Spine Toolbox Documentation

Release 0.5.0-beta.0

Pekka Savolainen, Manuel Marin, Erkka Rinne

Nov 17, 2020

Contents:

Spine Toolbox is an application, which provides means to define, manage, and execute complex data processing and computation tasks, such as energy system models.

You can either start reading from the first page onwards or go straight to the *Getting Started* section to get you started quickly. If you need help understanding the terms we use throughout the app and this User Guide, please check the *Terminology* section. If you want to contribute to this project, please see the *Contribution Guide for Spine Toolbox*. The last section contains the complete code reference of Spine Toolbox.

CHAPTER 1

Getting Started

Welcome to the Spine Toolbox's getting started guide. In this guide you will learn two ways of running a "Hello, World!" program on Spine Toolbox. The following topics are touched (although not exhaustively covered):

- *Spine Toolbox Interface*
- *Creating a Project*
- *Creating a Tool specification*
- *Adding a Tool item to the project*
- *Executing a Tool*
- *Editing a Tool specification*
- *Adding a Data Connection item to the project*
- *Adding data files to a Data Connection*
- *Connecting project items*

1.1 Spine Toolbox Interface

The central element in Spine Toolbox's interface is the *Design View*, where you can visualize and manipulate your project in a pictorial way. Alongside *Design View* there are a few 'dock widgets' that provide additional functionality:

- *Project* provides a more concise view of your project, including:
  - *Items* currently in the project, grouped by category: Data Stores, Data Connections, Tools, Views, Importers and Exporters.
  - *Tool specifications* available in the project.
- *Properties* provides an interface to interact with the currently selected project item.

- *Event Log* shows relevant messages about every performed action.
- *Process Log* shows the output of executed Tools.
- *Julia console* provides an interface to interact with the Julia programming language, and also allows Spine Toolbox to execute Julia Tools.
- *Python console* provides an interface to interact with the Python programming language, and also allows Spine Toolbox to execute Python Tools.

Tip: You can drag-and-drop the Dock Widgets around the screen, customizing the interface at your will. Also, you can select which ones are shown/hidden using either the **View/Dock Widgets** menu, or the main menu toolbar's context menu. Spine Toolbox remembers your configuration between sessions. Selecting **Restore Dock Widgets** from the **View/Dock Widgets** menu restores the widgets back to their default location.

Tip: Most elements in the Spine Toolbox's interface are equipped with *tool tips*. Leave your mouse cursor over an element (button, view, etc.) for a moment to make the tool tip appear.

1.2 Creating a Project

To create a new project, please do one of the following:

A) From the application main menu, select **File -> New project...**

B) Press *Ctrl+N*.

The *Select project directory (New project...)* dialog will show up. Browse to a folder of your choice and create a new directory called 'hello world' there. Then select the 'hello world' directory. Spine Toolbox will populate that directory with some files and directories it needs to store the project's data.

Congratulations, you have created a new project.

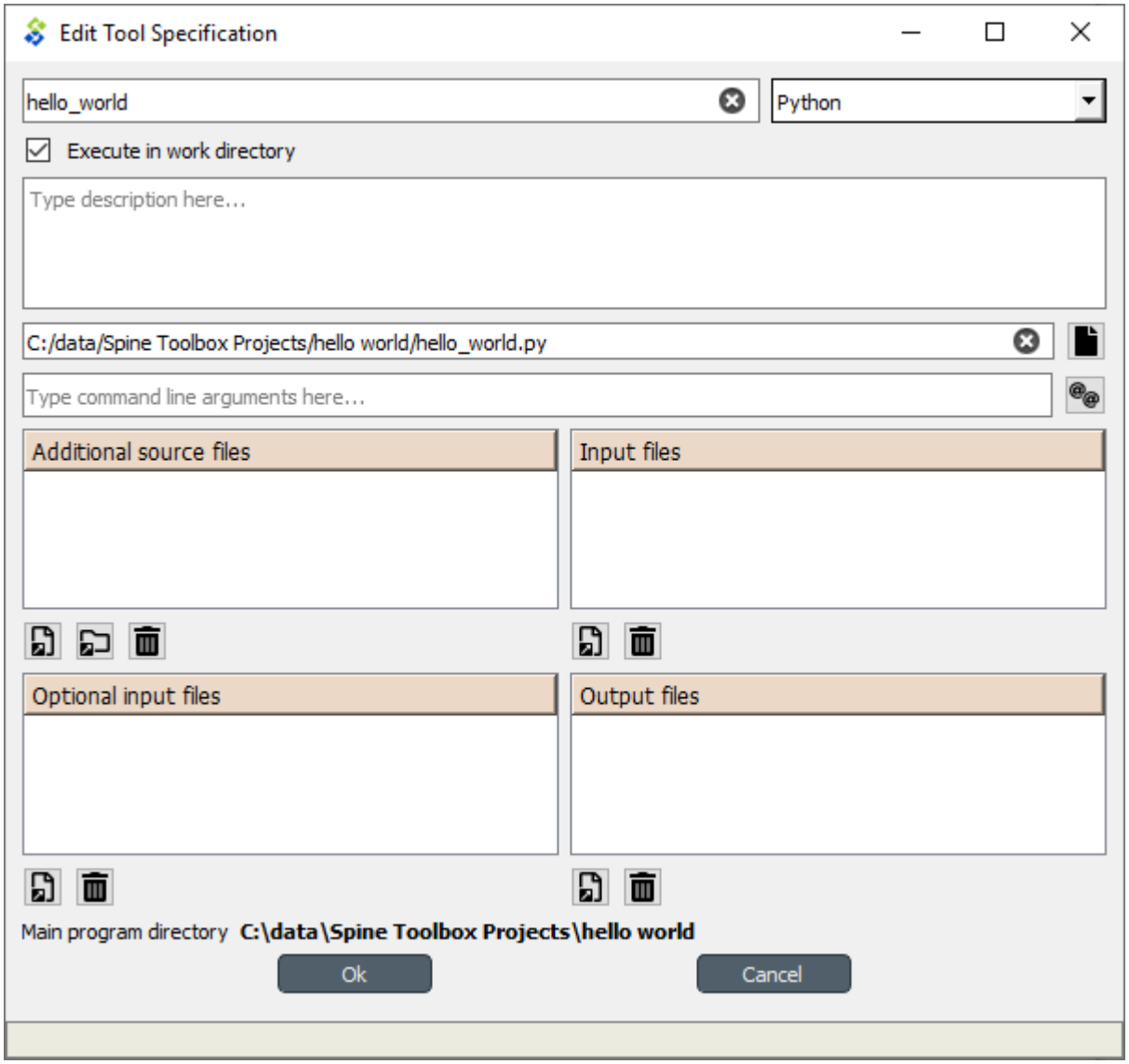
1.3 Creating a Tool specification

Note: Spine Toolbox is designed to run and connect multiple tools, which are specified using **Tool specifications**. You may think of a Tool specification as a self-contained program specification including a list of source files, required and optional input files, and expected output files. Once a Tool specification is added to a project, it can then be associated to a **Tool** item for its execution as part of the project workflow.

In the *Project* dock widget, click on the 'add tool specification button' () just below the *Tool specifications* list, and select **New** from the popup menu. The *Edit Tool specification* form will appear. Follow the instructions below to create a minimal Tool specification:

- Type 'hello_world' in the *Type name here...* field.
- Select 'Python' from the *Select type...* dropdown list,
- Click on the button right next to the field that reads *Add main program file here...*, and select the option **Make new main program** from the popup menu.
- A file browser dialog should open. Name the file *hello_world.py* and save it in a folder of your choice, e.g. in 'hello world'

After this, the *Edit Tool specification* form should be looking similar to this:



Click **Ok** at the bottom of the form. A new system dialog will appear, allowing you to select a file name and location to save the Tool specification we've just created. Don't change the default file name, which should be *hello_world.json*. Just select a folder from your system (it can be the same where you saved the main program file) and click **Save**.

Now you should see the new tool specification in the *Project* widget, *Tool specifications* list.

Tip: Saving the Tool specification into a file allows you to add and use the same Tool specification in another project. To do this, you just need to click on the add tool button (), select **Add existing...** from the popup menu, and then select the tool specification file from your system.

Congratulations, you have just created your first Tool specification.

However, the main program file *hello_world.py* was created empty, so for the moment this Tool specification does absolutely nothing. To change that, we need to add instructions to that program file so it actually does something when executed.

Right click on the 'hello_world' item in the *Tool specifications* list and select **Edit main program file...** from the

context menu. This will open the file *hello_world.py* in your default editor.

Enter the following into the file's content:

```
print("Hello, World!")
```

Save the file.

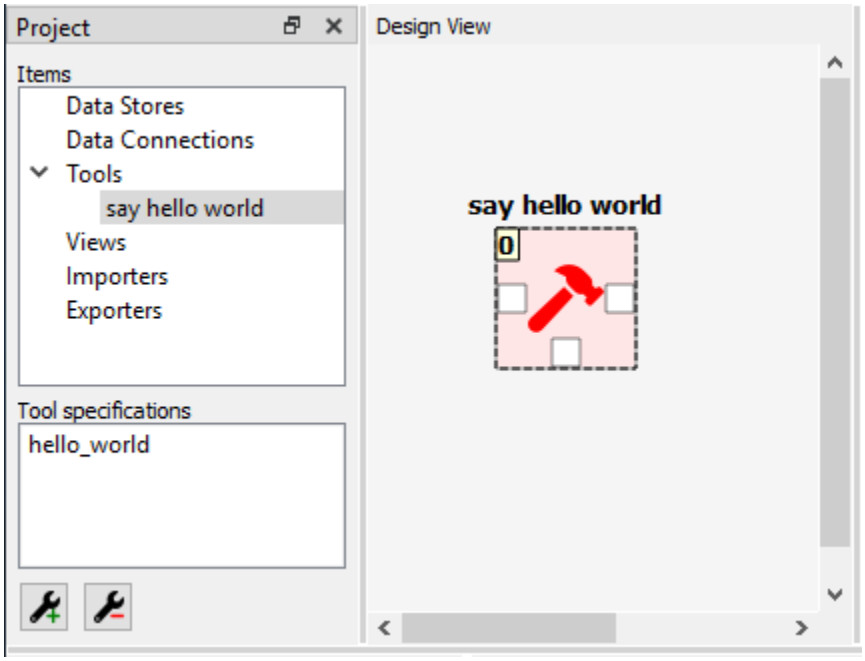
Now, whenever *hello_world.py* is executed, the sentence 'Hello, World!' will be printed to the standard output.

1.4 Adding a Tool item to the project

Note: The **Tool** item is used to run Tool specifications available in the project.

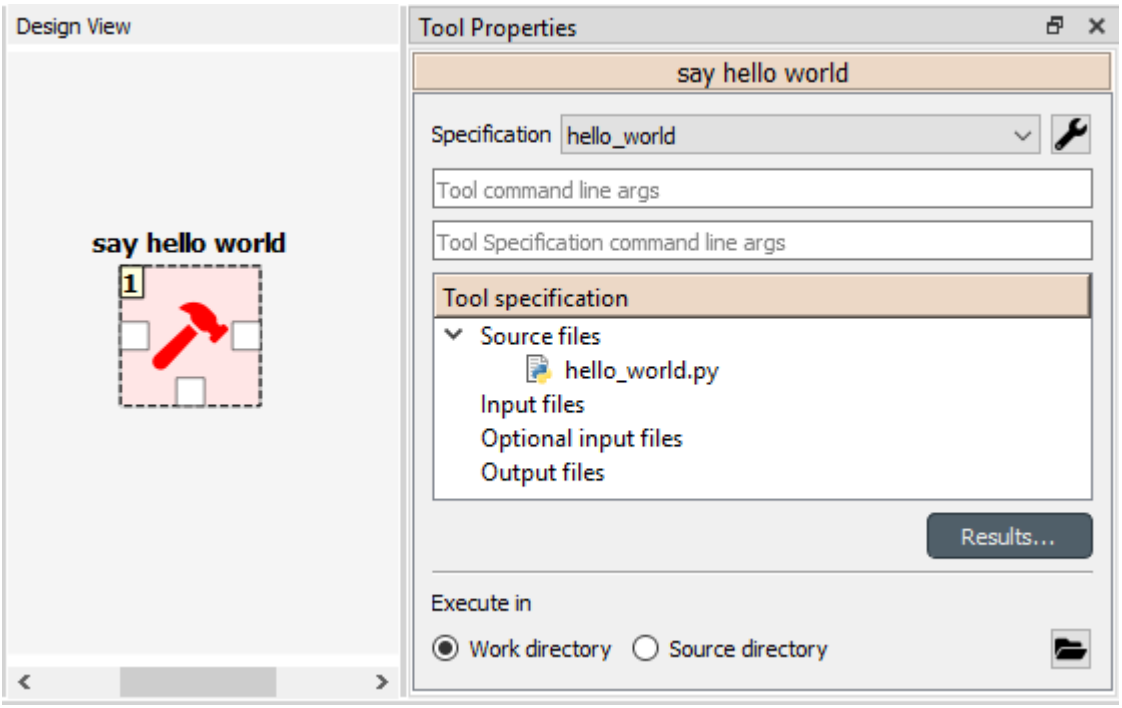
Let's add a Tool item to our project, so that we're able to run the Tool specification we created above. To add a Tool item drag-and-drop the Tool icon () from the *Drag & Drop Icon* toolbar onto the *Design View*.

The *Add Tool* form will popup. Type 'say hello world' in the name field, select 'hello_world' from the dropdown list just below, and click **Ok**. Now you should see the newly added Tool item as an icon in the *Design View*, and also as an entry in the *Project* dock widget, *Items* list, under the 'Tools' category. It should look similar to this:



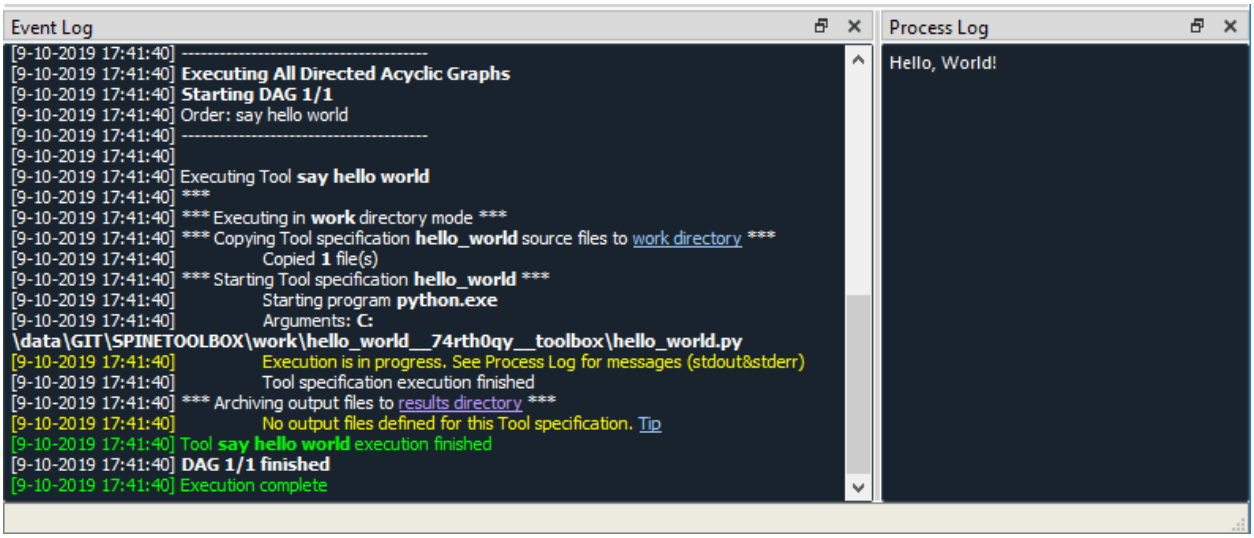
1.5 Executing a Tool

As long as the 'say hello world' Tool item is selected, you will be able to see its *Properties* on the right part of the window, looking similar to this:



Press *execute project* button on the toolbar. This will execute the Tool specification 'hello world', which in turn will run the main program file *hello_world.py* in a dedicated process.

You can see more details about execution in the *Event Log*. Once it's finished, you will see its output in the *Process Log* or in the *Python Console* depending on your settings (See *Settings*).



Note: If you encounter the following message in Event Log when trying to execute a Python Tool.

Couldn't determine Python version. Please check the Python interpreter option in Settings.

Please see *Setting up External Tools* for help.

Congratulations, you just ran your first Spine Toolbox project.

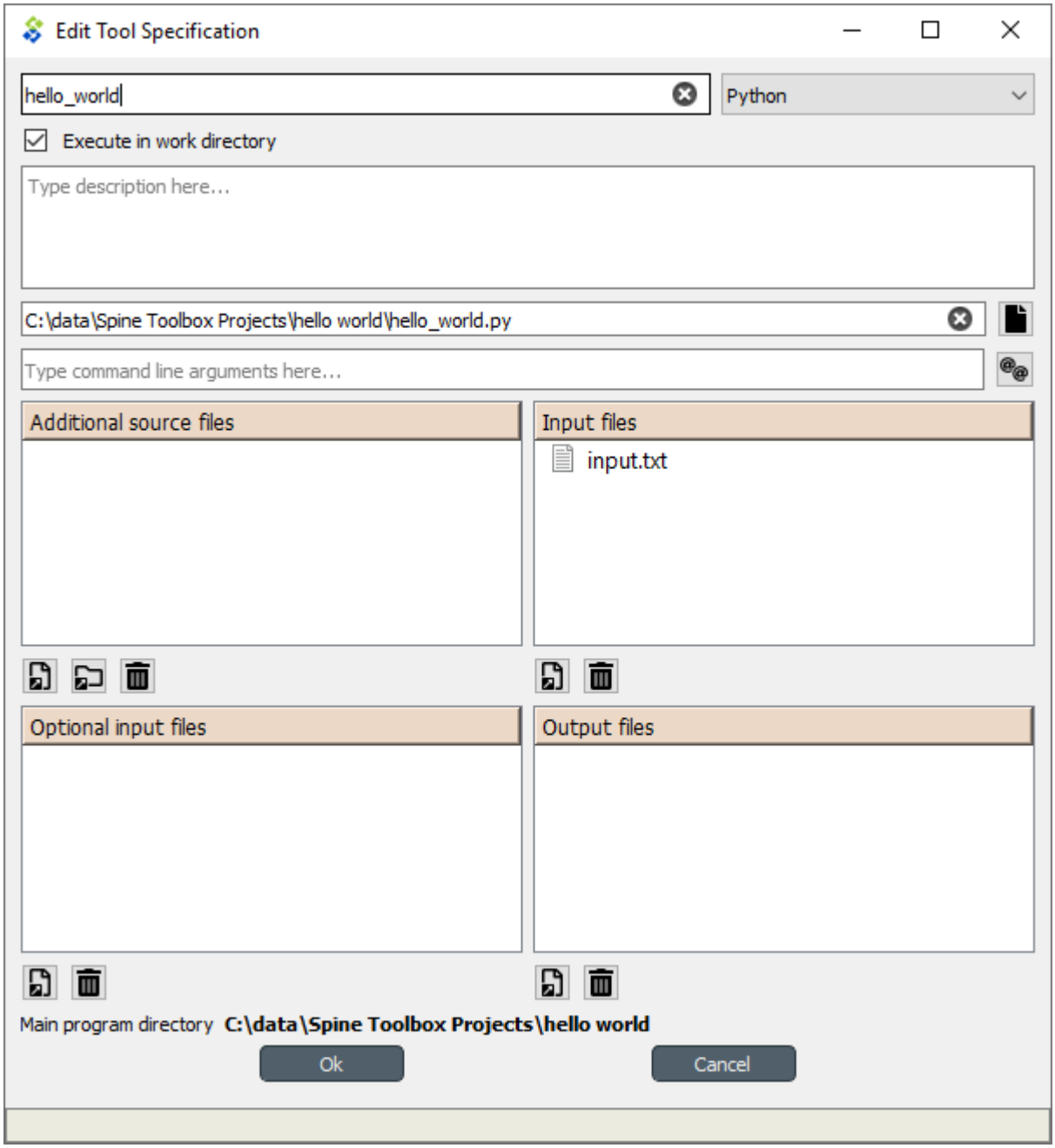
1.6 Editing a Tool specification

To make things more interesting, we will now specify an *input file* for our 'hello_world' Tool specification.

Note: Input files specified in the Tool specification can be used by the program source files, to obtain some relevant information for the Tool's execution. When executed, a Tool item looks for input files in **Data Connection** and **Data Store** items connected to its input.

Click on the 'Tool specification options' button () in 'say hello world' *Properties*, and select **Edit Tool specification** from the popup menu. This will open the 'Edit Tool specification' form pre-filled with data from the 'hello_world' specification.

Click the *add input files and/or directories* button right below the *Input files* list. A dialog will appear that lets you enter a name for a new input file. Type 'input.txt' and click **Ok**. The form should now look like this:



Click **Ok** at the bottom of the form.

Note: See *Tool specification editor* for more information on editing Tool specifications.

So far so good. Now let's use this input file in our program. Click on the 'Tool specification options' button () again, and this time select **Edit main program file...** from the popup menu. This will open the file *hello_world.py* in your default editor.

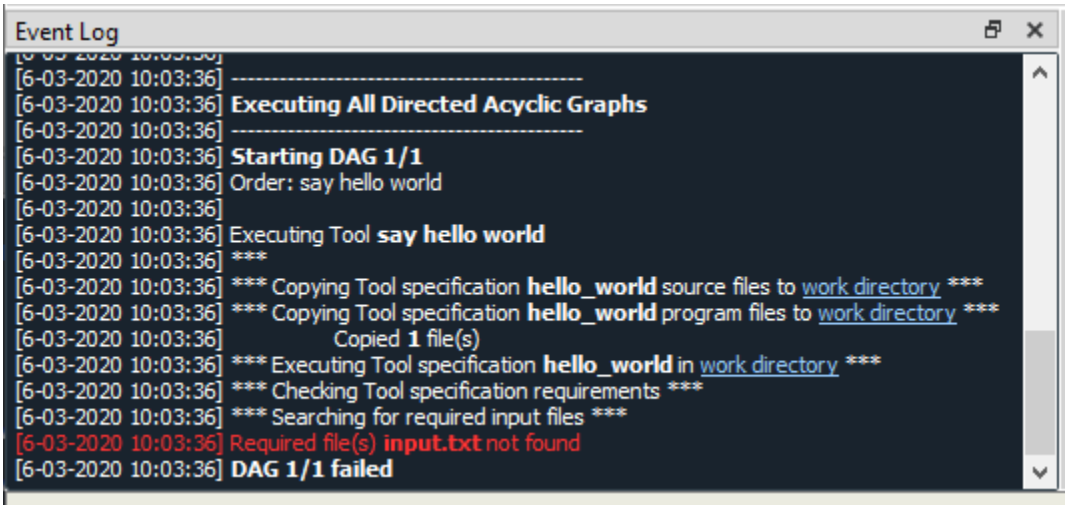
Delete whatever it's in the file and enter the following instead:

```
with open("input.txt") as input_file:
    print(input_file.read())
```

Save the file.

Now, whenever *hello_world.py* is executed, it will look for a file called 'input.txt' in the current directory, and print its content to the standard output.

Try executing the tool by pressing in the toolbar. *The execution will fail.* This is because the file 'input.txt' is not made available for the Tool:

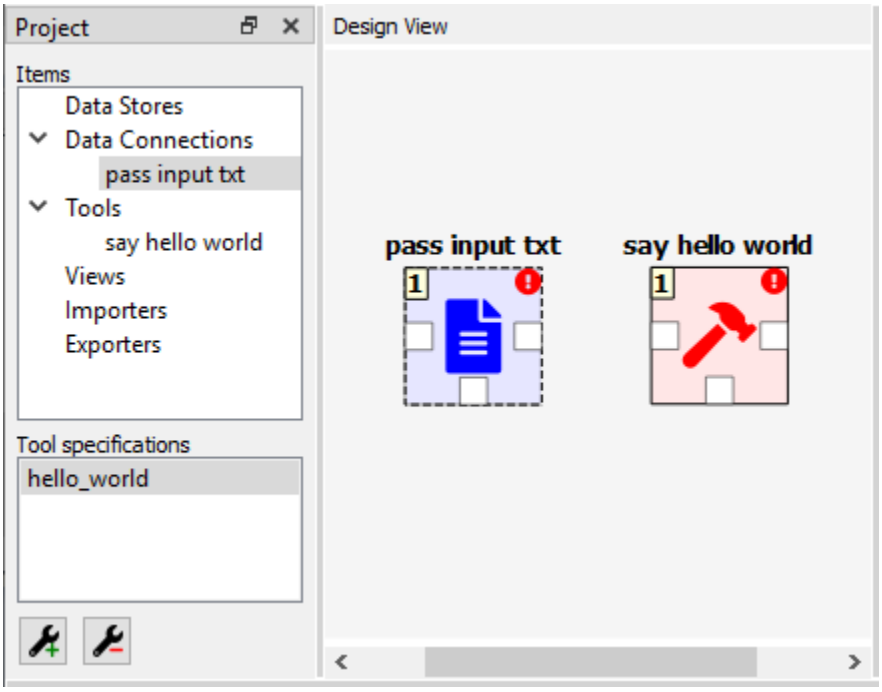


1.7 Adding a Data Connection item to the project

Note: The **Data Connection** item is used to hold generic data files, so that other items, notably Importer and Tool items, can make use of that data.

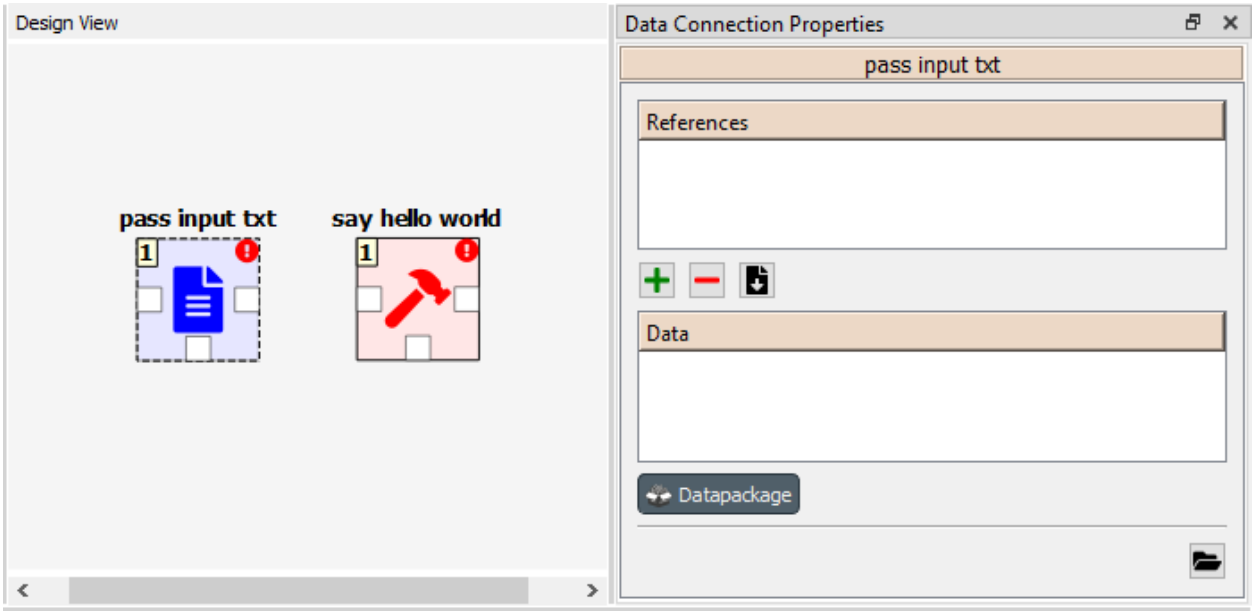
Let's add a Data Connection item to our project, so that we're able to pass the file 'input.txt' to 'say hello world'. To add a Data Connection item drag-and-drop the Data Connection icon () from the main window toolbar onto the *Design View*.

The *Add Data Connection* form will show up. Type 'pass input txt' in the name field and click **Ok**. Now you should see the newly added Data Connection item as an icon in the *Design View*, and also as an entry in the *Project* dock widget, *Items* list, under the 'Data Connections' category. It should look similar to this:



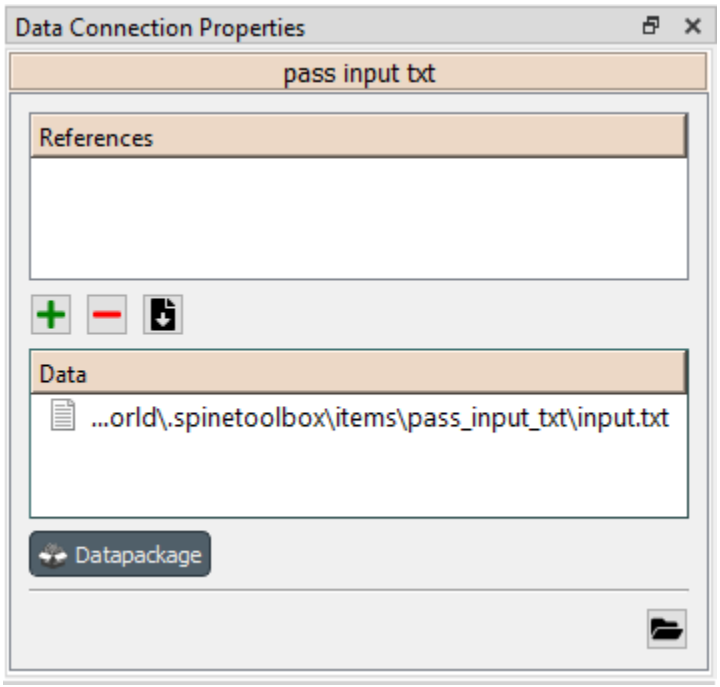
1.8 Adding data files to a Data Connection

As long as the 'pass input txt' Data Connection item is selected, you will be able to see its *Properties* on the right part of the window, looking similar to this:



Right click anywhere within the *Data* box and select **New file...** from the context menu. When prompted to enter a name for the new file, type 'input.txt' and click **Ok**.

Now you should see the newly created file in the *Data* list:



Double click on this file to open it in your default text editor. Then enter the following into the file's content:

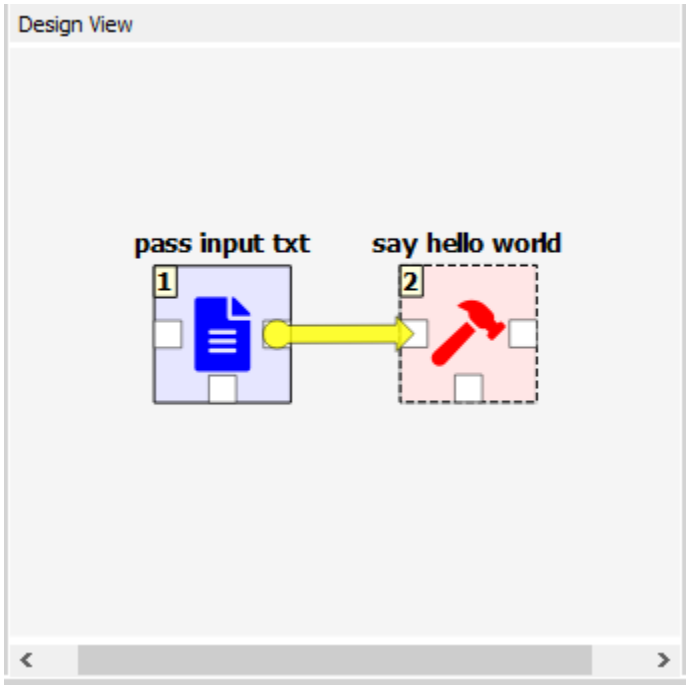
```
Hello again, World!
```

Save the file.

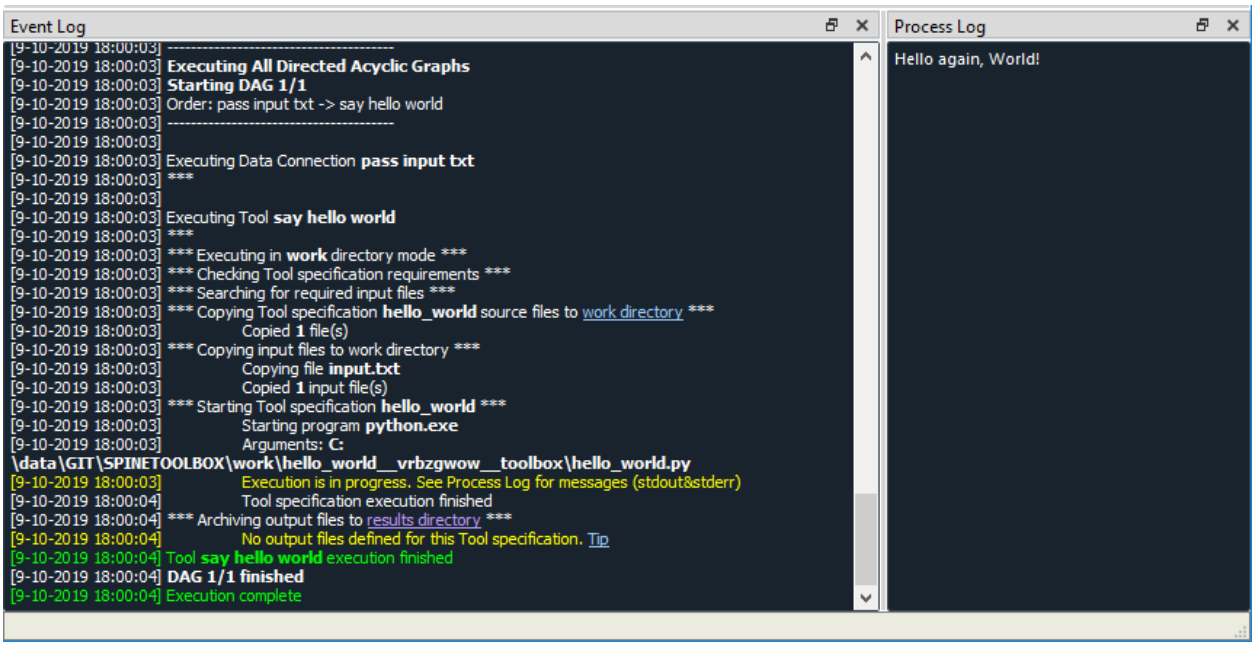
1.9 Connecting project items

As mentioned above, a Tool item looks for input files in Data Connection and Data Store items connected to its input. Thus, what we need to do now is create a *connection* from 'pass input txt' to 'say hello world', so the file 'input.txt' gets passed.

To do this, click on one of the *connector* slots at the edges of 'pass input txt' in the *Design view*, and then on a similar slot in 'say hello world'. This will create an arrow pointing from one to another, as seen below:



Press on the toolbar. The Tool will run successfully this time:



That's all for now. I hope you've enjoyed following this guide as much as I enjoyed writing it. See you next time.

CHAPTER 2

Setting up External Tools

This section describes how to set up Python, Julia, GAMS, and SpineOpt.jl for Spine Toolbox.

Executing Python or Julia Tools requires that Python or Julia are installed on your system. You can download Python from https://www.python.org/downloads/ and Julia from https://julialang.org/downloads/. In addition, you need an installation of GAMS to execute GAMS Tools and Exporter project items. GAMS can be downloaded from https://www.gams.com/download/.

2.1 Setting up Python

If you encounter the following message in Event Log when trying to execute a Python Tool.:

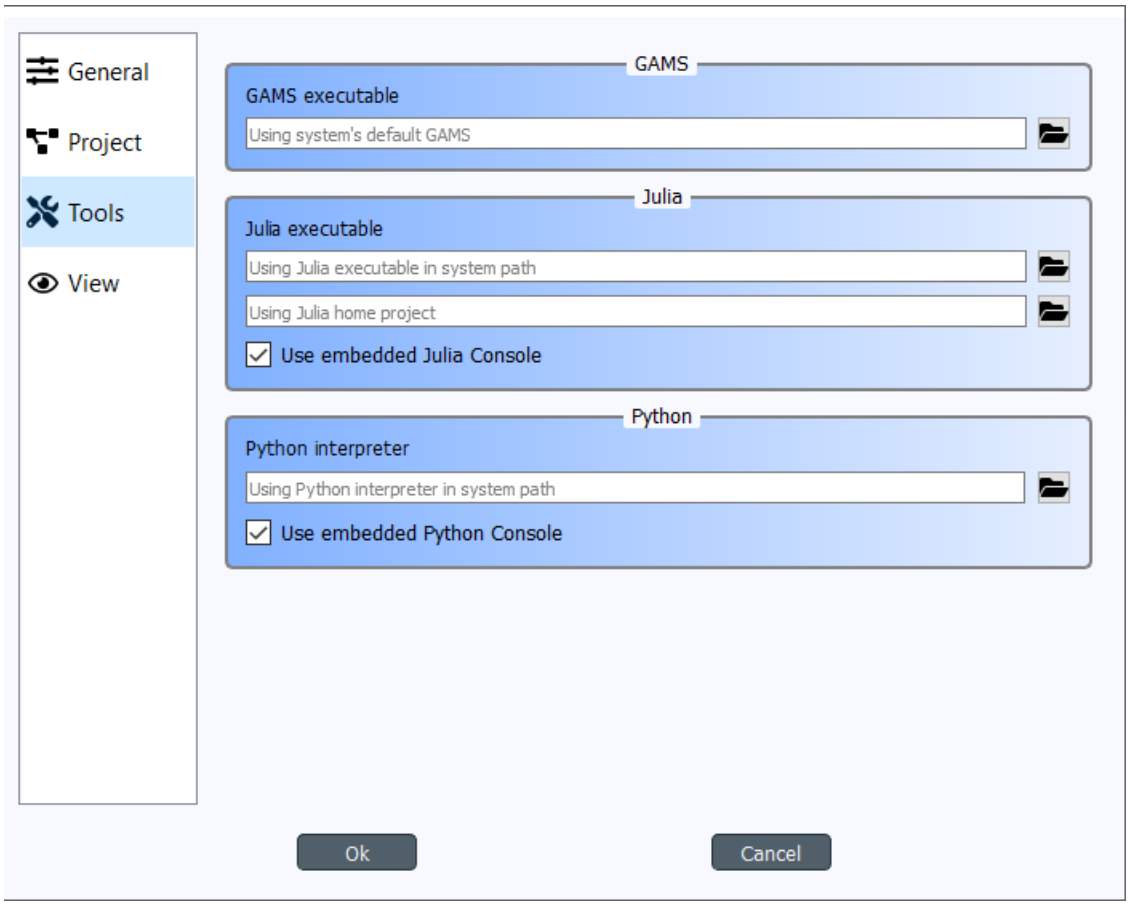
```
Couldn't determine Python version. Please check the Python interpreter option in␣
↪Settings.
```

After reading this section, you should know what this message means and how to set up Python for Spine Toolbox so you can successfully execute Python Tools.

