model → Model groups..." of the "Modelling" component ().



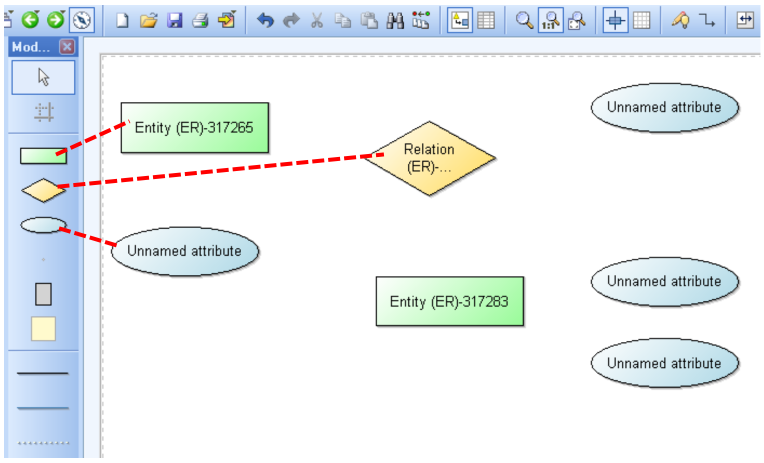
The picture above shows an example for a newly created and opened Entity-Relationship model. The **Modelling area** (**2.**) now shows the empty **Drawing area** (a white canvas) instead of the **Start Page**. The **Modelling window** (**3.**) shows the available types of objects and relations that can be used for ER modelling, e.g., *Entity*, *Relationship* etc. The created model can also be seen in the **Explorer window** (**4.**) under the selected model group. The **Navigator window** (**4.**) shows the complete model, enabling direct navigation and zooming, as well as the portion that is currently shown in the **Modelling area**.

Cloning existing models

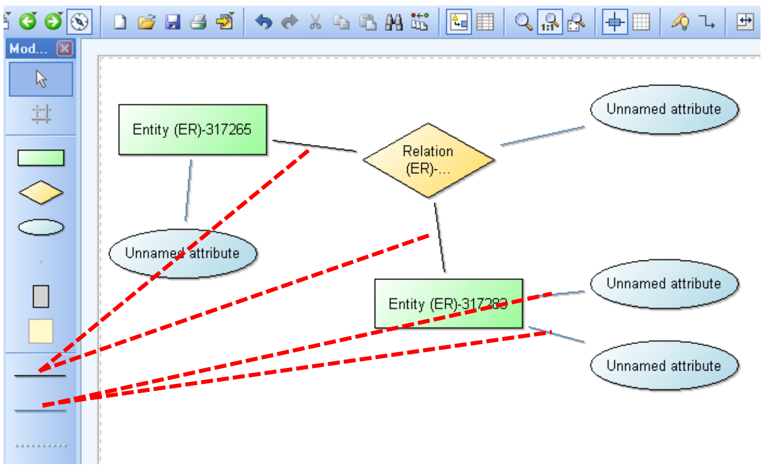
Bee-up 1.5 added a functionality to clone existing models, which can be helpful to create an intermediate save point. It is available through the "Model → Clone active model" or "Model → Clone set of models" menu while in the "Modelling" component (). The benefit of cloning is that it allows to clone a set of models at once while keeping them consistent, e.g. any Inter-Model references (pointers towards other objects / models) used in the clones will be adapted to point towards clone models of the same set.

Editing the model

To edit a model it hast to be opened in the Bee-Up tool. The easiest way to open a model is to double click on its name in the **Explorer window**. New objects can be added to the model by selecting the type of object that should be added in the **Modelling window** and then clicking in the **Modelling area** where the object should be placed. If necessary the **Drawing area** will be extended automatically. After adding a few objects, the **Modelling area** could look like this:



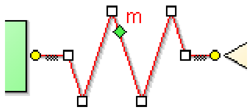
In order to connect objects with relations, first select the relation type from the **Modelling window**. Then click in the **Modelling area** on the object that is the **source** ("starting point") of the relation and then on the object that is the **target** ("ending point") of the relation. Certain types of relations can only be used between specific types of objects (e.g. "has Attribute" always has to target an "Attribute"). The previous example with some relations could look like this:



All objects can be moved in the **Modelling area** by selecting them and moving them accordingly. Some objects can also be resized, which works similar to resizing windows in the operating system (drag the side/corner when it is selected). For both the "Edit" function has to be selected in the **Modelling window** (looks like the default mouse cursor). After creating objects and relations you can quickly switch back to "Edit" by pressing the **right mouse button**. It is also possible to move/resize several objects at once by selecting them first (draw selection box around them, SHIFT+Click, CTRL+Click) and then performing the move or resizing. The difference between SHIFT+Click and CTRL+Click is that using SHIFT will select the object and all of the objects it contains if it is a container (like a "Package", "State", "Combined Fragment", "Lifeline" etc.) while using CTRL only select the object itself. SHIFT-Click is useful when a container should be moved with all of its contents at once.

Adding edges to relations

It is also possible to add edges to relations through so called **bend points**. Those force the relation to be drawn through that point and can increase readability of the created models. The following picture shows a relation with six bend points (small white rectangles):

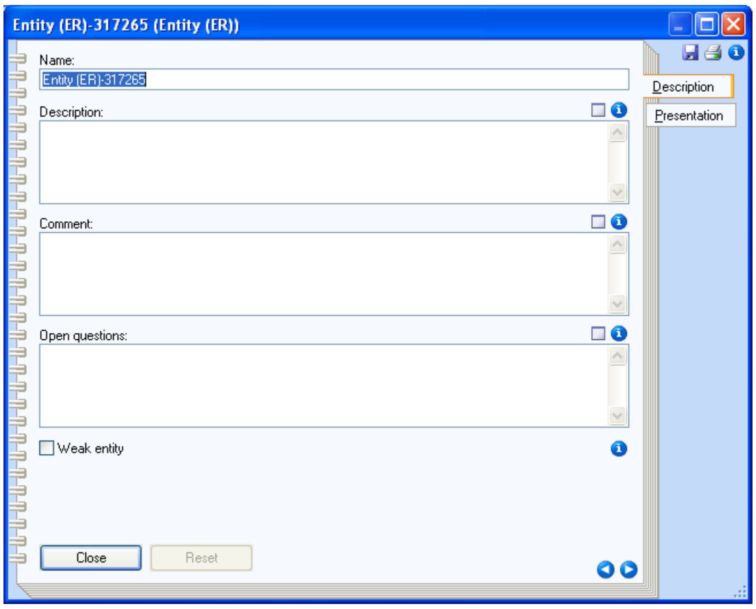


Bend points can be added either during the creation of the relation, or afterwards. To add **bend points** during the creation first click on the source, then click on the desired points in the **Modelling area** where a bend point (i.e. an edge) should be drawn, and finally click on the target object. To add them after the relation has been created, select the relation first and then click and drag a point of the relation (that isn't already a bend point) to the desired place on the **Modelling area**.

The source or the target of a relation can also be changed by selecting the relation and then clicking the yellow circle at the beginning or the end and dragging it to the new object that should be the source or target. The **green diamond** that is visible when a relation is selected can be used in many cases to move the text that is visualized next to it (e.g. the cardinality of the "Links" relation; 'm' in the figure above).

Editing attributes

All objects, models and relations can have editable attributes, which can also influence their visualization in the **Modelling area** (e.g. weak entities have a double outline instead of a single one). Those attributes can be accessed by double-clicking on the object or relation (or selecting it and then pressing Enter). This opens up the ADOxx **Notebook**, which contains the attributes that can be edited. To access the attributes of a model it has to be opened first and then select the menu item "Model → Model attributes" or press ALT+Enter. The following picture shows an example of a **Notebook**:



The attributes take up the largest portion of the **Notebook** and are categorized in different tabs, available on the right side of the **Notebook**. Depending on the attribute type different editors for the attribute value are available (e.g. single-line text field for the Name, multi-line text area for the Description, checkbox for the Weak entity etc.). For some attributes an alternative editor can be accessed through the ☐ button. Also additional information is available for most of the attributes and can be accessed by clicking on the ⓘ icon. A similar icon can be found at the top right (underneath the X for closing the window), which provides information about the type of the object. The two ◀ ▶ buttons at the lower right are an alternative to switching between the different tabs. They are also used to switch between pages of one tab, in the case that more attributes are available than can be shown in the **Notebook** window.

There are two special types of attributes that have to be described in more detail:

1. Inter-model references – they allow to link (reference) to one or several other objects or models and have three special icons: ✚ ✖ ➡. The first one (+) allows to add new references, while the second one (X) removes the selected references. The third one (→) is like a hyperlink that jumps to the referenced object. Often when the attribute value is visualized it also works as a hyperlink in the Modelling area.
2. Tables – they allow to store more complex attribute values in a structured way. They also have two special icons: ✚ ✖. The first one (+) is used to add a new row at the end and the second one (X) removes the selected rows. Note that in order to select rows the number on the left side has to be clicked. The context menu also provides several options to handle rows in a table (e.g. insert row, move row, etc.)

It is also possible to edit some of the attribute values that are visualized in the **Modelling area** without using the **Notebook**. For this simply select the object and then click on the visualized attribute value (e.g. the name: "Entity (ER)-317265"). Note that this takes precedence over opening the **Notebook** through double clicking, thus preventing it.

Checking models

The Bee-Up tool also provides some simple checks in some types of models (ER and EPC among others). While previously these have been performed automatically during modelling, it is now necessary to manually request some of them. To check if the elements and relations are used properly the menu item "Model → Check cardinalities" while in the "Modelling" component (✓) can be used. Note that the cardinality check is meant as a help and not a replacement for human knowledge. Therefore it can miss some errors in the model (like an "Event" following an "Event" when an operator is between them).

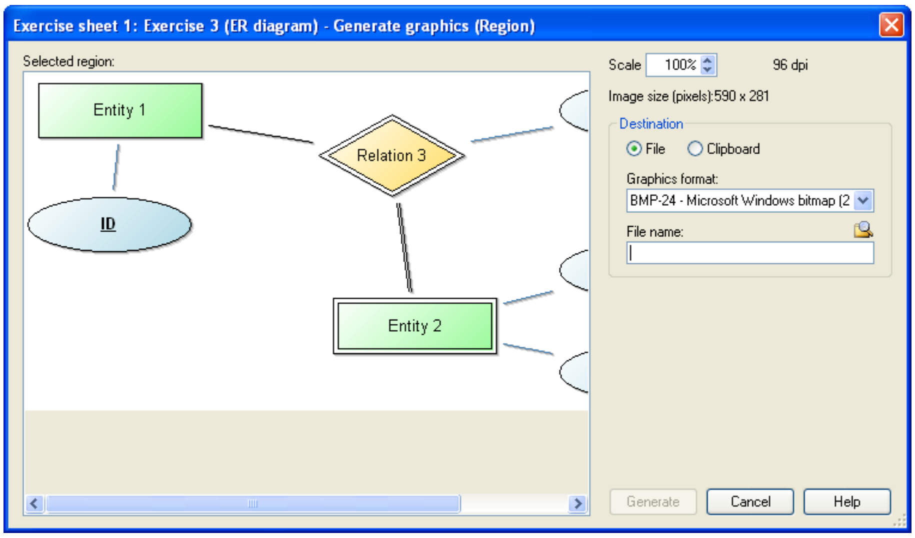
Exporting models

The tool also provides functionality to export the created models in different formats. Some of those will be described here. Most of them are available in the "Import/Export" component ( )

Creating a graphic from a model

It is possible to create a graphic of the currently opened model and store it in a file using the provided "**Generate graphics**" functionality (icon in the **Toolbar** or through the menu "Edit → Generate graphics → Region…", both available when using the "Modelling" component). This opens a dialog showing the region of the model for which the graphic will be created as well as options to scale the created picture and to either save it in a file (with different formats available) or copy it to the clipboard. Clicking on the "**Generate**" button will finish the process.

The region can be set beforehand by holding down the ALT key and click-and-dragging a box in the **Modelling area** with the mouse. This will create a teal rectangle showing what region will be used for generating the graphic and which can also be resized. The following picture shows the dialog for the previous example model, where a fitting region has been set:



Scale 100% 96 dpi
Image size (pixels):590 x 281
File [✓] Clipboard
Graphics format: BMP-24 - Microsoft Windows bitmap (2
File name:

Exporting the exercise

It is also possible to export an entire exercise at once. This can be achieved by using the menu item "Export → Export Exercises" (or using the icon in the **Toolbar**). This will show a dialog where the model group containing all of the solutions for the exercise can be selected. Afterwards a new dialog asks for a folder where the results should be stored. Once it is finished a message will inform you about it.

This functionality exports all of the models contained in the selected model group or one of its descendent sub-groups in ADL format and also creates individual graphic files for all of the models. The creation of the graphic can sometimes fail when the name and/or version contain a character that is not supported by the operating system as a filename[4]. **IMPORTANT**: It also removes empty space from the right and the bottom of the drawing area (in the **Modelling area**) AND saves the model before generating the graphic.

Exporting and importing models

One or several models as well as model groups can be exported to/imported from either the ADL format (proprietary) or XML format by using the according menu items in the "Model" menu (e.g. "Model → ADL export → Models/Attribute profiles…").

For the export a simple dialog is shown where the models and/or model groups are selected at the top. The middle of the dialog contains some checkboxes to control what should be exported (e.g. "Including models", "Including model groups" etc.) and at the bottom the file is specified where the models/model

[4] Common ones like „:", „/", „*" etc. are replaced by a „-" for the file name.

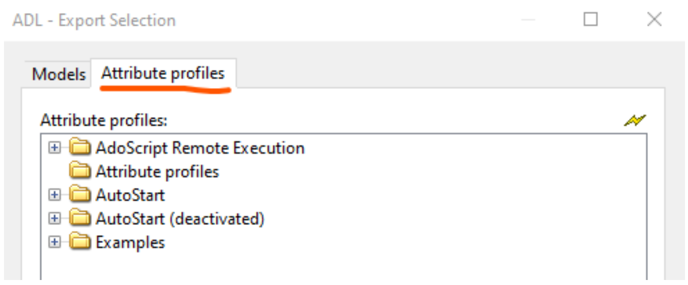
groups should be exported to. The export is started by clicking on the "Export" button and at the end a success or error message is shown.

For the import there are several tabs available. In the "File selection" tab the file containing the models/model groups is selected and whether to import from other libraries, like a different version of the tool. The "Model options" tab provides some choices on how to deal with collisions, e.g. what to do when two models have the same name. The last tab "Log file" allows logging the process in a file. After everything is selected click on the "OK" button. This will prepare the data from the previously selected file and open another dialog. Here the left side shows which models and/or model groups have been found in the file and you can select which of those should be imported. On the right side select into which model group the contents should be imported and click on the "Import" button. At the end a success or error message is shown.

Exporting and importing Attribute Profiles

Attribute Profiles can be imported and exported similar to models both in ADL or XML format using the same menu items (e.g. "Model → ADL export → Models/Attribute profiles..." from the Import/Export component). In this section only the relevant differences are being presented.

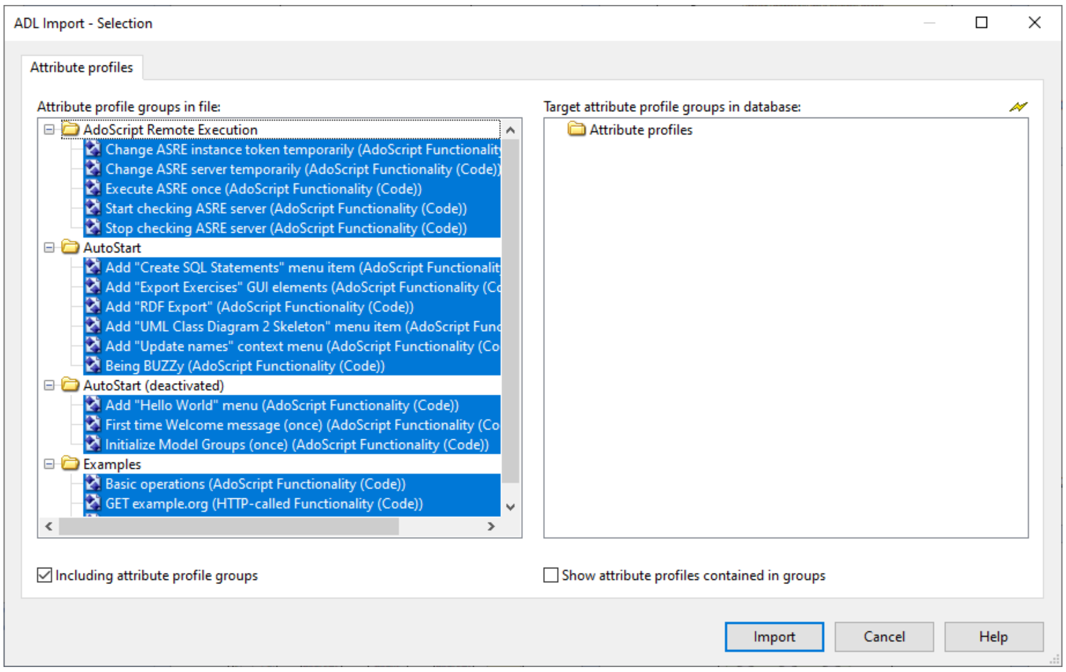
When exporting Attribute Profiles it is important to select the proper tab at the top of the dialogue as shown in the picture below.


ADL - Export Selection

Models | Attribute profiles

Attribute profiles:
- AdoScript Remote Execution
- Attribute profiles
- AutoStart
- AutoStart (deactivated)
- Examples

When Attribute Profiles are being imported it is not necessary to select a "Target attribute profile group". In some cases none should be selected, for example when importing the "Standard Bee-Up attribute profiles.adl", which have a specific structure of groups (directories). Also make sure that the Attribute Profiles on the left side and the "Including attribute profile groups" option in the bottom left are selected.


ADL Import - Selection

Attribute profiles

Attribute profile groups in file:
- AdoScript Remote Execution
  - Change ASRE instance token temporarily (AdoScript Functionality
  - Change ASRE server temporarily (AdoScript Functionality (Code))
  - Execute ASRE once (AdoScript Functionality (Code))
  - Start checking ASRE server (AdoScript Functionality (Code))
  - Stop checking ASRE server (AdoScript Functionality (Code))
- AutoStart
  - Add "Create SQL Statements" menu item (AdoScript Functionalit
  - Add "Export Exercises" GUI elements (AdoScript Functionality (C
  - Add "RDF Export" (AdoScript Functionality (Code))
  - Add "UML Class Diagram 2 Skeleton" menu item (AdoScript Func
  - Add "Update names" context menu (AdoScript Functionality (Co
  - Being BUZZy (AdoScript Functionality (Code))
- AutoStart (deactivated)
  - Add "Hello World" menu (AdoScript Functionality (Code))
  - First time Welcome message (once) (AdoScript Functionality (Co
  - Initialize Model Groups (once) (AdoScript Functionality (Code))
- Examples
  - Basic operations (AdoScript Functionality (Code))
  - GET example.org (HTTP-called Functionality (Code))

Target attribute profile groups in database:
- Attribute profiles

☑ Including attribute profile groups

☐ Show attribute profiles contained in groups

Import | Cancel | Help

Exporting models as RDF

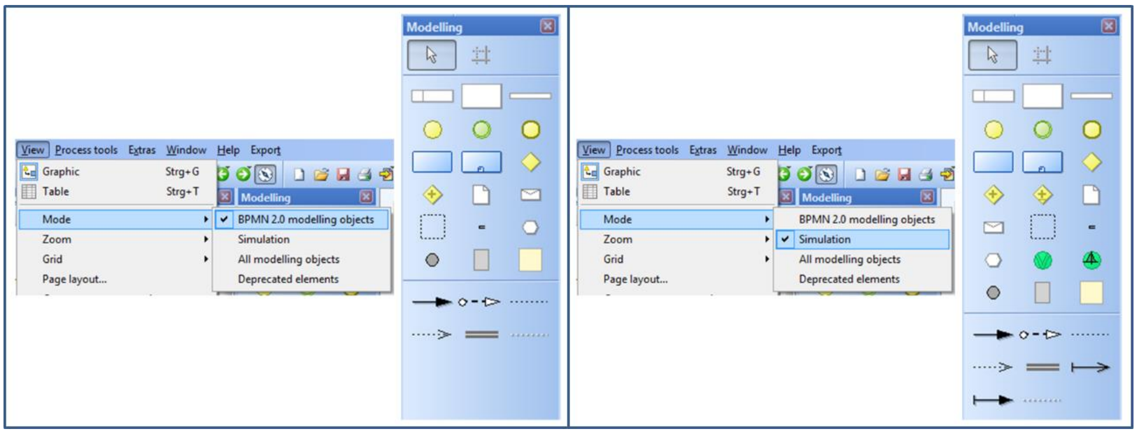
A new function in version 1.1 adds the possibility to export one or several models in RDF Format. This can be simply accessed through the menu item "Model → RDF Export". The first dialog asks which models should be exported. Here either directly the models or entire model groups can be selected. Afterwards a file selection dialog will ask where the RDF data should be stored and allows a choice of different formats (.trig is recommended). A third dialog will ask for a base URI to be used in the identifiers as a prefix. Here it is recommended to use a valid URL without the fragment (for example http://www.omilab.org/example# or http://www.example.net/#). It is not necessary for the URL to be actually used (i.e. the URL can return an error code like 404), just that the URL is valid. Once it is finished a message will inform you about it.

With Version 1.2 additional attributes have been added to (almost) all elements and models to enhance the RDF Export. These are found in the "RDF properties" tab of the **Notebook**. The "URI" attribute allows specifying a specific URI to be used for the element/model instead of automatically generating a URI. The "Additional Triples" table allows specifying additional triples (Subject, Predicate and Object) that will be added to the graph, where one row represents one triple. If a cell is left empty in the "Additional Triples" table, then it will be substituted with the URI of the element/model it is located in. Note that the values provided in those attributes will be treated as is as a complete URI (ignoring prefixes etc.). Therefore, it is necessary to enter the entire URI. Also with Version 1.2 the names of elements and models are exported explicitly as rdfs:label statements.

Additional hints and information

Specific information for BPMN modelling

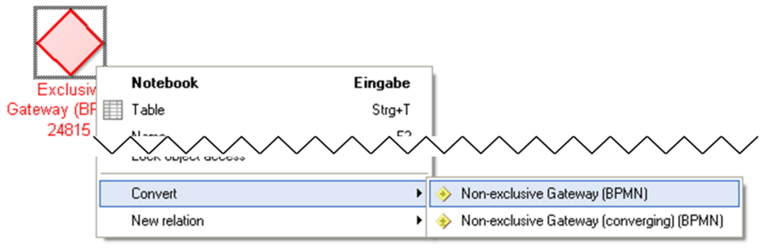
The BPMN implementation provides concepts to describe processes, as well as for describing input, output and execution of "Tasks". Different modes are available, which limit the available concepts. By default the "**BPMN 2.0**" mode is selected, which contains the typical BPMN concepts. However, the mode can be changed (through the menu "View → Mode") to "**Simulation**". This mode further adds concepts which are necessary to perform simulation of processes in the tool (e.g. converging gateways as their own types). The following picture shows the two modes and which types of elements they use:



The majority of the BPMN implementation should be straight forward. Some constraints are enforced by the tool (e.g. "Pool" cannot be the source or target of "Subsequent" relations). However, due to certain platform restrictions the Gateways (Exclusive, Inclusive, Parallel etc.) are handled a bit differently[5]. In the normal BPMN mode the Exclusive Gateway is available as its own type ("Exclusive Gateway"), however the Inclusive and Parallel Gateway are modelled through the "Non-exclusive Gateway". The type (Inclusive, Parallel or Complex) is then set through the attribute "Gateway type" (in the Notebook)[6]. To use simulation the "Non-exclusive Gateway" has to also be distinguished between the two different types of " Non-exclusive Gateway" and "Non-exclusive Gateway (converging)".

Previously the Intermediate event was split into two different types: "Intermediate Event (boundary)" and "Intermediate Event (sequence)". Since Version 1.1 the two have been merged into one and are distinguished through setting the "Attached to" Attribute. If the attribute has a value it will be considered on the Boundary of the set "Task". A new Mode has been added called "Deprecated", which allows the use of the two old Intermediate events in order to not destroy previously created models. Those events can easily be transformed into the new "Intermediate Event" by right clicking on them and selecting "Convert → Intermediate Event (BPMN)".

Certain types of objects can be converted into other types (e.g. "Exclusive Gateway" to "Non-exclusive Gateway") by selecting them and then using "Conversion" in the context menu. An object will become greyed out and cannot be selected, if it is converted to a type that is not available in the current mode. To transform it back (or delete it or change it etc.), simply change the mode to one that makes use of the type (e.g. "All modelling objects"). The picture below shows the available options for converting the "Exclusive Gateway":



[5] This is due to the way simulation is handled by the platform.

[6] "Simulation" mode additionally has a "Non-exclusive Gateway (converging)", which is necessary for the simulation.

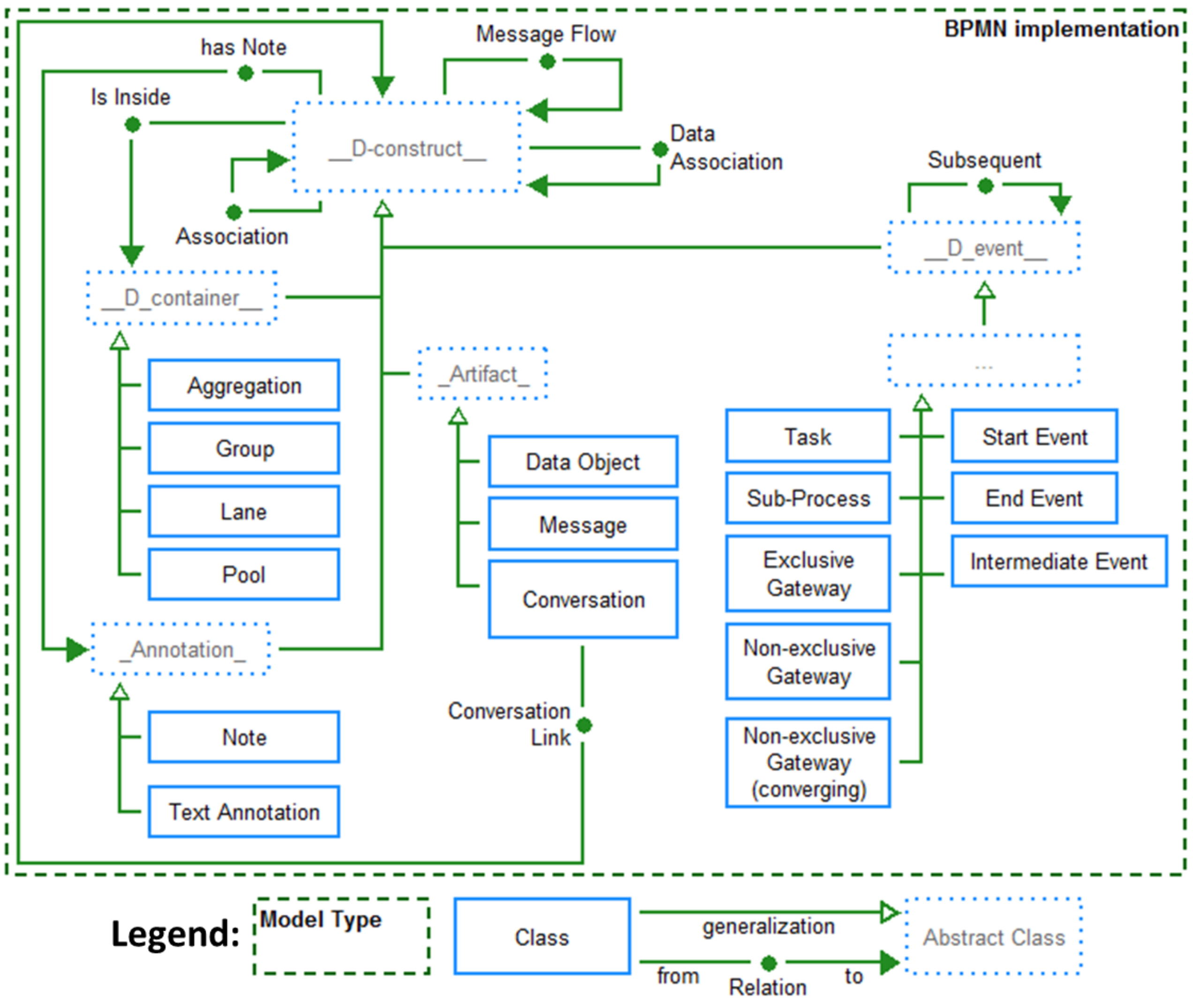
The availability of some attributes (in the Notebook) is dependent on the values of others. This is to prevent setting wrong values or changing irrelevant attributes. For example the available "Triggers" of a "Start Event" depend on its "Type" to prevent wrong selections. Another example is the "Loop condition (standard)" attribute of a "Task", which is only available when the "Loop type" is set to "Standard" (otherwise it is irrelevant).

The relation "Subsequent" has an attribute "Visualized values", which controls which attribute values are shown. Should the desired value not be shown on the drawing area (e.g. "Transition condition") then it might be because of the "Visualized values" attribute. "Subsequent" is also used in several different model types (e.g. EPC, UML Activity Diagram). Therefore it also contains attributes used in those model types. They are however grouped in their own categories (e.g. "UML properties").

For many different types of objects (e.g. "Start Event", "Exclusive Gateway" etc.) the visualization of the name can be controlled through the attribute "Show name". In some cases this is a simple choice if the name should be displayed (e.g. "Start Event"). In other cases more options are available (e.g. for "Exclusive Gateway" to show below, above or not at all).

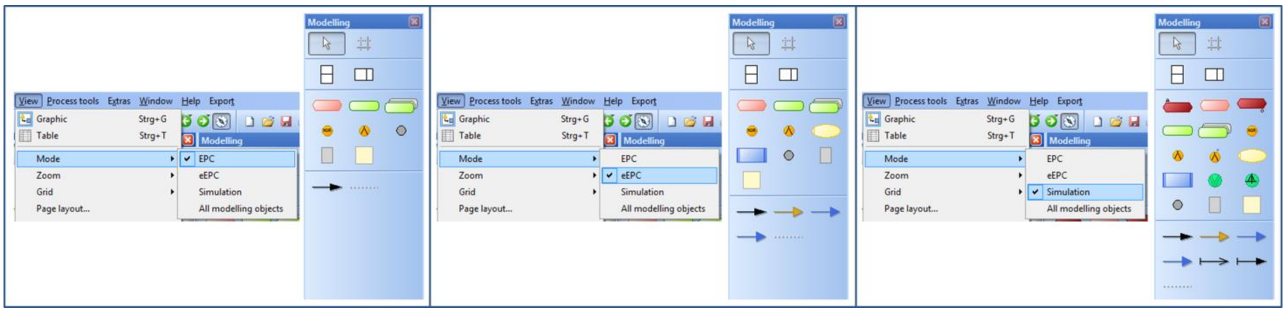
Version 1.2 also added the option to further describe Service Tasks through Petri Nets or Flowcharts using the "Automated service details" attribute. The attribute should reference the starting point in the Petri Net or Flowchart.

The following picture provides some detailed information about the implementation of BPMN in Bee-Up. More specifically it shows an excerpt of how the BPMN meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_event__ and Subsequent used for process simulation. The "…" abstract class is used to represent complex generalization structures in the meta-model in a simplified manner.



Specific information for EPC modelling

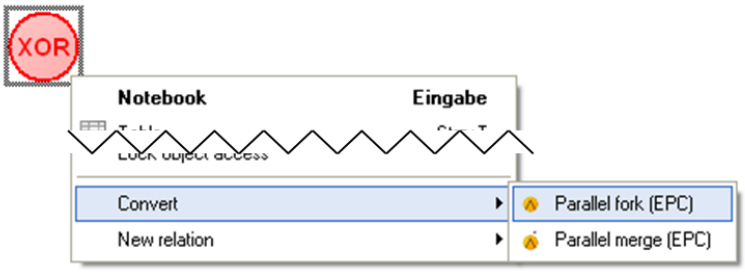
The EPC implementation provides the core concepts from Event-driven Process Chains to describe processes ("Event", "Function", logical operators), as well as some additional ones for describing input, output and execution of "Functions". Different modes can be selected, which limit the available concepts. By default the "**EPC**" mode is selected, which contains "Events", "Functions" and the basic logical operators from EPC with some additional "general" concepts. However, the mode can be changed (through the menu "View → Mode") to "**eEPC**" or "**Simulation**". "eEPC" (extended EPC) additionally contains "Organizational units", "Information objects" and relations for those new object types. The relations for denoting inputs and outputs for "Functions" are implemented as separate types. "Simulation" mode further adds concepts which are necessary to perform simulation of processes in the tool (e.g. "Start Event" which explicitly denotes the start of the process). The following picture shows the three modes and which types of elements they use:



The majority of EPC should be straight forward. However, due to certain platform restrictions the typical logical operators (XOR, OR, AND) are handled a bit differently[7]. In the basic "EPC" and "eEPC" the XOR operator is available as its own type ("XOR operator"), however the AND and OR operators are modelled through the "Parallel fork". The type (AND or OR) is then set through the attribute "Type" (in the Notebook)[8]. In "EPC" and "eEPC" mode the "Parallel fork" can be used both for splitting and merging paths. For simulation it is however necessary to use the "Parallel fork" to split and the "Parallel merge" to join paths again.

The relation "Subsequent" has an attribute "Visualized values", which controls which attribute values are shown. Should the desired value not be shown on the drawing area (e.g. "Transition condition") then it might be because of the "Visualized values" attribute. "Subsequent" is also used in several different model types (e.g. BPMN, UML Activity Diagram). Therefore it also contains attributes used in those model types. They are however grouped in their own categories (e.g. "UML properties").

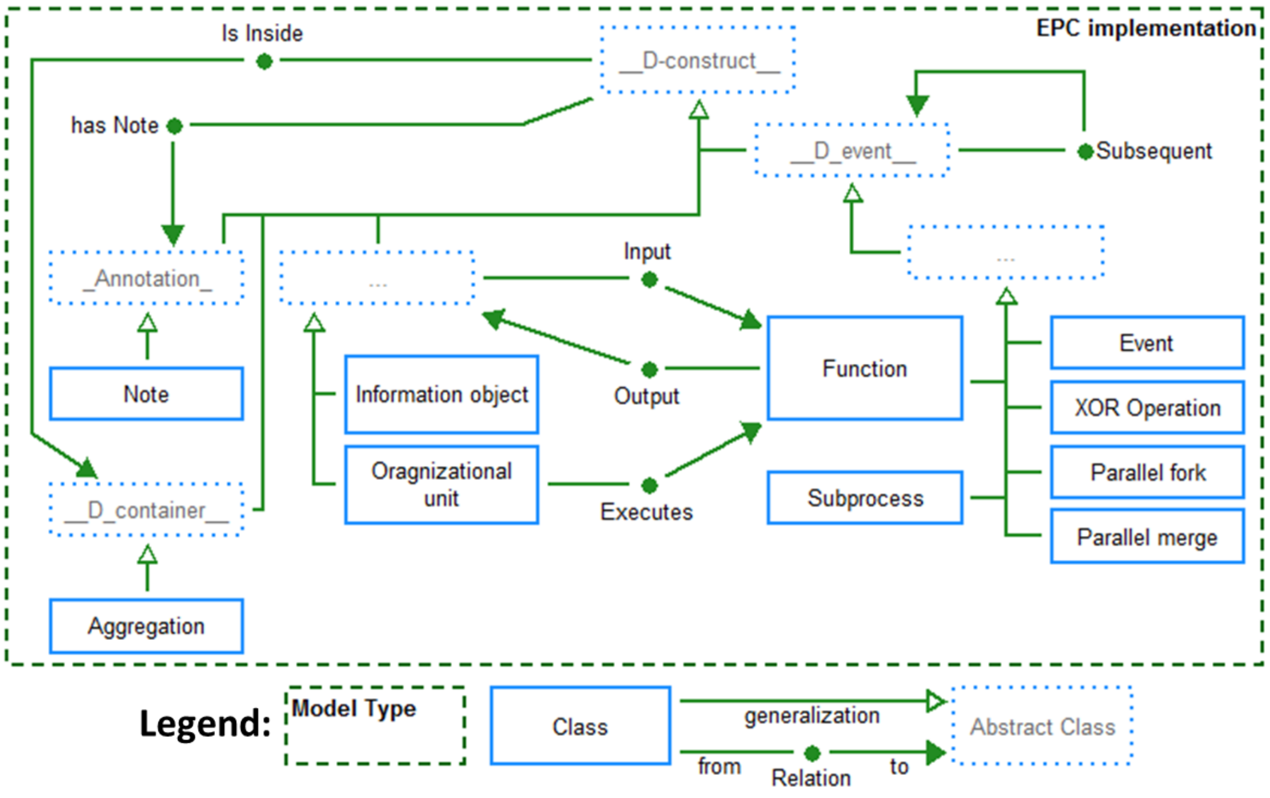
Certain types of objects can be converted into other types (e.g. "Event" to "Start Event" or "End Event", "XOR operator" to "Parallel fork" etc.) by selecting them and then using "Conversion" in the context menu. An object will become greyed out and cannot be selected, if it is converted to a type that is not available in the current mode. To transform it back (or delete it or change it etc.), simply change the mode to one that makes use of the type (e.g. "All modelling objects"). The picture below shows the available options for converting the "XOR operator":



[7] This is due to the way simulation is handled by the platform.
[8] "Simulation" mode additionally has a "Parallel merge". This distinction between fork and merge is necessary for the simulation algorithm.