There are two ways you can install Spine Toolbox.

1. Clone Spine Toolbox repository from https://github.com/Spine-project/Spine-Toolbox, checkout the branch you want and follow installation instructions on the page (README.md).
2. [On Windows] Use a single-file installation bundle (e.g. *spine-toolbox-0.4.0-x64.exe*). These are available for download in [Spine Toolbox Release Archive]

If you go with option 1, and you have successfully started the application, you already have a Python that can be used in executing Python Tools. If you go with option 2, you need to have a Python installed on your system to be able to execute Python Tools. You can select the Python you want to use on the Tools tab in Settings (See *Settings*).



The Python interpreter you select here is the Python that is used when executing Python Tools with or without the Embedded Python Console.

The default Python interpreter is the Python that is in your PATH environment variable. If you do not have Python in your PATH, you can explicitly set the Python you want to use by clicking on the button and selecting the Python interpreter file (*python.exe* on Windows). Note that you can use any Python in your system by setting a Python interpreter here.

Note: Embedded Python Console supports Python versions from 2.7 all the way to latest ones (3.8). Executing

Python Tools without using the embedded Python Console possibly supports even earlier Pythons than 2.7. You can start Spine Toolbox only with Python 3.6 or with 3.7, but you can still set up an embedded Python Console into Spine Toolbox that uses e.g. Python 2.7. This means, that if you still have some old Python 2.7 scripts lying around, you can incorporate those into a Spine Toolbox project and execute them without any modifications.

2.1.1 Step-by-step instructions

You can either execute Python Tools in the embedded Python Console or as in the shell. Here are the step-by-step instructions for setting up Spine Toolbox for both.

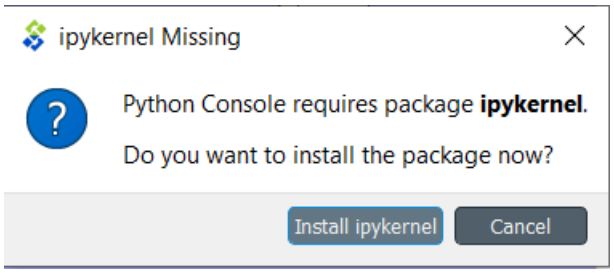
Shell execution

1. Go to https://www.python.org/downloads/ and download the Python you want
2. Run the Python installer and follow instructions
3. Either let the installer put Python in your PATH or memorize the path where you installed it (e.g. *C:\Python38*)
4. Start Spine Toolbox
5. Go to File -> Settings (or press F1) and click the Tools tab open
6. If the installed Python is now in your PATH, you can leave the Python interpreter line edit blank. Or you can set the Python interpreter explicitly by setting it to e.g. *C:\Python38\python.exe* by using the button.
7. Uncheck the *Use embedded Python Console* check box
8. Create a project with a Tool and a Python Tool specification (See *Getting Started*)
9. Press play to execute the project (See *Executing Projects*)
10. Executing your Tool project item starts. You can see the output (stdout and stderr) in the Process Log.

Python Console execution

If you want to use the embedded Python Console (and you should). There is an extra step involved since the Python Console requires a couple of extra packages (*ipykernel* and its dependencies) to be installed on the selected Python. In addition, kernel specifications for the selected Python need to be installed beforehand. **Spine Toolbox can install these for you automatically.**

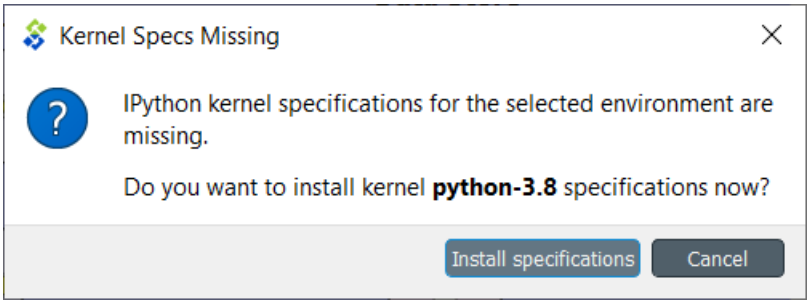
1. Go to https://www.python.org/downloads/ and download the Python you want
2. Run the Python installer and follow instructions
3. Either let the installer put Python in your PATH or memorize the path where you installed it (e.g. *C:\Python38*)
4. Start Spine Toolbox
5. Go to File -> Settings (or press F1) and click the Tools tab open
6. If the installed Python is now in your PATH, you can leave the Python interpreter line edit blank. Or you can set the Python interpreter explicitly by setting it to e.g. *C:\Python38\python.exe* by using the button.
7. Check the *Use embedded Python Console* check box
8. Create a project with a Tool and a Python Tool specification (See *Getting Started*)
9. Press play to execute the project (See *Executing Projects*)
10. You will see a question box



When you click on the *Install ipykernel* button, you can see the progress of the operation in Process Log. The following packages will be installed on your selected Python.:

```
backcall, colorama, decorator, ipykernel, ipython, ipython-genutils, jedi, jupyter-
↪client,
jupyter-core, parso, pickleshare, prompt-toolkit, pygments, python-dateutil, pywin32,␣
↪pyzmq, six,
tornado, traitlets, wcwidth
```

When this operation finishes successfully, you will see another guestion box.



Clicking on *Install specifications* button starts installing the kernel specs for the selected Python. On the tested system, this creates a new kernel into directory *C:\Users\ttepsa\AppData\Roaming\jupyter\kernels\Python-3.8*, which contains the *kernel.json* file required by the embedded Python Console (which is actually a jupyter qtconsole)

11. After the kernel specs have been installed, executing your Tool project item starts in the Python Console immediately. You can see the executed command and the Tool output in the Python Console.

Note: If you want to set up your Python environment ready for Python Console manually, the following commands are executed by Spine Toolbox under the hood

This installs all required packages:

```
python -m pip install ipykernel
```

And this installs the kernel specifications:

```
python -m ipykernel install --user --name python-3.8 --display-name Python3.8
```

2.1.2 What about Anaconda and Miniconda Pythons?

If you installed Spine Toolbox on a Conda environment, the Python you started Spine Toolbox with has been added to the conda environment variables. This means that you are ready to execute Python Tools without using the embedded Python Console out of the box. For setting up the Python Console you just need to let Spine Toolbox install the

ipykernel package and the kernel specifications for this Python. See section *Python Console execution* above for more info.

2.2 Setting up Julia

Spine Toolbox requires a Julia installation that must be set up before Julia Tools can be executed. The basic idea is the same as with Python. In File->Settings (Tools tab), there's a line edit for the Julia executable. If you leave this blank, Spine Toolbox uses the Julia that is in your PATH environment variable. Setting an explicit path to a Julia executable (e.g. *C:\Julia-1.2.0\bin\julia.exe*) overrides the Julia in PATH. As with Python Tools, you execute Julia Tools in the embedded Julia Console or without it (shell execution).

If you see this (or similar) message in Event Log when trying to execute a Julia Tool.:

```
julia.exe failed to start. Make sure that Julia is installed properly on your␣
↪computer.
```

This means that you either don't have a Julia installation on your system, Julia is not set up in your PATH environment variable or the Julia executable you have set in Settings is not valid.

2.2.1 Step-by-step instructions

Shell execution

1. Go to https://julialang.org/downloads/ and download the Julia you want
2. Run the Julia installer and follow instructions
3. Either let the installer put Julia in your PATH or memorize the path where you installed it (e.g. *C:\Julia-1.2.0*)
4. Start Spine Toolbox
5. Go to File -> Settings (or press F1) and click the Tools tab open
6. If the installed Julia is now in your PATH, you can leave the Julia executable line edit blank. Or you can set the Julia executable explicitly by setting it to e.g. *C:\Julia.1.2.0\bin\julia.exe* by using the button.
7. Uncheck the *Use embedded Julia Console* check box
8. Create a project with a Tool and a Julia Tool specification (See *Getting Started*)
9. Press play to execute the project (See *Executing Projects*)
10. Executing your Tool project item starts. You can see the output (stdout and stderr) in the Process Log.

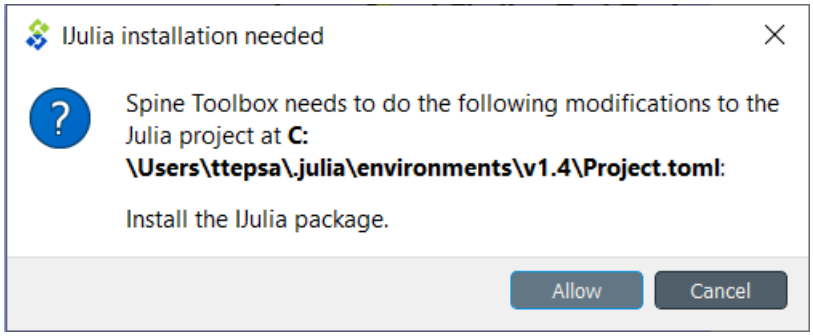
Julia Console execution

Like the Python Console, Julia Console requires some extra setting up. **Spine Toolbox can set this up for you automatically**.

If you want to use the embedded Julia Console (and you should). There is an extra step involved since the Julia Console requires a couple of extra packages (*IJulia*, etc.) to be installed and built.

1. Go to https://julialang.org/downloads/ and download the Julia you want
2. Run the Julia installer and follow instructions
3. Either let the installer put Julia in your PATH or memorize the path where you installed it (e.g. *C:\Julia-1.2.0*)

4. Start Spine Toolbox
5. Go to File -> Settings (or press F1) and click the Tools tab open
6. If the installed Julia is now in your PATH, you can leave the Julia executable line edit blank. Or you can set the Julia executable explicitly by setting it to e.g. *C:\Julia.1.2.0\bin\julia.exe* by using the button.
7. Check the *Use embedded Julia Console* check box
8. Create a project with a Tool and a Julia Tool specification (See *Getting Started*)
9. Press play to execute the project (See *Executing Projects*)
10. You will see a question box



When you click on the *Allow* button, installing IJulia starts and you can see the progress of the operation in Process Log. **This may take a few minutes**.

When you see the these messages in the Event Log, the Julia Console is ready to be used.:

```
IJulia installation successful.
*** Starting Julia Console ***
```

11. After the installation has finished, executing your Julia Tool project item starts in the Julia Console immediately. You can see the executed command and the Tool output in the Julia Console. If nothing seems to be happening in the Julia Console. Just click button and then try executing the project again by clicking the button.

Note: If you want to set up your Julia environment ready for Julia Console manually, you need to install IJulia and the Julia kernel specifications.

2.3 Setting up GAMS

Executing a GAMS Tool project item or executing an Exporter project item requires a GAMS installation on your system.

Note: You do not need to own a GAMS license as the demo version works just as well.

Note: The bitness (32 or 64bit) of GAMS has to match the bitness of the Python interpreter.

If you have GAMS in your PATH environment variable, you can leave the GAMS executable line edit in File->Settings blank and Spine Toolbox will find it. You can also override the GAMS in your PATH by setting an explicit path to the GAMS executable (e.g. *C:\GAMS\win64\28.2\gams.exe*) line edit.

2.4 Setting up SpineOpt.jl

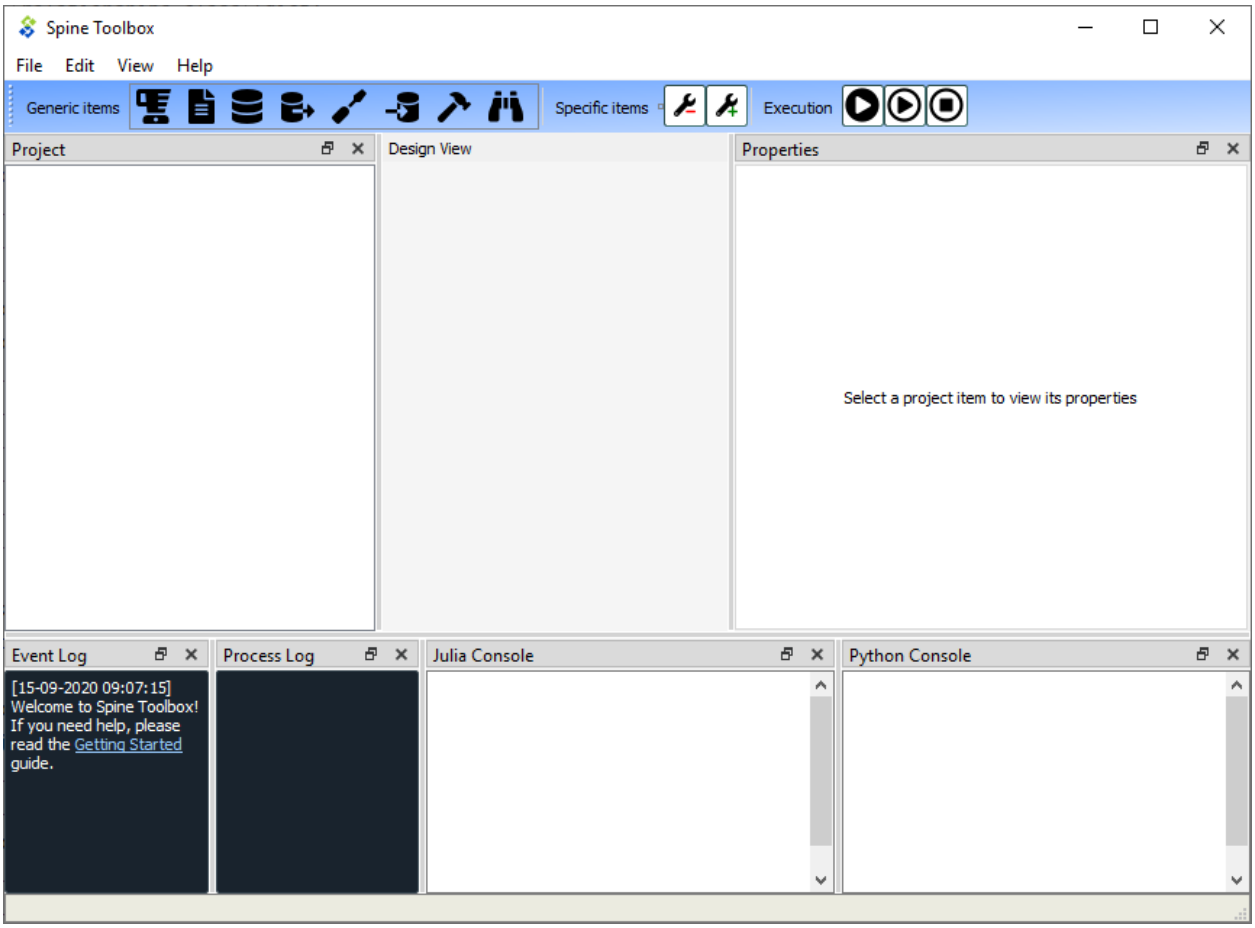
There's a built-in configuration assistant in Spine Toolbox that downloads and configures SpineOpt.jl automatically. You can find the configuration assistant in the main window menu **File->Tool configuration assistants...->SpineOpt.jl** Before you run this, you need to set up Julia for Spine Toolbox. See instructions above (*Setting up Julia*). After a Julia has been set up correctly, run the Tool configuration assistant and follow the instructions given.

CHAPTER 3

Main Window

This section describes the different components in the application main window.

The first time you start the application you will see the main window like this.



The application main window contains six dock widgets (*Project*, *Properties*, *Event Log*, *Process Log*, *Julia Console*, and *Python Console*), a tool bar, a *Design View*, and a menu bar with *File*, *Edit*, *View*, and *Help* menus. The *Project* dock widget contains a list of project items and Tool specifications that are available in your project. The *Properties* dock widget shows the properties of the selected project item. *Event Log* shows messages depending on what you do in Spine Toolbox. *Process Log* shows messages from processes that are spawned by the application, i.e. it shows the stdout and stderr streams of GAMS, Julia, Python (if Tools are executed without embedded Julia and Python Consoles, see *Settings* section), and executable programs. Julia and Python Consoles provide full iJulia and a iPython consoles. If you choose to execute Julia tools in the embedded Julia Console, the Julia code will be included into the Julia Console and executed there. You can interact with the iJulia in the Julia Console like you would with any iJulia you use.

Tip: You can configure the Julia and Python versions you want to use in `File->Settings`.

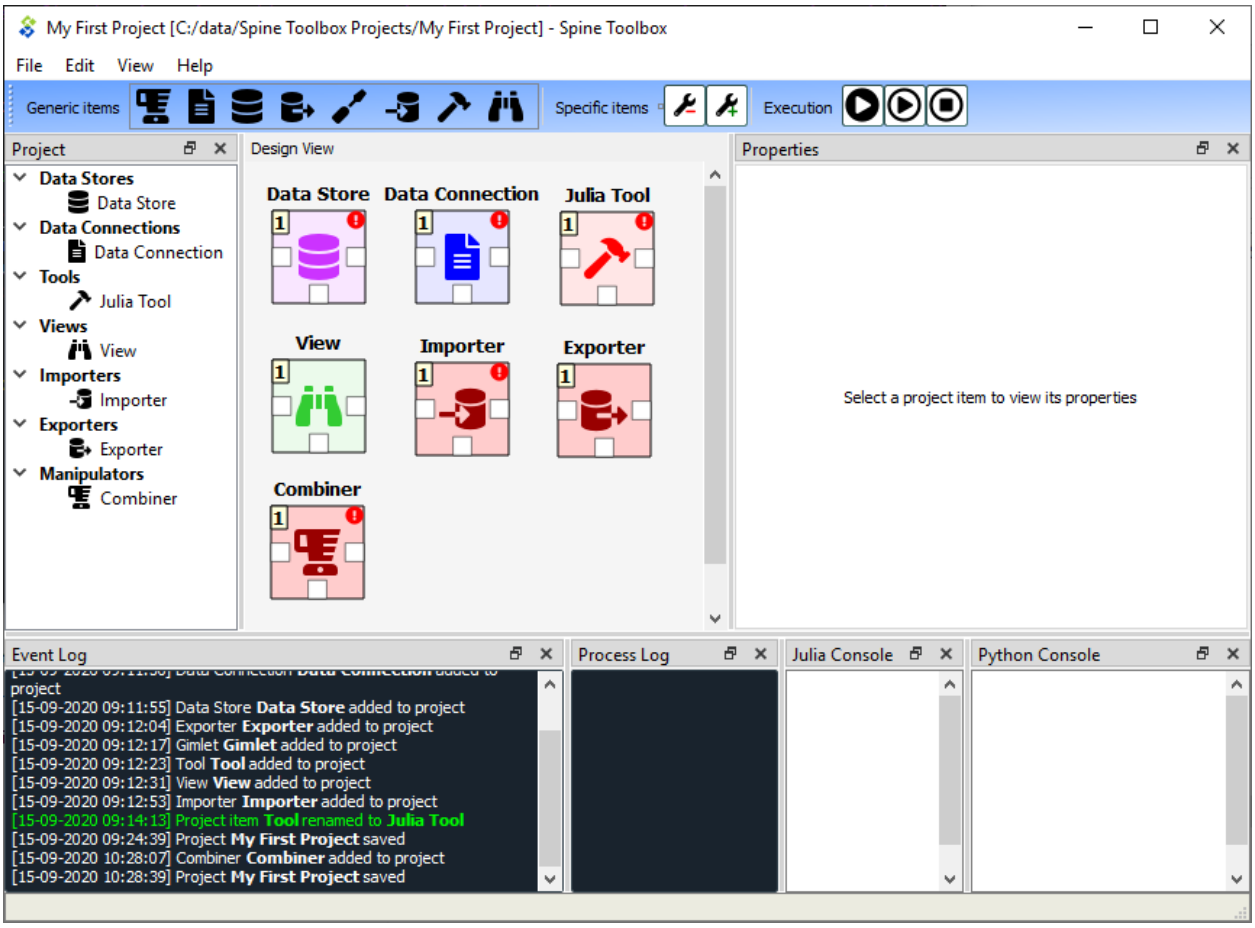
The menu bar in the top of the application contains *File*, *Edit*, *View*, and *Help* menus. In the *File* menu you can create a new project, open an existing project, save the project, upgrade an old project to modern directory-based project, and open the application Settings among other things. Spine Toolbox is project based, which means that you need to create a new project or open an existing one before you can do anything. You can create a new project by selecting `File->New project...` from the menu bar. *Drag & Drop Icon* tool bar contains the available *project item* types. The button can be used to remove all items from your project. The *Execute* icons control the execution of the items in the *Design view* where you build your project. The button executes all Directed-Acyclic Graphs (DAG) in the project in a row. The button executes the selected project items only. The button terminates the execution (if running).

You can add a new project item to your project by pointing your mouse cursor on any of the draggable items in the *Drag & Drop Icon* tool bar, then click-and-drag the item on to the *Design view*. After this you will be presented a dialog, which asks you to fill in basic information about the new project item (name, description, etc.).

The main window is very customizable so you can e.g. close the dock widgets that you do not need and/or you can resize the views to fit your needs and display size or resolution.

Note: If you want to restore all dock widgets to their default place use the menu item `View->Dock Widgets->Restore Dock Widgets`. This will show all hidden dock widgets and restore them to the main window.

Below is an example on how you can customize the main window. In the picture, a user has created a project *My First Project*, and created one project item from each of the seven categories. A Data Store called *Database*, a Data Connection called *Data files*, A Tool called *Julia model*, a View called *View*, an Importer called *Importer*, an Exporter called *Exporter*, and a Manipulator called *Combiner*. The project items are also listed in the *Project* dock widget.


My First Project [C:/data/Spine Toolbox Projects/My First Project] - Spine Toolbox
File
Edit
View
Help
Generic items
Specific items
Execution
Project
Data Stores
Data Store
Data Connections
Data Connection
Tools
Julia Tool
Views
View
Importers
Importer
Exporters
Exporter
Manipulators
Combiner
Design View
Data Store
Data Connection
Julia Tool
View
Importer
Exporter
Combiner
Properties
Select a project item to view its properties
Event Log
project
[15-09-2020 09:11:55] Data Store Data Store added to project
[15-09-2020 09:12:04] Exporter Exporter added to project
[15-09-2020 09:12:17] Gimlet Gimlet added to project
[15-09-2020 09:12:23] Tool Tool added to project
[15-09-2020 09:12:31] View View added to project
[15-09-2020 09:12:53] Importer Importer added to project
[15-09-2020 09:14:13] Project item Tool renamed to Julia Tool
[15-09-2020 09:24:39] Project My First Project saved
[15-09-2020 10:28:07] Combiner Combiner added to project
[15-09-2020 10:28:39] Project My First Project saved
Process Log
Julia Console
Python Console

CHAPTER 4

Project Items

Project items in the *Design view* and the connections between them make up the graph (Directed Acyclic Graph, DAG) that is executed when the or buttons are pressed.

See *Executing Projects* for more information on how a DAG is processed by Spine Toolbox. Those interested in looking under the hood can check the *Project item development* section.

4.1 Project Item Properties

Each project item has its own set of *Properties*. You can view and edit them by selecting a project item on the *Design View*. The Properties are displayed in the *Properties* dock widget on the main window. Project item properties are saved into the project save file (`project.json`), which can be found in `<proj_dir>/.spinetoolbox/` directory, where `<proj_dir>` is your current project directory.

In addition, each project item has it's own directory in the `<proj_dir>/.spinetoolbox/items/` directory. You can quickly open the project item directory in a file explorer by clicking on the button located in the lower right corner of each *Properties* form.

4.2 Project Item Descriptions

The following items are currently available:

4.2.1 Data Store

A Data store item represents a connection to a (Spine) database. Currently, the item supports sqlite and mysql dialects. The database can be accessed and modified in *Spine db editor* available from the item's properties or from a right-click context menu.

4.2.2 Data Connection

A Data connection item provides access to data files. It also provides access to the *Datapackage editor*.

4.2.3 Tool

Tool is the heart of a DAG. It is usually the actual model to be executed in Spine Toolbox but can be an arbitrary script or executable as well. A tool is specified by its *specification*.

4.2.4 Gimlet

While being able to run most scripts and copyable executables, Tool cannot handle system commands or executables meant to run from system's *path*. This is a job for Gimlet. A Gimlet can execute an arbitrary system command with given command line arguments, input files and work directory.

4.2.5 View

A View item is meant for inspecting data from multiple sources using the *Spine db editor*. Note that the data is opened in read-only mode so modifications are not possible from the View item.

4.2.6 Combiner

A Combiner item can be used to combine two or more databases into one.

4.2.7 Importer

This item provides the user a chance to define a mapping from tabulated data such as comma separated values or Excel to the Spine data model. See *Importing and exporting data* for more information.

4.2.8 Exporter

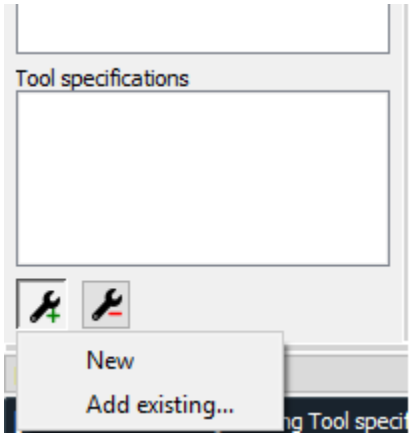
This item exports databases contained in a *Data Store* into `.gdx` format for GAMS Tools. See *Importing and exporting data* for more information.

CHAPTER 5

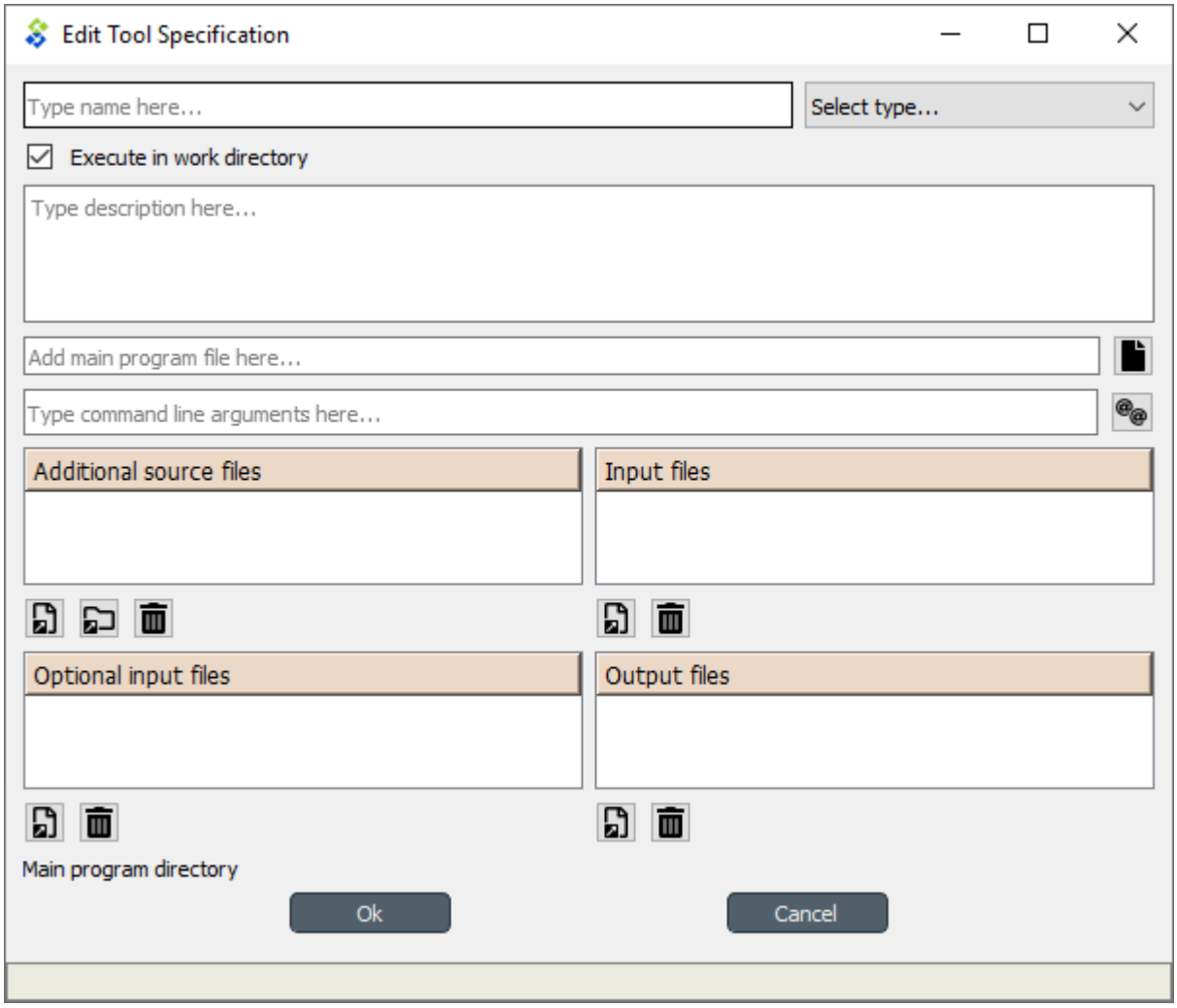
Tool specification editor

This section describes how to make a new Tool specification and how to edit existing Tool specifications.

To execute a Julia, Python, GAMS, or an executable script in Spine Toolbox, you must first create a Tool specification to your project. You can open the Tool specification editor in several ways. One way is to press the Tool icon with a plus button in the *Project* dock widget. This presents you a pop-up menu with the *New* and *Add existing...* options.



When you click on *New* the following form pops up.



Start by giving the Tool specification a name. Then select the type of the Tool. You have four options (Julia, Python, GAMS or Executable). Then select, whether you want the Tool specification to be executed in the work directory or in its source directory (See *Terminology* section). You can give the Tool specification a description, describing what the Tool specification does. Main program file is the main file of your simulation model, or an executable script. You can create a blank file into a new directory by pressing the button and selecting *Make new main program* or you can browse to find an existing main program file by pressing the same button and selecting *Select existing main program*.

Command line arguments can be appended to the actual command that Spine Toolbox executes in the background. For example, you may have a Windows batch file called *do_things.bat*, which accepts command line arguments *a* and *b*. Writing `a b` on the command line arguments field in the tool specification editor is the equivalent of running the batch file in command prompt with the command `do_things.bat a b`. See *Command line argument tag expansion* for more information on the command line arguments.

Additional source files is a list of files that the main program requires in order to run. You can add individual files or whole directories at once to this list.

Tip: You can also drag&drop a directory from your operating systems File Explorer into the *Additional source files* list.

Input files is a list of input data files that the program **requires** in order to execute. You can also add directories and subdirectories. Wildcards are **not** supported (see Optional input files).

Examples:

- **data.csv** -> File is copied to the same work directory as the main program
- **input/data.csv** -> Creates directory input/ to the work directory and copies file *data.csv* there
- **output/** -> Creates an empty directory output/ into the work directory

Optional input files are files that may be utilized by your program if they are found. Unix-style wildcards *?* and * are supported.

Examples:

- **data.csv** -> If found, file is copied to the same work directory as the main program
- ***.csv** -> All found .csv files are copied to the same work directory as the main program
- **input/data_?.dat** -> All found files matching the pattern *data_?.dat* are copied into input/ directory in the work directory.

Output files are files that will be archived into a timestamped result directory of the Tool's project directory after the Tool specification has finished execution. Unix-style wildcards *?* and * are supported.

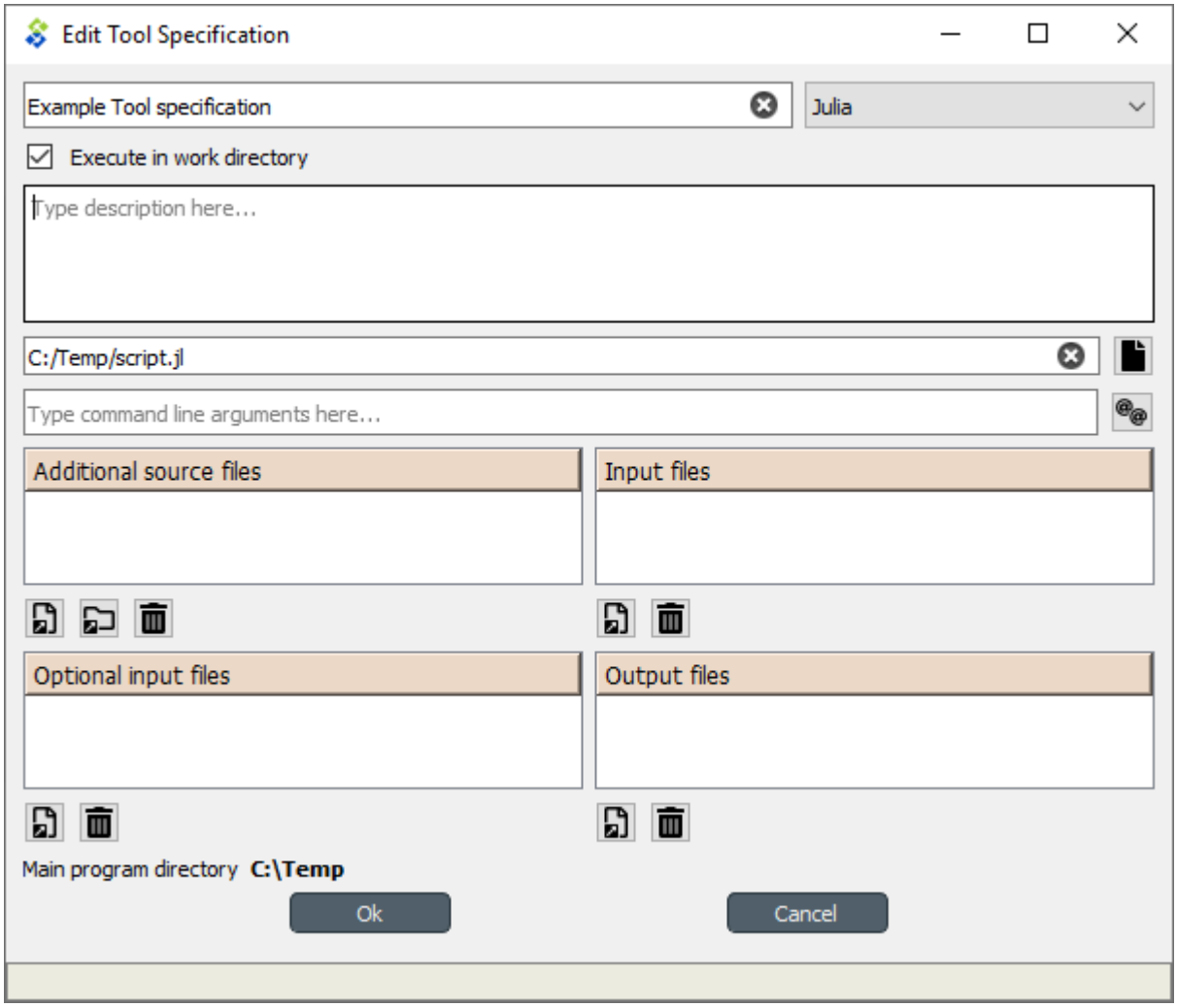
Examples:

- **results.csv** -> File is copied from work directory into results directory
- ***.csv** -> All .csv files from work directory are copied into results directory
- **output/*.gdx** -> All GDX files from the work directory's output/ subdirectory will be copied to into output/ subdirectory in the results directory.

When you are happy with your Tool specification, click Ok, and you will be asked where to save the Tool specification file. It is recommended to save the file into the same directory where the main program file is located. The Tool specification file is a text file in JSON format and has an extension *.json*

Tip: Only *name*, *type*, and *main program file* fields are required to make a Tool specification. The other fields are optional.

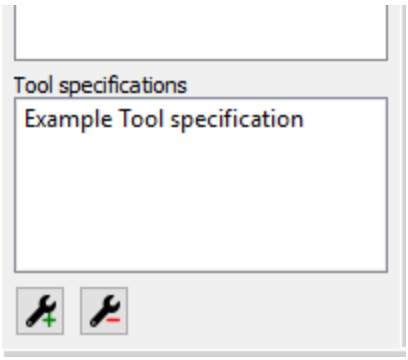
Here is a minimal Tool specification for a Julia script *script.jl*



Note: Under the hood, the contents of the Tool specification are saved to a *Tool specification file* in JSON format. Users do not need to worry about the contents of these files since reading and writing them is managed by the app. For the interested, here are the contents of the *Tool specification file* that we just created.:

```
{
    "name": "Example Tool specification",
    "description": "",
    "tooltype": "julia",
    "execute_in_work": true,
    "includes": [
        "script.jl"
    ],
    "inputfiles": [],
    "inputfiles_opt": [],
    "outputfiles": [],
    "cmdline_args": ""
}
```

After the user has clicked Ok and saved the file, the new Tool specification has been added to the project.

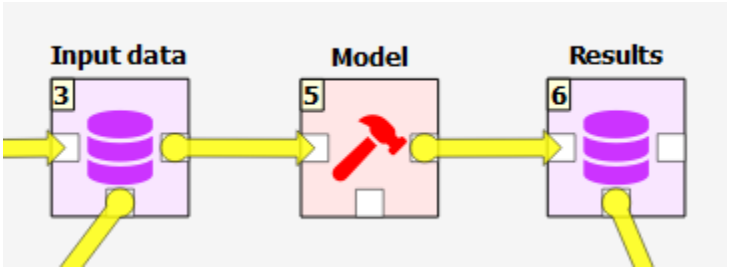


To edit this Tool specification, just right-click on the Tool specification name and select *Edit Tool specification* from the context-menu.

You are now ready to execute the Tool specification in Spine Toolbox. You just need to select a Tool item in the *Design view*, set the specification *Example Tool specification* to it, and click or button.

5.1 Command line argument tag expansion

Spine Toolbox supports a number of special command line arguments called *tags* that get replaced by information relevant to a Tool's current connections. For example, the `@@url-inputs@@` tag expands to a list of input database URLs. If the command line arguments for the *Model* tool in the image below were `--input-database=@@url-inputs@@` the tool would be executed by `python tool_script.py --input_database=sqlite:///input_database.sqlite` command in case *Input data*'s database URL was `sqlite:///input_database.sqlite`.



The button next to the command line arguments field in Tool Specification editor gives a quick access to insert the tags into the field.