Version 1.2 also added the option to further describe Functions through Petri Nets or Flowcharts using the "Automation details" attribute. The attribute should reference the starting point in the Petri Net or Flowchart.

The following picture provides some detailed information about the implementation of EPC in Bee-Up. More specifically it shows an excerpt of how the EPC meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_event__ and Subsequent used for process simulation. The "…" abstract class is used to represent complex generalization structures in the meta-model in a simplified manner.



Specific information for ER modelling

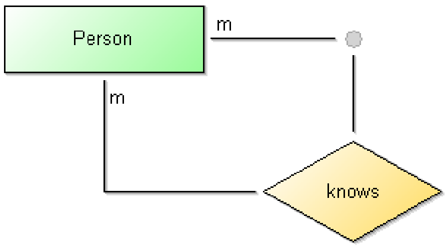
The ER Model implementation provides the general concepts used ("Entity", "Relation" and "Attribute") as well as the necessary connectors[9] ("Links" and "has Attribute") among other common elements ("Note", "has Note" etc.). A "Links" connector should start from either a "Relation" or a "Relation Node" and target an "Entity", a "Relation" or a "Relation Node". So it is necessary to click first on a "Relation" or a "Relation Node" when creating a "Links" connector. Cardinalities for the relation are also set on the "Links" connector. Note that for Chen-Notation the "m" is used for anything else than 1, meaning it should be used to represent Cardinalities like "m", "n", "o" etc. Think of "m" not as a specific number, but as "many". What notation is visualized in the model can be set through the model attribute "Used Notation (ER)" found in the "ER properties" tab.

The finer details are controlled through the attributes found in the notebook, which in some cases also influence the visualization (notation) of the objects. For example to show a "Weak Entity" use a normal "Entity" and check its "Weak entity" attribute in the Notebook. Also to specify the "Relation" that indicates on which stronger entity it relies use a "Relation" and set its "Relation type" attribute to "Weak entity dependency".

Should a "Relation" be between the same "Entity" (e.g. Person knows Person) then use the "Relation Node" on one of the connections. For a binary relation (e.g. Person knows Person): First connect the "Relation" to the "Entity" directly, then connect the "Relation" to a "Relation Node" and then connect the "Relation

[9] In this one section we refer to the lines as "connectors" instead of "relations" to not confuse them with the objects of type "Relation"

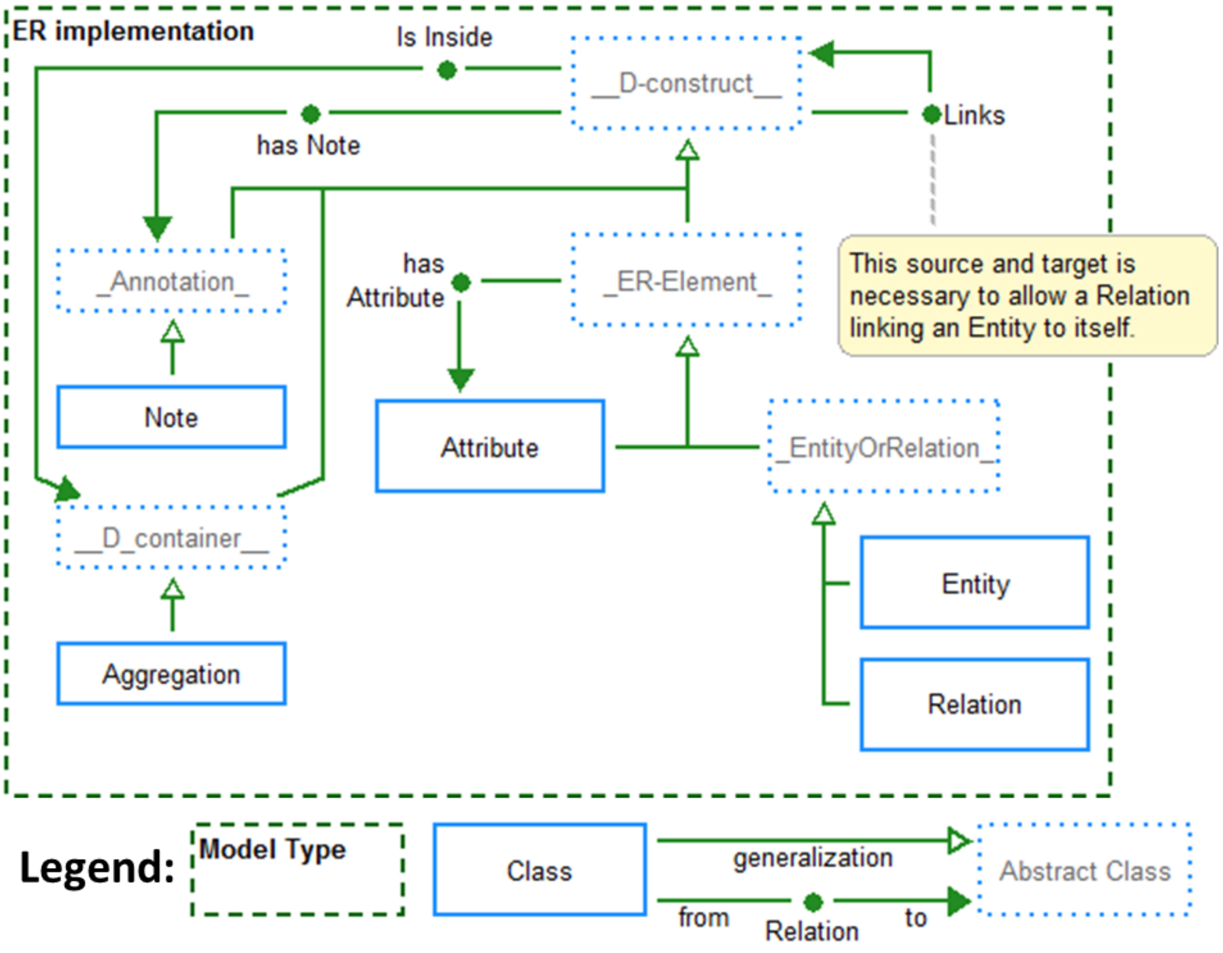
Node" to the "Entity". This is necessary because of how identifiers of connectors work (identified by their type, their source object and their target object). An example can be seen below:



To help manage the names for objects an "Update names" option has been added in Bee-Up 1.6 to the context menu. It updates the names of all selected elements (e.g. "Attributes") based on the value specified in their "Label" / "Denomination".

Functionality for creating SQL statements from an ER Model is also provided. In version 1.6 it has been moved to a button, which is accessible in the model attributes of an ER model with a description of the quirks and details found below the button. The functionality uses the currently active model and will ask for a file to store the created SQL code in. If the selection of a file is cancelled it will instead show the SQL code in a pop-up window from where it can be copied to the clipboard. Version 1.2 added two SQL properties to "Attribute": 1) one for directly entering the data type of the attribute and 2) to specify auto-increment (only works for MySQL). Version 1.3 changed how to handle inheritance through "IS-A" relations. Two options are available as Model attribute "IS-A Behaviour": 1) the "old" style where the table is copied and 2) [now default] which handles inheritance similar to Weak Entities.

The following picture provides some detailed information about the implementation of ER in Bee-Up. More specifically it shows an excerpt of how the ER meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_container__ used to automatically derive "Is Inside" relations.



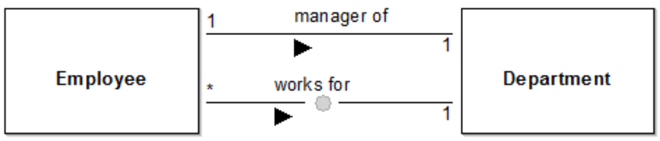
Specific information for UML modelling

UML and its implementation in the tool are big. Addressing all of the peculiarities would be difficult and also a lot of text to read. Therefore, they are addressed in general and some examples are provided.

Notations[10] are generally influenced by the attribute values that are specified for them (in the notebook):

- Most of the attributes that deal only with the visualization are located in a category called "Presentation". Examples for such attributes are "Color" (of the object background), "Representation" (of text) and "Presentation" (of class details).
- The "Subsequent" relation and the "Activity edge" use the attribute "Visualized values" to control which attribute values should be shown (e.g. Denomination, Transition condition, Weighting etc.).
- Relations often have an option on where the text should be shown, handled through a "Representation" attribute. In general "above/below" value should be used for parts of relations going horizontally and "left/right" value for parts of relations going vertically. As an example the "Association" used in "Class / Object Diagrams" can have text at three parts: at the start, at the middle and at the end. For the start and the end a different "Representation" value can be set. If for example the association class starts going from the object towards the right (horizontal) and then turns towards the bottom (vertical) then the "Representation start" should use "above/below", the "Representation end" should use "left/right".
- UML Specific attributes (e.g. "IsAbstract", "Visibility" etc.) are usually located in a category called "UML properties". In some cases they are located in the "Description" category as well (e.g. the "Type" of a "Final Node") for quicker access or have their own category (e.g. "Properties/Operations" of a "Class"). Some of them also influence the notation, like the "Final type" attribute of a "Final Node" in an "Activity Diagram" or the "Properties" entered in a "Class".
- Certain relations, like the "Message" from a "Sequence Diagram", have their sub-types controlled mostly through the attributes. So the typical types like "synchronous call", "asynchronous call" and "reply" are handled through the "Message sort" attribute of the "Message" relation.

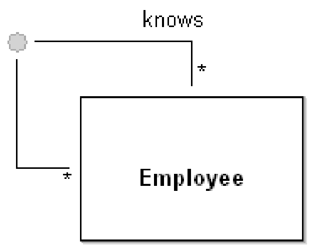
In order to draw several relations between the same two objects in the same direction (e.g. several "Associations" between the same two "Classes") the "Relation Node" has to be used. For every additional relation beyond the first one it is necessary to create two relations: one has to go from the source object to a "Relation Node" and the other from that "Relation Node" to the target object. This is necessary because of how identifiers of relations work[11]. For example when there are the classes "Employee" and "Department" and two associations "works for" and "manager of" between the two classes. The "manager of" association can go directly from "Employee" to "Department". However, the "works for" association has to be split in two: one association going from "Employee" to a "Relation Node" and another from the same "Relation Node" to the "Department". The attributes should also be split among those two relations accordingly (i.e. the multiplicity for the "Employee" side of "works for" has to go to the first relation, the multiplicity for the "Department" side of "works for" has to be in the second relation and the name can be in one of those). The example can be seen below:



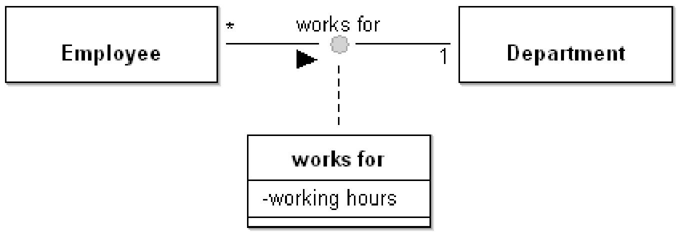
[10] The look of an object on screen or on paper.

[11] A relation is identified by its type, its source object and its target object. Duplicate identifiers are not allowed.

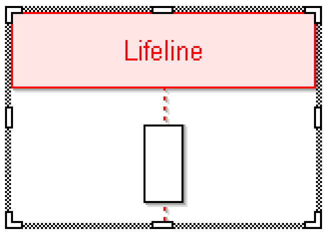
There are also cases where the source and the target of a relation should be the same object (e.g. an "Association" from a "Class" to the same "Class" or a "Transition" from a "State" to the same "State"). This also requires a "Relation Node", since the same object cannot be the source and the target of one relation. For this case simply make a relation from the object to the "Relation Node" and then from the "Relation Node" to the same object. For example when a relation "knows" should be from and to the class "Employee" first create the "Relation Node", then make an "Association" from "Employee" to the "Relation Node" and then from the "Relation Node" back to the "Employee". The example can be seen below:



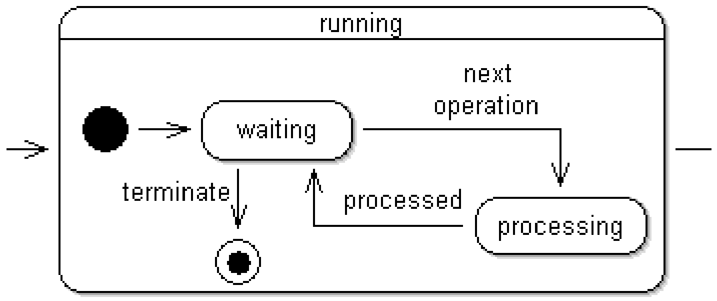
In UML it is also sometimes necessary to have a relation which originates or targets another relation. Again this is solved by using the "Relation Node". Simply put the "Relation Node" on the relation that should be the source or the target (this will split the relation in two) and use the "Relation Node" as the source or target of the other relation. For example when the association "works for", between "Employee" and "Department" should be linked to a class "works for" to indicate it is an association-class (so it can contain attributes like "working hours"): first put the "Relation Node" on the "works for" association and then make the "is Associationclass" relation from that "Relation Node" to the desired "works for" class. The example can be seen below:



The boundary of "Lifelines" should not overlap, due to the automatic assignment of "Execution Specifications" based on being inside of a "Lifeline". The exact boundary of an object is visible when the element is selected and is represented by the thick-chequered line as seen in the picture below:

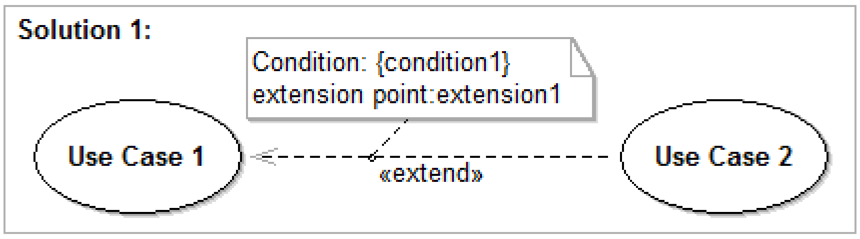


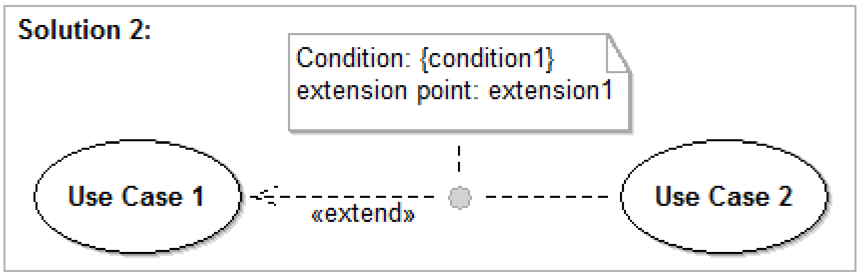
To create a "Composite State" (i.e. a "State" that contains other states) use the "State" type and set the attribute "Number of regions" to a value larger than 0, depending on how many regions are available. A simple example of a "Composite State" with only one region can be seen below:



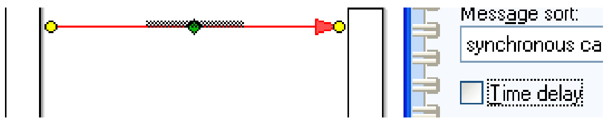
In a UML Use Case Diagram it is possible to add constraints to "Extend" relations using two approaches:

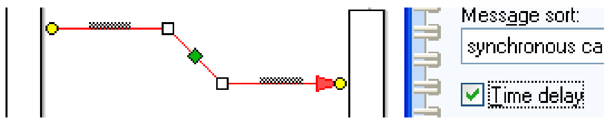
1. Use the "Condition" and "Points of extension" attributes of the "Extend" relation.
2. Create a "Constraint" object, place a "Relation Node" in the middle of the "Extend" relation and then connect the "Constraint" to the "Relation Node" through the "has Constraint" relation.



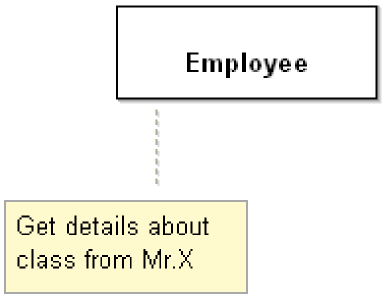


In "Sequence Diagrams" it is sometimes necessary to show a time delay by drawing "Message" relations diagonally. This is generally achieved by adding bend points to a relation. However, adding bend points can be difficult since the tool tries to draw horizontal (and vertical) relations whenever possible. Therefore the "Message" relation contains an attribute called "Time delay". Putting a check mark in this attribute will automatically add two bend points to the relation. Those can then be moved and other bend points can also be added more easily. Removing the check mark will also remove the bend points again. The two pictures below show a "Message" relation with the two possible states of the "Time delay" attribute:



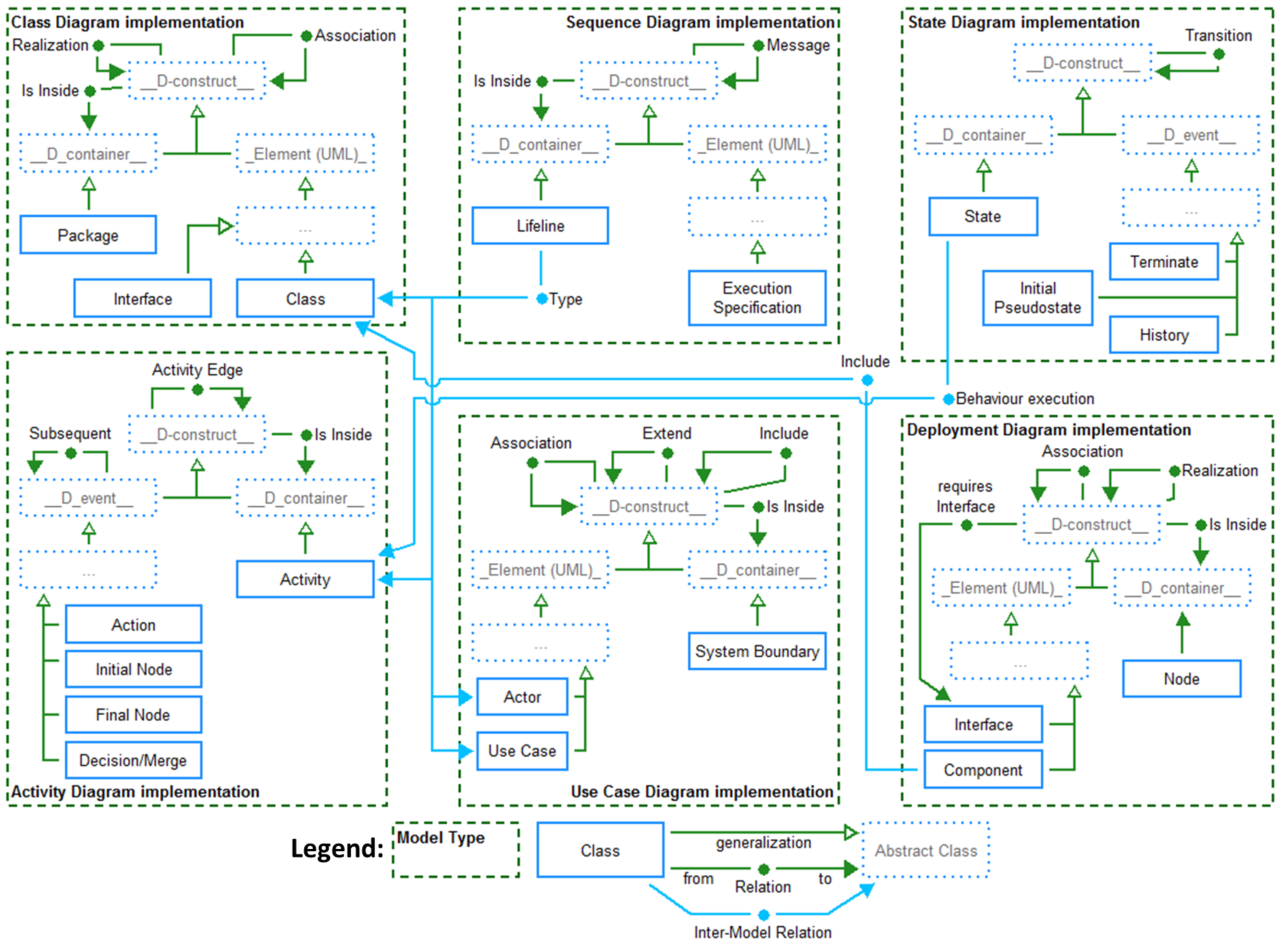


It is possible to leave notes and comments in the models by using the "Note" class and also assigning those notes to any object using the "has Note" relation. The text displayed is specified through the "Description" attribute of the "Note". An example can be seen below:



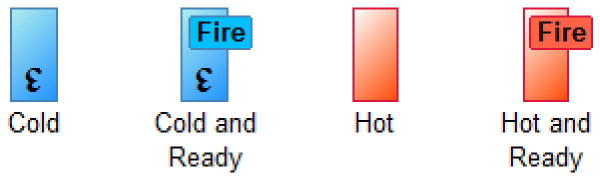
Bee-Up 1.6 added the "UML Class Diagram 2 Skeleton" functionality, which allows deriving source code out of a UML class diagram for Java, C++ and Python. The code should then be further extended with the desired functionality. It is available through the "Model" menu of the "Import/Export" component. Selecting "Cancel" when prompted for a folder to store the results will instead provide the code directly through a text-box. It does however have some limitations, e.g. doesn't work with "Relation Node".

The following picture provides some detailed information about the implementation of UML in Bee-Up. More specifically it shows an excerpt of how the UML meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_event__ and Subsequent used for process simulation (e.g. of Activity Diagrams). The "…" abstract class is used to represent complex generalization structures in the meta-model in a simplified manner.

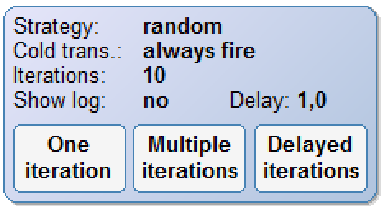


Specific information for PN modelling

The Petri Net implementation provides the base concepts ("Place", "Transition" and a connector called "Arc") as well as some additional ones for simulation and state storage. Tokens are modelled through the "Tokens" attribute of "Place" and are also visualized in them through small black circles and a number if there isn’t enough room to draw all tokens. "Transitions" are also categorized into "Hot" (drawn in red color) and "Cold" (drawn in blue color with a black "epsilon" looking character) transitions which is handled through the "Type" attribute. "Arcs" contain an attribute "Weight" which is used to denote how many tokens should be consumed/created by the attached "Transition". The picture below shows the different notations of a Transition:

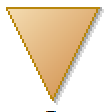


When the conditions to fire a transition are met (i.e. enough tokens in all preceding places and enough capacity in all succeeding places) then a "Fire" button will appear on the transition (see picture above). Clicking on this button will fire the transition, meaning that the necessary tokens will be consumed in preceding "Places" and new ones will be added to the succeeding "Places". In Version 1.1 the "Arcs" have been extended with additional "Ready behaviour" for their following transitions, which allows firing a transition only when certain conditions are met without consuming any tokens. For more information check the "Ready behaviour" attribute information of an "Arc".


Strategy: **random**
Cold trans.: **always fire**
Iterations: **10**
Show log: **no** Delay: **1,0**

To simulate the net a special concept called "Simulation Configurator" is used (see picture above). It contains the configuration for a simulation run and is also used to start the simulation. The configuration is handled through the attributes of the notebook. See the additional information available for each attribute to find out more. The simulation can be started either by using the buttons on the drawing area or the buttons in the notebook. Through them either one iteration, multiple iterations or a slow simulation with delays between each iteration can be run. One iteration tries to fire all ready "Transitions". Should there not be enough tokens to fire all ready "Transitions" (e.g. several transitions requiring a token from a "Place" that only has one) then the selected "Transition conflict strategy" will be employed. With Version 1.3 an additional style of Petri Net simulation has been added through the "Automated Transition Firing" where each iteration selects only one "Transition" to fire (check its help text for more information).

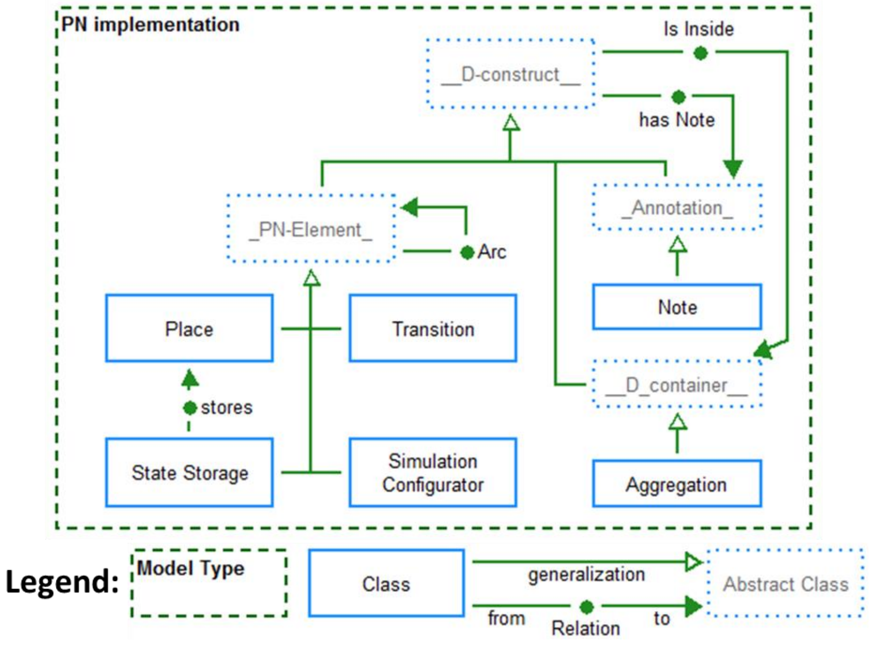
It is also possible to store the current state of a Petri Net and later restore it using "State Storage" (see picture below). In this context the state of the whole net is considered to be the amount of tokens in all the known places. When a "State Storage" is first added to the model it will store the state at that time in its attribute "Storage". This stored state can also be manipulated manually through that attribute. The notebook also provides two buttons: one to store the current state of the model (i.e. update the "State Storage" object with new values) and one to restore the state based on the "State Storage".



State Storage

Version 1.1 also added two Model attributes: "Visualize priorities" and "Visualize probabilities". Selecting those changes the notations of transitions. "Visualize priorities" shows their relative priority in the model with a green bar on the left. "Visualize probabilities" shows a yellow bar on the right of cold "Transitions". Version 1.3 added more Model attributes: "Visualize fire button" which, when selected, will hide the "Fire" buttons in the Petri Net and "Visualize relative frequencies" which will show the relative frequencies specified for the "Transitions" through a purple/violet bar at their left. It also added a "Capacity" attribute to the "Places" that is considered when checking if a "Transitions" is ready.

The picture below provides some detailed information about the implementation of PN in Bee-Up. More specifically it shows an excerpt of how the PN meta-model is implemented. Certain parts are provided by the platform to allow specific functionality, like __D_container__ used to automatically derive "Is Inside" relations.



General information for modelling

- Don’t forget to save (so you are safe from data loss).
- Context menus are available for many things (e.g. objects in the **Modelling area**, entries of the **Explorer window** etc.). Making use of them can make work easier.
- Should a window be gone/missing (e.g. **Explorer window**, **Modelling window** etc.) → They can be toggled on and off through the menu "Window → Tools"
- Most icons have a tool tip, which provide a hint on what an icon is about. In case of the icons of the **Modelling window** the tool tip show the name of the type (e.g. Entity, Relation, has Attribute etc.).
- The tool also provides some functions for convenience. They can be accessed through the **Toolbar** using the icons. From left to right they toggle the functionalities:
  - Align objects on the grid. The grid can be configured through the menu "View → Grid → Settings…"
  - Show the grid.
  - Use the modelling assistant. It supports the creation of new objects and relations from an existing object.
  - Automatically add bend points to relations when creating them to use right angles.
- Notations can contain hyperlinks to other models/objects if the proper attributes are set. For example if a "Class" has the "Referenced class" attribute set, then the visualized name will be based on the referenced class and also a hyperlink to that class.
- The size of the **Drawing area** is represented by the white rectangle with the grey border in the **Modelling area** and can be resized similar to a window. Note that it is automatically extended as needed to fit any new objects that are created or old elements when their position is changed.
- Some model types (e.g. EPC, BPMN) have different modes. Those control which types of objects are available and visualized in the **Modelling area**. They can be changed through the menu "View → Mode"
- Unintentional object access locks are reversible through the menu "View → Object access locks…"
- The tool has certain restrictions due to the things it uses as identifiers and also some limitations:
  - Models are identified through their type and a combination of their name and version ("[name] [version]"). Therefore two ER models, one with the name "Exercise" and version "3" and the other with the name "Exercise 3" are not allowed.
  - Objects in a model are identified through their type and their name. Therefore **no two objects of the same type in the same model can have the same name**. Because of that the "Attribute" in ER models uses "Denomination".
  - Relations in a model are identified by their type, their source object and their target object. Therefore **two relations of the same type linking the same objects in the same direction in the same model cannot exist**.
  - The **source and the target of a relation cannot be the same object**.
  - Relations cannot be the source or the target of other relations.
- To work around the limitations of relations the object type "**Relation Node**" (a small grey circle) is available in all model types:
  - It can be used to create multiple relations of the same type between the same two objects (e.g. several "Message flows" between two "Pools" in a BPMN model) by linking the first object to the "Relation Node" and then the "Relation Node" to the second object (this has to be done for each relation of the same type, between the same two objects, beyond the first direct relation).
  - It can be used to draw relations with the same source and target, by going through the "Relation Node" instead (e.g. when a "Class" is associated with itself). Place the "Relation Node", then draw the relation from the object to the "Relation Node" and then from the "Relation Node" back to the object. Kindly add bend points to the created relations to increase the readability.
  - It allows the use of relations as the source or target of another relation by using the "Relation Node" instead. Freely place the "Relation Node" on an existing relation (e.g. association between two "Classes" in UML) and create the new relation (e.g. "is Associationclass") from/to this "Relation Node" to/from the desired Object (e.g. the third "Class").

Available functionality

Bee-Up comes with many different functionalities out of the box besides modelling. Most of them are shortly described in this section.

Some of the simpler functionalities include:

- Import and export of models in different formats / as images has been described on page 13.
- Queries can be executed on models using the ADOxx Query Language (AQL) through the "Analysis" component in the menu "Analysis" → "Queries/Reports". Details about the AQL can be found on the ADOxx homepage (under "Documentation" → "Query Language AQL").
- Cloning of models as described on page 10.
- Performing various checks on models, identifying some possible errors. Some are performed automatically, others available manually through "Check cardinalities" in the "Model" menu.
- Converting between different types of similar objects through the context menu (right click).
- Directly executing AdoScript code. Available through the "Extras" menu.
- Checking if the latest version of Bee-Up is installed. Available through the "Help" menu.
- Using logging during some of the functionalities. Can be configured for the current session through the "Extras" menu or more persistently by editing its configuration file.

Process simulation

The Bee-Up tool supports the simulation of several process languages (BPMN, EPC, UML Activity Diagrams, Flowcharts) through the functionality provided by the ADOxx platform. It provides quantitative data by pretending to act out a process many times as "simulation runs" and gathering the information while doing so. The approach through simulation allows to handle processes that contain loops, which can technically lead to an infinite number of paths. Three types of simulation are available, two of which are described in more detail later:

- Path analysis – determines the possible paths in a process through several simulation runs and the quantitative data for each of them, including the paths probability, different types of durations and costs based on the values specified in the elements.
- Capacity analysis – determines quantitative data from process models in conjunction with one or several working environment models, finding performers to execute parts of the process. The data also doesn't focus on the possible paths, but on the connection between the process and the performers. In addition to durations and costs from the process the result also provides properties from the working environment model, like capacity workloads or personnel costs.
- Workload analysis – similar to the capacity analysis it works on a set of process models together with one or several working environment models. However, it focuses on analysing the dynamic behaviour of the processes by calculating waiting times using a queue model together with a process calendar.

The path analysis is executed only on a single process, gathering durations (e.g. Execution time, Waiting time) and costs while executing a simulation run. The results from all simulation runs are then collected based on the found paths and grouped together. As the name suggests its main focus is on analysing different paths through a process, but the average results over all paths for the entire simulation are also available. For a path analysis the values from "Execution time", "Waiting time", "Resting time", "Transport time" and "Costs" (all found in the "Time/Costs" tab of the Notebook) are relevant.

The capacity analysis executes on a set of processes and working environments, considering additional properties that result from their conjunction. As such it no longer focuses on determining individual paths, but the results gained from all simulation runs. The results can also be viewed focusing on different aspects over different timespans, like the results for a single process execution (durations, costs), the results for individual performers over a year (capacity, personnel costs) or other parts of the working environment. In addition to the values used in a path analysis it is also important to specify the performer for the active elements (Tasks, Actions etc.). This is achieved through the "Performer" attribute under "Simulation settings" in the Notebook, which expects an AQL which is evaluated during simulation. To simplify specifying the performer it is recommended to first also open the "Working Environment Models" that are to be used and use the alternative editor available (see section Editing attributes). Additionally, some details have to be specified at the starting element, found in the "Simulation settings" tab of the Notebook.

All types of simulations are available through the "Simulation" component

and have certain requirements on the process models:

- Only one starting point is allowed (Start Event, Initial Node, Start Terminal).
- Each path must end in an end-type object (End Event, Final Node, End Terminal).
- "Either or" decisions must be modelled with the corresponding type of element (Exclusive Gateway, XOR operator, Decision/Merge, Decision, Switch) and all outgoing "Subsequent" relations must specify a "Transition condition". The "Transition condition" should either specify a probability of the path as a number or a condition based on "Variable" elements from the model.
- Opening a parallelism must be modelled with the corresponding type of element (Non-exclusive Gateway, Fork, Parallel fork) and must be merged again with the corresponding type of element (Non-exclusive Gateway (converging), Join, Parallel merge). Each opened parallelism must be closed by its own merge.

Note that every time a model is changed, the simulation cache has to be cleared for the changes to be considered by the simulation. This can be achieved through the "Edit" menu ("Free simulation cache") or the corresponding icon on the toolbar.

Model execution

Two types of models can be directly executed in Bee-Up since version 1.3: Petri Nets and Flowcharts. This is achieved through implementing the language's semantics and further extending where appropriate. Details on how to execute Petri Nets have been described on page 24, allowing to fire transitions and move tokens throughout the model manually or automatically. Additionally, the "Transitions" of Petri Nets provide an "Effect" attribute which can specify AdoScript code that is triggered every time the "Transition" fires. This attribute has to be enabled first through the model attribute "Allow effects".

It is possible to create and execute Flowchart models with elements that have AdoScript code deposited in them. We will not focus here on how to create proper Flowcharts, instead presenting the relevant elements and attributes necessary for their execution in the Bee-Up tool.

To create an executable Flowchart it is necessary to have a "Start Terminal", which provides the button to start the execution. It is also possible to have several "Start Terminals" in a single model to cover different cases (e.g. varying ways of collecting input) or to have entirely separate Flowcharts in the same model. For example one "arithmetic operators" model could have several Flowcharts, one for each operator or an "array sorting" model providing Flowcharts for different approaches of sorting arrays. The code to be executed at each "Operation" should be provided in its "Operation code" attribute, using AdoScript and/or the additional keywords made available (only available in Flowcharts, see ℹ of the "Operation code" attribute for more details). The "Decision" uses the "Check expression" attribute to evaluate an expression and continue down the corresponding path. The expression should evaluate to either true (not 0) or false (0) and the outgoing "Subsequent" relations should specify under "Expression result" for which of the two outcomes they are used. The execution continues along the Flowchart until it can no longer find a valid "Subsequent" relation to follow. The "Switch"[12] selects one of several possible follow-up elements by evaluating an expression from the "Evaluate expression" attribute and checking its result with the results from evaluating the "Compare expression" attribute of the outgoing "Subsequent" relations and selecting the first one where both results are equal. The "Subsequent" relations also provide a "Flowchart condition", which is evaluated and if it is not fulfilled then the "Subsequent" relation will be skipped. The "End Terminal" can use its "Ending type" to specify whether it represents a success or a failure.

The separation of code into different function blocks is also possible by putting them in different models, or at least using different "Start Terminals" that can then be called through the "External Operation" element. This element has the "External type" attribute controlling what type of operation should be accessed. More details about the operation have to be provided in the respective tabs through the notebook. One of the

[12] Available by changing the mode to "Extended Flowchart" through the "View → Mode" menu.

possible options here is to execute a Flowchart by referencing a specific "Start Terminal". In this case it is important to consider the specified "Required variables" of the "Start Terminal" and also provide the "Passed Variables" and "Returned Variables" in the "External Operation". An "External Operation" can also execute the different Attribute Profiles described in the next section. Check the additional information ℹ in the corresponding notebooks for the here mentioned elements and attributes for more details.

Code generation

Bee-Up also supports some limited code generation for certain types of models:

- SQL Create statements from ER Models.
- Source code skeletons for Java, C++ and Python from UML Class Diagrams.

The SQL Create statement generation is available through the model attributes of an ER model under "ER properties". It processes the model and writes the necessary SQL code to create the tables in a relational database. Details about the peculiarities of the functionality can also be found in the model attributes under "ER properties", describing how certain specific cases are handled or what the model should contain to create better results.