Below is a list of the command line argument tags that are currently available:

- `@@url_inputs@@`: a space separated list of database URLs provided by all input data stores.
- `@@url_outputs@@`: a space separated list of database URLs provided by all output data stores.
- `@@url:<data store name>@@`: the url provided by a named data store connected to the tool.
- `@@optional_inputs@@`: a space separated list of tool's optional input files.

CHAPTER 6

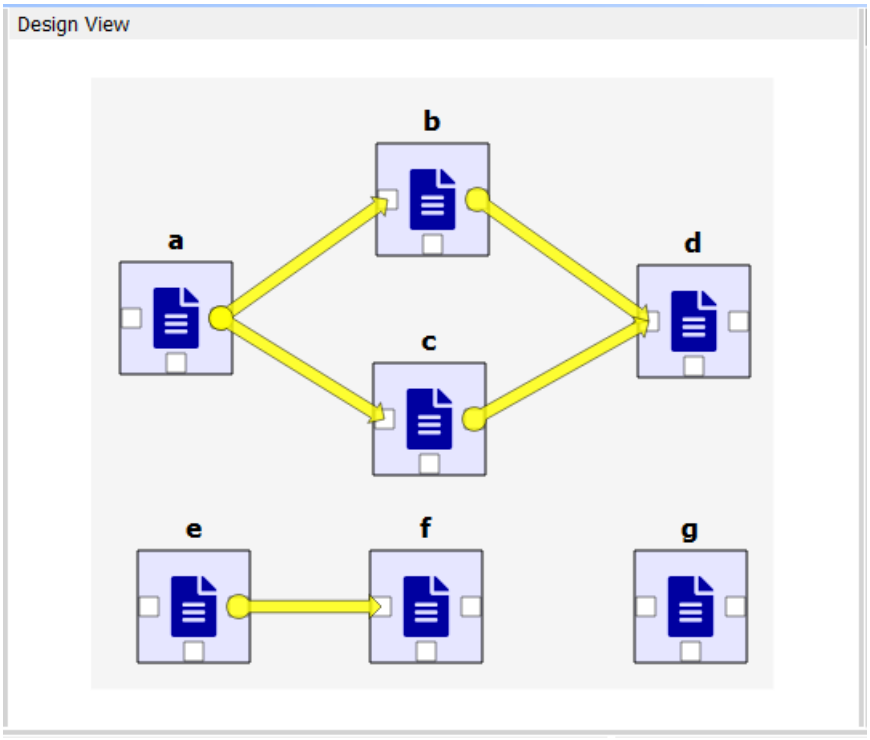
Executing Projects

This section describes how executing a project works and what resources are passed between project items at execution time. Execution happens by pressing the (Execute project) or the (Execute selection) buttons in the main window tool bar. A project consists of project items and connections (yellow arrows) that are visualized on the *Design View*. You use the project items and the connections to build a **Directed Acyclic Graph (DAG)**, with the project items as *nodes* and the connections as *edges*. A DAG is traversed using the **breadth-first-search** algorithm.

Rules of DAGs:

1. A single project item with no connections is a DAG.
2. All project items that are connected, are considered as a single DAG (no matter, which direction the arrows go). If there is a path between two items, they are considered as belonging to the same DAG.
3. Loops are not allowed (this is what acyclic means).

You can connect the nodes in the *Design View* how ever you want but you cannot execute the resulting DAGs if they break the rules above. Here is an example project with three DAGs.



- DAG 1: items: a, b, c, d. connections: a-b, a-c, b-d, c-d
- DAG 2: items: e, f. connections: e-f
- DAG 3: items: g. connections: None

When you press the button, all three DAGs are executed in a row. You can see the progress and the current executed item in the *Event Log*. Execution order of DAG 1 is *a->b->c->d* or *a->c->b->d* since items b and c are **siblings**. DAG 2 execution order is *e->f* and DAG 3 is just *g*. If you have a DAG in your project that breaks the rules above, that DAG is skipped and the execution continues with the next DAG.

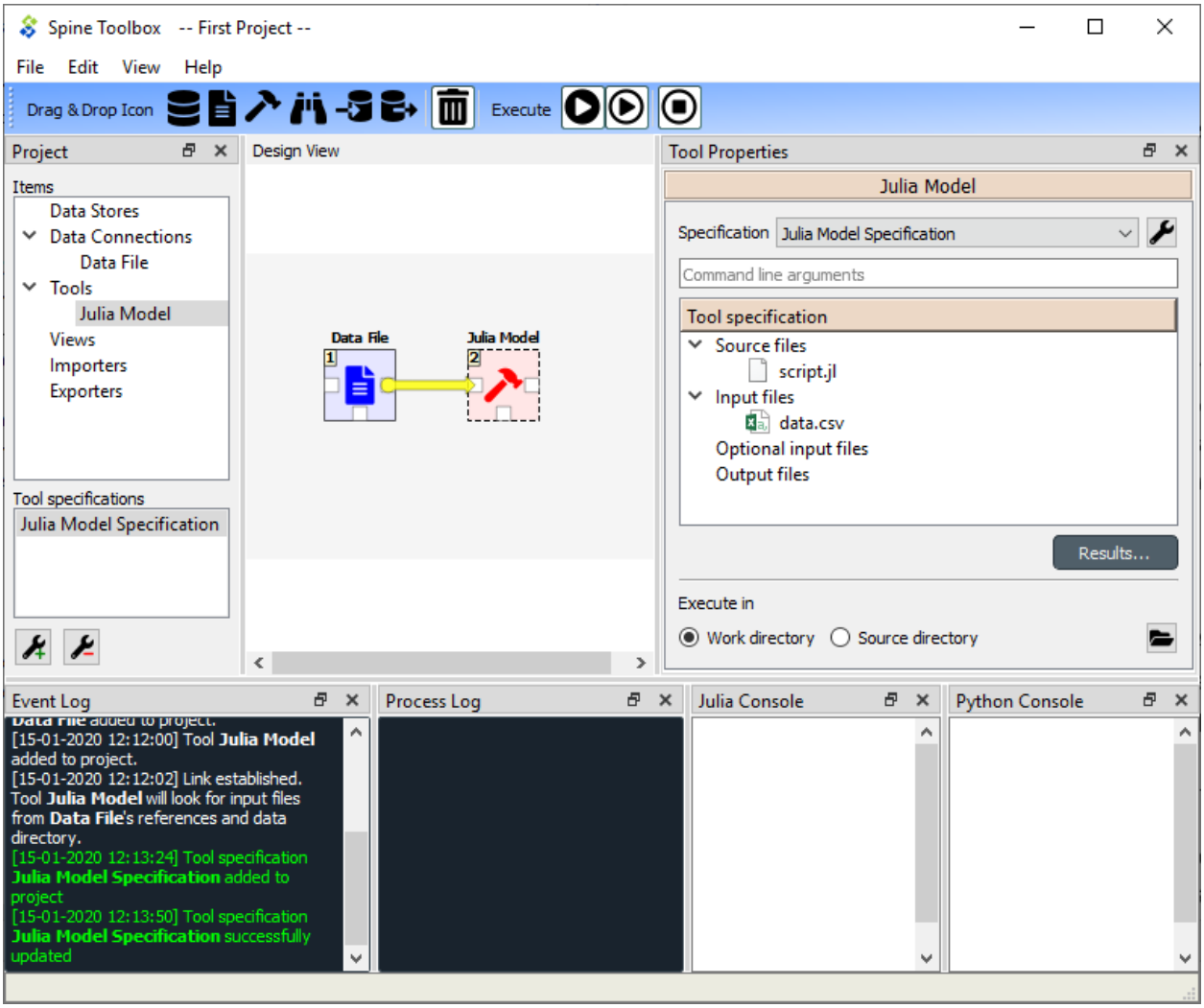
We use the words **predecessor** and **successor** to refer to project items that are upstream or dowstream from a project item. **Direct predecessor** is a project item that is the immediate predecessor. **Direct Successor** is a project item that is the immediate successor. For example, in DAG 1 above, the successors of *a* are project items *b*, *c* and *d*. The direct successor of *b* is *d*. The predecessor of *b* is *a*, which is also its direct predecessor.

You can also execute only the selected parts of a project by multi-selecting the items you want to execute and pressing the button in the tool bar. For example, to execute only items *b*, *d* and *f*, select the items in *Design View* or in the project item list in *Project* dock widget and then press the button.

Tip: You can select multiple project items by pressing the Ctrl-button down and clicking on desired items.

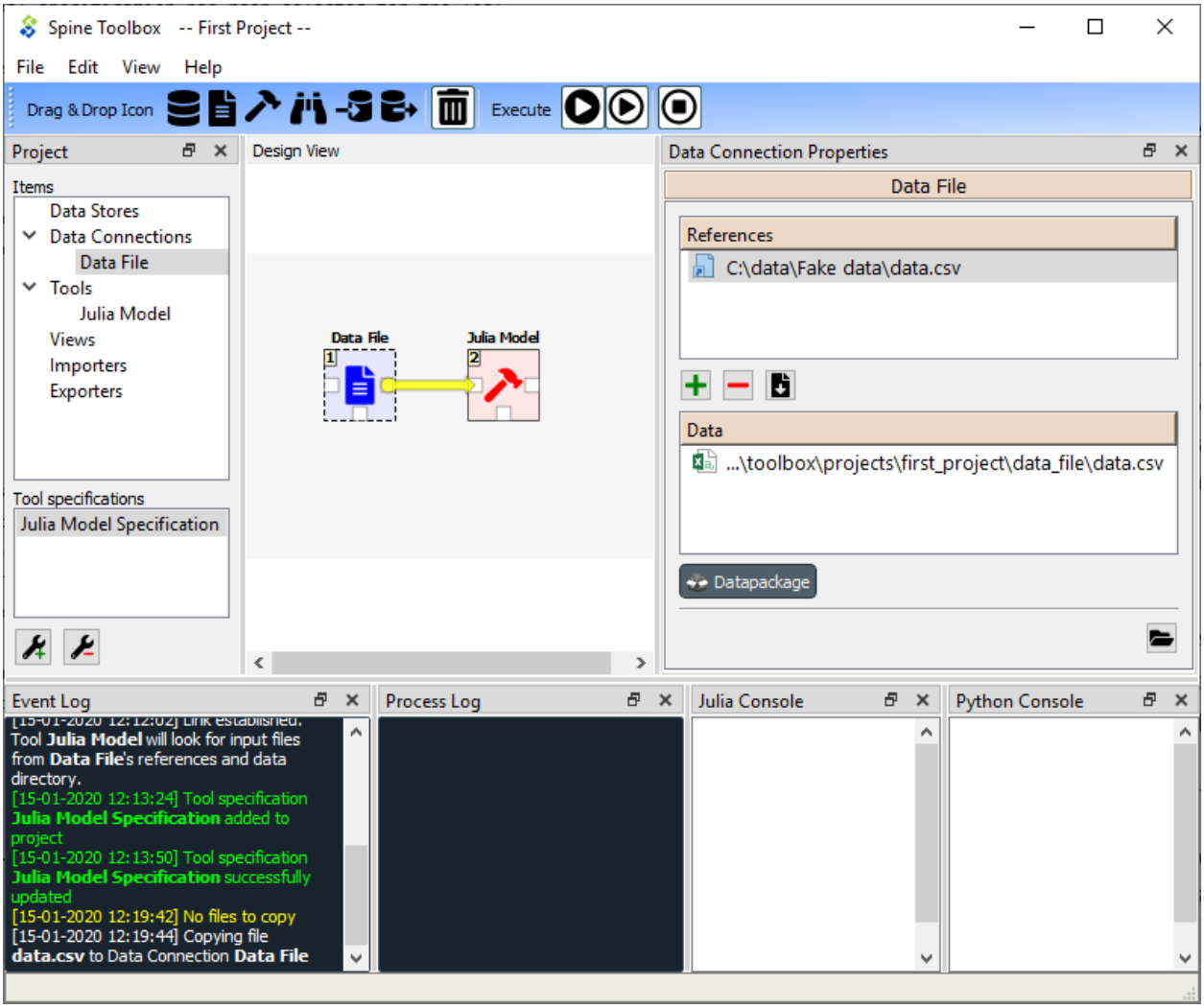
6.1 Example DAG

When you have created at least one Tool specification, you can execute a Tool as part of the DAG. The Tool specification defines the process that is depicted by the Tool project item. As an example, below we have two project items; *Julia Model* Tool and *Data File* Data Connection connected to each other.



Selecting the *Julia Model* shows its properties in the *Properties* dock widget. In the top of the Tool Properties, there is a specification drop-down menu. From this drop-down menu, you can select the Tool specification for this particular Tool item. The *Julia Model Specification* tool specification has been selected for the Tool *Julia Model*. Below the drop-down menu, you can see the details of the Tool specification, command line arguments, Source files (the first one is the main program file), Input files, Optional input files and Output files. *Results...* button opens the Tool's result archive directory in the File Explorer (all Tools have their own result directory). The *Execute in* radio buttons control, whether this Tool is first copied to a work directory and executed there, or if the execution should happen in the source directory where the main program file is located.

When you click on the button, the execution starts from the *Data File* Data Connection. When executed, Data Connection items *advertise* their files and references to project items that are in the same DAG and executed after them. In this particular example, the *Data File* item contains a file called *data.csv* as depicted in the picture below.

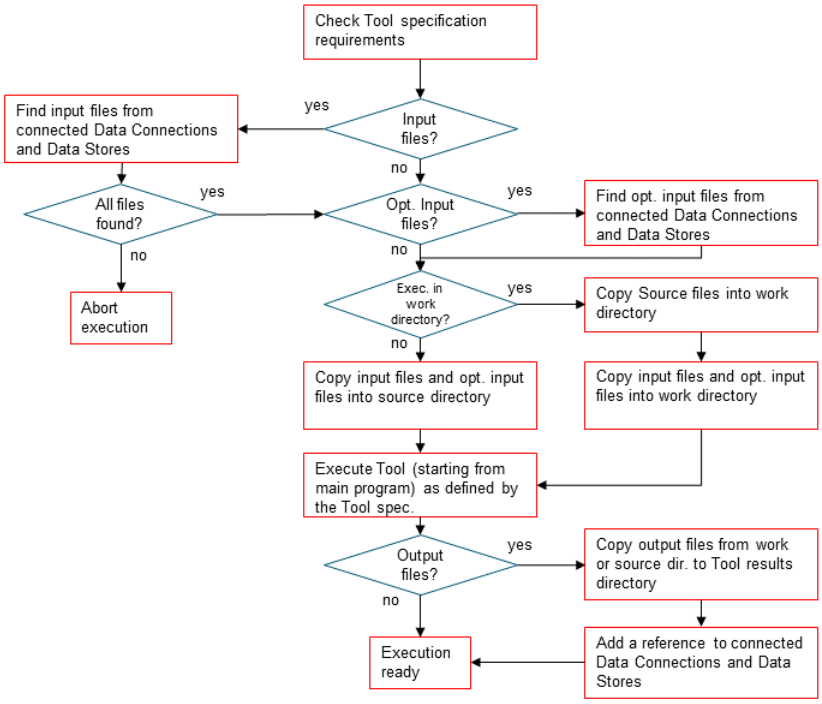


When it's the *Julia Model* tools turn to be executed, it checks if it finds the file *data.csv* from project items, that have already been executed. When the DAG is set up like this, the Tool finds the input file that it requires and then starts processing the Tool specification starting with the main program file *script.jl*. Note that if the connection would be the other way around (from *Julia Model* to *Data File*) execution would start from the *Julia Model* and it would fail because it cannot find the required file *data.csv*. The same thing happens if there is no connection between the two project items. In this case the project items would be in separate DAGs.

Since the Tool specification type was set as *Julia* and the main program is a Julia script, Spine Toolbox starts the execution in the Julia Console (if you have selected this in the application *Settings*, See *Settings* section).

6.2 Tool execution algorithm

The below figure depicts what happens when a Tool item with a valid Tool specification is executed.

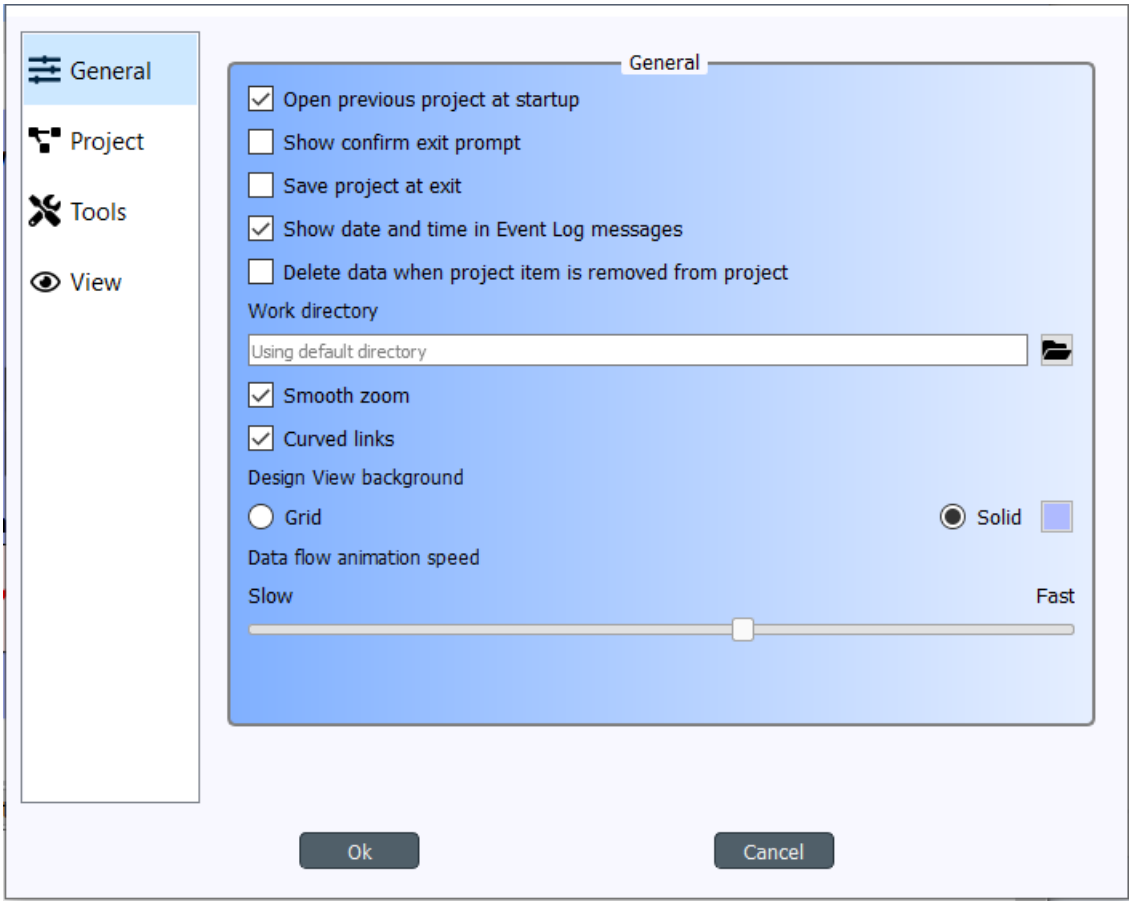

Check Tool specification requirements
Input files?
yes
Find input files from connected Data Connections and Data Stores
All files found?
yes
no
Abort execution
Opt. Input files?
yes
Find opt. input files from connected Data Connections and Data Stores
no
Exec. in work directory?
yes
Copy Source files into work directory
no
Copy input files and opt. input files into source directory
Copy input files and opt. input files into work directory
Execute Tool (starting from main program) as defined by the Tool spec.
Output files?
yes
Copy output files from work or source dir. to Tool results directory
no
Execution ready
Add a reference to connected Data Connections and Data Stores

CHAPTER 7

Settings

You can open Spine Toolbox settings from the main window menu `File->Settings...`, or by pressing **F1**. Settings are categorized into four tabs; *General*, *Project*, *Tool*, and *View*. In addition to application settings, each Project item has user adjustable properties (See *Project Items*)

7.1 General settings

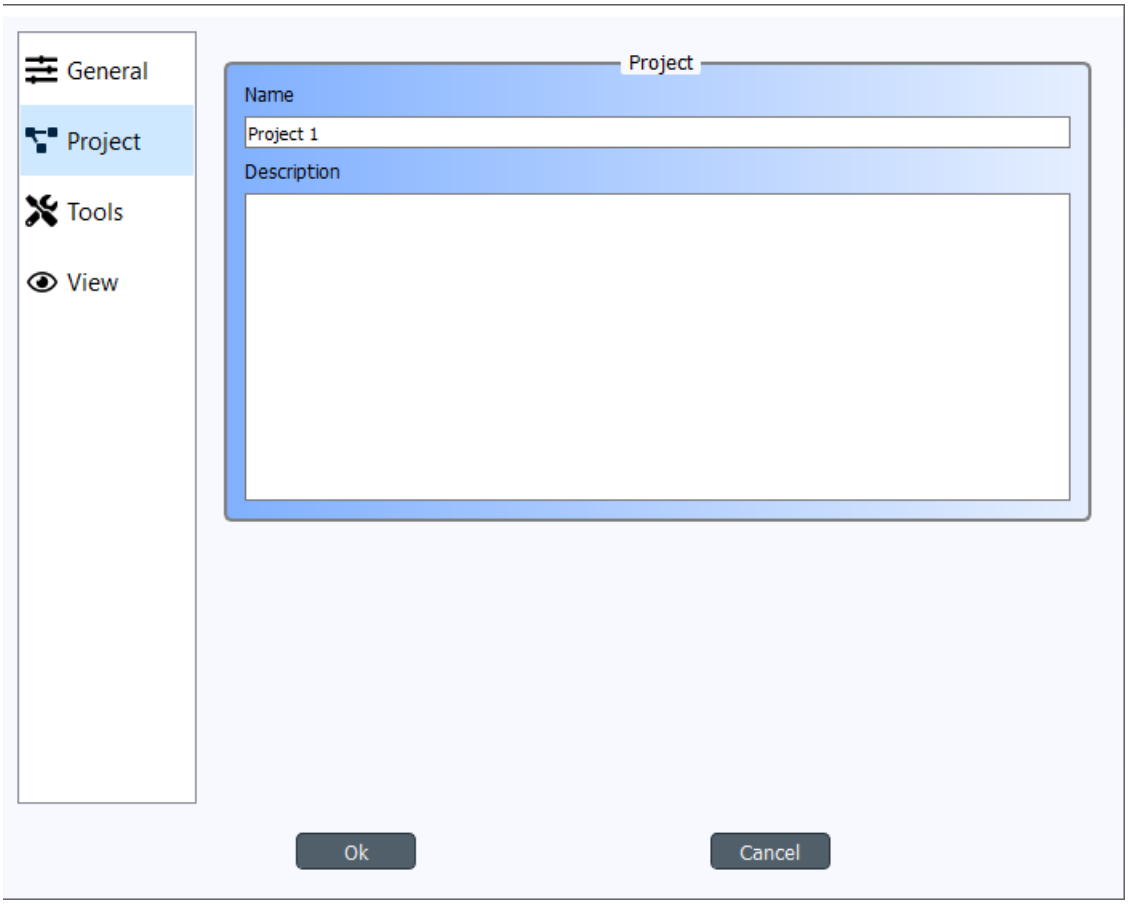


The General tab contains the general application settings.

- **Open previous project at startup** If checked, application opens the project at startup that was open the last time the application was shut down. If left unchecked, application starts without a project open.
- **Show confirm exit prompt** If checked, confirm exit prompt is shown. If unchecked, application exits without prompt.
- **Save project at exit** Unchecked: Does not save project and does not show message box. Partially checked: Shows message box (default). Checked: Saves project and does not show message box.
- **Show date and time in Event Log messages** If checked, every Event Log message is prepended with a date and time 'tag'.
- **Delete data when project item is removed from project** Check this box to delete project item's data when a project item is removed from project. This means, that the *project item directory* and its contents will be deleted from your hard drive. You can find the project item directories from the `<proj_dir>/.spinetoolbox/items/` directory, where `<proj_dir>` is your current project directory.
- **Work directory** Directory where processing the Tool takes place. Default place (if left empty) is the `/work` subdirectory of Spine Toolbox install directory. You can change this directory. Make sure to clean up the directory every now and then.
- **Smooth zoom** Controls the way zooming (by using the mouse wheel) behaves in Design View and in Spine database editor. Controls if the zoom in/out is continuous or discrete. On older computers, smooth zoom is not recommended (because it may be slower).

- **Curved links** Controls the look of the arrows (connections) on Design View.
- **Design View background** Choosing grid shows a black grid as the Design View background. Choosing Solid and then clicking on the square next to it let's you choose the background color.
- **Data flow animation speed** This slider controls the speed of the 'arrow' animation on Design View when execution is about to start.

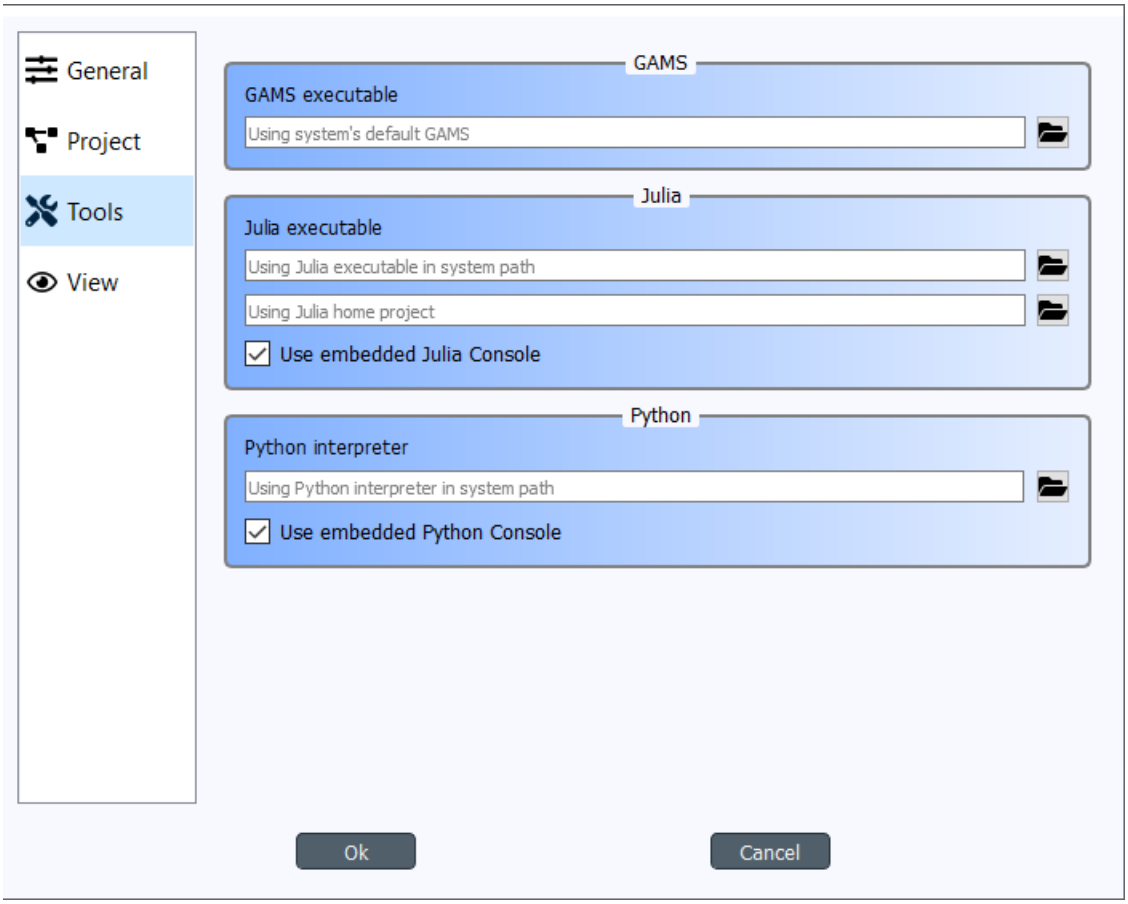
7.2 Project settings



These settings affect the project that is currently open. To save the project to a new directory use the `File->Save project as...` menu item. Or you can simply copy the project directory anywhere on your file system.

- **Name** The default name for new projects is the name of the project directory. You can change the project name here.
- **Description** You can type a description for your project here.

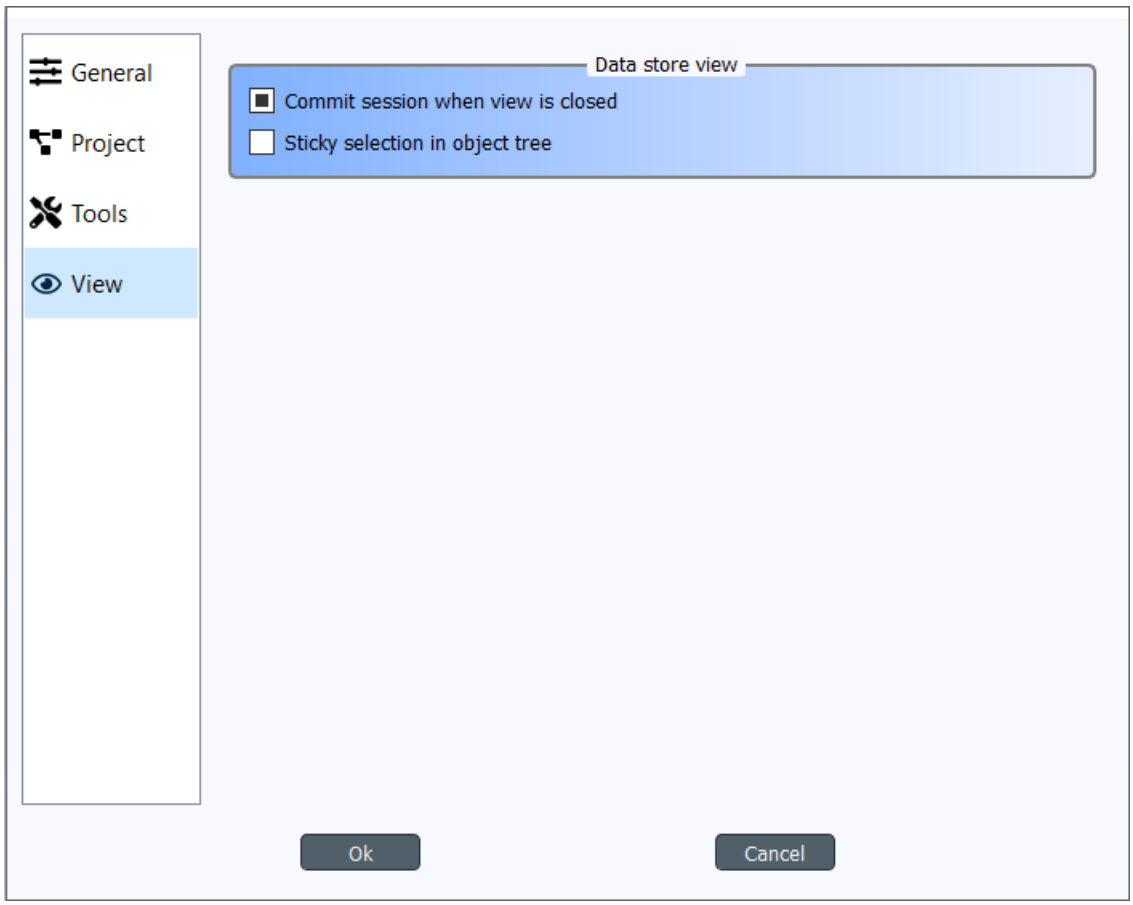
7.3 Tools settings



- **GAMS executable** Path to GAMS executable you wish to use to execute *Exporter* project items and *Tool* project items that use a GAMS Tool specification. Leave this empty to use the system GAMS (i.e. GAMS set up in your system PATH variable).
- **Julia executable** Path to Julia executable you wish to use to execute *Tool* project items that use a Julia Tool specification. This is the Julia executable that will be used in the embedded Julia Console and also the Julia that is used when executing Julia Tool specifications as in the shell. Leave this empty, if you wish to use the system Julia.
- **Julia home project** Set the Julia home project here.
- **Use embedded Julia Console** Check this box to execute *Tool* project items that use a Julia Tool specification in the built-in Julia Console. If you leave this un-checked, Julia Tool specifications will be executed as in the shell. I.e on Windows this would be the equivalent to running command `julia.exe example_script.jl` in the command prompt. If you decide to use the embedded Julia Console, the `example_script.jl` is *included* into the console and executed there. It is recommended to use the embedded Julia Console, since this gives a significant performance boost compared to shell execution.
- **Python interpreter** Path to Python executable you wish to use to execute *Tool* project items that use a Python Tool specification. This is the Python that will be used in the embedded Python Console and also the Python that is used when executing Python Tool specifications as in the shell. Leave this empty to use the system Python.
- **Use embedded Python Console** Check this box to execute Python Tool specifications in the embedded Python Console. If you un-check this box, Python Tool specifications will be executed as in the shell. I.e on Windows

this would be the equivalent to running command `python.exe script.py` in the command prompt. If you decide to use the embedded Python Console, `script.py` is executed there instead.

7.4 View settings



- **Commit session when view is closed** This checkbox controls what happens when you close the Spine database editor which has uncommitted changes. When this is unchecked, all changes are discarded without notice. When this is partially checked (default), a message box warning you about uncommitted changes is shown. When this is checked, a commit message box is shown immediately without first showing the message box.
- **Sticky selection in object tree** Controls how selecting items in Spine database editor's Object tree using the left mouse button works. If unchecked, single selection is enabled and pressing the Ctrl-button down enables multiple selection. If checked, Multiple selection is enabled and pressing the Ctrl-button down enables single selection.

7.5 Application preferences

Spine Toolbox remembers the size, location, and placement of most of the application windows from the previous session (i.e. when closing and restarting the app).

7.6 Where are the application settings stored?

Application settings and preferences (see above) are saved to a location that depends on your operating system. On Windows, there is no separate settings file. They are stored into registry key `HKEY_CURRENT_USER\Software\SpineProject\Spine Toolbox`. It is safe to delete this key if you want to reset Spine Toolbox to factory settings.

Note: If you are looking for information on project item properties, see *Project Items*.

CHAPTER 8

Welcome to Spine database editor's User Guide!

Spine database editor is a dedicated component of Spine Toolbox, that you can use to visualize and edit data in one or more Spine databases.

8.1 Getting started

8.1.1 Launching the editor

From Spine Toolbox

To open a single database in Spine database editor:

1. Create a *Data Store* project item.
2. Select the *Data Store*.
3. Enter the url of the database in *Data Store Properties*.
4. Press the **Open editor** button in *Data Store Properties*.

To open multiple databases in Spine database editor:

1. Repeat steps 1 to 3 above for each database.
2. Create a *View* project item.

3. Connect each *Data Store* item to the *View* item.
4. Select the *View* item.
5. Press **Open editor** in *View Properties*.

From the command line

To open a single SQLite database in Spine database editor, use the `open_spine_db_editor.py` script in the `bin` folder:

```
open_spine_db_editor.py "...path of the database file..."
```

8.1.2 Knowing the UI

The form has the following main UI components:

- *Entity trees* (*Object tree* and *Relationship tree*): they present the structure of classes and entities in all databases in the shape of a tree.
- *Stacked tables* (*Object parameter value*, *Object parameter definition*, *Relationship parameter value*, and *Relationship parameter definition*): they present object and relationship parameter data in the form of stacked tables.
- *Pivot table* and *Frozen table*: they present data for a given class in the form of a pivot table, optionally with frozen dimensions.
- *Entity graph*: it presents the structure of classes and entities in the shape of a graph.
- *Parameter value list*: it presents parameter value lists available in the database.
- *Parameter tag toolbar*: it presents parameter tags defined in the database.

Tip: You can show or hide form components using the **View** menu, or select among three predefined layout styles: **Stacked style**, **Pivot style**, and **Graph style**.

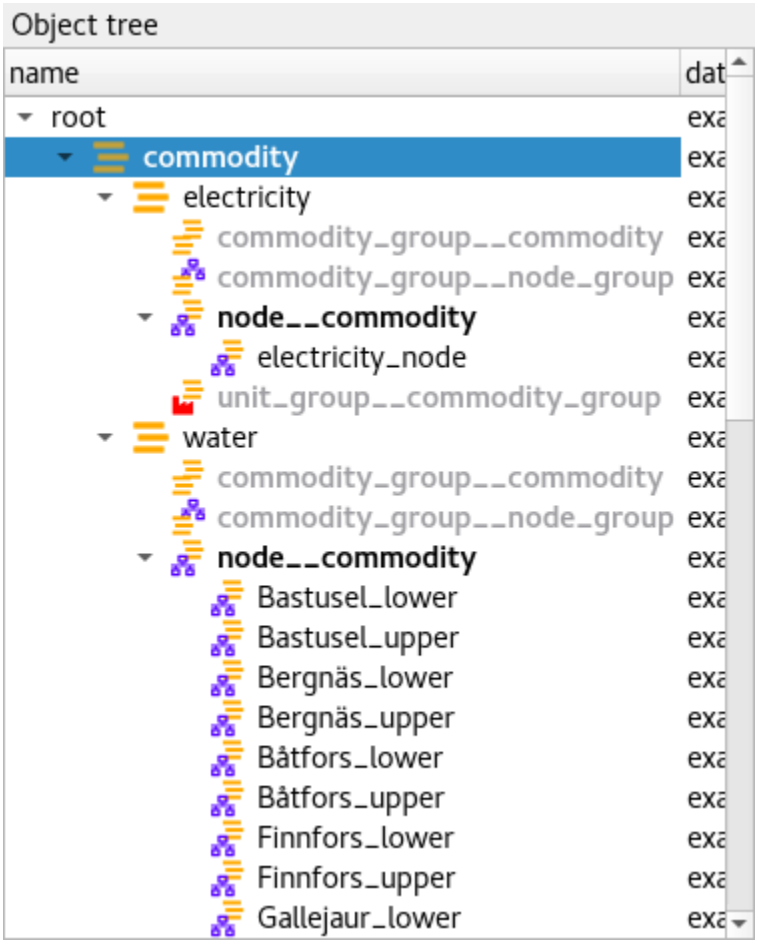
8.2 Viewing data

This section describes the available tools to view data.

8.2.1 Viewing entities and classes

Using *Entity trees*

Entity trees present the structure of classes and entities in all databases in the shape of a tree:



In *Object tree*:

- To view all object classes from all databases, expand the root item (automatically expanded when loading the form).
- To view all objects of a class, expand the corresponding object class item.
- To view all relationship classes involving an object class, expand any objects of that class.

- To view all relationships of a class involving a given object, expand the corresponding relationship class item under the corresponding object item.

In *Relationship tree*:

- To view all relationship classes from all databases, expand the root item (automatically expanded when loading the form).
- To view all relationships of a class, expand the corresponding relationship class item.

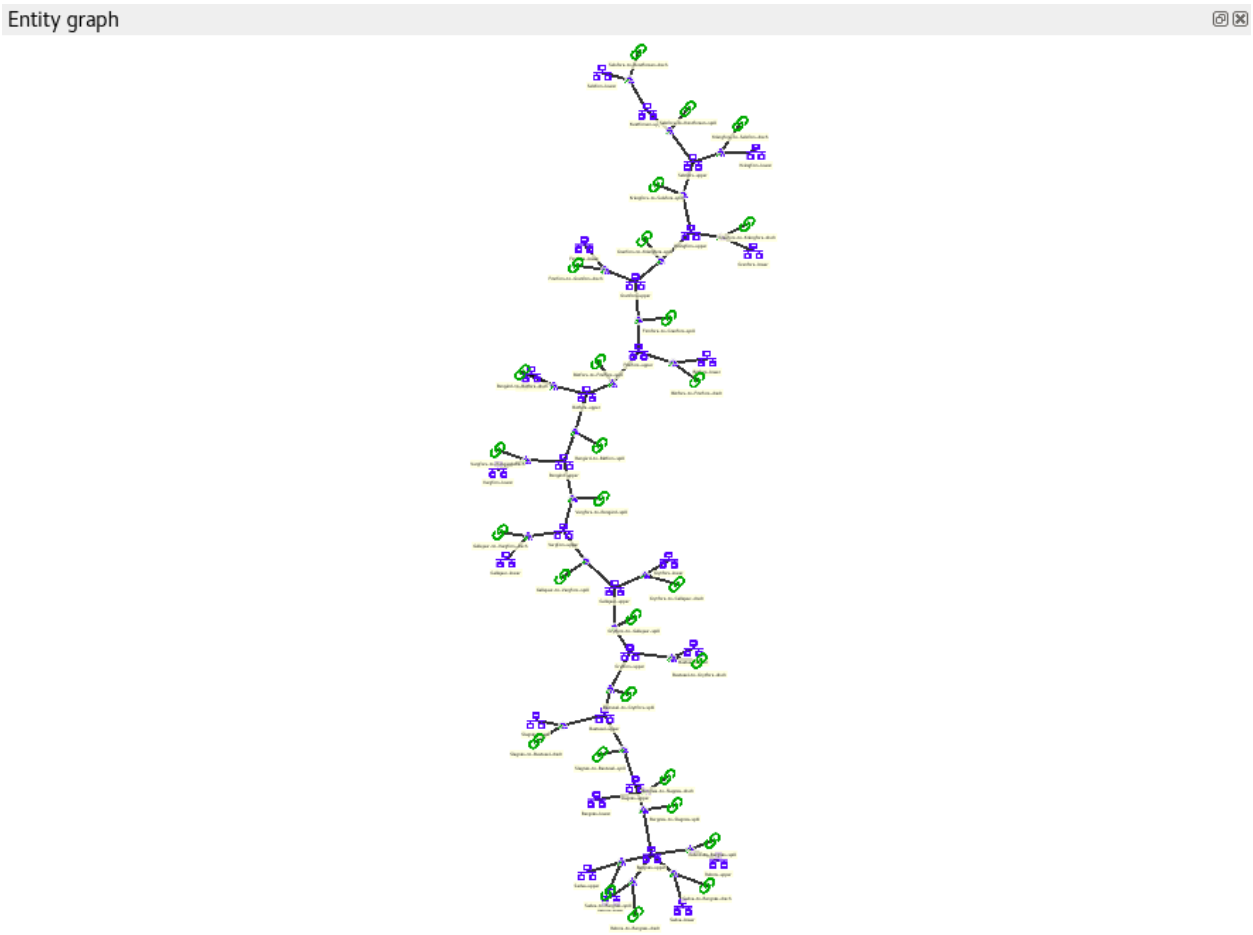
Note: To expand an item in *Object tree* or *Relationship tree*, double-click on the item or press the right arrow while it's active. Items in gray don't have any children, thus they cannot be expanded. To collapse an expanded item, double-click on it again or press the left arrow while it's active.

Tip: To expand or collapse an item and all its descentants in *Object tree* or *Relationship tree*, right click on the item to display the context menu, and select **Fully expand** or **Fully collapse.**

Tip: In *Object tree*, the same relationship appears in many places (as many as it has dimensions). To jump to the next ocurrence of a relationship item, either double-click on the item, or right-click on it to display the context menu, and select **Find next**.

Using *Entity graph*

Entity graph presents the structure of classes and entities from one database in the shape of a graph:



Tip: To see it in action, check out this video.

Building the graph

To build the graph, select any number of items in either *Object tree* or *Relationship tree*. What is included in the graph depends on the specific selection you make:

- To include all objects and relationships from the database, select the root item in either *Object tree* or *Relationship tree*.
- To include all objects of a class, select the corresponding class item in *Object tree*.
- To include all relationships of a class, select the corresponding class item in *Relationship tree*.
- To include all relationships of a specific class involving a specific object, select the corresponding relationship class item under the corresponding object item in *Object tree*.
- To include specific objects or relationships, select the corresponding item in either *Object tree* or *Relationship tree*.

Note: In *Entity graph*, a small unnamed vertex represents a relationship, whereas a bigger named vertex represents an object. An arc between a relationship and an object indicates that the object is a member in that relationship.

The graph automatically includes relationships whenever *all* the member objects are included (even if these relationships are not selected in *Object tree* or *Relationship tree*). You can change this behavior to automatically include relationships whenever *any* of the member objects are included. To do this, enable **Show cascading relationships** via the **Graph** menu, or via *Entity graph*'s context menu.

Tip: To *extend* the selection in *Object tree* or *Relationship tree*, press and hold the **Ctrl** key while clicking on the items.

Tip: *Object tree* and *Relationship tree* also support **Sticky selection**, i.e., extending the selection by clicking on items *without pressing Ctrl*. To enable **Sticky selection**, go to **File -> Settings** and check the corresponding box.

Note: At the moment, *Entity graph* only shows data from the first database open in the form.

Manipulating the graph

You can move items in the graph by dragging them with your mouse. To make relationship items stay in the same relative position with respect to their member objects, go to **File -> Settings** and check the box next to, *Move relationships along with objects in Entity graph*.

To save the position of items into the database, select the items in the graph and then choose **Graph -> Save positions** from the menu bar. To clear saved positions, select the items again and choose **Graph -> Clear saved positions**.

To hide part of the graph, select the items you want to hide and then choose **Graph -> Hide selected**. To show the hidden items again, select **Graph -> Show hidden**.

To prune the graph, select the items you want to prune and then choose **Graph -> Prune selected entities** or **Prune selected classes**. To restore specific prunned items, go to **Graph -> Restore prunned** and select the items you want to restore from the popup menu. To restore all prunned items at once, select **Graph -> Restore all prunned**.

Note: *Entity graph* supports extended selection and rubber-band selection. To extend a selection, press and hold **Ctrl** while clicking on the items. To perform rubber-band selection, press and hold **Ctrl** while dragging your mouse around the items you want to select.

Note: Prunned items are remembered across graph builds.

To zoom in and out, scroll your mouse wheel over *Entity graph* or use the buttons in **Graph -> Zoom**. To rotate clockwise or anti-clockwise, press and hold the **Shift** key while scrolling your mouse wheel, or use the buttons in **Graph -> Rotate**.

To export the current graph as a PDF file, select **File -> Export graph as PDF**.

8.2.2 Viewing parameter definitions and values

Using *Stacked tables*

Stacked tables present object and relationship parameter data from all databases in the form of stacked tables:

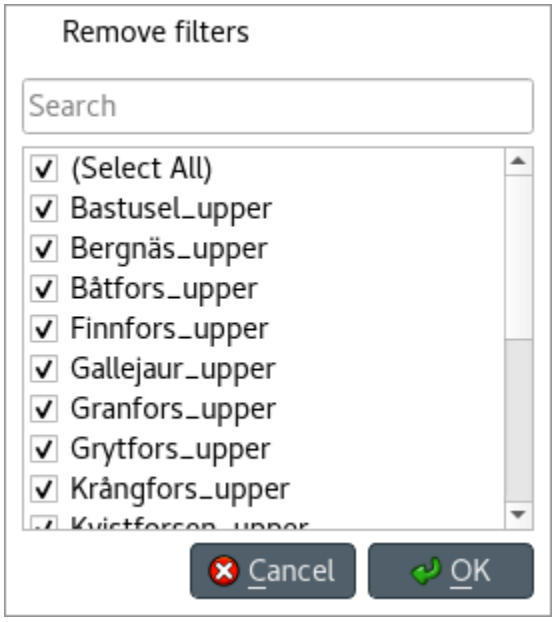
Object parameter value

object_class_name	object_name	parameter_name	value	database
model	instance	duration_unit	hour	example
model	instance	model_end	2019-01-08 00:00:00	example
model	instance	model_start	2019-01-01 00:00:00	example
node	Bastusel_upper	demand	-0.2579768519	example
node	Bastusel_upper	fix_node_state	Time series	example
node	Bastusel_upper	has_state	value_true	example
node	Bastusel_upper	node_state_cap	8208.0	example
node	Bergnäs_upper	demand	-22.29	example
node	Bergnäs_upper	fix_node_state	Time series	example
node	Bergnäs_upper	has_state	value_true	example
node	Bergnäs_upper	node_state_cap	216120.0	example
node	Båtfors_upper	demand	-2.0	example
node	Båtfors_upper	fix_node_state	Time series	example
node	Båtfors_upper	has_state	value_true	example
node	Båtfors_upper	node_state_cap	1330.0	example
node	Finnfors_upper	demand	0.0	example
node	Finnfors_upper	fix_node_state	Time series	example
node	Finnfors_upper	has_state	value_true	example

To filter *Stacked tables* by any entities and/or classes, select the corresponding items in either *Object tree*, *Relationship tree*, or *Entity graph*. To remove all these filters, select the root item in either *Object tree* or *Relationship tree*.

To filter parameter definitions and values by certain parameter tags, select those tags in *Parameter tag toolbar*.

To apply a custom filter on a *Stacked table*, click on any horizontal header. A menu will pop up listing the items in the corresponding column:



Uncheck the items you don't want to see in the table and press **Ok**. Additionally, you can type in the search bar at the top of the menu to filter the list of items. To remove the current filter, select **Remove filters**.

To filter a *Stacked table* according to a selection of items in the table itself, right-click on the selection to show the context menu, and then select **Filter by** or **Filter excluding**. To remove these filters, select **Remove filters** from the header menus of the filtered columns.

Tip: You can rearrange columns in *Stacked tables* by dragging the headers with your mouse. The ordering will be remembered the next time you open the form.

8.2.3 Viewing parameter values and relationships

Using *Pivot table* and *Frozen table*

Pivot table and *Frozen table* present data for an individual class from one database in the form of a pivot table, optionally with frozen dimensions:

Pivot table

			parameter	connection_...	fi
	connection	node1	node2		
	Bastusel_to_Grytfors_disch	Grytfors_upper	Bastusel_lower	1h	
	Bastusel_to_Grytfors_spill	Grytfors_upper	Bastusel_upper	150m	
	Bergnäs_to_Slagnäs_disch	Slagnäs_upper	Bergnäs_lower	1h	
	Bergnäs_to_Slagnäs_spill	Slagnäs_upper	Bergnäs_upper	1h	
	Båtfors_to_Finnfors_disch	Finnfors_upper	Båtfors_lower	3h	
	Båtfors_to_Finnfors_spill	Finnfors_upper	Båtfors_upper	3h	
	Finnfors_to_Granfors_disch	Granfors_upper	Finnfors_lower	3h	
	Finnfors_to_Granfors_spill	Granfors_upper	Finnfors_upper	3h	
	Gallejaur_to_Vargfors_disch	Vargfors_upper	Gallejaur_lower	30m	
	Gallejaur_to_Vargfors_spill	Vargfors_upper	Gallejaur_upper	150m	
	Granfors_to_Krångfors_disch	Krångfors_upper	Granfors_lower	3h	
	Granfors_to_Krångfors_spill	Krångfors_upper	Granfors_upper	3h	
	Grytfors_to_Gallejaur_disch	Gallejaur_upper	Grytfors_lower	15m	
	Grytfors_to_Gallejaur_spill	Gallejaur_upper	Grytfors_upper	15m	
	Krångfors_to_Selsfors_disch	Selsfors_upper	Krångfors_lower	3h	
	Krångfors_to_Selsfors_spill	Selsfors_upper	Krångfors_upper	3h	
	Rebnis_to_Bergnäs_disch	Bergnäs_upper	Rebnis_lower	2D	
	Rebnis_to_Bergnäs_spill	Bergnäs_upper	Rebnis_upper	2D	
	Rengård_to_Båtfors_disch	Båtfors_upper	Rengård_lower	3h	
	Rengård_to_Båtfors_spill	Båtfors_upper	Rengård_upper	3h	
	Sadva_to_Bergnäs_disch	Bergnäs_upper	Sadva_lower	2D	
	Sadva_to_Bergnäs_spill	Bergnäs_upper	Sadva_upper	2D	
	Selsfors_to_Kvistforsen_disch	Kvistforsen_upper	Selsfors_lower	3h	
	Selsfors_to_Kvistforsen_spill	Kvistforsen_upper	Selsfors_upper	3h	
	Slagnäs_to_Bastusel_disch	Bastusel_upper	Slagnäs_lower	4h	
	Slagnäs_to_Bastusel_spill	Bastusel_upper	Slagnäs_upper	4h	
	Vargfors_to_Rengård_disch	Rengård_upper	Vargfors_lower	3h	

To populate the tables with data for a certain class, just select the corresponding class item in either *Object tree* or *Relationship tree*.

Selecting the input type

Pivot table and *Frozen table* support three different input types:

- **Parameter value** (the default): it shows objects and parameter definitions in the headers, and corresponding parameter values in the table body.
- **Index expansion**: Similar to the above, but it also shows parameter indexes in the headers. Indexes are extracted from special parameter values, such as time-series.
- **Relationship**: it shows objects in the headers, and corresponding relationships in the table body. It only works when selecting a relationship class in *Relationship tree*.

You can select the input type from the **Pivot table** menu in the menu bar.

Note: In *Pivot table*, header blocks in the top-left area indicate what is shown in each horizontal and vertical header. For example, in **Parameter value** input type, by default, the horizontal header has a single row listing parameter names, whereas the vertical header has one or more columns listing object names.

Pivoting and freezing

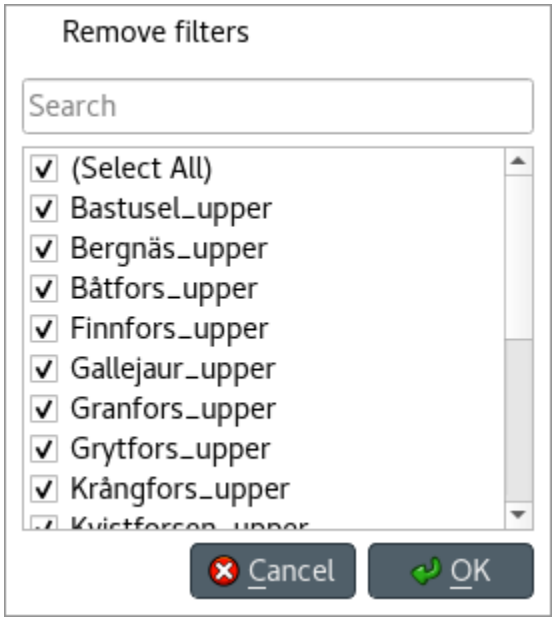
To pivot the data, drag a header block across the top-left area of the table. You can turn a horizontal header into a vertical header and viceversa, as well as rearrange headers vertically or horizontally.

To freeze a dimension, drag the corresponding header block from *Pivot table* into *Frozen table*. To unfreeze a frozen dimension, just do the opposite.

Note: Your pivoting and freezing selections for any class will be remembered when switching to another class.

Filtering

To apply a custom filter on *Pivot table*, click on the arrow next to the name of any header block. A menu will pop up listing the items in the corresponding row or column:



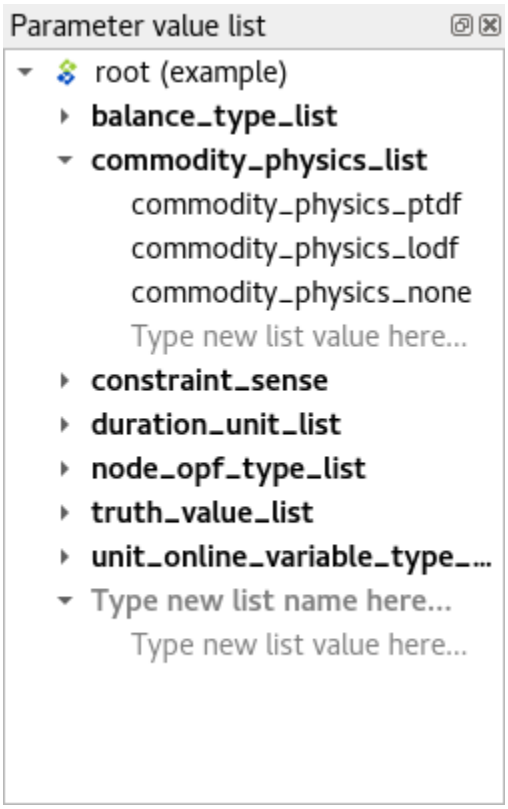
Uncheck the items you don't want to see in the table and press **Ok**. Additionally, you can type in the search bar at the top of the menu to filter the list of items. To remove the current filter, select **Remove filters**.

To filter the pivot table by an individual vector across the frozen dimensions, select the corresponding row in *Frozen table*.

Note: At the moment, *Pivot table* shows data for only one class at a time, and only for the first database open in the form.

8.2.4 Viewing parameter value lists

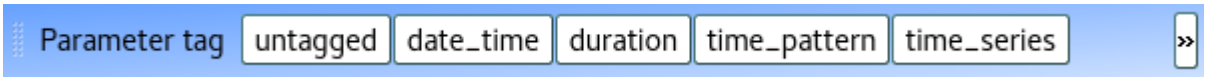
You can find parameter value lists from all databases under *Parameter value list*:



To view the parameter value lists from each database, expand the root item for that database. To view the values for each list, expand the corresponding list item.

8.2.5 Viewing parameter tags

You can find parameter tags from all databases in *Parameter tag toolbar*:



8.3 Adding data

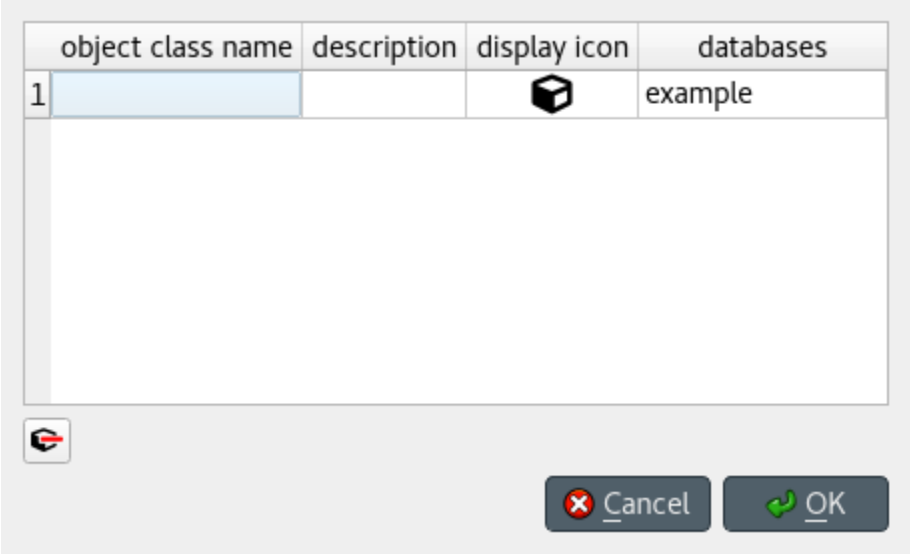
This section describes the available tools to add new data.

8.3.1 Adding object classes

Using *Add object classes* dialog

Select **Edit -> Add object classes** from the menu bar, or right-click on the root item in *Object tree* to display the context menu, and select **Add object classes**.

The *Add object classes* dialog will pop up:



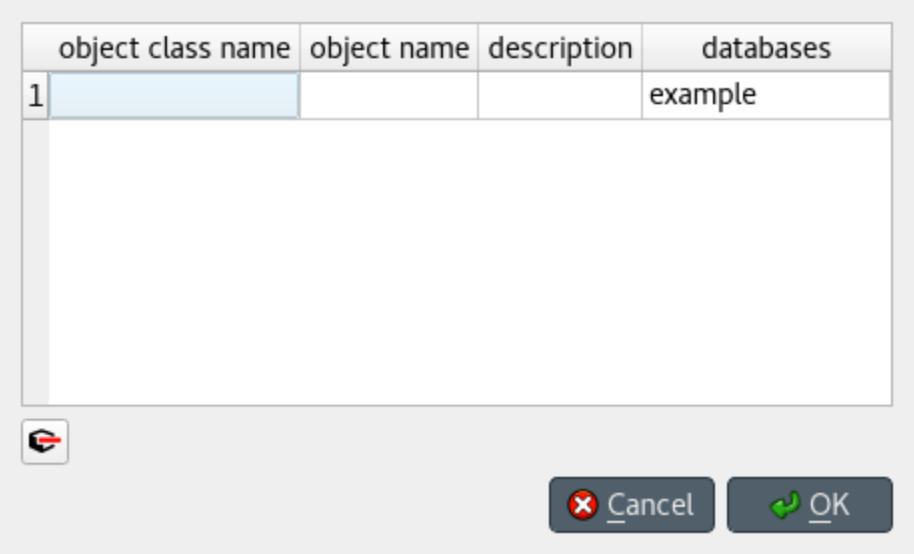
Enter the names of the classes you want to add under the *object class name* column. Optionally, you can enter a description for each class under the *description* column. To select icons for your classes, double click on the corresponding cell under the *display icon* column. Finally, select the databases where you want to add the classes under *databases*. When you're ready, press **Ok**.

8.3.2 Adding objects

Using *Add objects* dialog

Select **Edit -> Add objects** from the menu bar, or right-click on an object class item in *Object tree* to display the context menu, and select **Add objects**.

The *Add objects* dialog will pop up:



Enter the names of the object classes under *object class name*, and the names of the objects under *object name*. To display a list of available classes, start typing or double click on any cell under the *object class name* column. Optionally, you can enter a description for each object under the *description* column. Finally, select the databases where you want to add the objects under *databases*. When you're ready, press **Ok**.

Using *Pivot table*

To add an object to a specific class, bring the class to *Pivot table* using any input type (see *Using Pivot table and Frozen table*). Then, enter the object name in the last cell of the header corresponding to that class.

Duplicating objects

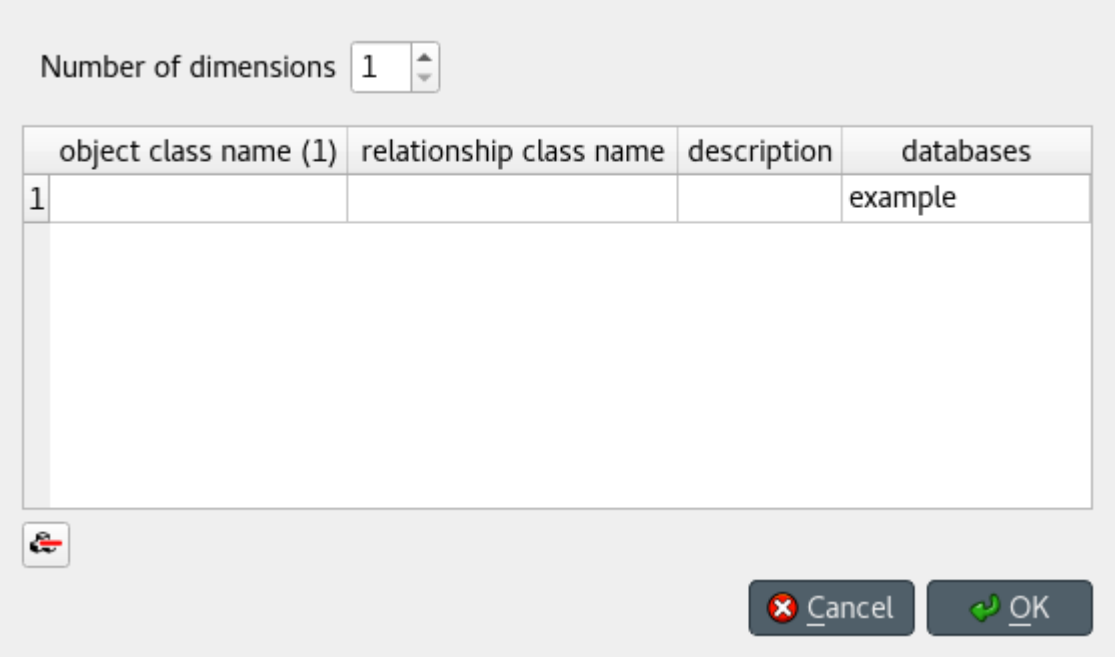
To duplicate an existing object with all its relationships and parameter values, right-click over the corresponding object item in *Object tree* to display the context menu, and select **Duplicate object**. Enter a name for the duplicate and press **Ok**.

8.3.3 Adding relationship classes

Using *Add relationship classes* dialog

From the menu bar, select **Edit -> Add relationship classes**. Alternatively, right-click on either the root item in *Relationship tree*, or on an object class item in *Object tree*, and select **Add relationship classes** from the context menu.

The *Add relationship classes* dialog will pop up:



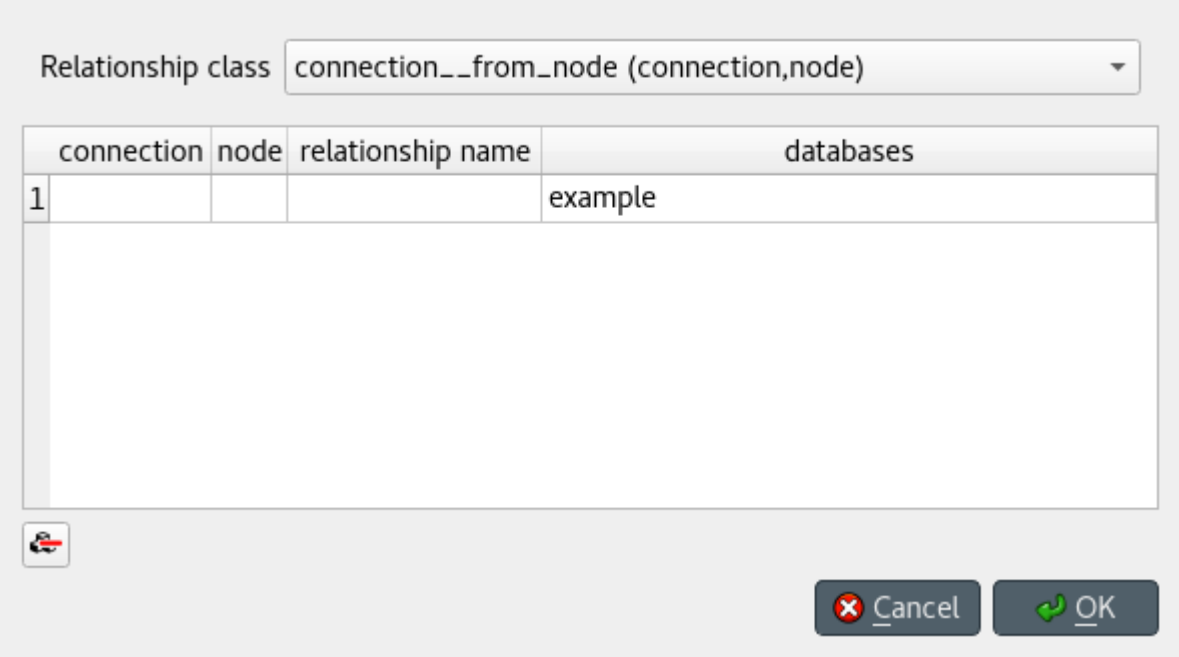
Select the number of dimensions using the spinbox at the top; then, enter the names of the object classes for each dimension under each *object class name* column, and the names of the relationship classes under *relationship class name*. To display a list of available object classes, start typing or double click on any cell under the *object class name* columns. Optionally, you can enter a description for each relationship class under the *description* column. Finally, select the databases where you want to add the relationship classes under *databases*. When you're ready, press **Ok**.

8.3.4 Adding relationships

Using *Add relationships* dialog

Select **Edit -> Add relationships** from the menu bar, or right-click on a relationship class item in either *Object tree* or *Relationship tree*, and select **Add relationships** from the context menu.

The *Add relationships* dialog will pop up:



Select the relationship class from the combo box at the top; then, enter the names of the objects for each member object class under the corresponding column, and the name of the relationship under *relationship name*. To display a list of available objects for a member class, start typing or double click on any cell under that class's column. Finally, select the databases where you want to add the relationships under *databases*. When you're ready, press **Ok**.

Using *Pivot table*

To add a relationship for a specific class, bring the class to *Pivot table* using the **Relationship** input type (see *Using Pivot table and Frozen table*). The *Pivot table* headers will be populated with all possible combinations of objects across the member classes. Locate the objects you want as members in the new relationship, and check the corresponding box in the table body.

Using *Entity graph*

Make sure all the objects you want as members in the new relationship are in the graph. To start the relationship, either double click on one of the objects, or right click to display the context menu and choose **Add relationships**. A menu will pop up showing the available relationship classes. Select the class you want; the mouse cursor will adopt a cross-hairs shape. Click on each of the remaining member objects one by one to make the relationships.

Tip: All the *Add...* dialogs support pasting tabular (spreadsheet) data from the clipboard. Just select any cell in the table and press **Ctrl+V**. If needed, the table will grow to accommodate the exceeding data. To paste data on multiple cells, select all the cells you want to paste on and press **Ctrl+V**.

8.3.5 Adding parameter definitions

Using *Stacked tables*

To add new parameter definitions for an object class, just fill the last empty row of *Object parameter definition*. Enter the name of the class under *object_class_name*, and the name of the parameter under *parameter_name*. To display a

list of available object classes, start typing or double click under the *object_class_name* column. Optionally, you can also specify a default value, a parameter value list, or any number of parameter tags under the appropriate columns. The parameter is added when the background of the cells under *object_class_name* and *parameter_name* become gray.

To add new parameter definitions for a relationship class, just fill the last empty row of *Relationship parameter definition*, following the same guidelines as above.

Using *Pivot table*

To add a new parameter definition for a class, bring the corresponding class to *Pivot table* using the **Parameter value** input type (see *Using Pivot table and Frozen table*). The *parameter* header of *Pivot table* will be populated with existing parameter definitions for the class. Enter a name for the new parameter in the last cell of that header.

8.3.6 Adding parameter values

Using *Stacked tables*

To add new parameter values for an object, just fill the last empty row of *Object parameter value*. Enter the name of the class under *object_class_name*, the name of the object under *object_name*, and the name of the parameter under *parameter_name*. Optionally, you can also specify the parameter value right away under the *value* column. To display a list of available object classes, objects, or parameters, start typing or double click under the appropriate column. The parameter value is added when the background of the cells under *object_class_name*, *object_name*, and *parameter_name* become gray.

To add new parameter values for a relationship class, just fill the last empty row of *Relationship parameter value*, following the same guidelines as above.

Note: To add parameter values for an object, the object has to exist beforehand. However, when adding parameter values for a relationship, you can specify any valid combination of objects under *object_name_list*, and a relationship will be created among those objects if one doesn't yet exist.

Using *Pivot table*

To add parameter value for any object or relationship, bring the corresponding class to *Pivot table* using the **Parameter value** input type (see *Using Pivot table and Frozen table*). Then, enter the parameter value in the corresponding cell in the table body.

Tip: All *Stacked tables* and *Pivot table* support pasting tabular (spreadsheet) data from the clipboard. Just select any cell in the table and press **Ctrl+V**. If needed, *Stacked tables* will grow to accommodate the exceeding data. To paste data on multiple cells, select all the cells you want to paste on and press **Ctrl+V**.

8.3.7 Adding parameter value lists

To add a new parameter value list, just enter the name of the list in the last row of *Parameter value list*, under the corresponding database item.

To add new values for the list, enter the values in the rows under the corresponding list item.

Note: To be actually added to the database, a parameter value list must have at least one value.

8.4 Updating data

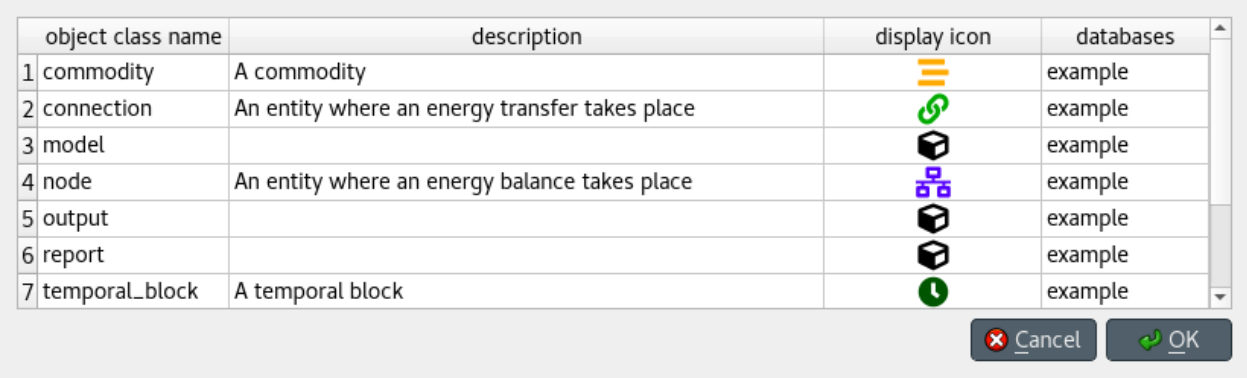
This section describes the available tools to update existing data.

8.4.1 Updating entities and classes

Using *Edit...* dialogs

Select any number of entity and/or class items in *Object tree* or *Relationship tree*, or any number of object and/or relationship items in *Entity graph*. Then, either select **Edit -> Edit selected items** from the menu bar, or right-click on the selection and choose **Edit selected items** from the context menu.

One separate *Edit...* dialog will pop up for each selected entity or class type, and the tables will be filled with the current data of selected items. E.g.:



Modify the field(s) you want under the corresponding column(s). Specify the databases where you want to update each item under the *databases* column. When you're ready, press **Ok**.

Using *Pivot table*

To rename an object of a specific class, bring the class to *Pivot table* using any input type (see *Using Pivot table and Frozen table*). Then, just edit the appropriate cell in the corresponding class header.

8.4.2 Updating parameter definitions and values

Using *Stacked tables*

To update parameter data, just go to the appropriate *Stacked table* and edit the corresponding row.

Using *Pivot table*

To rename parameter definitions for a class, bring the corresponding class to *Pivot table* using the **Parameter value** input type (see *Using Pivot table and Frozen table*). Then, just edit the appropriate cell in the *parameter* header.

To modify parameter values for an object or relationship, bring the corresponding class to *Pivot table* using the **Parameter value** input type (see *Using Pivot table and Frozen table*). Then, just edit the appropriate cell in the table body.

8.4.3 Updating parameter value lists

To rename a parameter value list or any of its values, just edit the appropriate row in *Parameter value list*.

8.5 Removing data

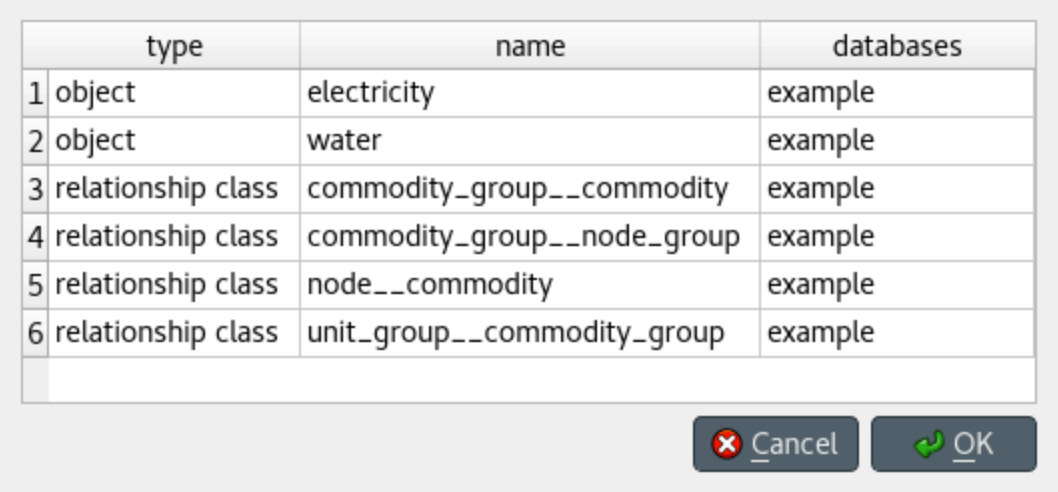
This section describes the available tools to remove data.

8.5.1 Removing entities and classes

Using *Remove items* dialog

Select the items in *Object tree* or *Relationship tree* corresponding to the entities and classes you want to remove. Then, either select **Edit -> Remove selected items** from the menu bar, or right-click on the selection and choose **Remove selected items** from the context menu.

The *Remove items* dialog will popup:



Specify the databases from where you want to remove each item under the *databases* column, and press **Ok**.

Using *Entity graph*

Select the items in *Entity graph* corresponding to the objects and/or relationships you want to remove. Then, either select **Edit -> Remove selected items** from the menu bar, or right-click on the selection and choose **Remove selected items** from the context menu.

Using *Pivot table*

To remove objects or relationships from a specific class, bring the class to *Pivot table* using the **Parameter value** input type (see *Using Pivot table and Frozen table*), and select the cells in the table headers corresponding to the objects and/or relationships you want to remove. Then, either select **Edit -> Remove selected items** from the menu bar, or right-click on the selection and choose **Remove selected items** from the context menu.

Alternatively, to remove relationships for a specific class, bring the class to *Pivot table* using the **Relationship** input type (see *Using Pivot table and Frozen table*). The *Pivot table* headers will be populated with all possible combinations of objects across the member classes. Locate the member objects of the relationship you want to remove, and uncheck the corresponding box in the table body.

8.5.2 Removing parameter definitions and values

Using *Stacked tables*

To remove parameter definitions or values, go to the relevant *Stacked table* and select any cell in the row corresponding to the items you want to remove Then, either select **Edit -> Remove selected items** from the menu bar, or right-click on the selection and choose **Remove selected items** from the context menu.