Source code skeletons (i.e. bare-bones code that has to be fleshed out) can be generated from UML Class Diagrams through the "UML Class Diagram 2 Skeleton" menu item found in the "Model" menu of the "Import/Export" component. It expects a UML Class Diagram to be opened in Bee-Up and allows to select one of the available languages (Java, C++ or Python) and creates the source code in a chosen folder or directly provides it as text in a box (if no folder is chosen). What is currently supported:

- Classes and interfaces (classes with stereotype "interface" or Interface objects) with their visibility (public, private ...) and IsAbstract.
- Properties with most of their information: visibility, name, type, multiplicity (used as is) and default value (assigned as is).
- Operations with most of their information: visibility, name, parameter list (used as is) and return type (used as is).
- Association, Composition and Aggregation are used to add additional properties unless there's indicators that it shouldn't (no role name, direction, not navigable ends etc.).
- Generalization (between two classes or between two interfaces) and realization (between class and interface).

The functionality tries to interpret the model as best as possible, assumes that the model is correct for the selected language and creates the corresponding parts of code. Some examples for the assumptions it makes:

- It assumes that the identifiers (e.g. class-names) only have characters allowed for the selected language.
- It assumes that there are no two variables with the same name or operations with the same signature.
- If for example the Model contains a structure like "Class A --Generalization--> Interface B" then the code for Java will say "class A extends B". The correct version of that should be "Class A --Realization--> Interface B" to create "class A implements B".
- Some parts are taken "as they are", like the parameters of operations or property types, so those have to fit the desired programming language.

There are also certain things that will not be considered or performed:

- Details from "Referenced class" and "Referenced interface" are ignored, only their name is used.
- Packages are ignored.
- Relations that have a Relation Node as the source/target are not handled and might lead to an error.
- Any "internal" dependencies between properties are not considered and are not ordered automatically as needed, e.g. default value of one property depends on value of another property being set beforehand.
- Sophisticated checks, e.g. when inheriting methods from interfaces: has an operation with the same signature already been inherited from somewhere else?

Extending Bee-Up functionality

Bee-Up provides different ways to add custom functionality not directly provided by the tool. It is possible to create and exchange add-ons for Bee-Up through this system (further just called the add-on system and add-ons). It allows to execute user-defined scripts, web-service calls or system applications. However, this add-on system does not allow to extend or change the available types of models, objects, relations and attributes[13] in Bee-Up. What can be done through add-ons are things like user interaction, processing of models, analysing of models and adaptation of models and their data. While it provides a lot of freedom in processing of the available models and data they can unfortunately also be damaged, for example due to bugs in one of the used add-ons. Use any add-ons at your own risk.

Following we will describe how additional functionality can be made available and used in the Bee-Up tool. It is useful, if not even necessary, to have knowledge about AdoScript, the scripting language provided by the ADOxx platform. More about AdoScript can be found in the ADOxx documentation available at https://www.adoxx.org/live/external-coupling-adoxx-functionalty. Additional descriptions of the available interfaces can be found at https://www.adoxx.org/AdoScriptDoc/index.html. Also a lot of the implementation of a specific add-on is handled through attributes available through the notebook of the elements. Please remember that each tab in a notebook can have multiple pages and check the additional information about attributes and elements in Bee-Up that is available through the ℹ button in their corresponding notebook.

Extension through Flowcharts

Since the extension of Bee-Up with executable Flowchart models in version 1.3 it is possible to exchange Flowcharts with other users through the already available export and import means described on page 13. The details on how to create executable Flowcharts have already been described on page 28.

Extension through Attribute Profiles

With Bee-Up 1.4 an additional approach for extending the functionality of Bee-Up is provided through different types of Attribute Profiles. They are like objects that are stored outside of the models and can be referenced by attributes of objects, for example in the "External Operation" element. The following types of Attribute Profiles are available to extend the functionality in Bee-Up:

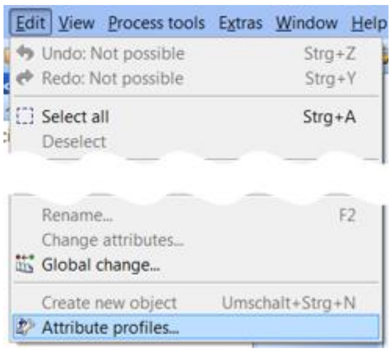
- Call Olive MSC Service – This represents a specific service available through the interface of an Olive Microservice Controller.
- HTTP-called Functionality - This represents a specific functionality provided by a service which can be accessed using HTTP.
- System-called Functionality - This represents a specific functionality provided by the system the tool is running on which can be accessed using a system call (e.g. through command-line).
- Complex Functionality - This represents a complex functionality which is further described through a Flowchart.
- AdoScript Functionality - This represents a specific functionality coded directly as AdoScript. In a sense this is the most powerful one, since it can recreate all the other three types of Attribute Profiles.

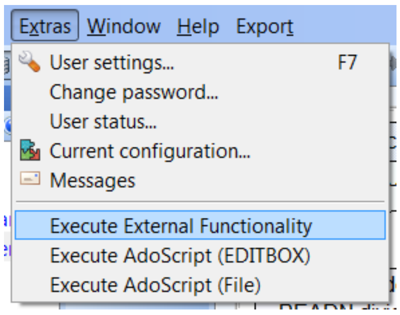
All of them allow to customize how the necessary parameters are determined and how the result should be processed in great detail providing as much freedom as possible for the best user interaction, but also require some solid knowledge of AdoScript to do so. Note that most of the Attribute Profile types do not entirely restrict the granularity of what is accessed / called. However, their instances have to flesh out the details once they are executed so that only a specific function is accessed. For example an Attribute Profile can represent a very specific functionality, like moving a robot arm, or a bunch of functionality from which later on one specific is chosen (e.g. through user interaction), like an Attribute Profile representing the entire robot arm and the user chooses a specific functionality like moving, picking up or resetting.

These Attribute Profiles can be created, viewed and edited through the "Edit → Attribute profiles..." menu item (see picture on the left) the while in the "Modelling" component (left most icon in the Toolbar).

[13] However, (almost) every element has an attribute called "General purpose attribute", which has no specific meaning for Bee-Up itself and can be used by such add-ons.

They can be executed through the "Extras → Execute External Functionality" menu item (see picture on the right).

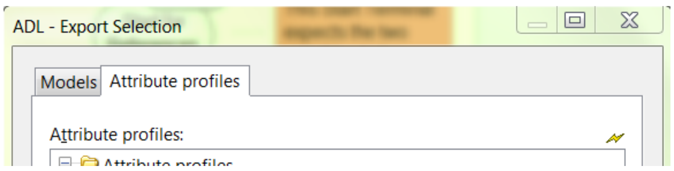




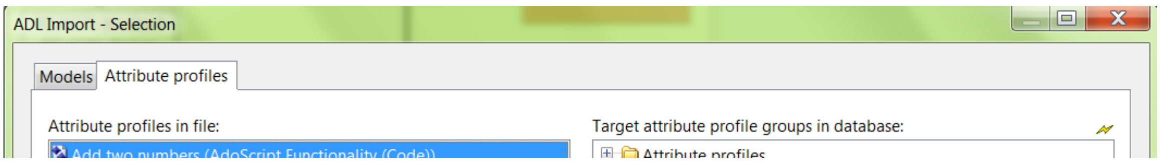
Each Attribute Profile is executed in their own scope and they generally work the same way:

- First the relevant parameters are set and made available as variables. Details about those variables can typically be found in the information text ℹ of the "Pre-processing" attribute.
- Second the code provided in the Pre-processing attribute is executed as is. This can be used for several things, like gathering the necessary input for executing the functionality (e.g. through user interaction, loading from the model etc.) or adapting the available variables and their values. Any further steps are skipped if Pre-processing ends with an error, identified by the ecode variable being not 0.
- Third the main functionality is executed with the previously set parameters. For the AdoScript functionality the map containing links to files (map_asf_filelinks) loads additional values from the "File links" table (unless the ID is already present) after the second step but before executing the code.
- Fourth and last the code provided in the Post-processing attribute is executed as is. This can be used for several things (provide the result to the user, adapt some models based on the provided result etc.) and has access to additional variables. Details about those variables can typically be found in the information text ℹ of the "Post-processing" attribute.

The Attribute Profiles can of course be exchanged with other users through the already available export and import functionality described on page 13. They can be exported together with models in one file, however the selection of Attribute Profiles is performed in a separate tab of the export dialog:



When importing there is also a separate tab to state where the Attribute Profiles from the file should be put in the tool:


ADL Import - Selection
Models
Attribute profiles
Attribute profiles in file:
Add two numbers (AdoScript Functionality (Code))
Target attribute profile groups in database:
Attribute profiles

AutoStart Attribute Profiles

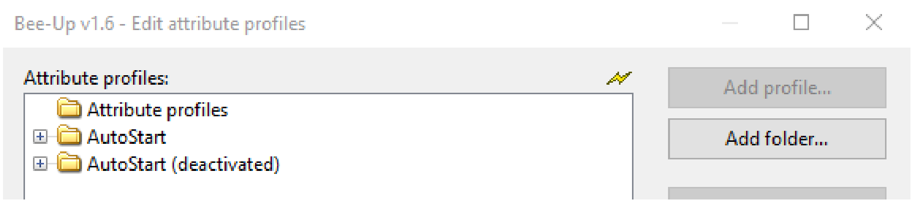
With version 1.6 of Bee-Up an AutoStart feature for Attribute Profiles has been added. Simply put it executes all the executable Attribute Profiles that are found in the "AutoStart" directory on starting the tool. A benefit this provides is the possibility to extend certain GUI elements of the Bee-Up tool through add-ons, for example using INSERT_MENU_ITEM or INSERT_ICON commands of the "Application" message port. Several of Bee-Ups functionalities are now actually added through the AutoStart feature. To deactivate some of those functionalities simply move the corresponding Attribute Profiles to the "AutoStart (deactivated)" directory.

It is also possible to create functionalities that are executed exactly once. The "First time Welcome message" and "Initialize Model Groups" are designed like this. In their case they can be run again by moving them from the "AutoStart (deactivated)" directory to the "AutoStart" directory. Their approach to executing only once calls a procedure in the Post-processing part of the code which moves them to the "AutoStart (deactivated)" directory (see sub-section AutoStart of section Available AdoScript functions for more details).

While it is possible to have dependencies between AutoStart functionalities it is generally discouraged in order to avoid having the user deal with load-order. Should it however be necessary for any reason, e.g. when extending an existing functionality: The Attribute Profiles are executed in a similar order as they are seen in the modelling tool[14].

Since Attribute Profiles used by AutoStart can be added and edited by the users, it is possible that the "AutoStart" can crash the tool (e.g. infinite loop, accessing something that it shouldn't etc.). However, a mechanism is in place to recover from such a case. If the tool crashes due to automatically executing an Attribute Profile then the "AutoStart" is disabled the next time the tool is started, providing several options to fix the issue: skip the AutoStart, view a log of successfully executed Attribute Profiles or disable AutoStart all together.

The following picture shows the two necessary groups (directories) for the "AutoStart" to work and the default "Attribute profiles" directory.



AdoScript Remote Execution

Bee-Up comes with an interface which allows its use with a deployed AdoScript Remote Execution (ASRE) server. The details about ASRE can be obtained from its project page. Simplified it allows to use a running Bee-Up instance as a remote AdoScript execution environment, accessing its models, processing AdoScript code etc. Conceptually it works by managing the jobs through the ASRE server and Bee-Up regularly checks the server if any new jobs are posted for it, and if so, it executes them and posts the result to the server. The ASRE functionality can be turned on in Bee-Up and temporarily configured by using the corresponding executable Attribute Profiles (through the "Extras" → "Execute External Functionality" menu). A more persistent setup of the ASRE configuration can be achieved through the corresponding configuration file (see section Editing configuration files). The relevant parameters are:

- serverUrl – specifies the URL where the ASRE API can be accessed under. Typically ends with the text "/AdoScriptRemoteExecution/api/".
- instanceToken – specifies the unique token the Bee-Up instance uses on the server. This token should also be used when posting jobs to the ASRE server.
- delay – specifies the minimum delay in milliseconds between checking the server for new jobs.

Handling large amounts of code

Depending on the complexity of the task to be solved by an add-on the amount of necessary code can vary. However, the attributes in the notebook where the code is provided only allow for a limited amount of characters to be stored, typically 32000 characters. There are however different approaches available to deal with long AdoScript code, which we will describe in the following.

In some cases it is possible and probably also more feasible to split the code into different objects, like separate operations in a Flowchart and thus evade the character limit. AdoScript also allows to define your own functions / procedures which can be called wherever necessary and thus help to reuse code (see the previously mentioned AdoScript documentation for more on procedures). Since the "External Operation" allows to execute other Flowcharts it is also an option to split the code into different Flowcharts for reuse.

[14] It is character based. Simplified: "1" before "11" before "2" before "A" before "a" before "B" before "b". Avoid relying on other characters to influence the order, since the internal sorting can deviate from what is seen in the modelling tool.

If splitting the code is not an option then the code could also be stored in a file and then loaded and executed in AdoScript. This can be achieved by using the EXECUTE command (to run the code from the file). The necessary files could be stored in the installation folder of Bee-Up, preferably in their own sub-folder. When a relative file path is specified for the EXECUTE command then it is considered relative to the current working directory folder. Note that relative file paths can cause a problem when used on installations for Linux or macOS, since the current working directory there is different than on Windows. To create add-ons that also work on Linux and macOS determine the tools path using the CC "Application" GET_PATH command instead of a relative path.

An extension is available for Visual Studio Code to help with writing AdoScript. More information about that can be found at: https://marketplace.visualstudio.com/items?itemName=ADOxxorg.adoxx-adoscript. Alternatively an AdoScript syntax highlighting for Notepad++ is available under the following link: https://www.adoxx.org/live/adoxx-development-languages-syntax-support

Editing configuration files

Many of the configurations, especially the ones used by new functionalities, are now stored in the internal database. To allow editing the configuration a special menu item "Edit internal configuration file" is available through the "Extras" menu. It allows to select one of the configuration files and edit it in the Bee-Up tool. The format of the configuration files is not strictly defined, so each functionality or add-on can use a format that suits best. The typical format used by functionalities provided by Bee-Up itself is a simple property file syntax:

- Configurations use simple key-value pairs.
- A line can contain one key-value pair.
- The key and the value are separated by an equals sign (=).
- Keys are text (strings), while values can also be different types.
- The characters # or ! can be used at the start of a line to indicate a comment until the end of the line.

Note that configurations are often loaded on starting the Bee-Up tool, so for changes to a configuration file to take effect a restart of Bee-Up can be necessary.

Checking global variables, functions and procedures

Bee-Up comes with a lot of internal variables, functions and procedures that are defined on the global scope. Creating extensions can run into the problem of overwriting the existing global variables, functions or procedures, rendering certain functions unavailable or broken. To mitigate this several procedures are provided to check if certain identifiers are already in use and catch any problems before they arise. They also allow to register own identifiers so other extensions can check if they are not competing with one another. Details about the procedures can be found in the section Extension management.

Handling cases where procedures are already available or not is up to the extension developer. The way AdoScript is implemented it is not possible to directly define a function or procedure on a conditional basis, e.g. in an IF. However, it is possible to place the definition of functions and procedures in a file and only load that file (using EXECUTE) if certain conditions are met, i.e. loading it in an IF. Also to ensure that other extension don't overwrite your used global identifiers it is recommended to register them using the corresponding procedures.

In general it is recommended to use some prefix for the identifiers variables, functions and procedures developed in the extension to reduce the chance of using the same identifiers as the others.

Available AdoScript functions and procedures

The recent updates to Bee-Up allow to further extend the functionalities of the tool using AdoScript. To facilitate the development of own add-ons for Bee-Up some helpful functions and procedures are made available by the tool and are shortly described in this section. Note that these do not list all of the available functions and procedures, nor does it contain any functions or commands directly provided by AdoScript. These functionalities have also been developed over several years, so expect differences, especially between different categories, e.g. some procedures show errors through GUI elements, some set the ecode to not zero, some recover to default values.

AdoScript Remote Execution

While the implementation in Bee-Up is set up to use a configuration to work with the AdoScript Remote Execution (ASRE), some of the following functions and procedures might be useful in other cases. For more details about ASRE check its documentation

- `asre_checkTimer(lastExec:integer, delay:integer)` – Returns true if enough time passed from the last execution based on the delay or false otherwise. The inputs should be in milli seconds, and lastExec should be in the relation to the same reference time as used by getTickCount().
- `ASRE_INIT_CONFIG configFile:string asreConfig:reference` – Initializes a default ASRE configuration map, updating its values based on the provided file-path (if available) and returns it through the asreConfig parameter. See section Editing configuration files for details on the supported format.
- ASRE_POLL_SERVER_ONCE scriptServer:string instanceToken:string – Checks the specified ASRE server once if there is anything to execute for the provided instance token and if so then executes it and returns the result. The server should be a valid URL to where the ASRE API is deployed, most often ending in " AdoScriptRemoteExecution/api/".

AutoStart

While the implementation in Bee-Up is set up to use a working configuration for AutoStart, some of its functions and procedures can also be useful for other functionalities. The default global variable containing the configuration is called g_autostart_config. If more control over the execution of Attribute Profiles is necessary, then check the section Executable attribute profiles.

- `AUTO_INIT_CONFIG configFile:string autostartConfig:reference` – Initializes a default AutoStart configuration map, updating its values based on the provided file-path (if available) and returns it through the autostartConfig parameter. See section Editing configuration files for details on the supported format.
- `AUTO_DEACTIVATE_AUTOSTART_PROFILE attrprofId:integer` – Moves the specified attribute profile into the "deactivated" directory for the default AutoStart configuration.
- `AUTO_GET_DIRECTORY dirName:string autostartDirId:reference` – Places the ID of the top-level directory with the specified name or 0 if it doesn't exist.
- `AUTO_ENSURE_DIRECTORY dirName:string autostartDirId:reference` – Places the ID of the top-level directory with the specified name. If the directory doesn't exist then it will be crated.
- `AUTO_GET_AUTOSTART_PROFILES autostartConfig:map attrprofIds:reference` – Finds all the attribute profiles according to the provided AutoStart configuration and places their IDs as a string list in attrprofIds.
- `AUTO_DEACTIVATE_PROFILE autostartConfig:map attrprofId:integer` – Moves the specified attribute profile into the "deactivated" directory for the provided AutoStart configuration.
- `AUTO_EXECUTE_MULTIPLE_PROFILES autostartConfig:map attrprofIds:string` – Runs the specified executable Attribute Profiles according to the provided configuration. Safety measures are circumvented when using this procedure.
- `AUTO_EXECUTE_AUTOSTART_PROFILES autostartConfig:map` – Runs multiple executable Attribute Profiles according to the provided configuration. Performs the typical AutoStart safety measures.
- `AUTO_RE_ENABLE_AUTOSTART autostartConfig:map` – Re-enables the AutoStart feature if it has been disabled due to some safety measures.

Cloning

Functions and procedures for cloning individual or multiple elements (models, objects etc.)

- `CLN_CLONE_MODEL_SET (string:str_modelids) id_targetmodgrp:integer str_suffix:string map_modelclones:reference` – Creates clones for the models specified as the main parameter in the target model group with the provided suffix attached at the end. Interrefs between elements of the clones are adapted to target the clones if possible. A map is placed in map_modelclones which contains the IDs of the original models as keys and their values are the ID of the clone.