Using *Pivot table*

To remove parameter definitions and/or values for a certain class, bring the corresponding class to *Pivot table* using the **Parameter value** input type (see *Using Pivot table and Frozen table*). Then:

1. Select the cells in the *parameter* header corresponding to the parameter definitions you want to remove.
2. Select the cells in the table body corresponding to the parameter values you want to remove.

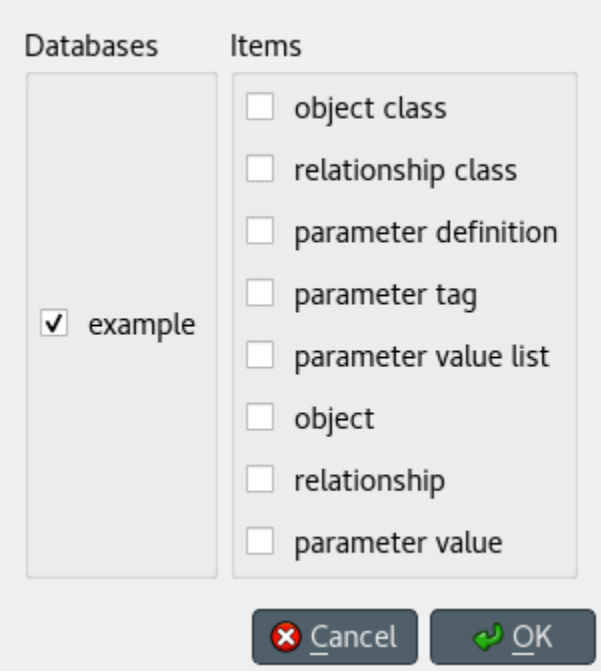
Finally, either select **Edit -> Remove selected items** from the menu bar, or right-click on the selection in *Pivot table* and choose **Remove selected items** from the context menu.

8.5.3 Removing parameter value lists

To remove parameter value list or any of their values, just select the appropriate rows in *Parameter value list*. Then, either select **Edit -> Remove selected items** from the menu bar, or right-click on the selection and choose **Remove selected items** from the context menu.

8.5.4 Mass-removing items

To remove all items of specific types, select **Edit -> Mass remove items** from the menu bar. The *Mass remove items* dialog will pop up:



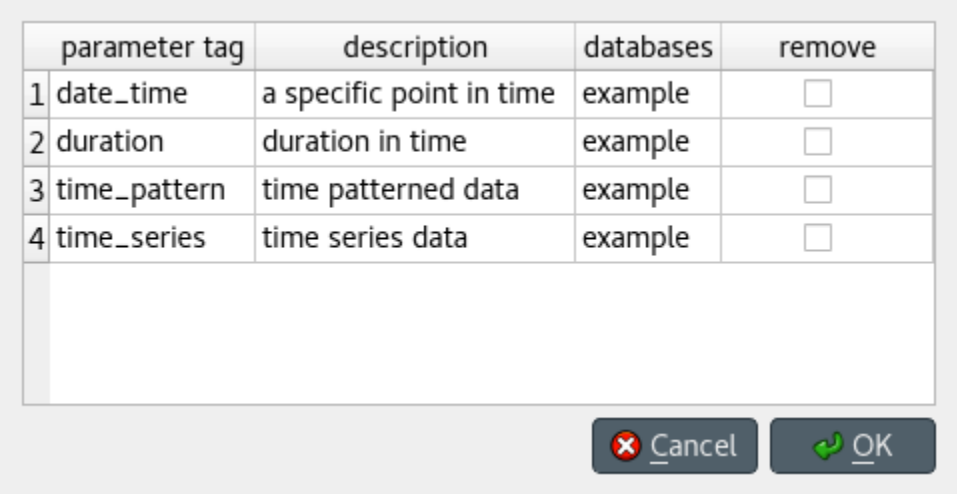
Select the databases from where you want to remove the items under *Databases*, and the type of items you want to remove under *Items*. Then, press **Ok**.

8.6 Managing data

This section describes the available tools to manage data, i.e., adding, updating or removing data at the same time.

8.6.1 Managing parameter tags

To add, update, and/or remove parameter tags, press **Manage parameter tags** on *Parameter tag toolbar*, or select **Edit -> Manage parameter tags** from the menu bar. The *Manage parameter tags* dialog will pop up:



To add new parameter tags, just fill the last empty row in the table: Enter the tag under *parameter tag*, and optionally a description under *description*. Finally, select the databases where you want to add the tag under *databases*.

To update existing parameter tags, just edit the appropriate row in the table, and select the databases where you want the changes to be effective under *databases*.

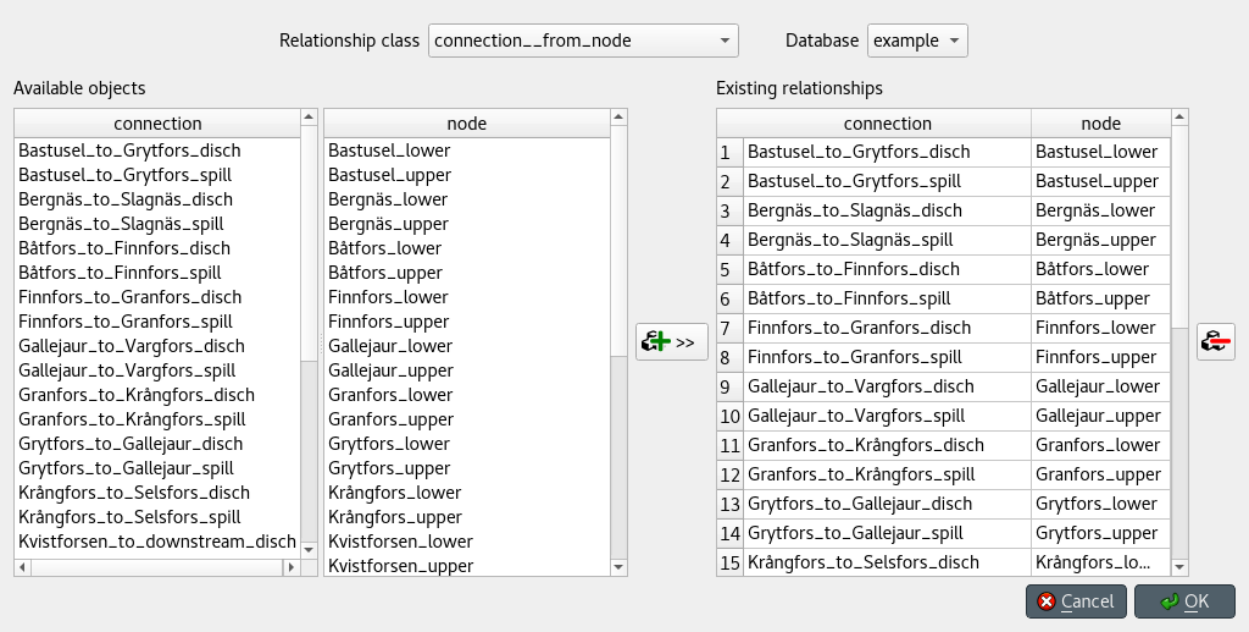
To remove parameter tags, just check the corresponding box under the *remove* column, and select the databases from where you want to remove the tag under *databases*.

When you're ready, press **Ok**.

Note: Changes made using the *Manage parameter tags* dialog are not applied to any databases until you press **Ok**.

8.6.2 Managing relationships

Select **Edit -> Manage relationships** from the menu bar. The *Manage relationships* dialog will pop up:



To get started, select a relationship class and a database from the combo boxes at the top.

To add relationships, select the member objects for each class under *Available objects* and press the **Add relationships** button at the middle of the form. The relationships will appear at the top of the table under *Existing relationships*.

To add multiple relationships at the same time, select multiple objects for one or more of the classes.

Tip: To *extend* the selection of objects for a class, press and hold the **Ctrl** key while clicking on more items.

Note: The set of relationships to add is determined by applying the *product* operation over the objects selected for each class.

To remove relationships, select the appropriate rows under *Existing relationships* and press the **Remove relationships** button on the right.

When you're happy with your changes, press **Ok**.

Note: Changes made using the *Manage relationships* dialog are not applied to the database until you press **Ok**.

8.7 Importing and exporting data

This section describes the available tools to import and export data.

8.7.1 Overview

Spine database editor supports importing and exporting data in three different formats: SQLite, JSON, and Excel. The SQLite import/export uses the Spine database format. The JSON and Excel import/export use a specific format described below.

Tip: To get a JSON or Excel file template you can simply export an existing Spine database into one of those formats.

Excel format

The Excel format consists of one sheet per object and relationship class. Each sheet can have one of four different formats:

1. Object class with scalar parameter data:

	A	B	C
1	Sheet type	Data type	object class name
2	object	Parameter	unit
3			
4	**commodity_group**	**Parameter_1**	**Parameter_2**
5	named_unit_1		1,618
6	named_unit_2	2,718	3,14

2. Object class with time-series parameter data:

	A	B	C
1	Sheet type	Data type	object class name
2	object	json array	unit
3			
4	node	named_unit_1	named_unit_2
5	json parameter	timseries_parameter	timseries_parameter
6	2018-01-01 00:00	1	3
7	2018-01-01 01:00	2	4
8	2018-01-01 02:00	3	

3. Relationship class with scalar parameter data:

	A	B	C	D	E
1	Sheet type	Data type	relationship class name	Number of relationship dimensions	Number of pivoted relationship dimensions
2	relationship	Parameter	commodity_group__commodity	2	0
3					
4	commodity_group	commodity	relationship_parameter1	relationship_parameter2	
5	electricity_group	electricity		1	
6	water_group	water	1	2	
7					

4. Relationship class with time-series parameter data:

	A	B	C	D	E	F	G
1	Sheet type	Data type	relationship class name	Number of relationship dimensions			
2	relationship	json array	unit__commodity	2			
3							
4	unit	SE1_spot	SE2_spot				
5	commodity	electricity	electricity				
6	json parameter	conversion_cost	conversion_cost				
7	2018-01-01 00:00	-24,03	-24,03				
8	2018-01-01 01:00	-24,03	-24,03				
9	2018-01-01 02:00	-24,02	-24,02				

JSON format

The JSON format consists of a single JSON object with the following `OPTIONAL` keys:

- **object_classes**: the value of this key `MUST` be a JSON array, representing a list of object classes. Each element in this array `MUST` be itself a JSON array and `MUST` have three elements:
  - The first element `MUST` be a JSON string, indicating the object class name.
  - The second element `MUST` be either a JSON string, indicating the object class description, or null.
  - The third element `MUST` be either a JSON integer, indicating the object class icon code, or null.
- **relationship_classes**: the value of this key `MUST` be a JSON array, representing a list of relationships classes. Each element in this array `MUST` be itself a JSON array and `MUST` have three elements:
  - The first element `MUST` be a JSON string, indicating the relationship class name.
  - The second element `MUST` be a JSON array, indicating the member object classes. Each element in this array `MUST` be a JSON string, indicating the object class name.
  - The third element `MUST` be either a JSON string, indicating the relationship class description, or null.
- **parameter_value_lists**: the value of this key `MUST` be a JSON array, representing a list of parameter value lists. Each element in this array `MUST` be itself a JSON array and `MUST` have two elements:
  - The first element `MUST` be a JSON string, indicating the parameter value list name.
  - The second element `MUST` be a JSON array, indicating the values in the list. Each element in this array `MUST` be either a JSON object, string, number, or null, indicating the value.
- **object_parameters**: the value of this key `MUST` be a JSON array, representing a list of object parameter definitions. Each element in this array `MUST` be itself a JSON array and `MUST` have five elements:
  - The first element `MUST` be a JSON string, indicating the object class name.
  - The second element `MUST` be a JSON string, indicating the parameter name.

  - The third element MUST be either a JSON object, string, number, or null, indicating the parameter default value.
  - The fourth element MUST be a JSON string, indicating the associated parameter value list, or null.
  - The last element MUST be either a JSON string, indicating the parameter description, or null.
- **relationship_parameters**: the value of this key MUST be a JSON array, representing a list of relationship parameter definitions. Each element in this array MUST be itself a JSON array and MUST have five elements:
  - The first element MUST be a JSON string, indicating the relationship class name.
  - The second element MUST be a JSON string, indicating the parameter name.
  - The third element MUST be either a JSON object, string, number, or null, indicating the parameter default value.
  - The fourth element MUST be a JSON string, indicating the associated parameter value list, or null
  - The last element MUST be either a JSON string, indicating the parameter description, or null.
- **objects**: the value of this key MUST be a JSON array, representing a list of objects. Each element in this array MUST be itself a JSON array and MUST have three elements:
  - The first element MUST be a JSON string, indicating the object class name.
  - The second element MUST be a JSON string, indicating the object name.
  - The third element MUST be either a JSON string, indicating the object description, or null.
- **relationships**: the value of this key MUST be a JSON array, representing a list of relationships. Each element in this array MUST be itself a JSON array and MUST have two elements:
  - The first element MUST be a JSON string, indicating the relationship class name.
  - The second element MUST be a JSON array, indicating the member objects. Each element in this array MUST be a JSON string, indicating the object name.
- **object_parameter_values**: the value of this key MUST be a JSON array, representing a list of object parameter values. Each element in this array MUST be itself a JSON array and MUST have four elements:
  - The first element MUST be a JSON string, indicating the object class name.
  - The second element MUST be a JSON string, indicating the object name.
  - The third element MUST be a JSON string, indicating the parameter name.
  - The fourth element MUST be either a JSON object, string, number, or null, indicating the parameter value.
- **relationship_parameter_values**: the value of this key MUST be a JSON array, representing a list of relationship parameter values. Each element in this array MUST be itself a JSON array and MUST have four elements:
  - The first element MUST be a JSON string, indicating the relationship class name.
  - The second element MUST be a JSON array, indicating the relationship's member objects. Each element in this array MUST be a JSON string, indicating the object name.
  - The third element MUST be a JSON string, indicating the parameter name.
  - The fourth element MUST be either a JSON object, string, number, or null, indicating the parameter value.

Example:

```
{
    "object_classes": [
        ["connection", "An entity where an energy transfer takes place",␣
↪280378317271233],
```

(continues on next page)

(continued from previous page)

```
        ["node", "An entity where an energy balance takes place", 280740554077951],
        ["unit", "An entity where an energy conversion process takes place",␣
↪281470681805429],
    ],
    "relationship_classes": [
        ["connection__node__node", ["connection", "node", "node"] , null],
        ["unit__from_node", ["unit", "node"], null],
        ["unit__to_node", ["unit", "node"], null],
    ],
    "parameter_value_lists": [
        ["balance_type_list", ["\"balance_type_node\"", "\"balance_type_group\"", "\
↪"balance_type_none\""]],
        ["truth_value_list", ["\"value_false\"", "\"value_true\""]],
    ],
    "object_parameters": [
        ["connection", "connection_availability_factor", 1.0, null, null],
        ["node", "balance_type", "balance_type_node", "balance_type_list", null],
    ],
    "relationship_parameters": [
        ["connection__node__node", "connection_flow_delay", {"type": "duration", "data
↪": "0h"}, null, null],
        ["unit__from_node", "unit_capacity", null, null, null],
        ["unit__to_node", "unit_capacity", null, null, null],
    ],
    "objects": [
        ["connection", "Bastusel_to_Grytfors_disch", null],
        ["node", "Bastusel_lower", null],
        ["node", "Bastusel_upper", null],
        ["node", "Grytfors_upper", null],
        ["unit", "Bastusel_pwr_plant", null],
    ],
    "relationships": [
        ["connection__node__node", ["Bastusel_to_Grytfors_disch", "Grytfors_upper",
↪"Bastusel_lower"]],
        ["unit__from_node", ["Bastusel_pwr_plant", "Bastusel_upper"]],
        ["unit__to_node", ["Bastusel_pwr_plant", "Bastusel_lower"]],
    ],
    "object_parameter_values": [
        ["node", "Bastusel_upper", "demand", -0.2579768519],
        ["node", "Bastusel_upper", "fix_node_state", {"type": "time_series", "data": {
↪"2018-12-31T23:00:00": 5581.44, "2019-01-07T23:00:00": 5417.28}}],
        ["node", "Bastusel_upper", "has_state", "value_true"],
    ],
    "relationship_parameter_values": [
        ["connection__node__node", ["Bastusel_to_Grytfors_disch", "Grytfors_upper",
↪"Bastusel_lower"], "connection_flow_delay", {"type": "duration", "data": "1h"}],
        ["unit__from_node", ["Bastusel_pwr_plant", "Bastusel_upper"], "unit_capacity",
↪ 127.5],
    ]
}
```

8.7.2 Importing

To import a file, go to **File –> Import**. The *Import file* dialog will pop up. Select the file type (SQLite, JSON, or Excel), enter the path of the file to import, and accept the dialog.

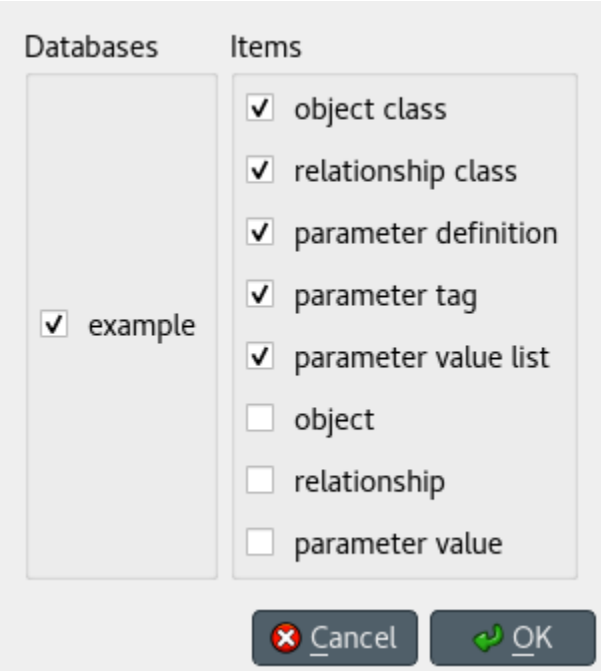
Note: Changes from import operations are not committed immediately to any databases. You need to commit them separately (see *Committing and rolling back*).

Tip: You can undo import operations using **Edit -> Undo**.

8.7.3 Exporting

Mass export

To export items in mass, go to **File –> Export**. The *Mass export items* dialog will pop up:



Select the databases you want to export under *Databases*, and the type of items under *Items*, then press **Ok**. The *Export file* dialog will pop up now. Select the file type (SQLite, JSON, or Excel), enter the path of the file to export, and accept the dialog.

Selective export

To export a specific subset of items, select the corresponding items in either *Object tree* and *Relationship tree*, right click on the selection to bring the context menu, and select **Export selected**.

The *Export file* dialog will pop up. Select the file type (SQLite, JSON, or Excel), enter the path of the file to export, and accept the dialog.

Session export

To export only uncommitted changes made in the current session, go to **File –> Export session**.

The *Export file* dialog will pop up. Select the file type (SQLite, JSON, or Excel), enter the path of the file to export, and accept the dialog.

Note: Export operations include all uncommitted changes.

8.7.4 Accessing/using exported files

Whenever you successfully export a file, a button with the file name is created in the *Exports* bar at the bottom of the form. To open the file in your registered program, press that button. To open the containing folder, click on the arrow next to the file name and select **Open containing folder** from the popup menu.

To add an exported SQLite file to a *Data Store* item in the current project, click on the arrow next to the file name and select **Add to project** from the popup menu. The *Add SQLite file to Project* dialog will pop up. Select a *Data Store* item from the list to become the host of the exported file. Alternatively, you can create a new *Data Store* item by typing in the last row. When you're done, press **OK**.

8.8 Committing and rolling back

Note: Changes are not immediately saved to the database(s). They need to be committed separately.

To commit your changes, select **Session -> Commit** from the menu bar, enter a commit message and press **Commit**. Any changes made in the current session will be saved into the database.

To undo *all* changes since the last commit, select **Session -> Rollback** from the menu bar.

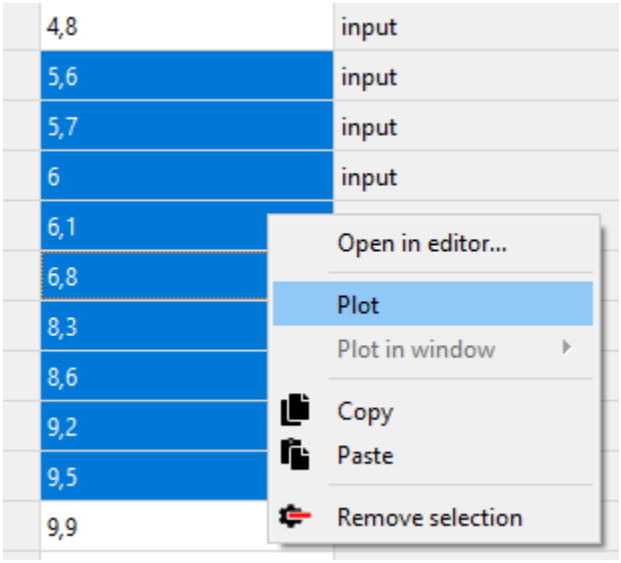
Tip: To undo/redo individual changes, use the **Undo** and **Redo** actions from the **Edit** menu.

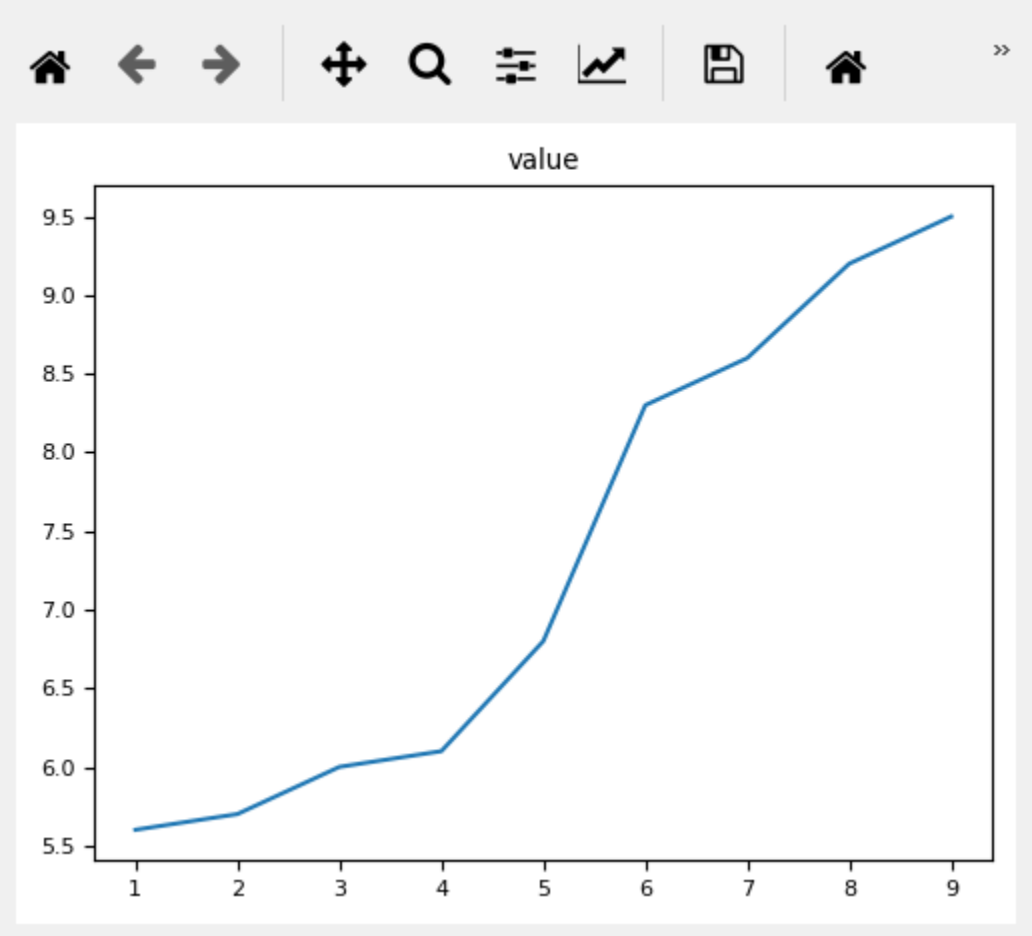
CHAPTER 9

Plotting

Basic data visualization is available in the Spine database editors. Currently, it is possible to plot plain parameter values as well as time series and maps.

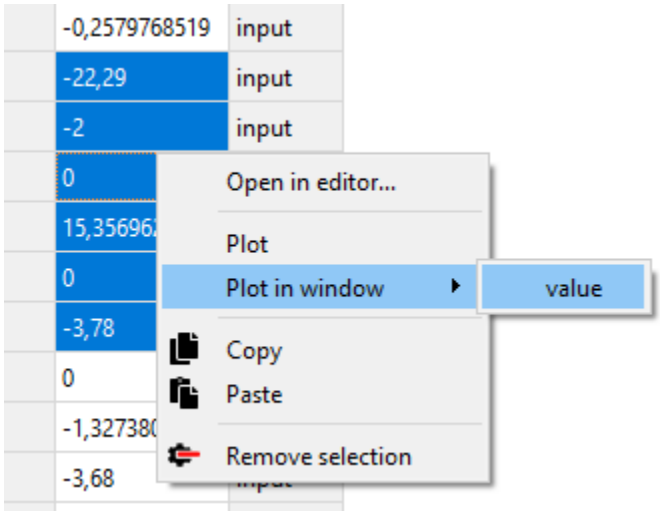
To plot a column, select the values from a table and then *Plot* from the **right click** popup menu.





Selecting data in multiple columns plots the selection in a single window.

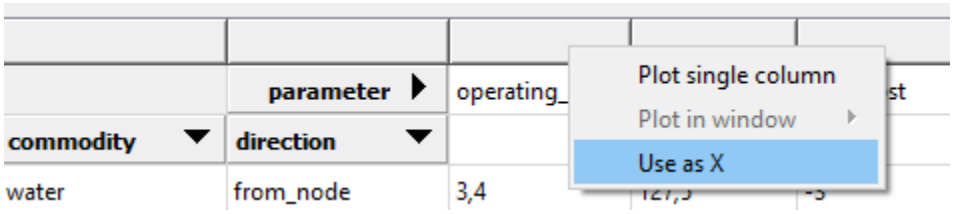
It is possible to add a plot to an existing plot window. Select the target plot window from the *Plot in window* submenu and the data will be added to the plot.



9.1 X axis for plain values

It is possible to plot plain values against X values given by a designated column in the pivot table.

To set a column as the X column **right click** the top empty area above the column header and select *Use as X* from the popup menu. An *(X)* in the topmost cell indicates that the column is designated as containing the X axis.



When selecting and plotting other columns in the same table the data will be plotted against the values in the X column instead of row numbers.

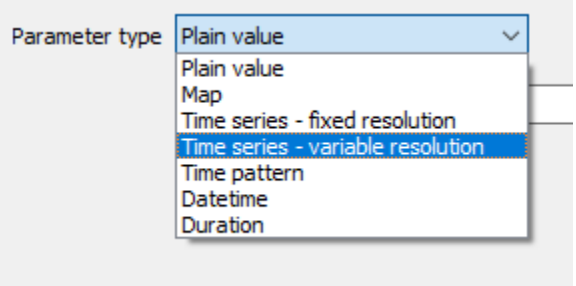
CHAPTER 10

Parameter value editor

Parameter value editor is used to edit object and relationship parameter values such as time series, time patterns or durations. It can also convert between different value types, e.g. from a time series to a time pattern.

The editor is available from a **right click** popup menu or by **double clicking** a parameter value in one of the Spine database editors.

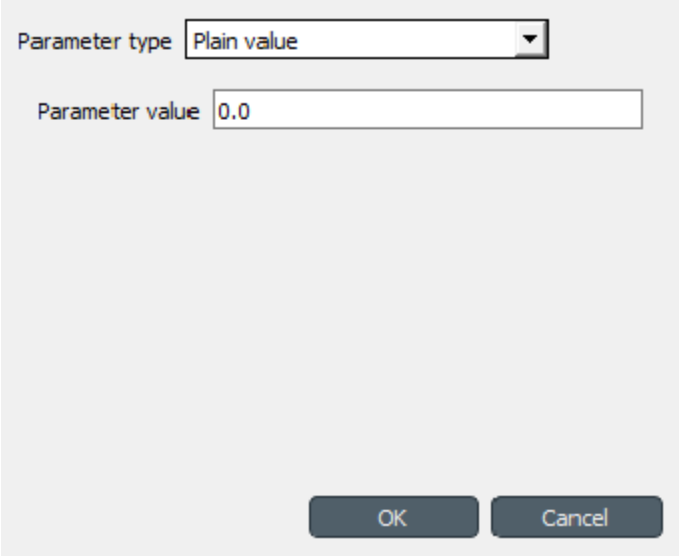
10.1 Choosing value type



The combo box at the top of the editor window allows changing the type of the current value.

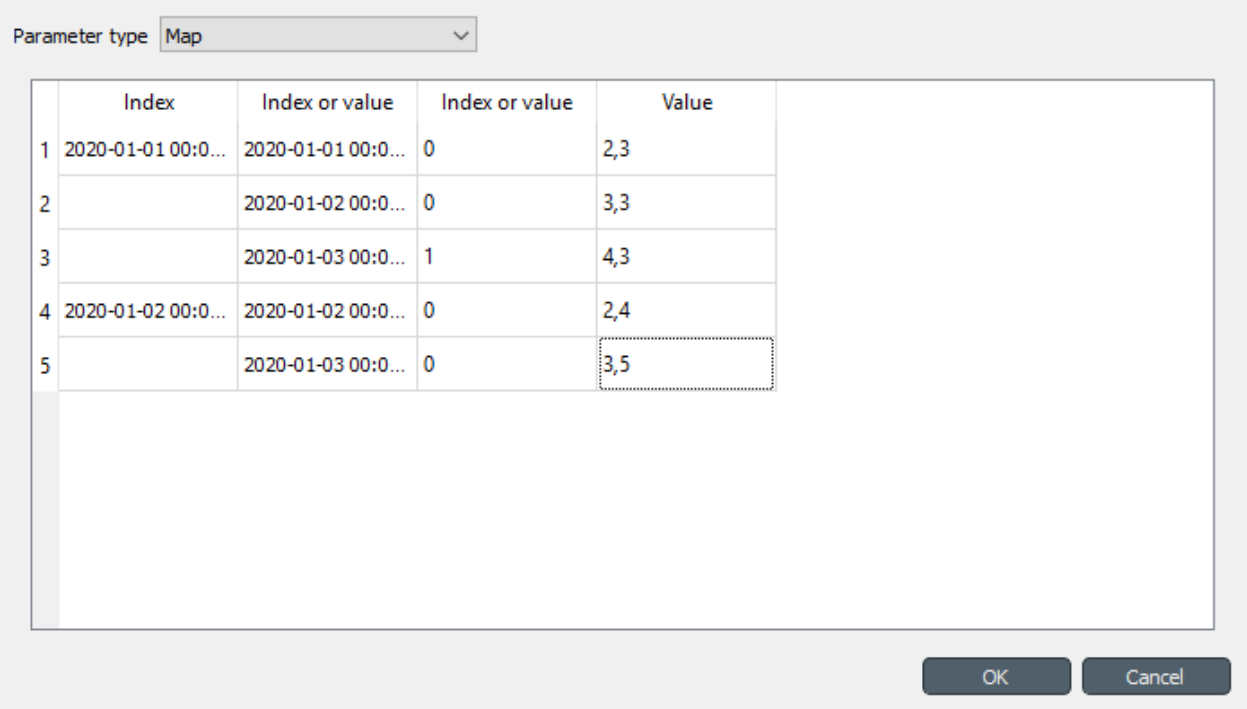
10.2 Plain values

The simplest parameter values are of the *Plain value* type. These are numbers or booleans which can be set by entering `true` or `false` on the *Parameter value* field.



10.3 Maps

Maps are nested data structures which can contain many different types of data including one and multi dimensional indexed arrays. The current support for maps in Parameter value editor is rather bare bones. The map is shown as a table where the last non-empty cells on each row contain the value while the preceding cells contain the value's index.

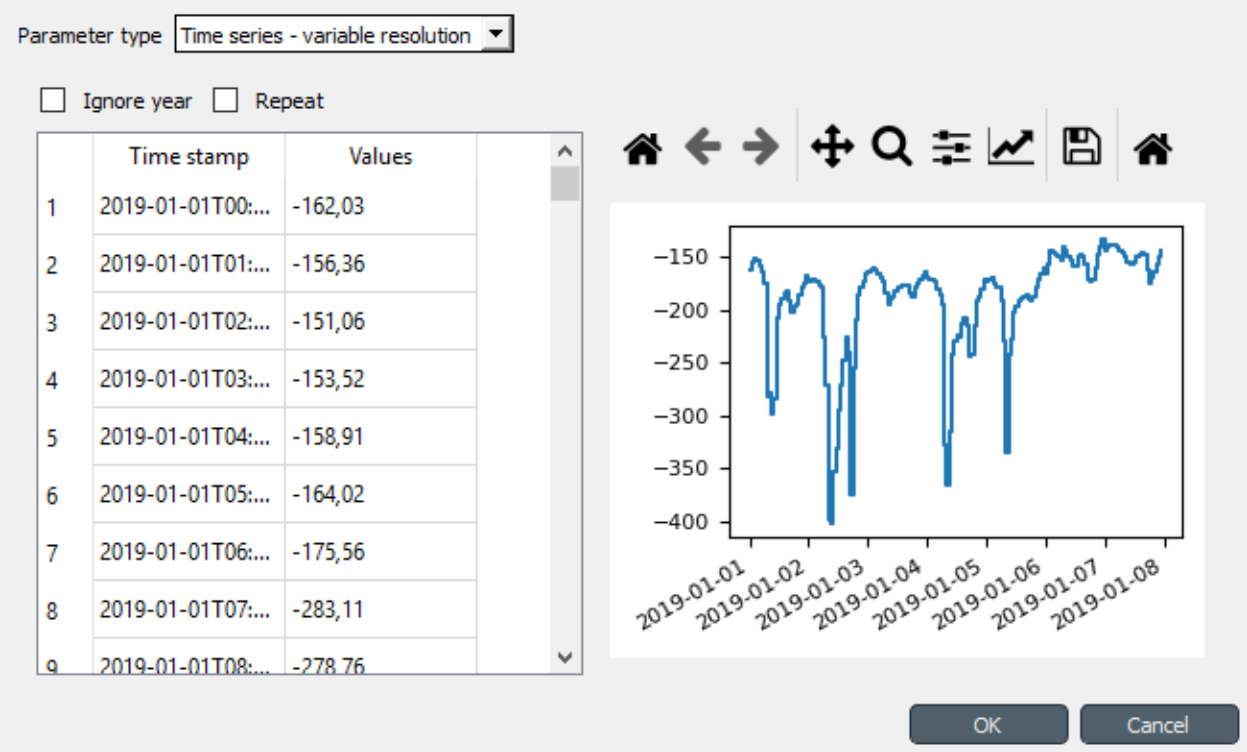


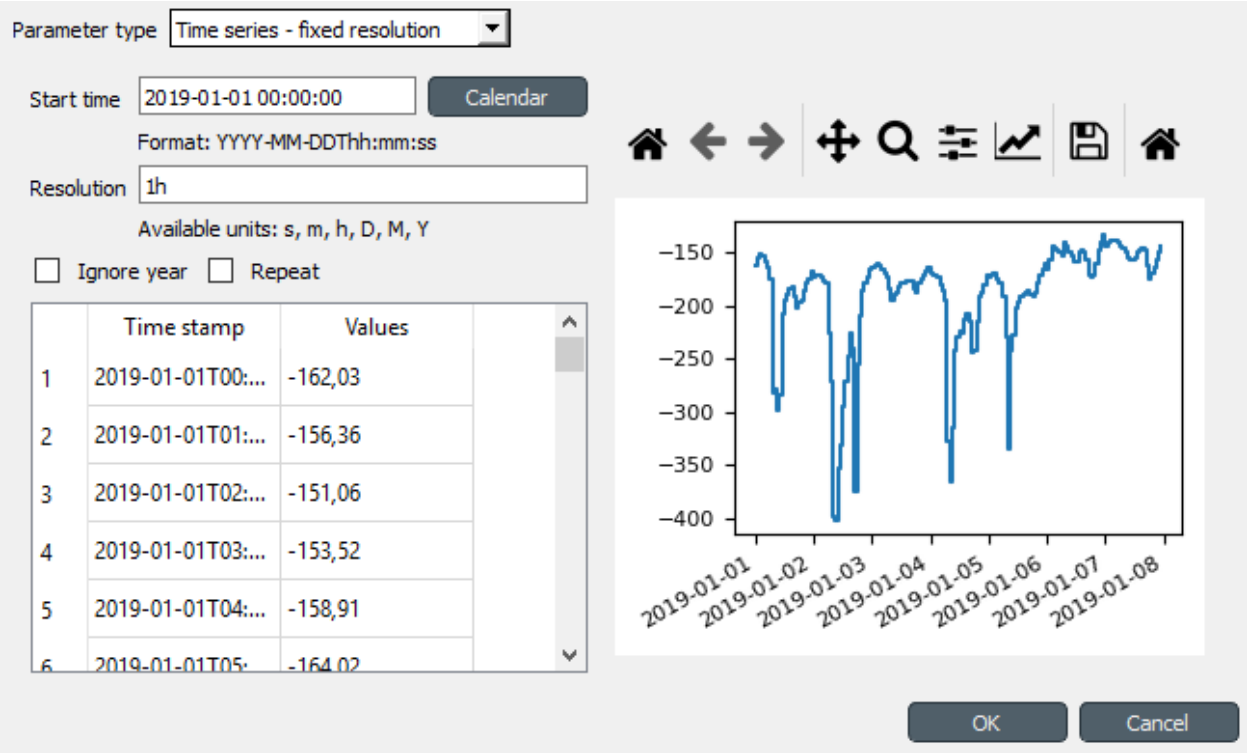
A **Right click** popup menu gives options to add rows or columns (effectively adds a new dimension to map) or trim empty columns from the right hand side.

At the moment the cell values have to be entered as JSON strings.

10.4 Time series

There are two types of time series: *variable* and *fixed resolution*. Variable resolution means that the time stamps can be arbitrary while in fixed resolution series the time steps between consecutive stamps are fixed.





The editor windows is split into two in both cases. The left side holds all the options and a table with all the data while the right side shows a plot of the series. The plot is not editable and is for visualization purposes only.

In the table rows can be added or removed from a popup menu available by a **right click**. Data can be copied and pasted by **Ctrl-C** and **Ctrl-V**. Copying from/to an external spreadsheet program is supported.