- `CLN_CLONE_MODEL (integer:id_model) id_targetmodgrp:integer str_suffix:string id_modelclone:reference` – Creates a clone of a single model specified as the main parameter in the target model group with the provided suffices attached at the end. Interrefs between elements of the clones are adapted to target the clones if possible. The ID of the clone is placed in id_modelclone.
- `CLN_CLONE_ATTRPROF integer:id_attrprof id_attrprofdir:integer str_suffix:string id_attrprofclone:reference` – Creates a clone of a single attribute profile specified as the main parameter in the target directory with the provided suffix attached at the end. The ID of the clone is placed in id_attrprofclone.
- `CLN_CLONE_INSTANCE (integer:id_object) id_targetmodel:integer id_objectclone:reference` – Creates a clone of a single modelling instance specified as the main parameter in the target model. The ID of the clone is placed in id_objectclone.
- `CLN_CLONE_RELATION (integer:id_relation) id_targetmodel:integer id_relationclone:reference` – Creates a clone of a single relation specified as the main parameter in the target model. The source and target objects must exist in the target model with the same name. The ID of the clone is placed in id_relationclone.
- `CLN_COPY_ATTR_VAL (integer:id_attr) id_source:integer id_target:integer` – Copies the attribute value provided as the main parameter from the source (model, object, etc.) to the garget (model, object etc.).
- `CLN_BEND_IREFS integer:id_object id_attr:integer map_modelclones:map` – Changes the interrefs of the provided object (main parameter) in the specified attribute to target objects/models from the clones instead if possible. The map_modelclones should contain the IDs of the original models as keys and their values should be the ID of their clone.
- `CLN_MOVE_ATTRPROF integer:id_attrprof id_attprofdir:integer` -
- `CLN_UI_CLONE_ACTIVE_MODEL` – Uses GUI elements to clone the currently active model.
- `CLN_UI_CLONE_SELECT_MODEL_SET` – Uses GUI elements to clone a set of models selected by the user.

Configuration files

Functions and procedures to simplify the use of configuration files through the use of ADOxx maps. Configuration files are assumed to be simple property files which contain key-value pairs, each pair in a separate line and the key and value separated by an equals sign (=). Comments are possible for an entire line when starting with an exclamation mark (!) or a hash/pound sign (#).

- `lam_cf_kvToConfigString(key, value)` – Creates the string representing the key-value pair in the configuration file. Both key and value can be of any type.
- `CF_CONFIG_STRING_TO_MAP configString:string configMap:reference` – Transforms a configuration text, e.g. from a configuration file, to an ADOxx map structure where keys are always of type string and the values use the best fitting type available. The result is placed in configMap.
- `CF_MAP_TO_CONFIG_STRING configMap:map configString:reference` – Transforms a map to the corresponding configuration text. The result is placed in configString.
- `CF_UPDATE_CONFIG_STRING configString:reference configMap:map strict:integer` – Updates the provided configuration text (provided through configString) with values from the map, replacing values where possible. If strict is not 0 then any key-value pairs found only in the configuration text will be commented out. Any comments that were part of the configuration text are kept. Also any key-value pairs in the map that would become a comment, i.e. when the key starts with an exclamation mark or a hash/pound sign, are skipped. The updated result is found in configString.
- `CF_LOAD_CONFIG_FROM_FILE configFile:string configMap:reference` – Loads the configuration from a file-path into the configMap reference.
- `CF_SAVE_CONFIG_TO_FILE configFile:string configMap:map` – Saves the configuration from the provided map in the specified file-path. If the file already exists then the values are updated, otherwise the file is created when possible.
- `CF_EDIT_CONFIGFILE_GUI configFile:string configMap:map` – Opens a GUI interface in ADOxx to edit the configuration from the specified file-path. The configuration map is used to update any values in the configuration text before it is presented to the user. If the file doesn't exist then this tries to create it.

Conversion

Functions and procedures for converting between different datatypes, escaping characters or similar.

- `lam_toString(input)` – The input can be of any type and returns the input as a string. Useful to ensure that something is provided as a string.
- `lam_toBoolean(input)` – The input can be of any type and returns a value that can be interpreted as a boolean.
- `lam_toNumber(input)` – The input can be of any type and returns a number if the input represents a number or returns undefined otherwise.
- `toNumber(input:string)` – Returns a number if the input represents a number or returns undefined otherwise.
- `toBestFittingType(input:string)` – Analyses the input and returns it as the best fitting data type. For example: if the input looks like a number then the number is returned; if the input looks like an array then the array is returned etc.
- `escapeStringUriStyle(input:string, allowedCharacters:string)` – Escapes all characters in the input that are not found in the list of allowedCharacters. The allowedCharacters should simply be a string containing all the characters that don't have to be escaped, e.g. for only keeping simple alpha-numeric characters "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".
- `escapeStringForPath(input:string)` – replaces all characters in the input that are not allowed as part of a file-path with a hyphen (-).
- `decToHex(input:integer)` – Transforms a decimal integer into a hexadecimal value. The returned value is a string.
- `hexToDec(input:string)` – Transforms a hexadecimal value to a decimal integer. The returned value is an integer.
- `lam_toJson(input)` – Returns a string with the best possible JSON representation of the input.
- `mapToJson(input:map)` – Returns a string with the JSON representation of the ADOxx map. ADOxx maps are considered to be the same as JSON objects. All the keys of the map will be transformed into strings, which can lead to losing some values. Other types are transformed as best as possible.
- `arrayToJson(input:array)` – Returns a string with the JSON representation of the ADOxx array. ADOxx maps are considered to be the same as JSON objects. All the keys of the map will be transformed into strings, which can lead to losing some values. Other types are transformed as best as possible.
- `fromJson(input:string)` – Returns an ADOxx value that best represents the provided JSON input. The mapping between the different datatypes is: JSON object → ADOxx map, JSON array → ADOxx array, JSON string → ADOxx string, simple JSON number → ADOxx integer or real, JSON true → 1, JSON false → 0, JSON null → "null". JSON numbers with an exponent notation will be stored as strings.
- `escapeStringForSQL(input:string)` – Function used by the "Create SQL Statements" functionality to escape invalid characters in names.

Date / time

These functions and procedures allow handling strings in ISO 8601 date and time formats. The supported format for a date and time is YYYY-MM-DDThh:mm:ss.sss and can also contain the optional time zone designator. Functions that use only the date or the time expect the corresponding parts. The separators (T, :, -) are more important than the exact amount of characters. Most functions also work if only parts of the date and/or time are provided, assuming default values for anything that is missing from the right side, e.g. 2021-08T14Z is considered to be 14:00:00 on the 1st of August in 2021. There are also functions available that work directly on map representations of dates/times. These follow the same naming structure, except instead of "…iso[Time/Date/DateTime]String…" they just use "[Time/Date/DateTime]".

- `timeToMap(time:string)` – Returns a map representation of the given time, including "hours", "minutes" and "seconds" as numbers and "timezone" as a string.
- `mapToIsoTime(timeMap:map)` – Returns an ISO 8601 string representation of the given map representing a time. It expects the map to have values for "hours", "minutes" and "seconds" as numbers and "timezone" as a string, otherwise it will use 0 / empty values.

- isoDateToMap(date:string) – Returns a map representation of the given date, including "years", "monthOfYear" and "dayOfMonth" as numbers.
- mapToIsoDate(dateMap:map) – Returns an ISO 8601 string representation of the given map representing a date. It expecets the map to have values for "years", "monthOfYear" and "dayOfMonth", otherwise it will use either 0 (for year) or 1 (for day and month).
- isoDateTimeToMap(datetime:string) – Returns a map representation of the given date and time, including "years", "monthOfYear", "dayOfMonth", "hours", "minutes" and "seconds" as numbers and "timezone" as a string.
- mapToIsoDateTime(datetimeMap:map) – Returns an ISO 8601 string representation of the given map representing a date and time. It expects the map to have values for "years", "monthOfYear", "dayOfMonth", "hours", "minutes" and "seconds" as numbers and "timezone" as a string, otherwise it will use the value 1 for "monthOfYear" and "dayOfYear", an empty value for "timezone" or 0 for any other missing part.
- applyOffsetToIsoTimeString(time:string, offset:string) – Returns the result of applying a time duration (hh:mm:ss.sss) to a time. The offset should start with either a + or - to indicate whether to add or subtract. The result can be negative or have hours larger than 23 to preserve moving to a different date.
- normalizeIsoTimeStringToUtc(time:string) – Returns the corresponding time in the UTC time zone for the provided input considering time zones.
- compareIsoTimeStrings(time1:string, time2:string) – Returns a positive number if time1 is larger, a negative number if time2 is larger or zero if both times are the same. Accounts for different time zones.
- isoDateToDays(date:string) – Returns the amount of days the date is compared to the reference date of 1970-01-01. Returns a positive number for dates after the reference date, zero for the reference date or a negative number for dates before the reference date. Accounts for leap years.
- daysToIsoDate(days:integer) – Returns a string with the date based on the amount of specified days in relation to the reference date of 1970-01-01. The input can be a positive (after) or negative (before) number. Using this function with a parameter of 0 will return "1970-01-01".
- isLeapYear(date:string) – Returns true if the year of the provided date is a leap year, or otherwise false.
- applyOffsetToIsoDateString(date:string, offset:string) – Returns the result of applying a duration in the format of YYYY-DDD to the date. The offset should start with either a + or – to indicate whether to add or subtract. A year in the offset is always considered 365 days.
- compareIsoDateStrings(date1:string, date2:string) – Returns a positive number if date1 is later, a negative number if date2 is later or zero if both dates are the same.
- normalizeIsoDateTimeStringToUtc(datetime:string) – Returns the corresponding date and time in the UTC time zone for the provided input considering time zones.
- compareIsoDateTimeStrings(datetime1:string, datetime2:string) – Returns a positive number if datetime1 is later, a negative number if datetime2 is later or zero if both are the same. Accounts for time zones.
- applyAdoxxTimeToIsoDateTimeString(datetime:string, duration:time) – Returns the result of applying a duration that is of the ADOxx type "time" (YY:DDD:hh:mm:ss) to the provided date and time. The function also works with a date or time alone by leaving the parts to the left or right side of "T" empty.

Executable attribute profiles

Functions and procedures to run the different types of executable Attribute Profiles available in Bee-Up. In general there are two procedures for each type of executable Attribute Profile provided: 1) a simple one which uses the data provided in the Attribute Profile and 2) an override version which allows to change the behaviour and parameters used during the execution. The 2nd versions are used for example when triggered from a Flowchart.

- EXECAP_EXECUTE_ATTRPROF_GUI – Opens the "Execute Attribute Profile" GUI and runs the selected Attribute Profile.

- `EXECAP_EXECUTE_ATTRPROF (integer:id_functionobj)` – Runs the specified Attribute Profile if possible.
- `EXECAP_HTTP_CALL (integer:id_functionobj) val_httpcode:reference map_respheaders:reference str_respbody:reference` – Executes an "HTTP call" type Attribute Profile whose ID is provided as the main parameter. The HTTP response code, headers and body are provided through the reference parameters.
- `EXECAP_HTTP_CALL_OVERRIDE (integer:id_functionobj) val_skipprecode:integer val_skippostcode:integer str_endpoint:string str_method:string str_auth:string str_username:string str_password:string map_headers:map str_body:string val_httpcode:reference map_respheaders:reference str_respbody:reference` – A version of EXECAP_HTTP_CALL that allows controlling many different parameters, which can also be influenced by the pre-processing code:
  - when val_skipprecode is not 0 then the "Pre-processing" code is skipped.
  - when val_skippostcode is not 0 then the "Post-processing" code is skipped.
  - the values for str_endpoint, str_method, str_username and str_password can be directly specified or use "" to get the values from the Attribute Profile.
  - str_auth can be specified as "Basic" to use basic authentication, or no authentication in any other case.
  - map_headers and str_body specifying the headers and body of the request.
  - NOTE: This procedure and its parameters CAN change in the future!
- `EXECAP_OLIVE_SERVICE_CALL (integer:id_functionobj) any_serviceresponse:reference` – Executes an "Olive operation call" type Attribute Profile whose ID is provided as the main parameter. The result is provided through the reference parameter.
- `EXECAP_OLIVE_SERVICE_CALL_OVERRIDE (integer:id_functionobj) val_skipprecode:integer val_skippostcode:integer str_endpoint:string str_microserviceid:string str_microservicename:string str_operationid:string val_usesimpleinput:integer map_input:map any_serviceresponse:reference` – A version of EXECAP_OLIVE_SERVICE_CALL. That allows controlling many different parameters, which can be also influenced by the pre-processing code:
  - when val_skipprecode is not 0 then the "Pre-processing" code is skipped.
  - when val_skippostcode is not 0 then the "Post-processing" code is skipped.
  - the values for str_endpoint, str_microserviceid, str_microservicename and str_operationid can be directly specified or use "" to get the values from the Attribute Profile.
  - Val_usesimpleinput can be specified as 1 (true), 0 (false) or -1 to use the setting from the Attribute Profile.
  - map_input specifies the specifying the headers and body of the request.
  - NOTE: This procedure and its parameters CAN change in the future!
- `EXECAP_SYSTEM_CALL (integer:id_functionobj)` – Executes a "System call" type Attribute Profile whose ID is provided as the main parameter.
- `EXECAP_SYSTEM_CALL_OVERRIDE (integer:id_functionobj) val_skipprecode:integer val_skippostcode:integer str_command:string str_commandtype:string val_showcmd:integer map_sys_inputvars:map str_sys_inputvarskeys:string map_sys_outputvars:reference str_sys_output:reference str_sys_errors:reference` – A version of EXECAP_SYSTEM_CALL that allows controlling many different parameters, which can also be influenced during the pre-processing code:
  - when val_skipprecode is not 0 then the "Pre-processing" code is skipped.
  - when val_skippostcode is not 0 then the "Post-processing" code is skipped.
  - the values for str_command and str_commandtype (Synchronous CMD, Asynchronous CMD or Application) can be directly specified or use "" to get the values from the Attribute Profile.
  - when val_showcmd is set to 0 then the console window will not be shown and values smaller than 0 will use the configuration from the Attribute Profile.
  - through map_sys_inputvars the variables and their values available for the arguments can be provided. The variable names should be the keys and str_sys_inputvarskeys should be a string list (separated by "~") containing all the keys of map_sys_inputvars.

  - through map_sys_outputvars some of the variables after post-processing can be retrieved. The value for each key is evaluated as an expression and stored as the new value in the map.
  - in str_sys_output and str_sys_errors the content written to the standard output and error streams is captured if possible.
  - NOTE: This procedure and its parameters CAN change in the future!
- `EXECAP_COMPLEX_CALL (integer:id_functionobj)` – Executes a "Complex" type Attribute Profile whose ID is provided as the main parameter.
- `EXECAP_COMPLEX_CALL_OVERRIDE (integer:id_functionobj) val_skipprecode:integer val_skippostcode:integer val_comp_execdepth:integer id_comp_start:integer val_comp_delay:real val_comp_output:integer val_comp_debug:integer map_comp_inputvars:map str_comp_inputvarskeys:string map_comp_outputvars:reference str_comp_stdout:reference str_comp_debout:reference` – A version of EXECAP_COMPLEX_CALL that allows controlling many different parameters, which can also be influenced during the pre-processing code:
  - when val_skipprecode is not 0 then the "Pre-processing" code is skipped.
  - when val_skippostcode is not 0 then the "Post-processing" code is skipped.
  - the val_comp_execdepth should be set to 1. It specifies how deep the execution currently is in case of recursive calls.
  - through id_comp_start the ID of the Start Terminal or smaller than 0 to get the value from the Attribute Profile.
  - through val_comp_delay, val_comp_output and val_comp_debug the delay, output behaviour and debug behaviour can be overridden from what is specified in the Flowchart.
  - through map_comp_inputvars the input variables and their values available for the execution from the Start Terminal can be provided. The variable names should be the keys and str_comp_inputvarskeys should be a string list (separated by "~") containing all the keys of map_comp_inputvars.
  - through map_comp_outputvars some of the variables after post-processing can be retrieved. The value for each key is evaluated as an expression and stored as the new value in the map.
  - in str_comp_stdout and str_comp_debout the content written to the standard output and debug output is captured if possible.
  - NOTE: This procedure and its parameters CAN change in the future!
- `EXECAP_ADOSCRIPT_CALL (integer:id_functionobj)` – Executes an "AdoScript" type Attribute Profile whose ID is provided as the main parameter.
- `EXECAP_ADOSCRIPT_CALL_OVERRIDE (integer:id_functionobj) val_skipprecode:integer val_skippostcode:integer map_asf_filelinks:map map_asf_inputvars:map str_asf_inputvarskeys:string map_asf_outputvars:reference` – A version of EXECAP_ADOSCRIPT_CALL that allows controlling many different parameters, which can also be influenced during the pre-processing code:
  - when val_skipprecode is not 0 then the "Pre-processing" code is skipped.
  - when val_skippostcode is not 0 then the "Post-processing" code is skipped.
  - the map_asf_filelinks allows to override the values for file links. Its structure should be similar to the File links table from the Attribute Profile: the keys should specify the ID column and their value the File column.
  - through map_asf_inputvars the input variables and their values available for the execution can be provided. The variable names should be the keys and str_asf_inputvarskeys should be a string list (separated by "~") containing all the keys of map_asf_inputvars.
  - through map_asf_outputvars some of the variables after post-processing can be retrieved. The value for each key is evaluated as an expression and stored as the new value in the map.
  - NOTE: This procedure and its parameters CAN change in the future!

Executable flowcharts

Functions and procedures to execute the Flowcharts in Bee-Up.

- `FC_EXEC_FC_FROM_START (integer:id_start) int_userinput:integer map_fc_outputvars:reference str_fc_stdout:reference str_fc_debout:reference` – Executes a Flowchart starting from the Start Terminal ID provided through the main parameter. When int_userinput evaluates to true, then the execution will prompt the user for every necessary input variable. The map_fc_outputvars can specify a map with values containing simple expressions to be evaluated and placed back in the map. Also any of the specified "Returned variables" of the Start Terminal are placed in the map as well. The std_fc_stdout and str_fc_debout references can specify a string variable to which any outputs written to the standard output (through PRINT/LN) or debug output are appended.

- `FC_EXEC_FC (integer:fc_id_startelement) val_fc_delay:real val_fc_output:integer val_fc_debug:integer val_fc_execdepth:integer val_fc_printtimes:integer val_fc_highlight:integer map_fc_inputvars:map str_fc_inputvarskeys:string map_fc_outputvars:reference str_fc_stdout:reference str_fc_debout:reference` – Executes a Flowchart starting from the object ID provided through the main parameter. While this can technically start at any object in the Flowchart, it means that some necessary variables are not set, due to skipping code that would otherwise be executed. The map_fc_outputvars, str_fc_stdout and str_fc_debout are handled the same as in FC_EXEC_FC_FROM_START. The other variables control the following parameters:
  - through val_fc_delay the delay between switching from one element to the next during the execution can be specified.
  - the standard output and debug output behaviour is controlled through val_fc_output and val_fc_debug: -1 will close the output window and not use it, 0 will not change the output window in any way, 1 will use the output window if it already exists, 2 will reuse the available output window or create a new one if necessary and 3 will reset the output window and create it if necessary.
  - the val_fc_execdepth specifies the depth of the execution, which mainly influences the values printed in the debug output.
  - if val_fc_printtimes is true then Start Terminals and End Terminals will print a timestamps to the standard output.
  - if val_fc_highlight is true then the currently processed element is highlighted in the Flowchart during execution.
  - through map_fc_inputvars the variables and their values available for the execution can be provided. The variable names should be the keys and str_fc_inputvarskeys should be a string list (separated by "~") containing all the keys of map_fc_inputvars.

Extension management

Functions and procedures that help managing extensions, most notably support checking if any extensions are competing for identifiers (global variables, functions, procedures) with one another. Note that only identifiers that are registered are checked by these procedures. Most of the procedures also set an ecode when checking or registering if the identifier is already known.

- `EM_CHECK_GLOBAL_VARIABLE (string:variableName) exists:reference` – Checks if the name is already used for a global variable and places the result in the exists reference.
- `EM_CHECK_GLOBAL_VARIABLES (string:variableNames) existingVariables:reference` – Checks if the names provided as the string list are already used by a global variable and places any names that are already in use in the provided reference.
- `EM_REGISTER_GLOBAL_VARIABLE (string:variableName)` – Registers a global variable name if possible, making it known to the extension management. If registration fails, e.g. because the name is already in use, then ecode will be set to true.
- `EM_REGISTER_GLOBAL_VARIABLES (string:variableNames) failedRegistrations:reference` – Registers several global variable names if possible, making them known to the extension management. If registration for any of them fails, e.g. because the name is already in use, then it will be added to the failed registrations and ecode will be set to true.
- `EM_LIST_GLOBAL_VARIABLES globalVariables:reference` – Places a list of all known global variable names in the provided reference parameter as a string list.
- `EM_CHECK_GLOBAL_FUNCTION (string:functionName) exists:reference` – Checks if the name is already used for a global function and places the result in the exists reference.

- EM_CHECK_GLOBAL_FUNCTIONS (string:functionNames) existingFunctions:reference – Checks if the names provided as the string list are already used by a global procedure and places any names that are already in use in the provided reference.
- EM_REGISTER_GLOBAL_FUNCTION (string:functionName) – Registers a global function name if possible, making it known to the extension management. If registration fails, e.g. because the name is already in use, then ecode will be set to true.
- EM_REGISTER_GLOBAL_FUNCTIONS (string:functionNames) failedRegistrations:reference – Registers several global function names if possible, making them known to the extension management. If registration for any of them fails, e.g. because the name is already in use, then it will be added to the failed registrations and ecode will be set to true.
- EM_LIST_GLOBAL_FUNCTIONS globalFunctions:reference – Places a list of all known global function names in the provided reference parameter as a string list.
- EM_CHECK_GLOBAL_PROCEDURE (string:procedureName) exists:reference – Checks if the name is already used for a global procedure and places the result in the exists reference.
- EM_CHECK_GLOBAL_PROCEDURES (string:procedureNames) existingProcedures:reference – Checks if the names provided as the string list are already used by a global procedure and places any names that are already in use in the provided reference.
- EM_REGISTER_GLOBAL_PROCEDURE (string:procedureName) – Registers a global procedure name if possible, making it known to the extension management. If registration fails, e.g. because the name is already in use, then ecode will be set to true.
- EM_REGISTER_GLOBAL_PROCEDURES (string:procedureNames) failedRegistrations:reference – Registers several global procedure names if possible, making them known to the extension management. If registration for any of them fails, e.g. because the name is already in use, then it will be added to the failed registrations and ecode will be set to true.
- EM_LIST_GLOBAL_PROCEDURES globalProcedures:reference – Places a list of all known global procedure names in the provided reference parameter as a string list.

HTTP/S Requests

Starting from Bee-Up 1.7 use the corresponding AdoScript command to perform HTTP requests:

```
CC "AdoScript" HTTP_REQUEST (strValue) [ method: strValue ]
    [ username: strValue ] [ password: strValue ]
    [ reqheader: mapValue ] [ reqbody: strValue ]
    [ base64-request: boolValue ] [ binary-request: boolValue ]
    [ base64: boolValue ] [ binary: boolValue ]
    [ certver-off: boolValue ] [ timeouts:tokenStr ] [ tries:intValue ]
    [ proxy:strValue ] [ proxyuser:strValue ] [ proxypw:strValue ]
```

Inputs:

- main parameter (strValue) - The URL to send the request to.
- method (strValue, optional) - The HTTP method / verb for the request, e.g. "GET", "POST", "PUT" etc. Should be specified in upper case. The default is "GET".
- username (strValue, optional) - The user name for basic authentication. The default is "".
- password (strValue, optional) - The password for basic authentication. The default is "".
- reqheader (mapValue, optional) - Additional headers to send with the request. The default is an empty map.
- reqbody (strValue, optional) - The body to be sent with the request. The default is "".
- base64-request (boolValue, optional) - When true, then the request body is first decoded using base64 before it is sent. The default is false.
- binary-request (boolValue, optional) - When true, then the request body is treated as binary data instead of text. The default is false.
- base64 (boolValue, optional) - When true, then the response body is encoded using base64 before it is returned. The default is false.
- binary (boolValue, optional) - When true, then the response body is treated as binary data instead of text. The default is false.

- `certver-off` (boolValue, optional) - When true, then verification of certificates (HTTPS) is turned off. The default is false.
- `timeouts` (tokenStr, optional) - Specifies the total timeout and the connection timeout for the request in milliseconds. The two values should be separated with a single white space. The default value is "90000 60000".
- `tries` (intValue, optional) - Set the maximum number of tries to perform the request. The default is 1.
- `proxy` (strValue, optional) - The proxy server to use. An empty value "" means to not use any proxy. The default is "".
- `proxyuser` (strValue, optional) - The user name to use with the proxy. The default is "".
- `proxypw` (strValue, optional) - The password to use with the proxy. The default is "".

Results:

- `ecode` (intValue) - Contains the error code or 0 in case of success. This only covers errors whether the request could be performed and a response received. Even if it is 0, the response can have an HTTP status code of 404, 500 etc.
- `errmsg` (strValue) - When ecode is non-0, then this contains a short description of the encountered error.
- `statuscode` (intValue) - The HTTP response status code, e.g. 200, 404 etc.
- `header` (mapValue) - The headers from the response.
- `response` (strValue) - The body from the response.
- `host` (strValue) - The "host" part of the URL provided as input.

A note on boolValue: AdoScript does not have a specific bool type. Instead, the other types (numbers, strings etc.) are interpreted as bool true (not 0, …) or bool false (0, any type of string, …).

Most of the old procedures are still available in Bee-Up for backwards compatibility, but they are considered deprecated (their further use is discouraged, and they can be removed in future versions). The old HTTP procedures that have the name "BYTES" in them are no longer supported.

Logging

Functions and procedures that help with logging certain events in different ways. Many of the procedures use a loggerConfig, where the global logger stored in the variable g_log_globalLogger can be used as the parameter. There are five types of events (with their level number in parentheses): fatal errors (1), errors (2), warnings (3), information (4) and debug (5).

- `LOGGER_INIT_CONFIG configFile:string loggerConfig:reference` – Initializes a default logger map, updating its values based on the provided file-path (if available) and returns it through the loggerConfig parameter. See section Editing configuration files for details on the supported format.
- `LOG_DEBUG (string:message)` – Logs a debug event with the specified message to the global logger.
- `LOG_INFO (string:message)` – Logs an info event with the specified message to the global logger.
- `LOG_WARN (string:message)` – Logs a warning event with the specified message to the global logger.
- `LOG_ERROR (string:message)` – Logs an error event with the specified message to the global logger.
- `LOG_FATAL (string:message)` – Logs a fatal error event with the specified message to the global logger.
- `LOGGER_SET_LOG_WINDOW (string:window) loggerConfig:reference` – Specifies the name of the window where events should be logged to. Set the window name to "" to not log to any window.
- `LOGGER_SHOW_LOG_WINDOW loggerConfig:reference` – Shows the log window of the logger. If it has no name specified for the log window it will assign the default name of "Event Log".
- `LOGGER_SET_LOG_FILE (string:file) loggerConfig:reference` – Specifies the path to a file where events should be appended to. Set the path to "" to not log to any file.
- `LOGGER_SET_LOG_FILE_GUI loggerConfig:reference` – Shows a GUI to set the log file.
- `LOGGER_SET_LOG_LEVEL (integer:logLevel) loggerConfig:reference` – Specifies up to which level events should be logged or set to 0 to not log anything.
- `LOGGER_SET_LOG_LEVEL_GUI loggerConfig:reference` – Shows a GUI to set the log level.

- `LOGGER_SET_LOG_POPUP_LEVEL (integer:popupLevel) loggerConfig:reference` – Specifies up to which level events should also trigger a popup message in the tool or set to 0 to not show such popups. Log level takes precedence of this.
- `LOGGER_SET_LOG_POPUP_LEVEL_GUI loggerConfig:reference` – Shows a GUI to set the log popup level.
- `LOGGER_SET_ALL_LOG_LEVELS_GUI loggerConfig:reference` – Shows a GUI to set the log level and the log popup level.

Olive

Functions and procedures to use Olive microservices, the Olive Microservice controller (Olive MSC) etc. Should an error during any of them occur then ecode is set to true (not 0).

- `mapToOliveJson(input)` – The provided simple map is transformed into the JSON data structure used by the Olive MSC for the inputs of service calls.
- `OLIVE_MSC_GET_MICROSERVICE_ID mscRestUrl:string microserviceName:string microserviceId:reference` – Retrieves the ID of the first microservice known under the name at the specified Olive MSC or empty if none is found.
- `OLIVE_MSC_CALL_MICROSERVICE_ID_OPERATION mscRestUrl:string microserviceId:string operationId:string input:string response:reference` – Calls an operation available through the Olive MSC. The input should be a valid JSON string as it is expected by the Olive MSC.
- `OLIVE_MSC_CALL_MICROSERVICE_OPERATION mscRestUrl:string microserviceName:string operationId:string input:string response:reference` – Calls an operation available through the Olive MSC. The input should be a valid JSON string as it is expected by the Olive MSC.

Other

An assortment of other functions and procedures available that don't quite fit into any other category.

- `RENAME_BASED_ON_ATTRS (string:str_objids) arr_attrnames:array` – Renames all the objects provided in the main parameter string list of IDs based on the values they have specified in one of its attributes. The attributes to use are specified through an array of attribute names, and the first attribute that is found to have a non-empty value is used for the name. Additionally to ensure unique names the ID of the object is appended in parentheses.
- `BEND_CONNECTOR_CENTER (integer:id_connector) val_xoff:real val_yoff:real` – Determines the centre of a connector specified as the main parameter and adds two bend points around it if it has fewer than two bend points. The val_xoff and val_yoff specify the offset between the two bend points.
- `UNBEND_CONNECTOR_CENTER (integer:id_connector)` – Does kind of the opposite of BEND_CONNECTOR_CENTER. It ensures that the connector specified as the main parameter has at most one bend point, removing any others.
- `GENERATE_MODELGROUPS string:str_mgroupnames id_supermgroup:integer` – Creates additional model groups based on the main parameter string list (separator "@") in the parent model group specified through id_supermgroup. If the parent model group ID is smaller or equal to 0 then it will use the "Models" model group.
- `GEN_ADL_FILE_RECURSIVE integer:id_mgroup str_folder:string` – Exports all the models found under any level of depth of the model group specified as the main parameter in an ADL file. The ADL file is in a folder of the system specified through str_folder and has the name of the selected model group.
- `GEN_GFX_FILES_RECURSIVE integer:id_mgroup str_folder:string str_gfxformat:string` – Exports all the models found under any level of depth of the model group specified as the main parameter as graphical files. The graphical files are placed in a folder of the system specified through str_folder, have the provided type if it is supported (e.g. bmp, gif, jpeg, png, tiff) and will be named after the model's name, version and type. The drawing area of the models will be trimmed as best as possible before generating the graphic.
- `ADD_MODELGROUP_MODELS_RECURSIVE integer:id_mgroup ref_modelids:reference str_sep:string` – Determines all the models that are found under any level of depth of the model group specified as the main parameter and places their IDs in the string list ref_modelids using the separator provided through str_sep.

- `TRIM_DRAWING_AREA (integer:id_model)` – Trims the drawing area of the model specified as the main parameter and saves it. Trimming only occurs from the bottom right and is based on the position and size of the elements.
- `TRIM_DRAWING_AREA_TWO (integer:id_model)` – Trims the drawing area of the model specified as the main parameter and saves it. Trimming only occurs from the bottom right and works by making the drawing area smaller in 1cm steps until it's no longer possible.
- `SUBSUP_EXPORT_GUI` – Shows the GUI for the "Export Exercises" functionality, exporting all models found under any level of depth of a selected model group as ADL and graphical files in a selected folder.
- `GET_CUSTOM_ATTRIBUTE_VALUES id_object:integer str_attrname:string ref_result:reference` – Provides in ref_result an array with all the values found in the "Custom attributes" table that have as "Property" set the specified str_attrname. The values of the table are always of type string, even when numbers are entered.
- `GET_CUSTOM_ATTRIBUTE_VALUE id_object:integer str_attrname:string ref_result:reference` – Provides in ref_result the first value found in the "Custom attributes" table that has the "Property" set to the specified str_attrname. The value is always provided as a string due to the type of the table.
- `SET_CUSTOM_ATTRIBUTE_VALUE id_object:integer str_attrname:string str_value:string` - Sets the value for the first row where "Property" has the specified str_attrname in the "Custom attributes" table. If no such row exists, then a new one is added. The value is always provided as a string due to the type of the table.

Petri Net specific

Functions and procedures that provide specific functionalities for Petri Net models, mostly dealing with simulation and analysis. Note that some of the procedures here rely on values evaluated from expression attributes, which aren't automatically updated when executing long AdoScripts. Use the AdoScript "Core" UPDATE_EXPR_ATTRS to update expression attributes.

- `safeLolaName(name:string, identifier:integer)` – Function that creates a name that can be used for the LoLA place/transition net format. Escapes characters and appends the identifier.
- `PN_PLACE_TRANSITION_NET_LM (integer:id_pnmodel) str_result:reference` – Generates the LoLA Place/Transition net format for the model with the ID specified as the main parameter and places it in the result parameter. The model is loaded and unloaded as needed.
- `PN_FIRE_TRANSITION (integer:id_trans) str_cthandling:string int_fired:reference` – Fires a specific Transition if possible. Through str_cthandling the behaviour of cold transitions can be controlled where the possible values are: "always fire", "based on probability" and "never fire". Whether the transition has really fired or not will be placed in the int_fired parameter.
- `PN_FIRE_ELIGIBLE_TRANSITIONS (integer:id_pnmodel) str_strategy:string str_cthandling:string str_fired:reference` – Fires all Transitions that are ready in the model specified through the main parameter. The str_strategy specifies what strategy should be employed if transitions require the same tokens to fire and the possible values are: "default", "random", "priority" and "priority then random". Through str_cthandling the behaviour of cold transitions can be controlled where the possible values are: "always fire", "based on probability" and "never fire". A string list of all Transitions that really have fired is placed in str_fired.
- `PN_FIRE_SEVERAL_ITERATIONS (integer:id_pnmodel) str_strategy:string str_cthandling:string val_iterations:integer str_log:reference` – Simulates several iterations of firing all transitions that are ready in the model specified through the main parameter. The str_strategy and str_cthandling parameters are the same as for PN_FIRE_ELIGABLE_TRANSITIONS. The amount of iterations to fire is specified through the val_iterations parameter. If str_log has the value "yes" then a log will be created during the execution and stored in the str_log parameter.
- `PN_SIMULATE_ITERATIONS_WITH_DELAY (integer:id_pnmodel) str_strategy:string str_cthandling:string val_delay:real str_log:reference` – Simulates the firing of all eligible Transitions in the model specified through the main parameter. Instead of executing a specific number of iterations it fires until it is cancelled. As such it relies on GUI elements and the user cancelling. Through val_delay a delay in seconds can be specified between the iterations. If val_delay is 0 or

smaller then the user is asked after each iteration whether to continue. The other parameters are the same as for PN_FIRE_SEVERAL_ITERATIONS.

- `PN_FIRE_ONE_ELIGIBLE_TRANSITION (integer:id_pnmodel) str_strategy:string str_cthandling:string str_fired:reference` – Fires one Transition in the model specified through the main parameter that is ready. The str_strategy controls how to decide which Transition to fire and its possible values are: "default", "random", "relative", "priority then default", "priority then random" and "priority then relative". Through str_cthandling the behaviour of cold transitions can be controlled where the possible values are: "always fire", "based on probability" and "never fire". The ID of the Transition that has fires is placed in the str_fired parameter, which can be empty if no Transition has fired.
- `PN_FIRE_SEVERAL_TRANSITIONS (integer:id_pnmodel) str_strategy:string str_cthandling:string val_iterations:integer str_log:reference` – Similar to PN_FIRE_ONE_ELIGABLE_TRANSITION, except that it performs multiple iterations whose amount is specified through val_iterations. Each iteration selects one Transition that can fire and does so. If str_log has the value "yes" then a log will be created during the execution and stored in the str_log parameter.
- `PN_SIMULATE_TRANSITIONS_WITH_DELAY (integer:id_pnmodel) str_strategy:string str_cthandling:string val_delay:real str_log:reference` – Simulates the firing of one eligible Transition in the model specified through the main parameter. Instead of executing a specific number of iterations it fires until it is cancelled. As such it relies on GUI elements and the user cancelling. Through val_delay a delay in seconds can be specified between the iterations. If val_delay is 0 or smaller then the user is asked after each iteration whether to continue. The other parameters are the same as for PN_FIRE_SEVERAL_TRANSITIONS.
- `PN_UPDATE_STATELOG (integer:id_pnmodel) str_firedtrans:string str_statelog:reference` – Updates a state log in CSV format for the model specified through the main parameter. The state of the model is appended to str_statelog if it is not empty, otherwise a statelog is initialized in that parmeter. Through str_firedtrans the transitions that have fired since the last update of the statelog can be specified to capture them in the log too.
- `PN_STORE_STATE (integer:id_statestore)` – Stores the state of the Petri Net model in the State Storage object specified as the main parameter. It uses the Petri Net the State Storage is located in.
- `PN_RESTORE_STATE (integer:id_statestore)` – Restores the state from a State Storage specified as the main parameter to its Petri Net model. It uses the Petri Net the State Storage is located in.
- `PN_UPDATE_INTERREFS (integer:id_statestore)` – Updates the interrefs of a State Storage specified as the main parameter to reference elements in its model.

Random distributions

Several functions and procedures are provided to create random numbers based on specific distributions.