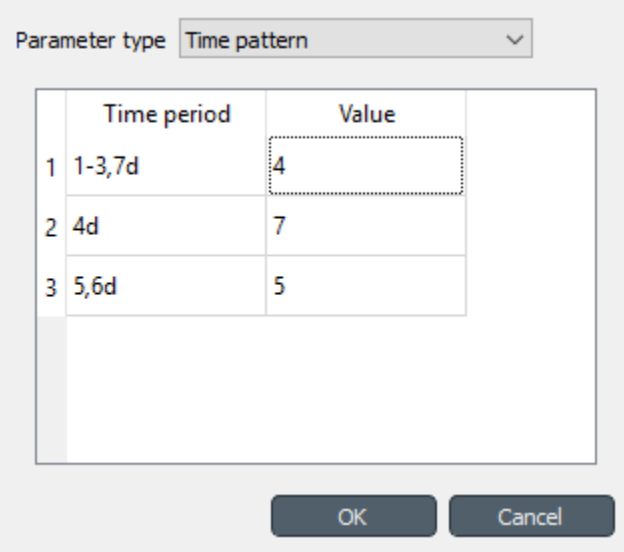
The time steps of a fixed resolution series are edited by the *Start time* and *Resolution* fields. The format for the start time is ISO8601. The *Resolution* field takes a single time step or a comma separated list of steps. If a list of resolution steps is provided then the steps are repeated so as to fit the data in the table.

The *Ignore year* option available for both variable and fixed resolution time series allows the time series to be used independent of the year. Only the month, day and time information is used by the model.

The *Repeat* option means that the time series is cycled, i.e. it starts from the beginning once the time steps run out.

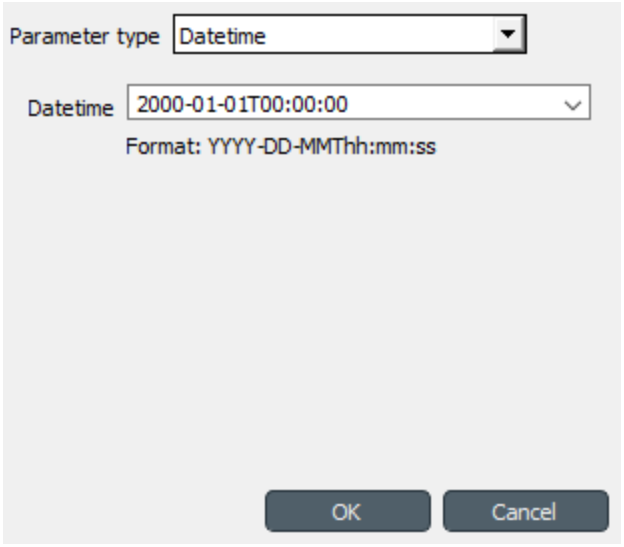
10.5 Time patterns

The time pattern editor holds a single table which shows the period on the right column and the corresponding values on the left. Inserting/removing rows and copy-pasting works as in the time series editor.



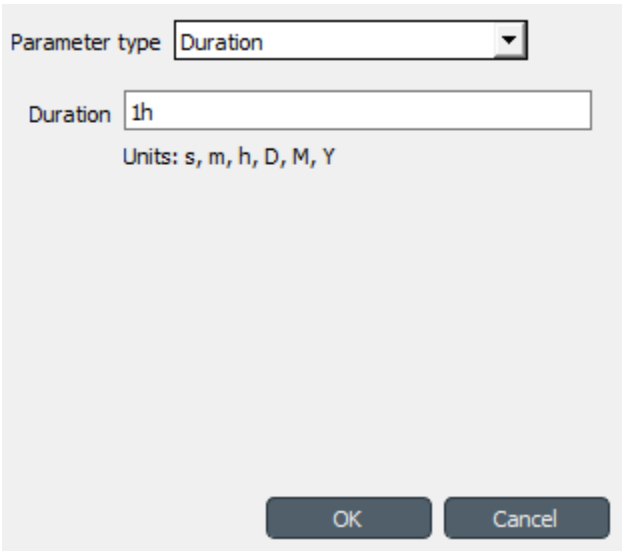
10.6 Datetimes

The datetime value should be entered in ISO8601 format.



10.7 Durations

A single value or a comma separated list of time durations can be entered to the *Duration* field.



CHAPTER 11

Importing and exporting data

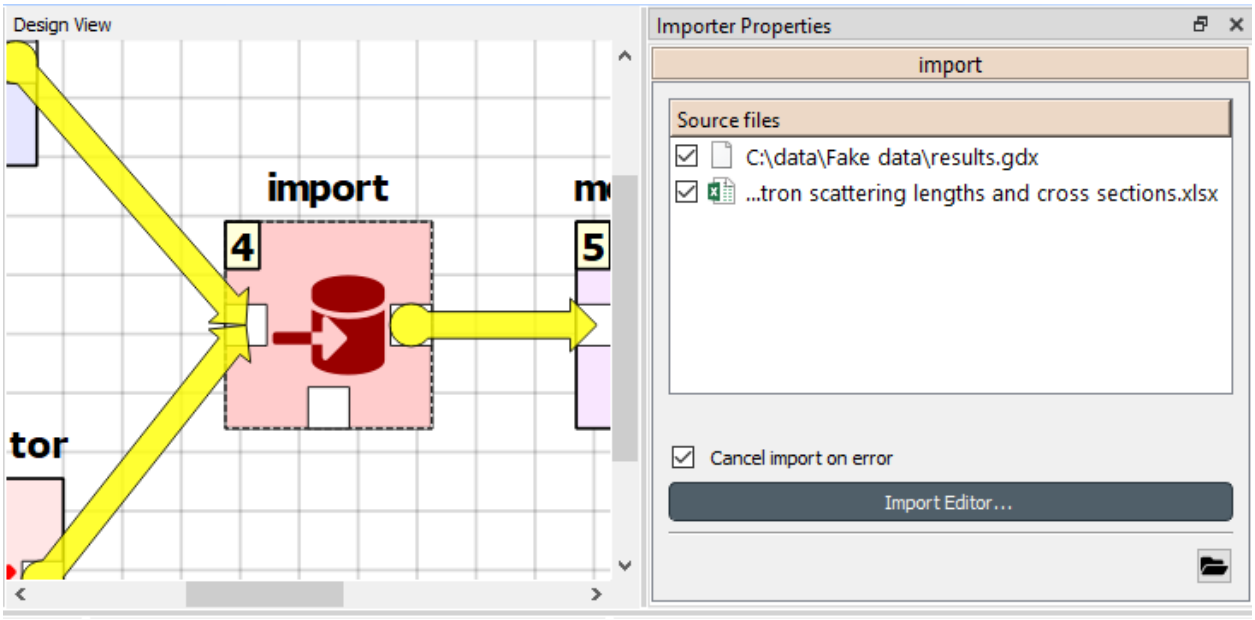
This section explains the different ways of importing and exporting data to and from a Spine database.

11.1 Importing data with Importer

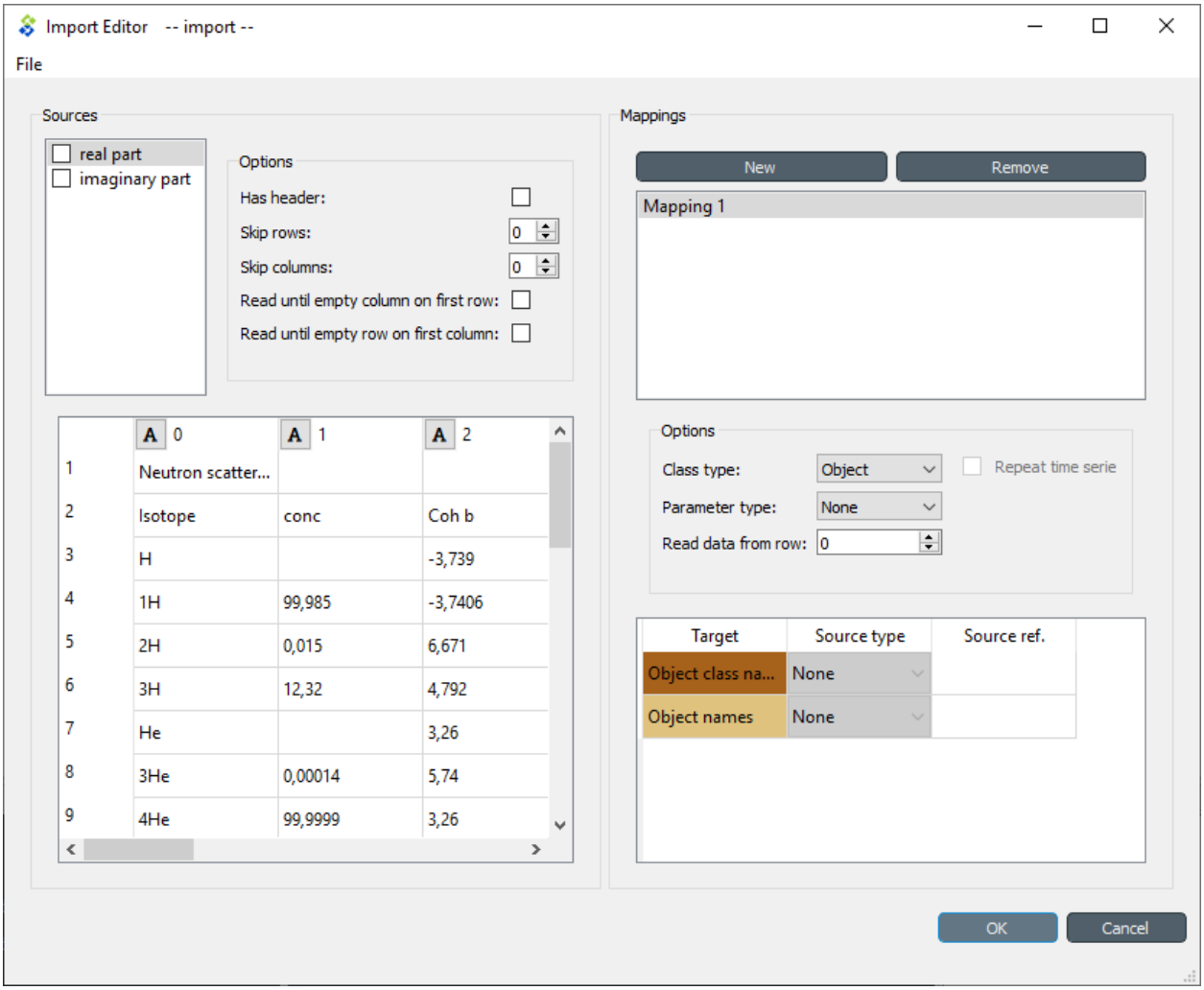
Data importing is handled by the Importer project item which can import tabulated and to some degree tree-structured data into a Spine database from various formats. The same functionality is also available in **Spine database editor** from **File->Import** but using an Importer item is preferred because then the process is documented and repeatable.

Tip: A Tool item can also be connected to Importer to import tool's output files to a database.

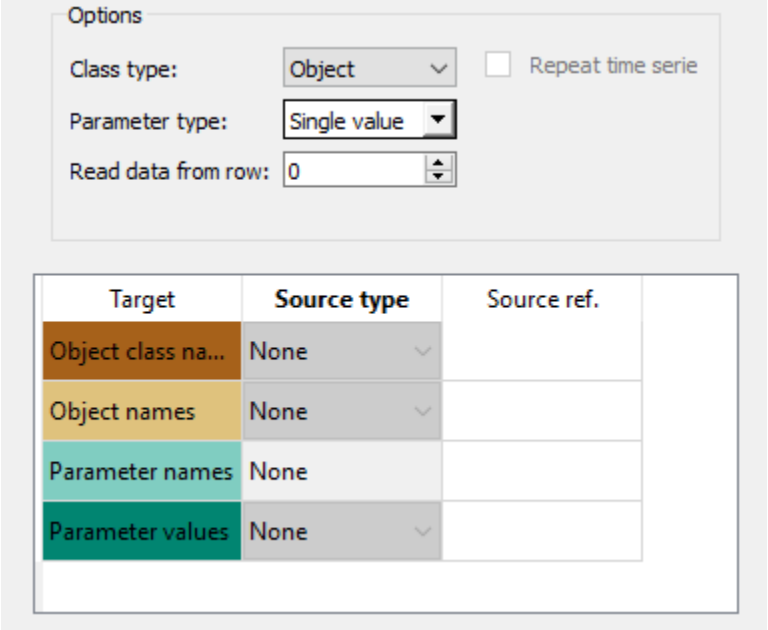
The heart of Importer is the **Import Editor** window in which the mappings from source data to Spine database entities are set up. The editor window can be accessed by the **Import Editor...** button in Importer's Properties tab. Note, that you have to select one of the files in the **Source files** list before clicking the button.



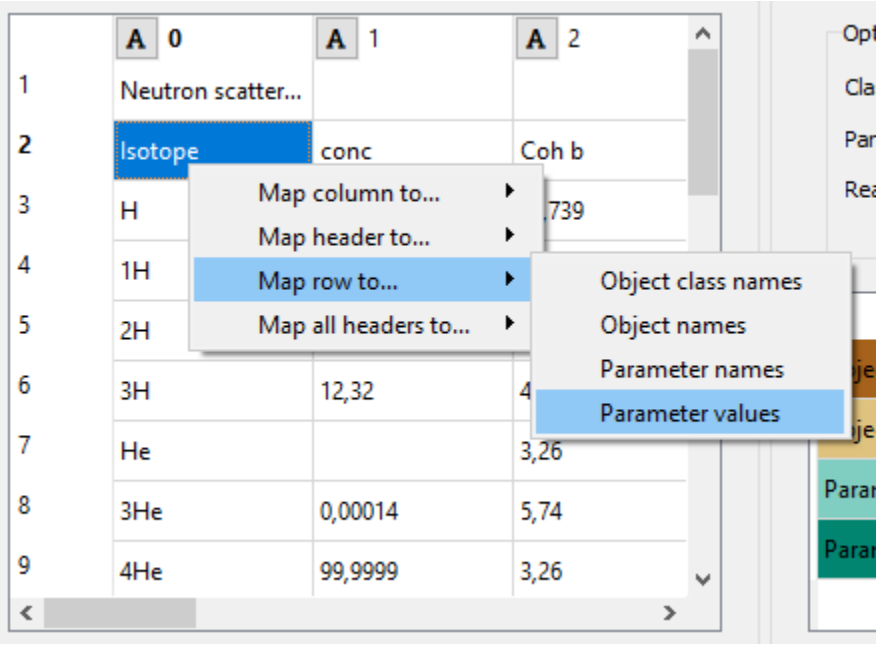
The **Import Editor** windows is divided into two parts: **Sources** shows all the 'sheets' contained in the file, some options for reading the file correctly, and a preview table to visualize and configure how the data on the selected sheet would be mapped. **Mappings**, on the other hand, shows the actual importing settings, the mappings from the input data to database entities.



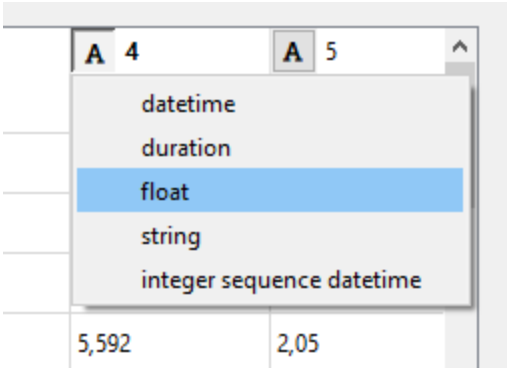
The options in the Mappings part declare if the currently selected sheet will be imported as an object or relationship and what type of parameters, if any, the sheet contains. The table can be used to configure how the input data is interpreted: which row or column contains the entity class names, parameter values, time stamps and so on.



It might be helpful to fill in the mapping options using the preview table in the Sources part. Right clicking on the table cells shows a popup menu that lets one to configure how the rows and colunms are read upon importing.



An important aspect of data import is whether each item in the input data should be read as a string, a number, a time stamp, or something else. By default all input data is read as strings. However, more often than not things like parameter values are actually numbers. It is possible to control what type of data each column (and, sometimes, each row) contains from the preview table. Clicking the data type indicator button on column headers pops up a menu with a selection of available data types. Right clicking the column header also gives the opportunity to change the data type of all columns at once.



11.2 Exporting to GAMS

Note: You need to have GAMS installed to use this functionality. However, you do not need to own a GAMS license as the demo version works just as well.

Note: The bitness (32 or 64bit) of GAMS has to match the bitness of the Python interpreter.

Databases can be exported to GAMS `.gdx` files by the *Exporter* project item. When a project is executed, *Exporter* writes its output files to its data folder and forwards file paths to project items downstream. If a *Tool* is to use such a file, remember to add the file as one of the *Tool specification*'s input files!

The mapping between entities in a Spine database and GAMS is as follows:

Database entity	GAMS entity
Object class	Universal set (or domain)
Object	Universal set member
Object parameter	Parameter
Relationship class	Subset of universal sets
Relationship	Subset member
Relationship parameter	Parameter

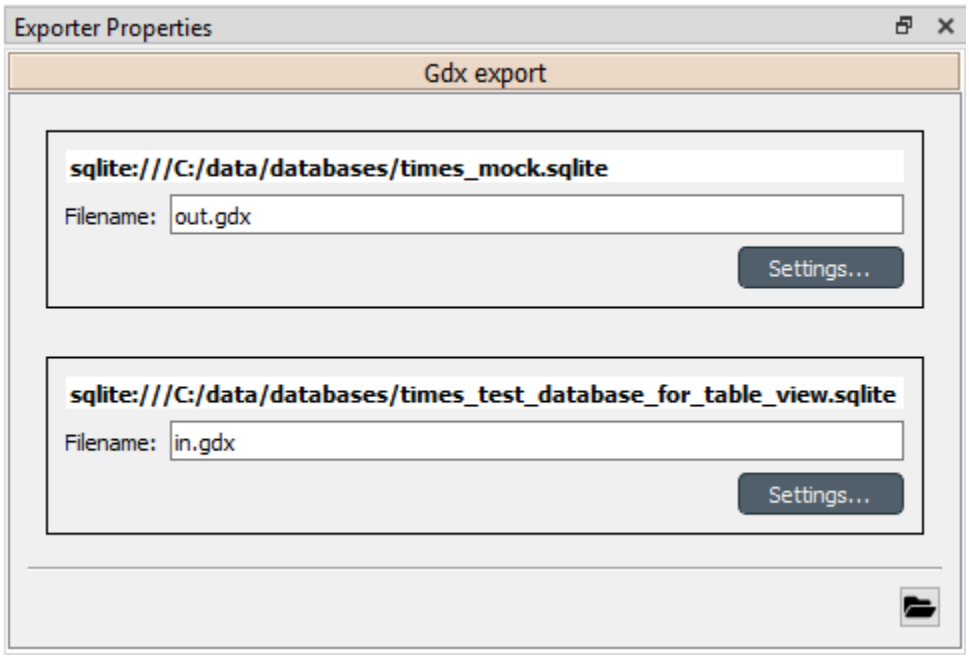
Note: Currently, it is not possible to use subsets (relationship classes) as dimensions for other subsets due to technical limitations. For example, if there is a domain **A(*)** and a subset **foo(A)**, a subset of **foo** has to be expressed as **bar(A)** instead of **bar(foo)**.

It is also possible to designate a single object class as a *Global parameter*. The parameters of the objects of that class will be exported as GAMS scalars.

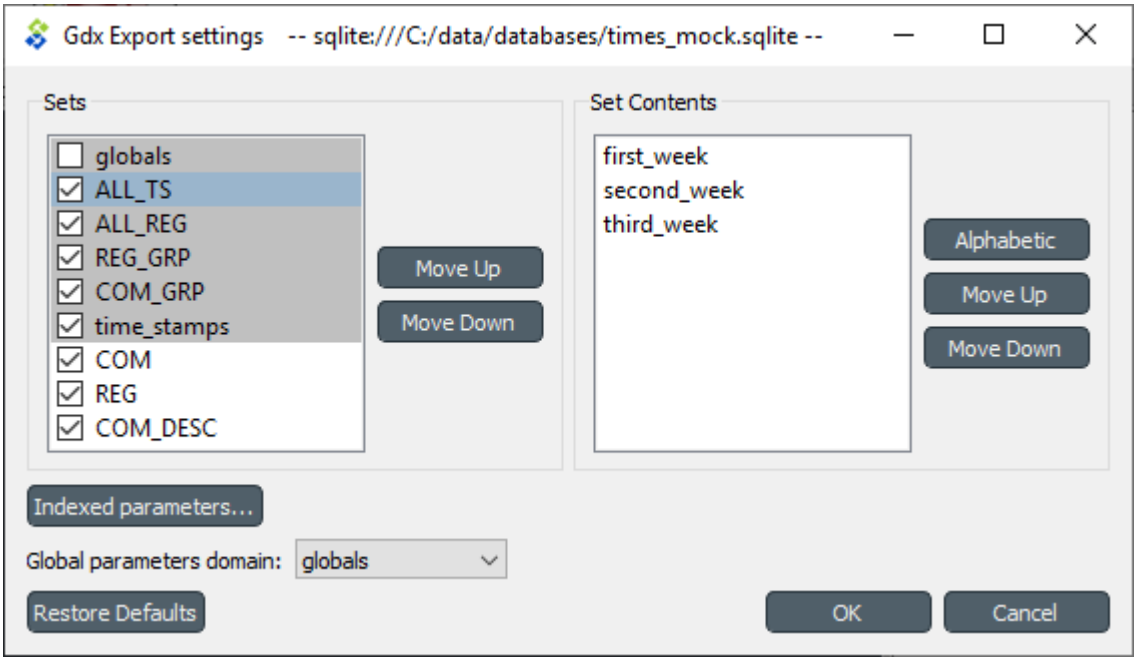
Some GAMS models need their data to be in a specific order in the `.gdx`. This is not directly supported by the database. Rather, user has to specify the desired exporting order using the *Exporter* item's settings.

11.2.1 Exporter Project Item

The image below shows the settings tab of *Exporter* with two *Data Sources* connected to it.



For each connected *Data Store* a box with the database's URL and export file name field is shown on the tab. The *Settings...* buttons open *Gdx Export settings* windows to allow editing database specific export parameters such as the order in which entities are exported from the database.



The *Gdx Export settings* window (see above) contains a *Sets* list which shows all GAMS sets (gray background) and subsets that are available in the database. The sets are exported in the order they are shown in the list. The *Move Up* and *Move Down* buttons can be used to move the selected set around. Note that you cannot mix sets with subsets so all sets always get exported before the subsets.

The checkbox next to the set name is used to control which sets are actually exported. Note that it is not possible to change this setting for certain sets. Global parameters domain is never exported, only its parameters which become GAMS scalars. Further, sets created for *Indexed paramaters* are always exported.

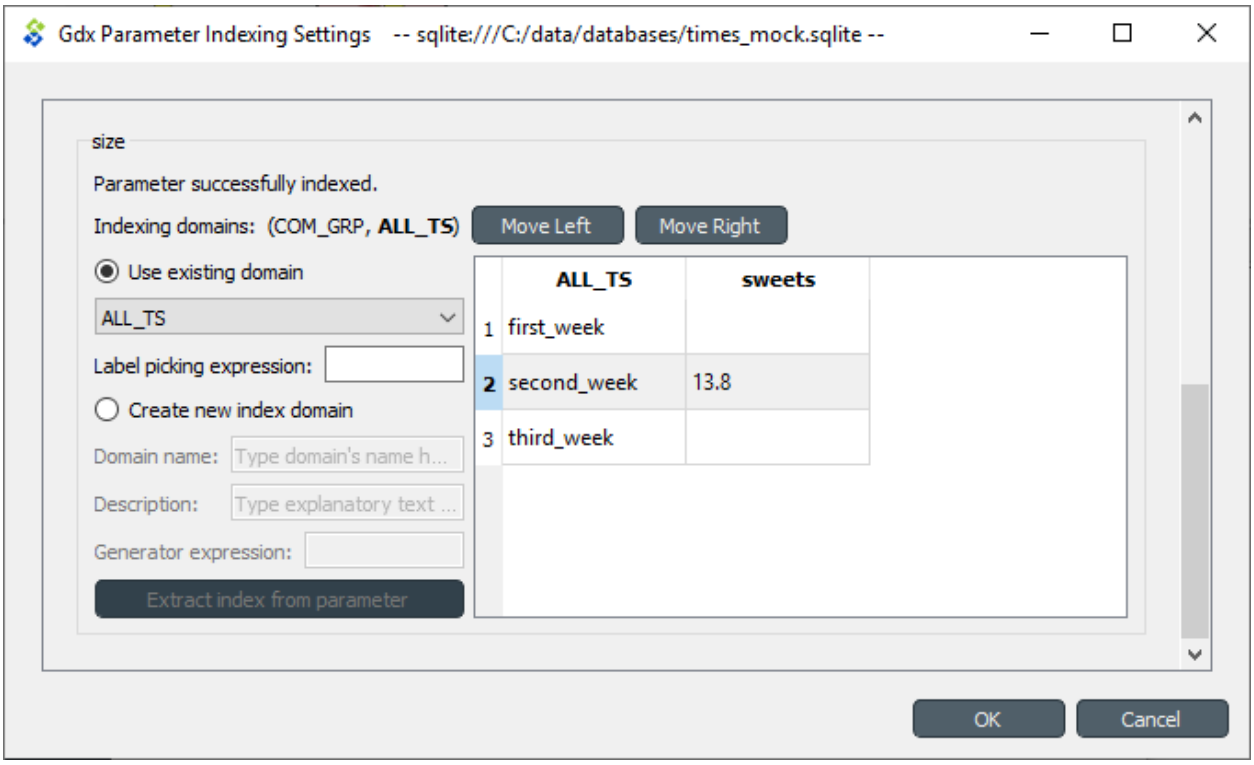
The *Set Contents* box lists the members of the selected set or subset. Their order of export can be changed the same way as with sets by *Move Up* and *Move Down*. The *Alphabetic* button sorts the members alphabetically.

Time series and time patterns cannot be exported as-is. They need to be tied up to a GAMS set. This can be achieved from the window that opens from the *Indexed parameters...* button. See the *Exporting time series and patterns* section below for more information.

Finally, one of the sets can be designated as the global parameter set. This is achieved by choosing the set's name in the *Global parameters domain* box. Note that this set is not exported, only its parameters are. They end up as GAMS scalars.

11.2.2 Exporting time series and patterns

Since GAMS has no notion of time series or time patterns these types need special handling when exported to a `.gdx` file. Namely, the time stamps or time periods (i.e. parameter indexes) need be available as GAMS sets in the exported file. It is possible to use an existing set or create a new one for this purpose. The functionality is available in *Gdx Parameter Indexing Settings* window accessible from the *Indexed Parameters...* button.

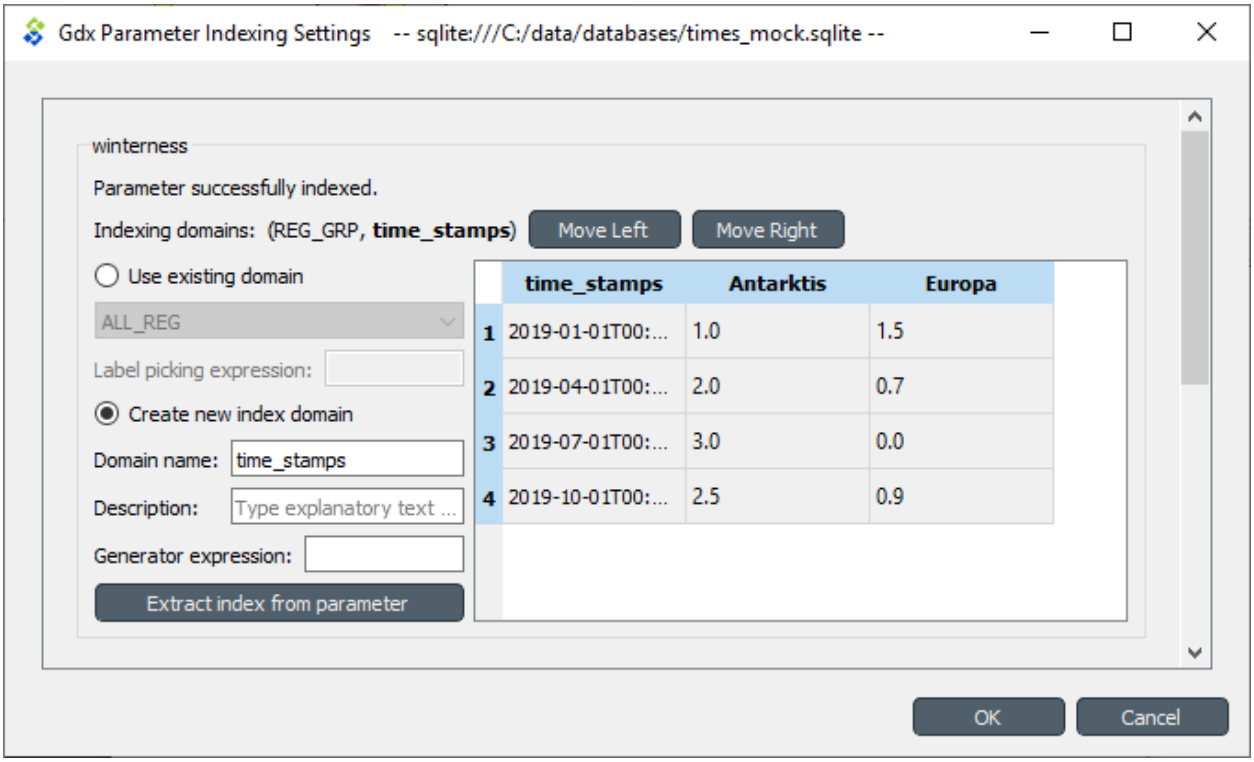


The above figure shows the indexing settings when an existing GAMS set is used to replace the original time stamps of a time series in a parameter called 'size'. The choice between using an existing set or creating a new one can be changed by the *Use existing domain* and *Create new index domain* radio buttons. When using an existing set it is selected by the combo box. In the above figure, *ALL TS* set is used for indexing.

In case of existing set it is possible that not all the set's contents are used for indexing. The table occupying the right side of the above figure shows which of the set's keys index which parameter values. The first column contains the keys of the currently selected set whereas the other columns contain the parameter's values, one column for each object that has the parameter. Selecting and deselecting rows in the table changes the indexing as only the keys on selected rows are used to index the parameter. **Shift**, **ctrl** and **ctrl-A** help in manual selection. If the selected indexes have certain pattern it might be useful to utilize the *Label picking expression* field which selects the set keys using a Python expression returning a boolean value. Some examples:

Expression	Effect
`i == 3`	Select the third row only
`i % 2 == 0`	Select even rows
`(i + 1) % 2 == 0 and i != 9`	Select odd rows except row 9

The *Indexing domains* list allows to shuffle the order of the parameter's dimensions. The **bold** dimension is the new dimension that is added to the parameter. It can be moved around by the *Move Left* and *Move Right* buttons.



It is possible to create a new indexing set by choosing *Create new index domain* as shown in the figure above. *Domain name* is mandatory for the new domain. A *Description* can also be provided but it is optional. There are two options to generate the index keys: extract the time stamps or time periods from the parameter itself or generate them using a Python expression. The *Extract index from parameter* button can be used to extract the keys from the parameter. The *Generator expression* field, on the other hand, is used to generate index keys for the new set. The expression should return Python object that is convertible to string. Below are some example expressions:

Expression	Keys
`i`	1, 2, 3,...
`f"{i - 1:04}"`	0000, 0001, 0002,...
`f"T{i:03}"`	T001, T002, T003,...

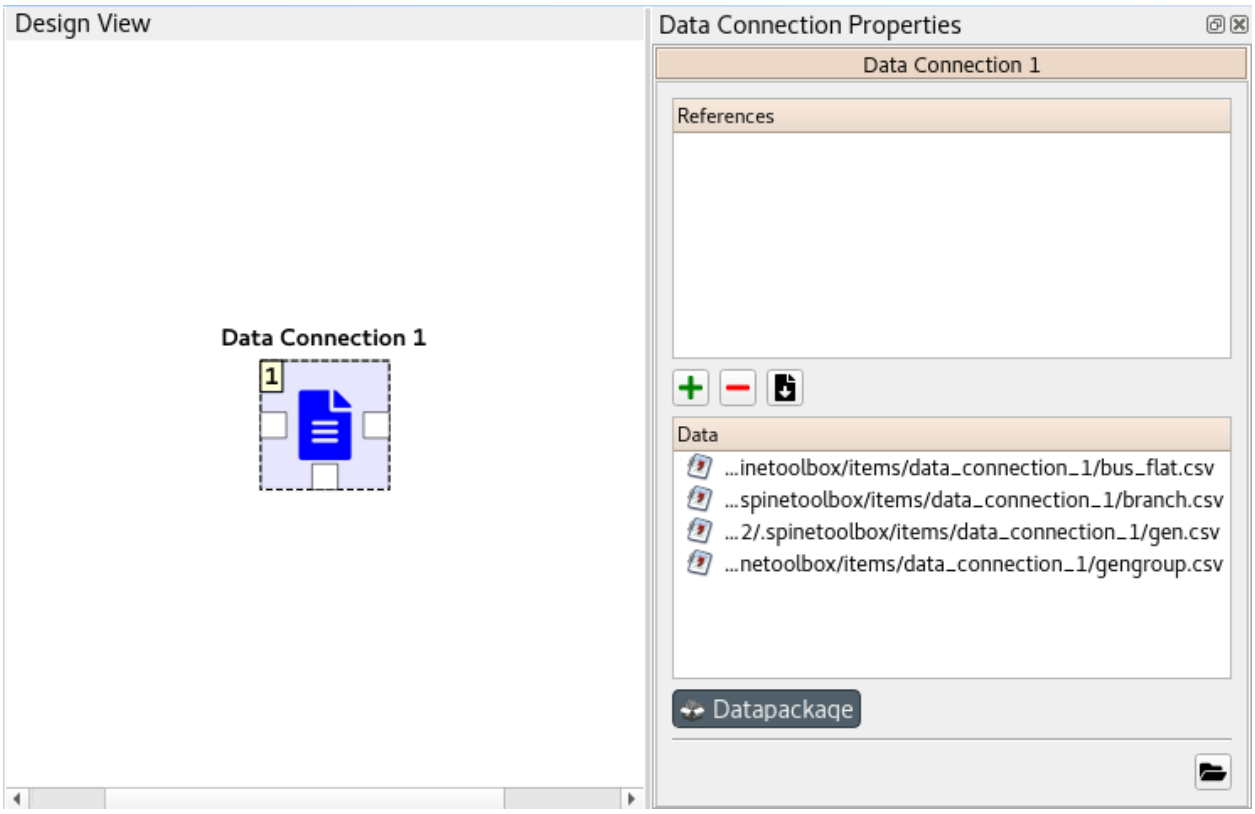
CHAPTER 12

Spine datapackage editor

Note: This section is a work in progress.

This section describes the Spine datapackage editor, used to interact with tabular data and export it into Spine format.

To open the Spine datapackage editor, select a **Data Connection** with *CSV files* in it, and press the **Datapackage** button in its *Properties*:



File Edit View

Resources

name	sourc
bus_flat	bus_flat.c
branch	branch.cs
gen	gen.csv
gengroup	gengroup

Data

f_bus	t_bus	br_r	br_x	br_b	tap	rate_a	rate_b	outage_rate	outage_duratio
1	2	0.003	0.014	0.461	0	193	200	0.24	16
1	3	0.055	0.211	0.057	0	208	220	0.51	10
1	5	0.022	0.085	0.023	0	208	220	0.33	10
2	4	0.033	0.127	0.034	0	208	220	0.39	10
2	6	0.05	0.192	0.052	0	208	220	0.48	10
3	9	0.031	0.119	0.032	0	208	220	0.38	10
3	24	0.002	0.084	0	1.015	510	600	0.02	768
4	9	0.027	0.104	0.028	0	208	220	0.36	10
5	10	0.023	0.088	0.024	0	208	220	0.34	10
6	10	0.014	0.061	2.459	0	193	200	0.33	35
7	8	0.016	0.061	0.017	0	208	220	0.3	10
8	9	0.043	0.165	0.045	0	208	220	0.44	10
8	10	0.043	0.165	0.045	0	208	220	0.44	10
9	11	0.002	0.084	0	1.03	510	600	0.02	768
9	12	0.002	0.084	0	1.03	510	600	0.02	768
10	11	0.002	0.084	0	1.015	510	600	0.02	768
10	12	0.002	0.084	0	1.015	510	600	0.02	768
11	13	0.006	0.048	0.1	0	600	625	0.4	11
11	14	0.005	0.042	0.088	0	600	625	0.39	11
12	13	0.006	0.048	0.1	0	600	625	0.4	11
12	23	0.012	0.097	0.203	0	600	625	0.52	11
13	23	0.011	0.087	0.182	0	600	625	0.49	11
14	16	0.005	0.059	0.082	0	600	625	0.38	11
15	16	0.002	0.017	0.036	0	600	625	0.33	11
15	21	0.006	0.049	0.103	0	600	625	0.41	11
15	21	0.006	0.049	0.103	0	600	625	0.41	11
15	24	0.007	0.052	0.109	0	600	625	0.41	11
16	17	0.003	0.026	0.055	0	600	625	0.35	11
16	19	0.003	0.023	0.049	0	600	625	0.34	11
17	18	0.002	0.014	0.03	0	600	625	0.32	11
17	22	0.014	0.105	0.221	0	600	625	0.54	11
18	21	0.003	0.026	0.055	0	600	625	0.35	11
18	21	0.003	0.026	0.055	0	600	625	0.35	11

Fields

name	type	primary key?
f_bus	integer	
t_bus	integer	
br_r	number	
br_x	number	
br_b	number	
tap	integer	
rate_a	integer	
rate_b	integer	
outage_rate	number	
outage_duration	integer	
weighting_factor	number	
br_status	integer	
angmin	integer	
angmax	integer	
shift	integer	
internal	integer	

Foreign keys

	fields	reference resource	reference fields
1			

CHAPTER 13

Terminology

Here is a list of definitions related to Spine project, SpineOpt.jl, and Spine Toolbox.

- **Arc** Graph theory term. See *Connection*.
- **Case study** Spine project has 13 case studies that help to improve, validate and deploy different aspects of the SpineOpt.jl and Spine Toolbox.
- **Connection** an arrow on Spine Toolbox Design View that is used to connect project items to each other to form a DAG.
- **Data Connection** is a project item used to store a collection of data files that may or may not be in Spine data format. It facilitates data transfer from original data sources e.g. spreadsheet files to Spine Toolbox. The original data source file does not need to conform to the format that Spine Toolbox is capable of reading, since there we can use an interpreting layer (Importer) between the raw data and the Spine format database (Data Store).
- **Data Package** is a data container format consisting of a metadata descriptor file (`datapackage.json`) and resources such as data files.
- **Data sources** are all the original, unaltered, sources of data that are used to generate necessary input data for Spine Toolbox tools.
- **Data Store** is a project item. It's a Spine Toolbox internal data container which follows the Spine data model. A data store is implemented using a database, it may be, for example, an SQL database.
- **Design View** A *sub-window* on Spine Toolbox main window, where project items and connections are visualized.
- **Direct predecessor** Immediate predecessor. E.g. in DAG *x*->*y*->*z*, direct predecessor of node *z* is node *y*. See also predecessor.
- **Direct successor** Immediate successor. E.g. in DAG *x*->*y*->*z*, direct successor of node *x* is node *y*. See also successor.
- **Directed Acyclic Graph (DAG)** Finite directed graph with no directed cycles. It consists of vertices and edges. In Spine Toolbox, we use project items as vertices and connections as edges to build a DAG that represents a data processing chain (workflow).
- **Edge** Graph theory term. See *Connection*

- **Exporter** is a project item that allows exporting a Spine data structure from a Data Store into a file which can be used as an input file in a Tool.
- **Importer** is a project item that can be used to import data from e.g. an Excel file, transform it to Spine data structure, and into a Data Store.
- **Node** Graph theory term. See *Project item*.
- **Predecessor** Graph theory term that is also used in Spine Toolbox. Preceding project items of a certain project item in a DAG. For example, in DAG *x*->*y*->*z*, nodes *x* and *y* are the predecessors of node *z*.
- **Project** in Spine Toolbox consists of project items and connections, which are used to build a data processing chain for solving a particular problem. Data processing chains are built and executed using the rules of Directed Acyclic Graphs. There can be any number of project items in a project.
- **Project item** Spine Toolbox projects consist of project items. Project items together with connections are used to build Directed Acyclic Graphs (DAG). Project items act as nodes and connections act as edges in the DAG.
- **Scenario** A scenario is a meaningful data set for the target tool.
- **Spine data structure** Spine data structure defines the format for storing and moving data within Spine Toolbox. A generic data structure allows representation of many different modelling entities. Data structures have a class defining the type of entity they represent, can have properties and can be related to other data structures. Spine data structures can be manipulated and visualized within Spine Toolbox while SpineOpt.jl will be able to directly utilize as well as output them.
- **SpineOpt.jl** An interpreter, which formulates a solver-ready mixed-integer optimization problem based on the input data and the equations defined in the SpineOpt.jl. Outputs the solver results.
- **Source directory** In context of Tool specifications, a source directory is the directory where the main program file of the Tool specification is located. This is also the recommended place for saving the Tool specification file (.json).
- **Successor** Graph theory term that is also used in Spine Toolbox. Following project items of a certain project item in a DAG. For example, in DAG *x*->*y*->*z*, nodes *y* and *z* are the successors of node *x*.
- **Tool** is a project item that is used to execute Python, Julia, GAMS, executable scripts, or simulation models. This is done by creating a Tool specification defining the script or program the user wants to execute in Spine Toolbox. Then you need to attach the Tool specification to a Tool project item. Tools can be used to execute a computational process or a simulation model, or it can also be a process that converts data or calculates a new variable. In general, Tools may take some data as input and produce an output.
- **Tool specification** is a JSON structure that contains metadata required by Spine Toolbox to execute a computational process or a simulation model. The metadata contains; type of the program (Python, Julia, GAMS, executable), main program file (which can be e.g. a Windows batch (.bat) file or for Python scripts this would be the .py file where the __main__() method is located), All additional required program files, any optional input files (e.g. data), and output files. Also any command line arguments can be defined in a Tool specification. SpineOpt.jl is a Tool specification from Spine Toolbox's point-of-view.
- **Use case** Potential way to use Spine Toolbox. Use cases together are used to test the functionality and stability of Spine Toolbox and SpineOpt.jl under different potential circumstances.
- **Vertice** Graph theory term. See *Project item*.
- **View** A project item that can be used for visualizing project data.
- **Work directory** Tool specifications can be executed in *Source directory* or in *work directory*. When a Tool specification is executed in a work directory, Spine Toolbox creates a new *work* directory, copies all required and optional files needed for running the Tool specification to this directory and executes it there. After execution has finished, output or result files can be copied into a timestamped (archive) directory from the work directory.

CHAPTER 14

Dependencies

Spine Toolbox requires Python 3.6 or Python 3.7. Python 3.8 is not supported yet.

Spine Toolbox uses code from packages and/or projects listed in the table below. Required packages must be installed for the application to start. Users can choose the SQL dialect API (pymysql, pyodbc psycopg2, and cx_Oracle) they want to use. These can be installed in Spine Toolbox when needed. If you want to deploy the application by using the provided *setup.py* file, you need to install *cx_Freeze* package (6.0b1 version or newer is recommended). All version numbers are minimum versions except for pyside2, where the version should be 5.14. Version 5.15 is not supported (yet).

14.1 Required packages

The following packages are installed when spinetoolbox package is installed via `setup.py` or `requirements.txt`

Package name	Version	License
pyside2	>=5.14, <5.15	LGPL
datapackage	1.2.3	MIT
jupyter-client	<5.3.2	BSD
qtconsole	4.3.1	BSD
sqlalchemy	1.2.6	MIT
spinedb_api	0.1.15	LGPL
spine_engine	0.4.0	LGPL
openpyxl	2.5.0	MIT/Expat
numpy	1.15.1	BSD
matplotlib	3.0	BSD
scipy	1.1.0	BSD
networkx	2.2	BSD
pymysql	0.9.2	MIT
pyodbc	4.0.23	MIT
psycopg2	2.7.4	LGPL
cx_Oracle	6.3.1	BSD
python-dateutil	2.8.0	PSF
pandas	0.24.0	BSD
jsonschema	2.6	MIT
gdx2py	2.0.4	MIT

14.1.1 Developer packages

The developer packages are available from `dev-requirements.txt`. Sphinx and sphinx_rtd_theme packages are needed for building the user guide. Black is used for code formatting while pylint does linting. Pre-commit hook enables automatic code formatting at git commit.

Package name	Version	License
black	19.3b0	MIT
pre-commit	1.16.1	MIT
pylint	2.3.0	GPL
sphinx	1.7.5	BSD
sphinx_rtd_theme	0.4.0	MIT
recommonmark	0.5.0	MIT
sphinx-autoapi	1.1.0	MIT

CHAPTER 15

Contribution Guide for Spine Toolbox

All are welcome to contribute! This guide is based on a set of best practices for open source projects [JF18].

15.1 Reporting Bugs

15.1.1 Due Diligence

Before submitting a bug report, please do the following:

Perform basic troubleshooting steps.

1. **Make sure you're on the latest version.** If you're not on the most recent version, your problem may have been solved already! Upgrading is always the best first step.
2. **Try older versions.** If you're already on the latest release, try rolling back a few minor versions (e.g. if on 1.7, try 1.5 or 1.6) and see if the problem goes away. This will help the devs narrow down when the problem first arose in the commit log.
3. **Try switching up dependency versions.** If you think the problem may be due to a problem with a dependency (other libraries, etc.). Try upgrading/downgrading those as well.
4. **Search the project's bug/issue tracker to make sure it's not a known issue.** If you don't find a pre-existing issue, consider checking with the maintainers in case the problem is non-bug-related.

15.1.2 What to Put in Your Bug Report

Make sure your report gets the attention it deserves: bug reports with missing information may be ignored or punted back to you, delaying a fix. The below constitutes a bare minimum; more info is almost always better:

1. What version of the Python interpreter are you using? E.g. Python 2.7.3, Python 3.6?
2. What operating system are you on? Windows? (Vista, 7, 8, 8.1, 10). 32-bit or 64-bit? Mac OS X? (e.g. 10.7.4, 10.9.0) Linux (Which distro? Which version of that distro? 32 or 64 bits?) Again, more detail is better.

3. Which version or versions of the software are you using? If you have forked the project from Git, which branch and which commit? Otherwise, supply the application version number (Help->About menu). Also, ideally you followed the advice above and have ruled out (or verified that the problem exists in) a few different versions.
4. How can the developers recreate the bug? What were the steps used to invoke it. A screenshot demonstrating the bug is usually the most helpful thing you can report (if applicable) Relevant output from the Event Log or debug messages from the console of your run, should also be included.

15.2 Feature Requests

The developers of Spine Toolbox are happy to hear new ideas for features or improvements to existing functionality. The format for requesting new features is free. Just fill out the required fields on the issue tracker and give a description of the new feature. A picture accompanying the description is a good way to get your idea into development faster. But before you make a new issue, check that there isn't a related idea already open in the issue tracker. If you have an idea on how to improve an existing issue, just join the conversation.

15.3 Submitting features/bugfixes

If you feel like you can fix a bug that's been bothering you or you want to add a new feature to the application but the devs seem to be too busy with something else, please follow the instructions in the following sections on how to contribute code.

15.3.1 Coding Style

Follow the style you see used in the repository! Consistency with the rest of the project always trumps other considerations. It doesn't matter if you have your own style or if the rest of the code breaks with the greater community - just follow along.

Spine Toolbox coding style follows PEP-8 style guide for Python code with the following variations:

- Maximum line length is 120 characters. Longer lines are acceptable if there's a sound reason.
- Google style docstrings with the title and input parameters are required for all classes, functions, and methods. For small functions or methods only the summary is necessary. Return types are highly recommended but not required if it is obvious what the function or method returns.
- Other deviations from PEP-8 can be discussed if there are good reasons.

15.3.2 Commit messages

The commit message should tell *what* was changed and *why*. Details on *how* it was done can usually be left out, if the code itself is self-explanatory (remeber source comments too!). Separate the subject line from the body with a blank line. The subjet line (max. 50 chars) should explain in condensed form what happened using imperative mood, i.e. using verbs like 'change', 'fix' or 'add'. Start the subject line with a capital letter. Do not use the issue number on the subject line, as it does not tell much to a person who's not aware of that particular issue. For more info see Chris Beams' 'Seven rules of of a great Git commit message' [CB14].

A good example (insipred by [CB14])

```
Fix bugs when updating parameters in foo and bar

Body of the commit message starts after a blank line. Explain here in more
detail the reasons why you made the change, how things worked before and how they␣
↪work now.
Also explain why

You can use hyphens to make bulleted lists:
- Foo was added because of bar
- Baz was not used so it was deleted

Add references to issue tracker (if any) at the end.

Solves: #123
See also: #456, #789
```

15.3.3 Contributing to the User Guide

Spine Toolbox uses Sphinx to create HTML pages from restructured text (.rst) files. The .rst files are plain text files that are formatted in a way that Sphinx understands and is able to turn them into HTML. Please see this brief introduction for more on reStructured text. You can modify the existing or create new .rst files into `docs/source` directory. When you are done editing, run `bin/build_doc.bat` on Windows or `bin/build_doc.sh` on Linux to build the HTML pages. The created pages are found in `docs/build/html` directory.

15.3.4 Contributing to the Spine Toolbox Graphical User Interface

If you want to change or add new widgets into the application, you need to use the `bin\build_ui.bat` (Windows) or `bin/build_ui.sh` (Linux) scripts. The main design of the widgets should be done with Qt Designer (`designer.exe` or `designer`) that is included with PySide2. The files produced by Qt Designer are XML files (.ui). You can also embed graphics (e.g. icons, logos, etc.) into the application by using Qt Designer. When you are done modifying widgets in the designer, you need to run the `build_ui` script for the changes to take effect. This script uses tools provided in the PySide2 package to turn .ui files into Python files, in essence rebuilding the whole Spine Toolbox user interface.

Styling the widgets should be done with Qt Style Sheets in code. Avoid using style sheets in Qt Designer.

15.3.5 Version Control Branching

Always make a new branch for your work, no matter how small. This makes it easy for others to take just that one set of changes from your repository, in case you have multiple unrelated changes floating around. A corollary: don't submit unrelated changes in the same branch/pull request! The maintainer shouldn't have to reject your awesome bugfix because the feature you put in with it needs more review.

Name your new branch descriptively, e.g. `issue#XXX-fixing-a-serious-bug` or `issue#ZZZ-cool-new-feature`. New branches should in general be based on the latest `dev` branch. In case you want to include a new feature still in development, you can also start working from its branch. The developers will backport any relevant bug-fixes to previous or upcoming releases under preparation.

If you need to use code from an upstream branch, please use git-rebase *if you have not shared your work with others yet*. For example: You started working on an issue, but now the upstream branch (`master`) has some new commits you would like to have in your branch too. If you have not yet pushed your branch, you can now rebase your changes on top of the upstream branch:

```
$ git pull origin master:master
$ git checkout my_branch
$ git rebase master
```

Avoid merging the upstream branch to your issue branch if it's not necessary. This will lead to a more linear and cleaner history.

Finally, make a pull request from your branch so that the developers can review your changes. You might be asked to make additional changes or clarifications or add tests to prove the new feature works as intended.

15.3.6 Test-driven development is your friend

Any bug fix that doesn't include a test proving the existence of the bug being fixed, may be suspect. Ditto for new features that can't prove they actually work.

It is recommended to use test-first development as it really helps make features better designed and identifies potential edge cases earlier instead of later. Writing tests before the implementation is strongly encouraged.

15.3.7 Full example

Here's an example workflow. Your username is `yourname` and you're submitting a basic bugfix.

Preparing your Fork

1. Click 'Fork' on Github, creating e.g. `yourname/Spine-Toolbox`
2. Clone your project: `git clone git@github.com:yourname/Spine-Toolbox`
3. `cd Spine-Toolbox`
4. Create a virtual environment and install requirements
5. Create a branch: `git checkout -b foo-the-bars master`

Making your Changes

1. Add changelog entry crediting yourself.
2. Write tests expecting the correct/fixed functionality; make sure they fail.
3. Hack, hack, hack.
4. Run tests again, making sure they pass.
5. Commit your changes: `git commit -m "Foo the bars"`

Creating Pull Requests

1. Push your commit to get it back up to your fork: `git push origin HEAD`
2. Visit Github, click handy 'Pull request' button that it will make upon noticing your new branch.
3. In the description field, write down issue number (if submitting code fixing an existing issue) or describe the issue + your fix (if submitting a wholly new bugfix).
4. Hit 'submit'! And please be patient - the maintainers will get to you when they can.

15.4 References

CHAPTER 16

Developer documentation

Here you can find developer specific documentation on Spine Toolbox.

16.1 Project item development

This document discusses the structure of *project items*, how they interact with the Toolbox GUI and how they are executed.

The core of every project item consists of two classes: a *static* project item class which is a subclass of `spinetoolbox.project_item.ProjectItem` and an *executable*, a subclass of `spinetoolbox.executable_item_base.ExecutableItemBase`. The static item is responsible for integrating the item with the Toolbox while its executable counterpart exists only during execution.

Additional classes are needed to fully define a project item:

- `spinetoolbox.project_item.ProjectItemFactory` assists Toolbox in constructing project items.
- Toolbox needs to know an item's type and category. This is achieved by `spinetoolbox.project_item_info.ProjectItemInfo`
- The item's Design view icon is inherited from `spinetoolbox.graphics_items.ProjectItemIcon`.
- Properties tab widget and other UI widgets are needed to change the item's settings.
- An *add project item* widget is used by Toolbox right after a new item has been created. Some project items use the general purpose `spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget` while others may use more specialized widgets.
- Items that support specifications need `spinetoolbox.project_item_specification_factory.ProjectItemSpecificationFactory`

16.1.1 Getting started

Probably the most convenient way to start developing a new project item is to work with a copy of some simple project item. For example, **View** provides a nice starting point.

16.1.2 Project item packages

Project items are mostly self-contained Python packages. Toolbox expects certain modules to exist in the package:

- `__init__.py` which contains an `ItemFactory` class which must be a subclass of *spinetoolbox.project_item.ProjectItemFactory* and an `ItemInfo` class which must be a subclass of *spinetoolbox.project_item_info.ProjectItemInfo*
- `executable_item.py` which contains an `ExecutableItem` class, a subclass of *spinetoolbox.executable_item_base.ExecutableItemBase*
- optional for items that support specifications: `specification_factory.py` which contains a `SpecificationFactory` class, a subclass of *spinetoolbox.project_item_specification_factory.ProjectItemSpecificationFactory*

It is customary to structure the project item packages like the Toolbox itself: `mvcmodels` submodule for Qt's models, `ui` module for automatically generated UI forms and `widgets` for the widget's business logic.

16.1.3 Item info

spinetoolbox.project_item_info.ProjectItemInfo is used by Toolbox to query two important pieces of knowledge from a project item: *type* and *category*. Type identifies the project item while category is used by the Toolbox GUI to group project items with similar function.

Categories are predefined by Toolbox. Currently available categories are: *Data Connections*, *Data Stores*, *Importers*, *Exporters*, *Manipulators*, *Tools* and *Views*.

16.1.4 Executable item

Usually, most of project item's code is for setting up the item via Toolbox GUI and for integrating the item into the Design View. The code that is run during execution by Spine Engine, the *executble item*, is usually contained in a single class which must be a subclass of *spinetoolbox.executable_item_base.ExecutableItemBase*.

Executable items live in a separate environment to the rest of the project item. They are constructed by the Toolbox only during execution and mainly interact with Spine Engine. As such, the executable items are expected to not use any GUI code or have any interaction with users.

One common aspect between executable items and 'static' project items (subclasses of *spinetoolbox.project_item.ProjectItem*) are resources. However, executable items cannot pass `transient_file` type resources since all file URLs need to point to existing files during execution.

16.1.5 Factories

Toolbox utilizes *spinetoolbox.project_item.ProjectItemFactory* to instantiate new project items in the Design View. For this purpose, the class provides methods to create an icon to show in Toolbox toolbar, an *add item dialog*, an icon to show on the Design view (a subclass of *spinetoolbox.graphics_items.ProjectItemIcon*), construct the project item itself, and some methods to deal with items that support specifications.

16.1.6 Specifications

Project item specifications are template or predefined configurations for certain tasks. For example, a tool might have a specification which defines input files, command line parameters and other settings for running a specific model

generator. Specifications are an opt-in feature and project items need to implement the corresponding methods in *spinetoolbox.project_item.ProjectItemFactory* such that Toolbox knows the item supports them.

16.1.7 Toolbox GUI integration

Toolbox shows a project item's icon which it gets from the item factory's *spinetoolbox.project_item.ProjectItemFactory.icon()* method on the toolbar. The method returns an URL to the icon's resource in Toolbox' resources. Items that support specifications may get their icon in the specifications toolbar as well, if a proper specification has been added to the project.

After dragging and dropping a project item from the toolbar onto the design view, Toolbox calls *spinetoolbox.project_item.ProjectItemFactory.make_icon()* to construct the item on the design view. This icon is a subclass of *spinetoolbox.graphics_items.ProjectItemIcon*. To prompt the user for the new item's name and optionally other initial properties, Toolbox shows the Add item dialog it gets from `spinetoolbox.project_item.ProjectItemFactory.make_add_item_widget()`

Once the item is on the design view, the main interaction with it goes through the properties tab which is created by `spinetoolbox.project_item.ProjectItemFactory.make_properties_widget()`. The properties tab widget should have all the needed controls to set up the item.

Every time a DAG on the design view changes, Toolbox calls *spinetoolbox.project_item.ProjectItem._do_handle_dag_changed()* on the affected items. This method should be reimplemented to update the project item and check its status, e.g. if all required inputs are available. Issues can be reported by *spinetoolbox.project_item.ProjectItem.add_notification()* and the notification cleared by *spinetoolbox.project_item.ProjectItem.clear_notifications()*

16.1.8 Saving and restoring project items

Project items are saved in JSON format as part of the *project.json* file. Item saving is handled by *spinetoolbox.project_item.ProjectItem.item_dict()* which should return a JSON compatible `dict` and contain at least the information in the `dict` returned by the base class method.

File system paths are handled specifically during saving: all paths outside the project directory should be absolute while the paths in the project directory should be relative. This is to enable self-contained projects which include all needed files and can be easily transferred from system to system. As such, paths are saved as special dictionaries. `spinetoolbox.helpers.serialize_path()`, `spinetoolbox.helpers.serialize_url()` and `spinetoolbox.helpers.deserialize_path()` help with dealing with the paths.

`spinetoolbox.project_item.ProjectItem.from_dict()` is responsible for reconstructing a save project item from the dictionary. `spinetoolbox.project_item.ProjectItem.parse_item_dict()` can be utilized to deserialize the basic data needed by the base class.

16.1.9 Passing data between items: resources

Project items share data by files or via databases. One item writes a file which is then read by another item. *Project item resources* are used to communicate the URLs of these files and databases.

Resources are instances of the *spinetoolbox.project_item_resource.ProjectItemResource* class.

Both static items and their executable counterparts pass resources. The major difference is that static item's may pass resource *promises* such as files that are generated during the execution. The full path to the promised files or even their final names may not be known until the items are executed.

During execution resources are propagated only to item's *direct* predecessors and successors. Static items offer their resources to direct successors only. Resources that are communicated to successor items are basically output files that

the successor items can use for input. Currently, the only resource that is propagated to predecessor items is database URLs by Data Store project items. As Data Stores leave the responsibility of writing to the database to other items it has to tell these items where to write their output data.

The table below lists the resources each project item type provides during execution.

Item	Notes	Provides to predecessor	Provides to successor
Combiner		n/a	n/a
Data Connection	[1]	n/a	File URLs
Data Store	[2]	Database URL	Database URL
Exporter		n/a	File URLs
Gimlet		n/a	Resources from predecessor
Importer		n/a	n/a
Tool	[3]	n/a	File URLs
View		n/a	n/a

The table below lists the resources that might be used by each item type during execution.

Item	Notes	Accepts from predecessor	Accepts from successor
Combiner	[4]	Database URL	Database URL
Data Connection		n/a	n/a
Data Store		n/a	n/a
Exporter		Database URL	n/a
Gimlet	[5]	File URLs, database URLs	Database URLs
Importer	[6]	File URLs	Database URL
Tool	[7]	File URLs, database URLs	Database URLs
View		Database URLs	n/a

16.1.10 Execution

The executable counterparts for project items in a DAG are created before execution. The current settings of each item are passed to the executable which is then sent to Spine Engine for execution.

The DAG is executed in two phases: first backwards then forwards. During backward execution, the DAG is executed in an inverted order and resources are propagated to direct predecessors. No current project item actually executes any other code besides storing these resources for later use. Forward execution is when the project items do their actions.

When executing in either direction:

1. *spinetoolbox.executable_item_base.ExecutableItemBase.execute()* is invoked with a list of available resources and current execution direction.
2. The resources returned by *spinetoolbox.executable_item_base.ExecutableItemBase.output_resources()* are accumulated and passed to the `execute()` of the successor item.

[1] Data connection provides paths to local files.

[2] Data Store provides a database URL to direct successors and predecessors. Note, that this is the only project item that provides resources to it's predecessors.

[3] Tool's output files are specified by a *Tool specification*.

[4] Combiner requires a database URL from its successor for writing the output data.

[5] Gimlet's resources can be passed to the command as command line arguments but are otherwise ignored.

[6] Importer requires a database URL from its successor for writing the mapped data. This can be provided by a Data Store.

[7] *Tool specification* specifies tool's optional and required input files. Database URLs can be passed to the tool *program* via command line arguments but are otherwise ignored by the Tool project item. Currently, there is no mechanism to know if a URL is actually required by a tool *program*. For more information, see *Tool specification editor*.

The `execute()` method further delegates the execution to the overridable *spinetoolbox.executable_item_base.ExecutableItemBase._execute_forward()* and *spinetoolbox.executable_item_base.ExecutableItemBase._execute_backward()* methods. Similarly, `output_resources()` calls the `spinetoolbox_executable_item_base.ExecutableItemBase._output_resources_forward()` and `spinetoolbox_executable_item_base.ExecutableItemBase._output_resources_backward()` methods.

The executable items need additional properties to function. The table below lists the properties for each item. Basically, these are the arguments that are provided to each executable's `__init__` method.

Item	Notes	Properties
Combiner		Log directory
	[8]	Cancel on error flag
Data Connection	[9]	File references
	[10]	Data files
Data Store		Database URL
Gimlet		Shell name
	[11]	Command
		Work directory
		Data files
Exporter		Export settings
		Output directory
	[12]	GAMS system directory
	[13]	Cancel on error flag
Importer		Mapping settings
		Log directory
	[14]	Python system directory
	[15]	GAMS system directory
	[16]	Cancel on error flag
Tool		Work directory
		Output directory
		Tool specification
		Command line arguments
View		n/a

[8] A flag indicating if the combine database operation should stop when an error is encountered.
[9] Path to files which can be anywhere in the file system.
[10] Files which reside in the item's data directory.
[11] Including command line arguments.
[12] Path to the directory which contains a GAMS installation. Required to find the libraries needed for writing `.gdx` files.
[13] Path to the directory which contains a Python installation. Required to run the import operation in a separate process.
[14] Path to the directory which contains a GAMS installation. Required to find the libraries needed for reading `.gdx` files.
[15] A flag indicating if the export operation should stop when an error is encountered.
[16] A flag indicating if the import operation should stop when an error is encountered.

CHAPTER 17

API Reference

This page contains auto-generated API reference documentation[1].

17.1 spinetoolbox

spinetoolbox package.

17.1.1 Subpackages

spinetoolbox.configuration_assistants

Standard assistants.

author

M. Marin (KTH)

date 17.2.2020

Subpackages

spinetoolbox.configuration_assistants.spine_opt

SpineOpt.jl config assistant.

author

M. Marin (KTH)

date 17.2.2020

[1] Created with sphinx-autoapi

Submodules

spinetoolbox.configuration_assistants.spine_opt.configuration_assistant

Widget for assisting the user in configuring SpineOpt.jl.

author

M. Marin (KTH)

date 9.1.2019

Module Contents

Classes

SpineOptConfigurationAssistant	A widget with a state machine.

class spinetoolbox.configuration_assistants.spine_opt.configuration_assistant.**SpineOptConf**

Bases: *spinetoolbox.widgets.state_machine_widget.StateMachineWidget*

A widget with a state machine.

Initializes class.

Parameters

- **window_title** (*str*) –
- **parent** (*QMainWindow*) –

_required_julia_version = 1.1.0

py_call_program_check_needed

spine_opt_process_failed

py_call_installation_needed

py_call_reconfiguration_needed

py_call_process_failed

spine_opt_ready

resolve_julia(*self*)
Returns Julia executable and project according to user's Settings.

resolve_python(*self*)
Returns the full path to Python interpreter according to user's Settings.

check_kernel_is_ok(*self*, *kernel_name*, *prgm*)
Checks that kernel spec is valid for the configuration assistant to continue.

Parameters

- **kernel_name** (*str*) – Kernel name
- **prgm** (*str*) – Either "Python" or "Julia", determines which kernel type to check

Returns True if kernel is ok, False otherwise

Return type bool

find_julia_version(*self*)

_make_processing_state(*self*, *name*, *text*)

_make_report_state(*self*, *name*, *text*)

_make_prompt_state(*self*, *name*, *text*)

_make_report_julia_not_found(*self*)

_make_report_bad_julia_version(*self*)

_make_welcome(*self*)

_make_updating_spine_opt(*self*)

_make_prompt_to_install_latest_spine_opt(*self*)

_make_installing_latest_spine_opt(*self*)

_make_report_spine_opt_installation_failed(*self*)

_make_checking_py_call_program(*self*)

_make_prompt_to_reconfigure_py_call(*self*)

_make_prompt_to_install_py_call(*self*)

_make_report_spine_opt_ready(*self*)

_make_reconfiguring_py_call(*self*)

_make_installing_py_call(*self*)

_make_report_py_call_process_failed(*self*)

update_spine_opt(*self*)

install_spine_opt(*self*)

_handle_spine_opt_process_finished(*self*, *ret*)

check_py_call_program(*self*)

_handle_check_py_call_program_finished(*self*, *ret*)

reconfigure_py_call(*self*)
Starts process that reconfigures PyCall to use selected Python interpreter.

_handle_reconfigure_py_call_finished(*self*, *ret*)

install_py_call(*self*)
Starts process that installs PyCall in current julia version.

_handle_install_py_call_finished(*self*, *ret*)

set_up_machine(*self*)

Package Contents

Classes

make_assistant	A widget with a state machine.

class spinetoolbox.configuration_assistants.spine_opt.**make_assistant**(*toolbox*)

Bases: *spinetoolbox.widgets.state_machine_widget.StateMachineWidget*

A widget with a state machine.

Initializes class.

Parameters

- **window_title** (*str*) –
- **parent** (*QMainWindow*) –

_required_julia_version = 1.1.0

py_call_program_check_needed

spine_opt_process_failed

py_call_installation_needed

py_call_reconfiguration_needed

py_call_process_failed

spine_opt_ready

resolve_julia(*self*)

Returns Julia executable and project according to user's Settings.

resolve_python(*self*)

Returns the full path to Python interpreter according to user's Settings.

check_kernel_is_ok(*self*, *kernel_name*, *prgm*)

Checks that kernel spec is valid for the configuration assistant to continue.

Parameters

- **kernel_name** (*str*) – Kernel name
- **prgm** (*str*) – Either "Python" or "Julia", determines which kernel type to check

Returns True if kernel is ok, False otherwise

Return type bool

find_julia_version(*self*)

_make_processing_state(*self*, *name*, *text*)

_make_report_state(*self*, *name*, *text*)

_make_prompt_state(*self*, *name*, *text*)

_make_report_julia_not_found(*self*)

_make_report_bad_julia_version(*self*)

_make_welcome(*self*)

_make_updating_spine_opt(*self*)

_make_prompt_to_install_latest_spine_opt(*self*)

_make_installing_latest_spine_opt(*self*)

_make_report_spine_opt_installation_failed(*self*)

_make_checking_py_call_program(*self*)

_make_prompt_to_reconfigure_py_call(*self*)

_make_prompt_to_install_py_call(*self*)

_make_report_spine_opt_ready(*self*)

_make_reconfiguring_py_call(*self*)

_make_installing_py_call(*self*)

_make_report_py_call_process_failed(*self*)

update_spine_opt(*self*)

install_spine_opt(*self*)

_handle_spine_opt_process_finished(*self*, *ret*)

check_py_call_program(*self*)

_handle_check_py_call_program_finished(*self*, *ret*)

reconfigure_py_call(*self*)
Starts process that reconfigures PyCall to use selected Python interpreter.

_handle_reconfigure_py_call_finished(*self*, *ret*)

install_py_call(*self*)
Starts process that installs PyCall in current julia version.

_handle_install_py_call_finished(*self*, *ret*)

set_up_machine(*self*)

spinetoolbox.configuration_assistants.spine_opt.**assistant_name = SpineOpt.jl**

spinetoolbox.import_editor

This subpackage contains GUI files for the Import editor.

authors

A. Soininen (VTT)

date 13.5.2020

Subpackages

spinetoolbox.import_editor.mvcmodels

Contains Import editor's MVC models.

author

A. Soininen (VTT)

date 5.8.2020

Submodules

spinetoolbox.import_editor.mvcmodels.mapping_list_model

Contains the mapping list model.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

MappingListModel	A model to hold a list of Mappings.

class spinetoolbox.import_editor.mvcmodels.mapping_list_model.**MappingListModel**(*mapping_specifications*, *table_name*, *undo_stack*)

Bases: PySide2.QtCore.QAbstractListModel

A model to hold a list of Mappings.

Parameters

- **mapping_specifications** (*list of MappingSpecificationModel*) – mapping specifications
- **table_name** (*str*) – source table name
- **undo_stack** (*QUndoStack*) – undo stack

mapping_specifications

flags(*self*, *index*)
Returns flags for given index.

get_mappings(*self*)

mapping_specification(*self*, *name*)

mapping_name_at(*self*, *row*)

rowCount(*self*, *index=None*)

row_for_mapping(*self*, *name*)

data_mapping(*self*, *index*)

data(*self*, *index*, *role=Qt.DisplayRole*)

setData(*self*, *index*, *value*, *role=Qt.EditRole*)
Renames a mapping.

rename_mapping(*self*, *row*, *name*)
Renames a mapping.

Parameters

- **row** (*int*) – mapping's row
- **name** (*str*) – new name

add_mapping(*self*)

insert_mapping_specification(*self*, *name*, *row*, *specification*)

remove_mapping(*self*, *row*)

check_mapping_validity(*self*)
Checks if there are any issues with the mappings.

Returns a map from mapping name to discovered issue; contains only mappings that have issues

Return type dict

reset(*self*, *item_mappings*, *table_name*)
Resets the model.

Parameters

- **item_mappings** (*dict*) – item mappings
- **table_name** (*str*) – name of the source table

spinetoolbox.import_editor.mvcmodels.mapping_specification_model

Contains the mapping specification model.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

MappingSpecificationModel	A model to hold a Mapping specification.

Functions

_name_index(name)	Parses an index from a string which ends with that number.

spinetoolbox.import_editor.mvcmodels.mapping_specification_model.**_MAP_TYPE_DISPLAY_NAME**

spinetoolbox.import_editor.mvcmodels.mapping_specification_model.**_DISPLAY_TYPE_TO_TYPE**

spinetoolbox.import_editor.mvcmodels.mapping_specification_model.**_TYPE_TO_DISPLAY_TYPE**

class spinetoolbox.import_editor.mvcmodels.mapping_specification_model.**MappingSpecificatio**

Bases: PySide2.QtCore.QAbstractTableModel

A model to hold a Mapping specification.

Parameters

- **table_name** (*str*) – source table name
- **mapping_name** (*str*) – mapping name
- **mapping** (*spinedb_api.ItemMappingBase*) – the item mapping to model
- **undo_stack** (*QUndoStack*) – undo stack

mapping_read_start_row_changed
Emitted after mapping's read start row has been changed.

row_or_column_type_recommendation_changed
Emitted when a change in mapping prompts for change in column or row type.

multi_column_type_recommendation_changed
Emitted when all but given columns should be of given type.

about_to_undo
Emitted before an undo/redo action.

mapping

mapping_name

source_table_name

skip_columns

map_type

last_pivot_row

dimension

import_objects

parameter_type

is_pivoted

read_start_row

set_read_start_row(*self*, *row*)

set_import_objects(*self*, *flag*)

set_mapping(*self*, *mapping*)

set_dimension(*self*, *dim*)

change_item_mapping_type(*self*, *new_type*)
Change item mapping's type.

Parameters **new_type** (*str*) – name of the type

change_parameter_type(*self*, *new_type*)
Change parameter type

set_parameter_mapping(*self*, *mapping*)
Changes the parameter mapping.

Parameters **mapping** (*ParameterDefinitionMapping*) – new mapping

update_display_table(*self*)

get_map_type_display(*self*, *mapping*, *name*)

get_map_value_display(*self*, *mapping*, *name*)

data(*self*, *index*, *role=Qt.DisplayRole*)

static data_color(*display_name*)

_mapping_issues(*self*, *row*)
Returns a message string if given row contains issues, or an empty string if everything is OK.

rowCount(*self*, *index=None*)

columnCount(*self*, *index=None*)

headerData(*self*, *section*, *orientation*, *role=Qt.DisplayRole*)

flags(*self*, *index*)

setData(*self*, *index*, *value*, *role=Qt.DisplayRole*)

change_component_mapping(*self*, *component_name*, *type_name*, *reference*)
Pushes `SetComponentMappingType` to the undo stack.

Parameters

- **component_name** (*str*) – name of the component whose type to change
- **type_name** (*str*) – name of the new type
- **reference** (*str or int*) – component mapping reference

set_type(*self*, *name*, *value*)
Changes the type of a component mapping.

Parameters

- **name** (*str*) – component name
- **value** (*str*) – mapping type name

set_value(*self*, *name*, *value*)
Sets the reference for given mapping.

Parameters

- **name** (*str*) – name of the mapping
- **value** (*str*) – a new value

Returns True if the reference was modified successfully, False otherwise.

Return type bool

_get_component_mapping_from_name(*self*, *name*)

_set_component_mapping_from_name(*self*, *name*, *mapping*)

_row_for_component_name(*self*, *name*)

_recommend_string_type(*self*, *mapping*)

_recommend_float_type(*self*, *mapping*)

_recommend_datetime_type(*self*, *mapping*)

_recommend_mapping_reference_type_change(*self*, *mapping*, *convert_spec*)

_recommend_parameter_value_mapping_reference_type_change(*self*, *mapping*)

set_skip_columns(*self*, *columns=None*)

set_time_series_repeat(*self*, *repeat*)
Toggles the repeat flag in the parameter's options.

set_map_dimensions(*self*, *dimensions*)

set_map_compress_flag(*self*, *compress*)
Sets the compress flag for Map type parameters.

Parameters **compress** (*bool*) – flag value

mapping_has_parameters(*self*)
Returns True if the item mapping has parameters.

model_parameters(*self*)
Returns the mapping's parameters.

to_dict(*self*)
Serializes the mapping specification into a dict.

Returns serialized specification

Return type dict

static from_dict(*specification_dict*, *table_name*, *undo_stack*)
Restores a serialized mapping specification.

Parameters

- **specification_dict** (*dict*) – serialized specification model
- **table_name** (*str*) – source table name
- **undo_stack** (*QUndoStack*) – undo stack

Returns mapping specification

Return type *MappingSpecificationModel*

spinetoolbox.import_editor.mvcmodels.mapping_specification_model.**_name_index**(*name*)
Parses an index from a string which ends with that number.

Parameters **name** (*str*) – a string that ends with a number

Returns the number at the end of the given string minus one

Return type int

spinetoolbox.import_editor.mvcmodels.source_data_table_model

Contains the source data table model.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

SourceDataTableModel	A model for import mapping specification.

class spinetoolbox.import_editor.mvcmodels.source_data_table_model.**SourceDataTableModel**(*par*

Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A model for import mapping specification.

Highlights columns, rows, and so on, depending on Mapping specification.

Table model for outlining simple tabular data.

Parameters **parent** (*QObject*) – the parent object

column_types_updated

row_types_updated

mapping_changed

about_to_undo
Emitted when an undo/redo command is going to be executed.

mapping_specification(*self*)

clear(*self*)
Clear all data in model.

reset_model(*self*, *main_data=None*)
Reset model.

set_mapping(*self*, *mapping*)
Set mapping to display colors from

Parameters **mapping** (MappingSpecificationModel) – mapping model

validate(*self*, *section*, *orientation=Qt.Horizontal*)

get_type(*self*, *section*, *orientation=Qt.Horizontal*)

get_types(*self*, *orientation=Qt.Horizontal*)

set_type(*self*, *section*, *section_type*, *orientation=Qt.Horizontal*)

set_types(*self*, *sections*, *section_type*, *orientation*)

set_all_column_types(*self*, *excluded_columns*, *column_type*)

_mapping_data_changed(*self*)

update_colors(*self*)

data_error(*self*, *index*, *role=Qt.DisplayRole*, *orientation=Qt.Horizontal*)

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the item referred to by the index.