- `randomStandardUniformDist()` – Returns a random floating-point number for a uniform distribution between (incl.) 0.0 and (excl.) 1.0.
- `randomUniformDist(low:ral, high:real)` – Returns a random floating-point number for a uniform distribution between (incl.) low and (excl.) high.
- `randomStandardNormalDist()` – Returns a random floating-point number for a standard normal distribution (μ = 0, standard deviation = 1).
- `randomNormalDist(mu:real, stddev:real)` – Returns a random floating-point number for a normal distribution with the specified μ and standard deviation.
- `randomTriangularDist(low:real, mode:real, high:real)` – Returns a random floating-point number for a triangular distribution with the specified low, mode and high.
- `randomExponentialDist(rate:real)` – Returns a random floating-point number for an exponential distribution with the specified rate / inverse scale.
- `randomDiscreteDistPositions(probabilities:array)` – Returns a random integer number based on the probabilities specified in the array. The returned value corresponds to the position in the array, starting at 0. The sum of all probabilities must be 1.
- `randomDiscreteDistValues(probabilities:map)` – Returns a random value based on the probabilities specified in the map. The values should be specified as the keys of the map under which

their probabilities are found e.g. {"a":0.4, "b":0.2, "c":0.2, "d":0.1, "e":0.1}. The sum of all probabilities must be 1.

- `randomDiscreteUniformDist(low:integer, high:integer)` – Returns a random integer number for a discrete uniform distribution between (incl.) low and (excl.) high.
- `randomBernoulliDist(probability:real)` – Returns either 1 or 0 based on the Bernoulli distribution. The provided probabilities is for the 1-value.
- `randomRademacherDist()` – Returns either 1 or -1 based on the Rademacher distribution, where both values have a 50% probability.
- `randomCoinToss()` – Returns either 1 or 0 based on a fair coin toss, where both values have a 50% probability.

Type support

Support functions and procedures to help with common functions for different types of data, like maps or strings.

- `mapKeysArray(input:map)` – Returns all the keys from the map as an array with their proper type.
- `mapKeysTypes(input:map)` – Returns all the types of keys that are used in a map and returns them as a string list (separated with " "). The length of the list can be shorter than the map.
- `mapKeysList(input:map)` – Returns all the keys from the map as a string list (separated with "~". All the keys must be of type string or else this will cause an error.
- `mapKeysTypedList(input:map)` – Returns all the keys from the map as a string list (separated with "~"). The values in the string list reflect their type, e.g. strings are put in double quotes, so they can be transformed to their proper value using the eval() function.
- `countSubstringOccurrence(input:string, substr:string)` – Returns how often the specified sub-string is found in the input.
- `countTokenOccurrence(list:string, token:string, separator:string)` – Returns how often the specified token is found in a string list using the provided separator.
- `replaceTextInArea(text:string, newtext:string, start:integer, end:integer)` – Returns a new string where the area between (incl.) start and (excl.) end in text will be replaced with newtext. The new text can be longer or shorter than the area it replaces.
- `TS_RECORD_SORT_SIMPLE objectId:integer recordAttrName:string recordColumnName:string direction:string` - Sorts the rows of a record/table according to the values in one of its columns. Direction should specify "desc" to order in descending order, otherwise the values are sorted in ascending order.
- `TS_RECORD_FIND_ROWS_FROM_VALUE objectId:integer recordAttrName:string recordColumnName:string columnValue:string rowIds:reference` - Finds all rows of a record/table which contain a specific value in a specified column.
- `TS_RECORD_FIND_FIRST_ROW_FROM_VALUE objectId:integer recordAttrName:string recordColumnName:string columnValue:string rowId:reference` - Finds the first row of a record/table which contains a speicifc value in a specified column.
- `TS_RECORD_AS_ARRAY objectId:integer recordAttrName:string result:reference` - Transforms the content from a record/table into an array representation. Rows are represented as maps.
- `TS_RECORD_ROW_AS_MAP rowId:integer result:reference` - Transforms a row from a record/table into a map representation.

Version checks

Functions and procedures that help checking if a new version is available online and for comparing version numbers. The procedures use a configuration map that contains the details of the tool or component and where to check for a new version. Therefore, it can also be used for checking versions of other things besides Bee-Up. The version checks are performed as the last activity when the tool is started, i.e. executed after the AutoStart functionality of Bee-Up. It also automatically performs checks for each configuration that is found in a global array stored in the variable g_vercheck_otherConfigs, so automatically checking for new versions of components can be achieved by appending configurations to the array.

- `getSemanticVersion(input:string)` – Returns a map representation of the semantic version number, splitting it into "major", "minor" and "patch" numbers and "label". The input can start with some text (may not contain digits) until the version number which will be ignored.
- `compareSemanticVersions(version1:string, version2:string)` – Returns a positive number if version1 is larger, a negative number if version2 is larger or zero if both are the same. Excepts both versions to be compatible with the getSemanticVersion function. The "label" is only considered when one of the versions has an empty label (i.e. no detailed label comparison performed).
- `compareCompatibleSemanticVersions(version1:string, version2:string)` – Returns a positive number if version1 is larger or version2 is not backwards compatible, a negative number if version2 is larger and backwards compatible or zero if both are the same. Excepts both versions to be compatible with the getSemanticVersion function. The "label" is only considered when one of the versions has an empty label (i.e. no detailed label comparison performed).
- `VERCHECK_INIT_CONFIG configFile:string vercheckConfig:reference` – Initializes a default version check configuration map, updating its values based on the provided file-path (if available) and returns it through the vercheckConfig parameter. See section Editing configuration files for details on the supported format.
- `VERCHECK_GET_LATEST_VERSION_INFO vercheckConfig:map map_latestVersionInfo:reference` – Uses the configuration provided through vercheckConfig to retrieve the label and the date of the latest version and returns them through map_latestVersionInfo.
- `VERCHECK_CHECK_FOR_NEW_VERSION vercheckConfig:reference` – Checks if a new version is available and uses GUI elements to notify the user about the outcome. This check should be initiated by the user.
- `VERCHECK_PERFORM_AUTOMATIC_CHECK vercheckConfig:reference` – Checks if a new version is available and uses GUI elements to notify the user if this is the case. This check should be used when automatically performing a check, e.g. at start-up.
- `VERCHECK_UPDATE_CONFIG_FILE vercheckConfig:map` – Updates the configuration file that stores the version check configuration.

Components / Modules used in Bee-Up

The following lists some generally available components and modules that Bee-Up uses. Details and information about them can be looked up at the corresponding sources.

ADOxx add-ons

- AdoScript Configuration Files – v1.0.0
- AdoScript Date and Time – v1.0.0
- AdoScript Extended Conversions – v1.0.2
- AdoScript Logging Procedures – v1.0.0
- AdoScript Random Distributions – v1.0.0
- AdoScript Remote Execution – v1.0.1
- AdoScript Type Support – v1.1.0
- ADOxx Add-on Extension – v1.1.0
- ADOxx Auto Start – v1.0.0
- ADOxx Cloning Procedures – v1.0.0
- ADOxx Olive Integration – v1.0.1
- ADOxx Version Check – v1.0.0
- RDF Transformation – v3.2.1

The Extended HTTP Requests module has been removed from Bee-Up 1.7, because the used ADOxx version now provides a direct AdoScript command (CC "AdoScript" HTTP_REQUEST) with better options. Most of the procedures provided by the previously used module are still available for backwards compatibility, building upon the AdoScript command. The procedures providing results as "bytes" no longer work (instead, the AdoScript command can load data as binary): `HTTP_SEND_REQUEST_BYTES`, `HTTP_SEND_AUTH_REQUEST_BYTES`, `HTTP_SEND_REQUEST_BYTES_INBASE`, `HTTP_SEND_AUTH_REQUEST_BYTES_INBASE`

Change history

Changes in version 1.7

- Updated to ADOxx version 1.8. Some of the relevant changes this brings:
  - It is now a 64-bit Application.
  - Use of SQLite as the database system.
  - Automatic import of relevant Attribute Profiles specifying Bee-Up functionality during installation.
  - Enhanced AdoScript command HTTP_REQUEST, making old HTTP_SEND_REQUSET_... procedures obsolete.
  - Specify an HTTP/S URL to an image for the "External Graphic" of "Note" objects.
  - For further changes: check the ADOxx 1.8.0 changelog.
- Other minor fixes and improvements (Firing Petri Net transitions marks a model as modified, success message after "Create SQL Statements" etc.).

Changes in version 1.6

- The Bee-Up Handbook and the IMKER case study are now available through the “Help” menu when they are installed with Bee-Up. Note that these are the local files and updated versions might exist online.
- Updated the used "Extended HTTP Requests" to the new version, now supporting HTTPS.
- Fixed issues with the Petri Net State Storage having bad entries when copy and pasting models / contents.
- Moved the "Create SQL Create statements" functionality from the "Model" menu ("Import/Export" component) to a model attribute found under "ER properties" of the ER Model.
- Added a context menu item to rename selected elements based on their "Denomination" or "Label" attribute or in some cases other referenced objects.
- Added the "UML Class Diagram 2 Skeleton" prototype, which converts a class diagram into Java, Python or C++ code.
- Added checking for new versions of Bee-Up: manually through "Help" menu and automatically on start-up.
- Added "Visualized value" attribute to "Note" elements.
- Added a "Switch" element, conditions (guards) to "Subsequent" relations and "Ending type" to "End Terminal" in Flowcharts.
- The preservation of ecode values throughout Flowchart executions has changed to better fit with the "Ending type" of an "End Terminal", i.e. the ecode is "saved" after executing an "Operation" or "External Operation" and "restored" before executing custom code of an "Operation", "External Operation", "Decision" or "Switch". Furthermore the "saved" ecode is also passed on beyond a "Neutral" type "End Terminal".
- Added an implementation to work with an ASRE Server (AdoScript Remote Execution).
- Added an "AutoStart" feature, which allows executing specific functionalities when the tool is started. Some Bee-Up functionalities are already realized with AutoStart.
- Added a table attribute "Custom attributes" to each object. This can be used when certain values/data should be stored at the objects, but none of the other already available attributes fit. It can be processed for example in custom implementations of functionalities / scripts. Additionally added two procedures "GET_CUSTOM_ATTRIBUTE_VALUES", "GET_CUSTOM_ATTRIBUTE_VALUE" and "SET_CUSTOM_ATTRIBUTE_VALUE" to make working with the values from the table easier. For more control use "TS_RECORD_FIND_ROWS_FROM_VALUE" or "TS_RECORD_FIND_FIRST_ROW_FROM_VALUE".
- Configuration files for several functionalities are stored in the internal database. They use a simple syntax (key=value and lines are commented with #) and can be edited through "Extras -> Edit internal configuration file".
- Basic logging procedures are now available in AdoScript and several menu items in the "Extras" menu help configure them.

- Restructured some internal files. This might also have changed some function names (e.g. lam_tokenCount -> countTokenOccurrence)! Also added new functions (fromJson, mapToJson, arrayToJson, etc.).
- Removed last remaining dependency on "ASC_Map_functions_150317.asc" and thus removed that file.
- Removed broken attributes from "Performance indicator overview" table.
- Other minor fixes and improvements.

Changes in version 1.5

- Installation scripts for Linux and macOS created.
- Added functionality for cloning models, available through the "Modelling" component in the "Model" menu. Cloning a set of models at once will also adapt the Inter-Model references of the cloned models to point towards other cloned models of the same set if possible.
- Flowcharts: When executing through the button of the Start Terminal the user will be asked to provide values for all the required variables. The kind of popup and message shown depends on the selected type for that variable.
- General: Added a "General purpose attribute" to every element and relation. This can be used when certain values/data should be stored at the objects, but none of the other already available attributes fit. This can be used for example in custom implementations of functionalities / scripts.
- Minor improvements to notations, help/information texts, handling of functionalities etc.

Changes in version 1.4

- UML: Improvements to some elements (hide name of Lifeline, Diamond shape for Relation Node to use with UML Associations etc.).
- Flowcharts: Added proper help-texts to the objects and improved the execution functionality:
  - Previously operations could use one "keyword-extension" (PRINTLN, READS, INC etc.) as its code to make simple operations easier to understand. This has been further improved. It is now possible to use multiple "keyword-extensions" in the same operation. Each used "keyword-extension" has to be put in its own separate line and will be considered "as a whole" from the beginning of the line to the end of the line.
  - During the execution the Operation and Decision have access to some preset variables (Do not overwrite their values directly or you might not get them back!):
    - "this_modelid" which contains the ID of the Flowchart model.
    - "this_objid" which contains the ID of the currently executing object.
    - "str_fc_stdout" which contains everything written using PRINT or PRINTLN.
    - "str_fc_debout" which contains everything written to the debug output, even if the debug window is not shown.
  - Added the option to disable highlighting of elements during the execution. This can increase the speed quite drastically if the debug output is also disabled.
  - Added an "External Operation" object which allows executing functionality described somewhere else (e.g. a different Flowchart, a RESTful web-service etc.).
- External web services and other similar functionality can be added as Attribute Profiles and called using the new "Execute External Functionality" found in the "Extras" menu. This allows the modeler to extend the available functionality in Bee-Up to some degree and the service descriptions as Attribute Profiles can be exchanged using the ADL-Import/Export functionality. In a sense this is a "Plug-in" mechanism for Bee-Up.
- Integrated the patch for extended HTTP requests in AdoScript.

Changes in version 1.3

- In BPMN and EPC: Added possibility to further describe the decisions made in tasks/functions through elements based on DMN Version 1.1.
  - (BPMN) Tasks and (EPC) Functions can reference a (DMN) Decision through their "Make decision" attribute.

  - (BPMN) Data Objects and (BPMN) Data Associations can be used to further details where and how (DMN) Input Data is set.
- In BPMN and EPC: Made the notation of relations more distinguishable from one another.
- In PN: Improved and extended the execution and simulation capabilities:
  - The Delayed Simulation is now more responsive (Cancelling is now quicker), works with tenths of a second (it can be faster than 1 second now) and also highlights the fired transitions (to better see the fired transitions: switch to "Grayscale mode").
  - The created simulation log has changed. Instead of only containing the places and the amount of tokens, it now also contains the transitions and whether they have fired or not. For details check the information text of the "Show log" attribute.
  - A new style for firing transitions was added through the "Automated Transition Firing" element. See its information text for more details.
  - Added an "Effect" attribute to transitions and an "Allow effects" model attribute. "Allow effects" activates the "Effect" attribute, which can then be used to specify AdoScript code, which is executed when the transition is fired (after removing tokens, before creating tokens).
  - Places can now have a maximum capacity for tokens specified.
  - The transition's "Fire" buttons can be hidden using the model attribute "Visualize fire button".
- In ER: The Create SQL statements functionality now has two options for handling inheritance 1) the old style where the table is copied and 2) [now default] which handles inheritance similar to Weak Entities. They can be switched through the model attribute "IS-A Behaviour".
- Added an option to show models using mostly only black, white and gray for all model types except UML. This can be enabled for each model through a new model attribute "Grayscale mode".
- Added functionality which executes Flowcharts. The provided code in Flowcharts (Operation code/Check expression) can use AdoScript as well as some additionally provided keywords. See the information text for "Execute flowchart from Start" in a Start Terminal for more details.
- Added Functions for AdoScript which provide a random value based on different types of distributions. Those are:
  - randomStandardUniformDist() --> a random value from a uniform distribution between (including) 0.0 and (excluding) 1.0, so it is very close to the Standard Uniform Distribution.
  - randomUniformDist(lower_limit, upper_limit) --> a random value from a uniform distribution between (including) the lower limit and (excluding) the upper limit.
  - randomStandardNormalDist() --> a random value from a standard normal distribution (i.e. expectancy value = 0, standard deviation = 1) based on Box-Muller transformation using a natural logarithm.
  - randomNormalDist(expectancy_value, standard_deviation) --> a random value from a normal distribution with a specific expectancy and standard deviation based on Box-Muller transformation using a natural logarithm.
  - randomTriangularDist(lower_limit, mode, upper_limit) --> a random value from a triangular distribution based on inverse CDF from "Beyond Beta - Other Continuous Families of Distributions with Bounded Support and Applications". The triangle is build from lower_limit to upper_limit with its peak at mode.
  - randomExponentialDist(inverse_scale) --> a random value from an exponential distribution based on inverse CDF using the inverse scale provided (lambda).
  - randomDiscreteDistPositions(probabilities) --> a random value from a discrete set of probabilities. The probabilities have to be an array and the returned value is a position index (0 to (LEN probabilities)-1) from the array. The sum of all probabilities should be 1.0.
  - randomDiscreteDistValues(value_probabilities) --> a random value from a discrete set of values and their corresponding probabilities. The value_probabilities have to be a map (key-value pairs), where the keys are the possible values (either strings or numbers) and their values should be their probability. The sum of all probabilities should be 1.0.
  - randomDiscreteUniformDist(lower_limit, upper_limit) --> a random value from a discrete uniform distribution of integers between (including) the lower limit and (excluding) the upper limit.
  - randomBernoulliDist(probability) --> either 1 or 0 based on the Bernoulli distribution, with the parameter indicating the probability of the value 1. A probability of 0.5 can be considered a coin-toss.

  - randomRademacherDist() --> either 1 or -1 based on the Rademacher distribution, where the probabilities of both cases are 50%.
  - randomCoinToss() --> either 1 or 0 based on a fair coin, with 50/50 chance. So same as randomRademacherDist, only with other outcomes.
- The automatic change of model names by adding the model group name at their beginning has been removed, since it has led to problems when importing models.
- Additional minor improvements and bug-fixes.

Major changes in version 1.2

- In BPMN and EPC: Added possibility to describe automated tasks (BPMN Service tasks, EPC functions) through Petri Nets or Flowcharts.
- In ER: Added two properties "Data type (direct)" and "Auto increment" for ER attributes, providing more options to the generation of SQL-Create statements.
- Added additional attributes to elements for enhancing the RDF-Export (e.g. specify element URI, provide additional triples, meaningful references between models/model elements).

Major changes in version 1.1

- In BPMN: Merged "Intermediate Event (boundary)" and "Intermediate Event (sequence)" into "Intermediate Event". The old classes are still available and can be converted to allow model compatibility to Version 1.0
- In PN: Allowed two special conditions on Arcs which control the firing of the transition without consuming tokens.
- In PN: Added two model attributes "Visualize priorities" and "Visualize probabilities" to turn on and off the visualization of those transition attributes in the model.
- Added RDF Export functionality for all models.

Development team

The Bee-Up modelling tool has been realized by the following team:

- Patrik Burzynski (patrik.burzynski@omilab.org): chief developer
- Dimitris Karagiannis: project owner

Additional tools used

The following additional tools, implementations, binary codes etc. are used/included in Bee-Up and their according licenses apply:

- Java Runtime Environment 1.8 – is used by the RDF Export functionality. Java can be found at https://java.com
- Apache Jena 3.1.0 – is used by the RDF Export functionality. Apache Jena website is available here: http://jena.apache.org/
- JDOM 2.0.6 – Developed by the JDOM Project (http://www.jdom.org/), it is used in the RDF Export functionality.