Parameters

- **index** (*QModelIndex*) – Index of item
- **role** (*int*) – Data role

Returns Item data for given role.

data_color(*self*, *index*)
Returns background color for index depending on mapping.

Parameters **index** (*PySide2.QtCore.QModelIndex*) – index

Returns color of index

Return type QColor

index_in_mapping(*self*, *mapping*, *index*)
Checks if index is in mapping

Parameters

- **mapping** (*MappingBase*) – mapping
- **index** (*QModelIndex*) – index

Returns True if mapping is in index

Return type bool

mapping_column_ref_int_list(*self*)
Returns a list of column indexes that are not pivoted

Returns [List[int]] – list of ints

spinetoolbox.import_editor.mvcmodels.source_table_list_model

Contains SourceTableListModel and associated list item classes

author

A. Soininen (VTT)

date 6.8.2019

Module Contents

Classes

SourceTableItem	A list item for _SourceTableListModel
SourceTableListModel	Model for source table lists which supports undo/redo functionality.

class spinetoolbox.import_editor.mvcmodels.source_table_list_model.**SourceTableItem**(*name*, *checked*)
A list item for _SourceTableListModel

Initialize self. See help(type(self)) for accurate signature.

class spinetoolbox.import_editor.mvcmodels.source_table_list_model.**SourceTableListModel**(*und*
Bases: PySide2.QtCore.QAbstractListModel

Model for source table lists which supports undo/redo functionality.

Parameters **undo_stack** (*QUndoStack*) – undo stack

checked_table_names(*self*)

data(*self*, *index*, *role=Qt.DisplayRole*)

flags(*self*, *index*)

headerData(*self*, *section*, *orientation*, *role=Qt.DisplayRole*)

reset(*self*, *items*)

rowCount(*self*, *parent=QModelIndex()*)

setData(*self*, *index*, *value*, *role=Qt.EditRole*)

set_checked(*self*, *row*, *checked*)
Sets the checked status of a list item.

Parameters

- **row** (*int*) – item row
- **checked** (*bool*) – True for checked, False for unchecked

set_multiple_checked_undoable(*self*, *rows*, *checked*)
Sets the checked status of multiple list items.

This action is undoable.

Parameters

- **rows** (*Iterable of int*) – item rows
- **checked** (*bool*) – True for checked, False for unchecked

table_at(*self*, *row*)

table_index(*self*, *table*)

table_names(*self*)

spinetoolbox.import_editor.ui

Automatically generated UI modules for Import editor.

authors

A. Soininen (VTT)

date 13.5.2020

Submodules

spinetoolbox.import_editor.ui.import_editor_window

Module Contents

Classes

Ui_MainWindow	

class spinetoolbox.import_editor.ui.import_editor_window.**Ui_MainWindow**
Bases: object

setupUi(*self*, *MainWindow*)

retranslateUi(*self*, *MainWindow*)

spinetoolbox.import_editor.widgets

Interface logic for Import editor.

authors

A. Soininen (VTT)

date 13.5.2020

Submodules

spinetoolbox.import_editor.widgets.import_editor

Contains ImportEditor widget and MappingTableMenu.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

ImportEditor	Provides an interface for defining one or more Mappings associated to a data Source (CSV file, Excel file, etc).
MappingTableMenu	A context menu for the source data table, to let users define a Mapping from a data table.
TableMenu	Menu for tables in data source

Functions

_sanitize_data(data, header)	Fills empty data cells with None.

class spinetoolbox.import_editor.widgets.import_editor.**ImportEditor**(*ui*, *ui_error*, *undo_stack*, *connector*, *mapping_settings*)

Bases: PySide2.QtCore.QObject

Provides an interface for defining one or more Mappings associated to a data Source (CSV file, Excel file, etc).

Parameters

- **ui** (*QWidget*) – importer window's UI
- **ui_error** (*QErrorMessage*) – error dialog
- **undo_stack** (*QUndoStack*) – undo stack
- **connector** (ConnectionManager) – a connector
- **mapping_settings** (*dict*) – serialized mappings

table_checked

mapped_data_ready

source_table_selected

preview_data_updated

checked_tables

set_model(*self*, *model*)

set_mapping(*self*, *model*)

set_loading_status(*self*, *status*)
Disables/hides widgets while status is True

request_new_tables_from_connector(*self*)
Requests new tables data from connector

_change_selected_table(*self*, *selected*, *deselected*)
Sets selected table and requests data from connector

_select_table_row(*self*, *row*)

select_table(*self*, *table*)
Selects given table in the source table list.

Parameters **table** (*str*) – source table name

request_mapped_data(*self*)

update_tables(*self*, *tables*)
Updates list of tables

update_preview_data(*self*, *data*, *header*)

_restore_mappings(*self*, *settings*)

import_mappings(*self*, *mappings_dict*)
Restores mappings from a dict.

Parameters **mappings_dict** (*dict*) – serialized mappings

get_settings_dict(*self*)
Returns a dictionary with type of connector, connector options for tables, mappings for tables, selected tables.

Returns dict with settings

Return type dict

close_connection(*self*)
Close connector connection.

_new_column_types(*self*)

_new_row_types(*self*)

_update_display_row_types(*self*)

show_source_table_context_menu(*self*, *pos*)
Shows context menu for source tables.

Parameters **pos** (*QPoint*) – Mouse position

_copy_mappings(*self*, *table*)
Copies the mappings of the given source table.

Parameters **table** (*str*) – source table name

Returns copied mappings

Return type dict

_options_to_dict(*self*, *table*)
Serializes mapping options to a dict.

Parameters **table** (*str*) – source table name

Returns serialized options

Return type dict

paste_mappings(*self*, *table*, *mappings*)
Pastes mappings to given table

Parameters

- **table** (*str*) – source table name
- **mappings** (*dict*) – mappings to paste

paste_options(*self*, *table*, *options*)
Pastes all mapping options to given table.

Parameters

- **table** (*str*) – source table name
- **options** (*dict*) – options

class spinetoolbox.import_editor.widgets.import_editor.**MappingTableMenu**(*parent=None*)
Bases: PySide2.QtWidgets.QMenu

A context menu for the source data table, to let users define a Mapping from a data table.

Parameters **parent** (*QWidget*) – parent widget

set_model(*self*, *model*)
Sets target mapping specification.

Parameters **model** (MappingSpecificationModel) – mapping specification

set_mapping(*self*, *name=''*, *map_type=None*, *value=None*)

request_menu(*self*, *pos=None*)

class spinetoolbox.import_editor.widgets.import_editor.**TableMenu**(*parent*, *position*, *can_paste_option*, *can_paste_mapping*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Menu for tables in data source

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen

spinetoolbox.import_editor.widgets.import_editor.**_sanitize_data**(*data*, *header*)

Fills empty data cells with None.

spinetoolbox.import_editor.widgets.import_editor_window

Contains ImportPreviewWindow class.

authors

P. Savolainen (VTT), A. Soininen (VTT), P. Vennström (VTT)

date 10.6.2019

Module Contents

Classes

ImportEditorWindow	A QMainWindow to let users define Mappings for an Importer item.

class spinetoolbox.import_editor.widgets.import_editor_window.**ImportEditorWindow**(*importer*, *filepath*, *connector*, *connector_settings*, *mapping_settings*, *toolbox*)

Bases: PySide2.QtWidgets.QMainWindow

A QMainWindow to let users define Mappings for an Importer item.

Parameters

- **importer** (spinetoolbox.project_items.importer.importer.Importer) – Project item that owns this preview window
- **filepath** (*str*) – Importee path
- **connector** (SourceConnection) – Asynchronous data reader
- **mapping_settings** (*dict*) – Default mapping specification
- **toolbox** (*QMainWindow*) – ToolboxUI class

settings_updated

connection_failed

_insert_undo_redo_actions(*self*)

show_error(*self*, *message*)

restore_dock_widgets(*self*)
Docks all floating and or hidden QDockWidgets back to the window.

begin_style_change(*self*)
Begins a style change operation.

end_style_change(*self*)
Ends a style change operation.

apply_classic_ui_style(*self*)
Applies the classic UI style.

import_mapping_from_file(*self*)
Imports mapping spec from a user selected .json file to the preview window.

export_mapping_to_file(*self*)
Exports all mapping specs in current preview window to .json file.

apply_and_close(*self*)
Apply changes to mappings and close preview window.

start_ui(*self*)

restore_ui(*self*)
Restore UI state from previous session.

closeEvent(*self*, *event=None*)
Handles close window.

Parameters **event** (*QEvent*) – Closing event if 'X' is clicked.

spinetoolbox.import_editor.widgets.import_mapping_options

ImportMappingOptions widget.

author

P. Vennström (VTT)

date 12.5.2020

Module Contents

Classes

ImportMappingOptions	Provides methods for managing Mapping options (class type, dimensions, parameter type, ignore columns, and so on).

class spinetoolbox.import_editor.widgets.import_mapping_options.**ImportMappingOptions**(*ui*, *undo_st*

Bases: PySide2.QtCore.QObject

Provides methods for managing Mapping options (class type, dimensions, parameter type, ignore columns, and so on).

Parameters

- **ui** (*QWidget*) – importer window's UI
- **undo_stack** (*QUndoStack*) – undo stack

about_to_undo

set_num_available_columns(*self*, *num*)

change_skip_columns(*self*, *skip_cols*)

set_mapping_specification_model(*self*, *model*)

update_ui(*self*)
updates ui to RelationshipClassMapping, ObjectClassMapping or ObjectGroupMapping model

_change_item_mapping_type(*self*, *new_type*)
Pushes a SetItemMappingType command to the undo stack

Parameters **new_type** (*str*) – item's new type

set_item_mapping_type(*self*, *source_table_name*, *mapping_specification_name*, *new_type*)
Sets the type for an item mapping.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **new_type** (*str*) – name of the type

set_item_mapping(*self*, *source_table_name*, *mapping_specification_name*, *mapping*)
Sets item mapping.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **mapping** (*ItemMappingBase*) – item mapping

_change_dimension(*self*, *dimension*)
Pushes a SetItemMappingDimension command to the undo stack.

Parameters **dimension** (*int*) – mapping's dimension

set_dimension(*self*, *source_table_name*, *mapping_specification_name*, *dimension*)
Changes the item mapping's dimension and emits about_to_undo.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **dimension** (*int*) – new dimension value

_change_parameter_type(*self*, *type_name*)
Pushes a SetParameterType command to undo stack.

Parameters **type_name** (*str*) – new parameter type's name

set_parameter_type(*self*, *source_table_name*, *mapping_specification_name*, *type_name*)
Sets parameter type for an item mapping.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **type_name** (*src*) – new parameter type's name

set_parameter_mapping(*self*, *source_table_name*, *mapping_specification_name*, *parameter_mapping*)
Sets parameter mapping for an item mapping.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **parameter_mapping** (*ParameterDefinitionMapping*) – new parameter

_change_import_objects(*self*, *state*)
Pushes SetImportObjectsFlag command to the undo stack.

Parameters **state** (*bool*) – new flag value

set_import_objects_flag(*self*, *source_table_name*, *mapping_specification_name*, *import_objects*)
Sets the import objects flag.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **import_objects** (*bool*) – flag value

_change_read_start_row(*self*, *row*)
Pushes `SetReadStartRow` to the undo stack.

Parameters **row** (*int*) – new read start row

set_read_start_row(*self*, *source_table_name*, *mapping_specification_name*, *start_row*)
Sets item's parameter's read start row.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification

- **start_row** (*int*) – new read start row value

_change_time_series_repeat_flag(*self*, *repeat*)
Pushes `SetTimeSeriesRepeatFlag` to the undo stack.

Parameters **repeat** (*bool*) – True is repeat is enable, False otherwise

set_time_series_repeat_flag(*self*, *source_table_name*, *mapping_specification_name*, *repeat*)
Sets the time series repeat flag to given value.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **repeat** (*bool*) – new repeat flag value

_change_map_dimensions(*self*, *dimensions*)
Pushes `SetMapDimensions` to the undo stack.

Parameters **dimensions** (*int*) – new map dimensions

set_map_dimensions(*self*, *source_table_name*, *mapping_specification_name*, *dimensions*)
Sets map dimensions.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **dimensions** (*int*) – new map dimensions

_change_map_compression_flag(*self*, *compress*)
Pushes `SetMapCompressFlag` to the undo stack.

Parameters **compress** (*CheckState*) – if `Qt.Checked`, Maps will be compressed

set_map_compress(*self*, *source_table_name*, *mapping_specification_name*, *compress*)
Sets map compress flag.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **compress** (*bool*) – new flag value

_update_time_series_options(*self*)
Updates widgets that concern time series type parameters

_update_map_options(*self*)
Updates widgets that concern map type parameters.

spinetoolbox.import_editor.widgets.import_mappings

ImportMappings widget.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

ImportMappings	Provides methods for managing Mappings (add, remove, edit, visualize, and so on).

spinetoolbox.import_editor.widgets.import_mappings.**MAPPING_CHOICES = ['Constant', 'Column',**

class spinetoolbox.import_editor.widgets.import_mappings.**ImportMappings**(*ui*, *undo_stack*)

Bases: PySide2.QtCore.QObject

Provides methods for managing Mappings (add, remove, edit, visualize, and so on).

Parameters

- **ui** (*QWidget*) – importer window's UI
- **undo_stack** (*QUndoStack*) – undo stack

mapping_selection_changed
Emitted when a new mapping specification is selected from the Mappings list.

mapping_data_changed
Emits the new MappingListModel.

about_to_undo
Emitted before an undo/redo action.

set_mappings_model(*self*, *source_table_name*, *model*)
Sets new mappings.

Parameters

- **source_table_name** (*str*) – source table's name
- **model** (MappingListModel) – mapping list model

focus_on_changing_specification(*self*, *source_table_name*, *mapping_name*)
Selects the given mapping from the list and emits about_to_undo.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_name** (*str*) – name of the mapping specification

data_changed(*self*)
Emits the mappingDataChanged signal with the currently selected data mappings.

new_mapping(*self*)
Pushes a CreateMapping command to the undo stack

create_mapping(*self*)

insert_mapping_specification(*self*, *source_table_name*, *name*, *row*, *mapping_specification*)

delete_selected_mapping(*self*)
Pushes a DeleteMapping command to the undo stack.

delete_mapping(*self*, *source_table_name*, *name*)

_select_row(*self*, *row*)

change_mapping(*self*, *selected*, *deselected*)

spinetoolbox.import_editor.widgets.multi_checkable_list_view

Contains *MultiCheckableListView*.

author

A. Soininen (VTT)

date 13.8.2020

Module Contents

Classes

MultiCheckableListView	A list view which allows all selected items to be checked/unchecked with space bar.

class spinetoolbox.import_editor.widgets.multi_checkable_list_view.**MultiCheckableListView**
Bases: PySide2.QtWidgets.QListView

A list view which allows all selected items to be checked/unchecked with space bar.

keyPressEvent(*self*, *event*)
Handles key press events.

spinetoolbox.import_editor.widgets.options_widget

Contains OptionsWidget class.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

OptionsWidget	A widget for handling simple options.

Functions

_emit_spin_box_option_changed(i, option_key, options_widget)	A 'slot' to transform changes in QSpinBox into changes in options.
_emit_line_edit_option_changed(text, option_key, options_widget)	A 'slot' to transform changes in QLineEdit into changes in options.

Continued on next page

Table 16 – continued from previous page

_emit_check_box_option_changed(state, option_key, options_widget)	A 'slot' to transform changes in QCheckBox into changes in options.
_emit_combo_box_option_changed(text, option_key, options_widget)	A 'slot' to transform changes in QComboBox into changes in options.

class spinetoolbox.import_editor.widgets.options_widget.**OptionsWidget**(*connector*, *undo_stack*)

Bases: PySide2.QtWidgets.QWidget

A widget for handling simple options.

Parameters

- **connector** (ConnectionManager) – the connection manager whose current table's options are show on the widget
- **undo_stack** (*QUndoStack*) – undo stack

options_changed
Emitted whenever an option in the widget is changed.

about_to_undo
Emitted before undo action.

connector
The connection manager linked to this options widget.

undo_stack

undo_enabled

current_source_table

_build_ui(*self*)
Builds ui from specification in dict

_set_options(*self*, *source_table*, *options=None*)
Sets state of options

Parameters

- **source_table** (*str*) – name of the source table
- **options** (*dict, optional*) – Dict with option name as key and value as value (default: {None})

set_option_without_undo(*self*, *source_table*, *option_key*, *value*)

_fetch_options_from_connector(*self*)
Read options from the connector.

spinetoolbox.import_editor.widgets.options_widget.**_emit_spin_box_option_changed**(*i*, *option_key*, *options_widget*)

A 'slot' to transform changes in QSpinBox into changes in options.

Parameters

- **text** (*str*) – text for undo/redo
- **option_key** (*str*) – option's key

- **options_widget** (OptionsWidget) – options widget

spinetoolbox.import_editor.widgets.options_widget.**_emit_line_edit_option_changed**(*text*, *option_key*, *options_widget*)

A 'slot' to transform changes in QLineEdit into changes in options.

Parameters

- **text** (*str*) – text for undo/redo
- **option_key** (*str*) – option's key
- **options_widget** (OptionsWidget) – options widget

spinetoolbox.import_editor.widgets.options_widget.**_emit_check_box_option_changed**(*state*, *option_key*, *options_widget*)

A 'slot' to transform changes in QCheckBox into changes in options.

Parameters

- **text** (*str*) – text for undo/redo
- **option_key** (*str*) – option's key
- **options_widget** (OptionsWidget) – options widget

spinetoolbox.import_editor.widgets.options_widget.**_emit_combo_box_option_changed**(*text*, *option_key*, *options_widget*)

A 'slot' to transform changes in QComboBox into changes in options.

Parameters

- **text** (*str*) – text for undo/redo
- **option_key** (*str*) – option's key
- **options_widget** (OptionsWidget) – options widget

spinetoolbox.import_editor.widgets.table_view_with_button_header

Classes for handling models in PySide2's model/view framework.

author

P. Vennström (VTT)

date 11.5.2020

Module Contents

Classes

TableViewWithButtonHeader	Customized table with data type buttons on horizontal and vertical headers
HeaderWithButton	Class that reimplements the QHeaderView section paint event to draw a button

Functions

_create_allowed_types_menu(parent, trigger_slot)	Returns a menu which contains actions for each allowed data type.

spinetoolbox.import_editor.widgets.table_view_with_button_header.**_ALLOWED_TYPES**

spinetoolbox.import_editor.widgets.table_view_with_button_header.**_TYPE_TO_FONT_AWESOME_ICO**

spinetoolbox.import_editor.widgets.table_view_with_button_header.**Margin**

class spinetoolbox.import_editor.widgets.table_view_with_button_header.**TableViewWithButtonH**

Bases: PySide2.QtWidgets.QTableView

Customized table with data type buttons on horizontal and vertical headers

Parameters **parent** (*QWidget*) – a parent widget

scrollContentsBy(*self*, *dx*, *dy*)
Scrolls the table's contents by given delta.

update_buttons(*self*, *orientation*)
Updates the header buttons for given orientation.

Parameters **orientation** (*int*) – Qt.Horizontal or Qt.Vertical

_create_horizontal_header_menu(*self*)
Returns a new menu for the horizontal header

_create_vertical_header_menu(*self*)
Returns a new menu for the vertical header.

_show_horizontal_header_menu(*self*, *pos*)
Opens the context menu of the horizontal header.

_show_vertical_header_menu(*self*, *pos*)
Opens the context menu of the vertical header.

_set_all_column_data_types(*self*, *action*)
Sets all columns data types to the type given by action's text.

_set_all_row_data_types(*self*, *action*)
Sets all rows data types to the type given by action's text.

set_undo_stack(*self*, *undo_stack*, *about_to_undo_slot*)
Sets undo stack for the table.

Parameters

- **undo_stack** (*QUndoStack*) – undo stack
- **about_to_undo_slot** (*object*) – a slot that takes the source table name as its argument

class spinetoolbox.import_editor.widgets.table_view_with_button_header.**HeaderWithButton**(*orie*
par
ent

Bases: PySide2.QtWidgets.QHeaderView

Class that reimplements the QHeaderView section paint event to draw a button that is used to display and change the type of that column or row.

about_to_undo
Emitted when an undo/redo command is going to be executed.

display_all

sections_with_buttons

_menu_pressed(*self*, *action*)
Sets the data type of a row or column according to menu action.

change_data_types(*self*, *sections*, *type_str*)
Pushes SetColumnOrRowType to the undo stack.

Parameters

- **sections** (*Iterable or int or NoneType*) – row/column index
- **type_str** (*str*) – data type name

set_data_types(*self*, *source_table_name*, *sections*, *convert_specification*)
Sets the data type for given sections.

Parameters

- **source_table_name** (*str*) – name of the source table
- **sections** (*Iterable*) – section indexes
- **convert_specification** (ConvertSpec) – data conversion specification

update_buttons(*self*)
Updates the buttons.

widget_width(*self*)
Width of widget

Returns Width of widget

Return type int

widget_height(*self*)
Height of widget

Returns Height of widget

Return type int

_hide_or_show_button(*self*, *logical_index*)
Hides or shows the button depending on the logical index.

Parameters **logical_index** (*int*) –

mouseMoveEvent(*self*, *mouse_event*)
Moves the button to the correct section so that interacting with the button works.

enterEvent(*self*, *event*)
Shows the button.

leaveEvent (*self*, *event*)
Hides button.

_set_button_geometry (*self*, *button*, *index*)
Sets a buttons geometry depending on the index.

Parameters

- **button** (*QWidget*) – QWidget that geometry should be set
- **index** (*int*) – logical_index to set position and geometry to.

_section_resize (*self*, *i*)
When a section is resized.

Parameters **i** (*int*) – logical index to section being resized

paintSection (*self*, *painter*, *rect*, *logical_index*)
Paints a section of the QHeader view.

Works by drawing a pixmap of the button to the left of the original paint rectangle. Then shifts the original rect to the right so these two doesn't paint over each other.

sectionSizeFromContents (*self*, *logical_index*)
Add the button width to the section so it displays right.

Parameters **logical_index** (*int*) – logical index of section

Returns Size of section

Return type QSize

_section_move (*self*, *logical*, *old_visual_index*, *new_visual_index*)
Section being moved.

Parameters

- **logical** (*int*) – logical index of section beeing moved.
- **old_visual_index** (*int*) – old visual index of section
- **new_visual_index** (*int*) – new visual index of section

fix_widget_positions (*self*)
Update position of interaction button

set_margins (*self*, *margins*)
Sets the header margins.

setModel (*self*, *model*)
Sets the model for this view.

Parameters **model** (*QAbstractItemModel*) – a model

set_undo_stack (*self*, *undo_stack*)
Sets undo stack for the header making menu actions to work.

Parameters **undo_stack** (*QUndoStack*) – undo stack

set_source_table (*self*, *table_name*)
Sets the current source table.

Parameters **table_name** (*str*) – source table name

spinetoolbox.import_editor.widgets.table_view_with_button_header.**_create_allowed_types_menu**

Returns a menu which contains actions for each allowed data type.

Parameters

- **parent** (*QWidget*) – a parent widget
- **trigger_slot** (*Slot*) – a slot which is connected to QMenu's 'triggered' signal

Returns a menu

Return type QMenu

Submodules

spinetoolbox.import_editor.commands

Contains undo and redo commands for Import editor.

author

A. Soininen (VTT)

date 4.8.2020

Module Contents

Classes

_Id	Enum where members are also (and must be) ints
PasteMappings	Command to paste copied mappings
PasteOptions	Command to paste copied mapping options.
SetTableChecked	Command to change a source table's checked state.
RenameMapping	A command to change the name of a mapping.
SetComponentMappingType	Sets the type of a component mapping.
SetComponentMappingReference	Sets the reference for a component mapping.
SetConnectorOption	Command to set a ConnectorManager option.
CreateMapping	Creates a new mapping.
DeleteMapping	Command to delete a mapping.
SetItemMappingType	Command to change item mapping's type.
SetImportObjectsFlag	Command to set item mapping's import objects flag.
SetParameterType	Command to change the parameter type of an item mapping.
SetReadStartRow	Command to change item mapping's read start row option.
SetItemMappingDimension	Command to change item mapping's dimension option.
SetTimeSeriesRepeatFlag	Command to change the repeat flag for time series.
SetMapDimensions	Command to change the dimensions of a Map parameter value type.
SetMapCompressFlag	Command to change the Map compress flag.
SetColumnOrRowType	Command to change the type of columns or rows.

Continued on next page

Table 19 – continued from previous page

RestoreMappingsFromDict	Restores mappings from a dict.

class spinetoolbox.import_editor.commands.**_Id**

Bases: enum.IntEnum

Enum where members are also (and must be) ints

Initialize self. See help(type(self)) for accurate signature.

SET_OPTION

class spinetoolbox.import_editor.commands.**PasteMappings**(*import_editor*, *source_table_name*, *copied_mappings*, *previous_mappings*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to paste copied mappings

Parameters

- **import_editor** (ImportEditor) – import editor
- **source_table_name** (*src*) – name of the target source table
- **copied_mappings** (*Iterable*) – mappings to paste
- **previous_mappings** (*Iterable*) – mappings before pasting

redo(*self*)

Pastes the copied mappings

undo(*self*)

Restores mappings to their previous state.

class spinetoolbox.import_editor.commands.**PasteOptions**(*import_editor*, *source_table_name*, *copied_options*, *previous_options*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to paste copied mapping options.

Parameters

- **import_editor** (ImportEditor) – import editor
- **source_table_name** (*src*) – name of the target source table
- **copied_options** (*dict*) – options from the internal clipboard
- **previous_options** (*dict*) – previous options

redo(*self*)

Pastes the options.

undo(*self*)

Restores the options to their previous values.

class spinetoolbox.import_editor.commands.**SetTableChecked**(*table_name*, *table_list_model*, *row*, *checked*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change a source table's checked state.

Parameters

- **table_name** (*str*) – source table name
- **table_list_model** (SourceTableListModel) – source table model
- **row** (*int*) – table row on the list
- **checked** (*bool*) – new checked state

redo(*self*)
Changes the checked state.

undo(*self*)
Restores the previous checked state.

class spinetoolbox.import_editor.commands.**RenameMapping**(*row*, *mapping_list_model*, *name*, *previous_name*)

Bases: PySide2.QtWidgets.QUndoCommand

A command to change the name of a mapping.

Parameters

- **mapping_list_model** (MappingListModel) – model holding the mapping names
- **name** (*str*) – new name
- **previous_name** (*str*) – original name

redo(*self*)
Renames the mapping.

undo(*self*)
Reverts renaming of the mapping.

class spinetoolbox.import_editor.commands.**SetComponentMappingType**(*component_display_name*, *mapping_specification_model*, *mapping_type*, *previous_type*, *previous_reference*)

Bases: PySide2.QtWidgets.QUndoCommand

Sets the type of a component mapping.

Parameters

- **component_display_name** (*str*) – component name on the mapping specification table
- **mapping_specification_model** (MappingSpecificationModel) – specification model
- **mapping_type** (*str*) – name of the new type
- **previous_type** (*str*) – name of the original type
- **previous_reference** (*str or int*) – original mapping's reference

redo(*self*)
Changes a component mapping's type.

undo(*self*)
Restores component mapping's original type.

class spinetoolbox.import_editor.commands.**SetComponentMappingReference**(*component_display_name*, *mapping_specification_model*, *reference*, *previous_reference*, *previous_mapping_type_was_non*

Bases: PySide2.QtWidgets.QUndoCommand

Sets the reference for a component mapping.

Parameters

- **component_display_name** (*str*) – component name on the mapping specification table
- **mapping_specification_model** (MappingSpecificationModel) – specification model
- **reference** (*str or int*) – new value for the reference
- **previous_reference** (*str or int*) – preference's original value
- **previous_mapping_type_was_none** (*bool*) – True if the mapping was originally a NoneMapping

redo(*self*)
Sets the reference's value.

undo(*self*)
Restores the reference's value and, if necessary, mapping type to their original values.

class spinetoolbox.import_editor.commands.**SetConnectorOption**(*source_table*, *option_key*, *options_widget*, *value*, *previous_value*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to set a ConnectorManager option.

Parameters

- **source_table** (*str*) – source table name
- **option_key** (*str*) – option's key
- **options_widget** (OptionsWidget) – connector options widget
- **value** (*str or int or bool*) – option's new value
- **previous_value** (*str or int or bool*) – option's previous value

id(*self*)
This command's id.

Returns id

Return type int

mergeWith(*self*, *command*)
Merges command with another *SetConnectorOption*.

Parameters **command** (*QUndoCommand*) – a command to merge with

Returns True if merge was successful, False otherwise

Return type bool

redo(*self*)
Changes the connector's option.

undo(*self*)
Restores the option back to its original value.

class spinetoolbox.import_editor.commands.**CreateMapping**(*source_table_name*, *import_mappings*, *row*)
Bases: PySide2.QtWidgets.QUndoCommand

Creates a new mapping.

Parameters

- **source_table_name** (*src*) – source table name
- **import_mappings** (ImportMappings) – mappings manager
- **row** (*int*) – row where the new mapping should be created

redo(*self*)
Creates a new mapping at the given row in mappings list.

undo(*self*)
Deletes the created mapping.

class spinetoolbox.import_editor.commands.**DeleteMapping**(*source_table_name*, *import_mappings*, *mapping_name*, *row*)
Bases: PySide2.QtWidgets.QUndoCommand

Command to delete a mapping.

Parameters

- **source_table_name** (*src*) – source table name
- **import_mappings** (ImportMappings) – mappings manager
- **mapping_name** (*str*) – name of the mapping to delete
- **row** (*int*) – mapping's row in the mapping list

redo(*self*)
Deletes the mapping.

undo(*self*)
Restores the deleted mapping.

class spinetoolbox.import_editor.commands.**SetItemMappingType**(*source_table_name*, *mapping_specification_name*, *options_widget*, *new_type*, *previous_mapping*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change item mapping's type.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping
- **options_widget** (ImportMappingOptions) – options widget
- **new_type** (*str*) – name of the new mapping type
- **previous_mapping** (*ItemMappingBase*) – the previous mapping

redo(*self*)
Sets the mapping type to its new value.

undo(*self*)
Resets the mapping type to its former value.

class spinetoolbox.import_editor.commands.**SetImportObjectsFlag**(*source_table_name*, *mapping_specification_name*, *options_widget*, *import_objects*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to set item mapping's import objects flag.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **options_widget** (ImportMappingOptions) – options widget
- **import_objects** (*bool*) – new flag value

redo(*self*)
Changes the import objects flag.

undo(*self*)
Restores the import objects flag.

class spinetoolbox.import_editor.commands.**SetParameterType**(*source_table_name*, *mapping_specification_name*, *options_widget*, *new_type*, *previous_parameter*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change the parameter type of an item mapping.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **options_widget** (ImportMappingOptions) – options widget
- **new_type** (*str*) – name of the new parameter type
- **previous_parameter** (*ParameterDefinitionMapping*) – previous parameter mapping

redo(*self*)
Changes a parameter's type.

undo(*self*)
Restores a parameter to its previous type

class spinetoolbox.import_editor.commands.**SetReadStartRow**(*source_table_name*, *mapping_specification_name*, *options_widget*, *start_row*, *previous_start_row*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change item mapping's read start row option.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **options_widget** (ImportMappingOptions) – options widget
- **start_row** (*int*) – new read start row
- **previous_start_row** (*int*) – previous read start row value

redo(*self*)
Changes item mapping's read start row to a new value.

undo(*self*)
Restores item mapping's read start row to its previous value.

class spinetoolbox.import_editor.commands.**SetItemMappingDimension**(*source_table_name*, *mapping_specification_name*, *options_widget*, *dimension*, *previous_dimension*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change item mapping's dimension option.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **options_widget** (ImportMappingOptions) – options widget
- **dimension** (*int*) – new dimension

- **previous_dimension** (*int*) – previous dimension

redo(*self*)
Changes the item mapping's dimension to the new value.

undo(*self*)
Changes the item mapping's dimension to its previous value.

class spinetoolbox.import_editor.commands.**SetTimeSeriesRepeatFlag**(*source_table_name*, *mapping_specification_name*, *options_widget*, *repeat*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change the repeat flag for time series.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **options_widget** (ImportMappingOptions) – options widget
- **repeat** (*bool*) – new repeat flag value

redo(*self*)
Sets the repeat flag to given value.

undo(*self*)
Restores the repeat flag to its previous value.

class spinetoolbox.import_editor.commands.**SetMapDimensions**(*source_table_name*, *mapping_specification_name*, *options_widget*, *dimensions*, *previous_dimensions*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change the dimensions of a Map parameter value type.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **options_widget** (ImportMappingOptions) – options widget
- **dimensions** (*int*) – new dimensions
- **previous_dimensions** (*int*) – previous dimensions

redo(*self*)
Sets the Map dimensions to the new value.

undo(*self*)
Restores the previous Map dimensions value.

class spinetoolbox.import_editor.commands.**SetMapCompressFlag**(*source_table_name*, *mapping_specification_name*, *options_widget*, *compress*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change the Map compress flag.

Parameters

- **source_table_name** (*str*) – name of the source table
- **mapping_specification_name** (*str*) – name of the mapping specification
- **options_widget** (ImportMappingOptions) – options widget
- **compress** (*bool*) – compress flag value

redo(*self*)
Sets the compress flag.

undo(*self*)
Resets the compress flag to previous value.

class spinetoolbox.import_editor.commands.**SetColumnOrRowType**(*source_table_name*, *header_widget*, *sections*, *new_type*, *previous_type*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to change the type of columns or rows.

Parameters

- **source_table_name** (*src*) – name of the source table
- **header_widget** (HeaderWithButton) – widget of origin
- **sections** (*Iterable of int*) – row or column indexes
- **new_type** (ConvertSpec) – conversion specification for the rows/columns
- **previous_type** (ConvertSpec) – previous conversion specification for the rows/columns

redo(*self*)
Sets column/row type.

undo(*self*)
Restores column/row type to its previous value.

class spinetoolbox.import_editor.commands.**RestoreMappingsFromDict**(*import_editor*, *mapping_dict*)

Bases: PySide2.QtWidgets.QUndoCommand

Restores mappings from a dict.

Parameters

- **import_editor** (ImportEditor) – import editor
- **mapping_dict** (*dict*) – mappings to

redo (*self*)
Restores the mappings.

undo (*self*)
Reverts back to previous mappings.

spinetoolbox.import_editor.mapping_colors

Contains colors used in Import editor's tables.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

spinetoolbox.import_editor.mapping_colors.**MAPPING_COLORS**

spinetoolbox.import_editor.mapping_colors.**ERROR_COLOR**

spinetoolbox.mvcmodels

Modules in this package contain classes that represent Spine Toolbox's models (internal data structures) in the Model-View-Controller design pattern. The model classes define an interface that is used by views and delegates to access data in the application.

author

P. Savolainen (VTT)

date 24.9.2019

Submodules

spinetoolbox.mvcmodels.array_model

Contains logic for the fixed step time series editor widget.

author

A. Soininen (VTT)

date 14.6.2019

Module Contents

Classes

ArrayModel	Model for the Array parameter_value type.
_ErrorCell	A sentinel class to mark erroneous cells in the table.

class spinetoolbox.mvcmodels.array_model.**ArrayModel**
Bases: PySide2.QtCore.QAbstractTableModel

Model for the Array parameter_value type.

Even if the array is empty this model's rowCount() will still return 1. This is to show an empty row in the table view.

array(*self*)
Returns the array modeled by this model.

batch_set_data(*self*, *indexes*, *values*)
Sets data at multiple indexes at once.

columnCount(*self*, *parent=QModelIndex()*)
Returns 1.

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns model's data for given role.

flags(*self*, *index*)
Returns table cell's flags.

headerData(*self*, *section*, *orientation*, *role=Qt.DisplayRole*)
Returns header data.

insertRows(*self*, *row*, *count*, *parent=QModelIndex()*)
Inserts rows to the array.

removeRows(*self*, *row*, *count*, *parent=QModelIndex()*)
Removes rows from the array.

reset(*self*, *value*)
Resets the model to a new array.

Parameters **value** (*Array*) – a new array to model

rowCount(*self*, *parent=QModelIndex()*)
Returns the length of the array.

Note: returns 1 even if the array is empty.

set_array_type(*self*, *new_type*)
Changes the data type of array's elements.

setData(*self*, *index*, *value*, *role=Qt.EditRole*)
Sets the value at given index.

_set_data(*self*, *index*, *value*)
Sets data for given index.

In case of errors the value at index is replaced by an _ErrorCell sentinel.

Parameters

- **index** (*QModelIndex*) – an index
- **value** (*str*) – value in database format

class spinetoolbox.mvcmodels.array_model.**_ErrorCell**(*edit_value*, *tooltip*)
A sentinel class to mark erroneous cells in the table.

Parameters

- **edit_value** (*str*) – the JSON string that caused the error

- **tooltip** (*str*) – tooltip that should be shown on the table cell

spinetoolbox.mvcmodels.compound_table_model

Models that vertically concatenate two or more table models.

authors

M. Marin (KTH)

date 9.10.2019

Module Contents

Classes

CompoundTableModel	A model that concatenates several sub table models vertically.
CompoundWithEmptyTableModel	A compound parameter table model where the last model is an empty row model.

class spinetoolbox.mvcmodels.compound_table_model.**CompoundTableModel**(*parent=None*, *header=None*)

Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A model that concatenates several sub table models vertically.

Initializes model.

Parameters **parent** (*QObject*) – the parent object

refreshed

map_to_sub(*self*, *index*)

Returns an equivalent submodel index.

Parameters **index** (*QModelIndex*) – the compound model index.

Returns the equivalent index in one of the submodels

Return type QModelIndex

map_from_sub(*self*, *sub_model*, *sub_index*)

Returns an equivalent compound model index.

Parameters

- **sub_model** (MinimalTableModel) – the submodel
- **sub_index** (*QModelIndex*) – the submodel index.

Returns the equivalent index in the compound model

Return type QModelIndex

item_at_row(*self*, *row*)

Returns the item at given row.

Parameters **row** (*int*) –

Returns object

sub_model_at_row (*self*, *row*)
Returns the submodel corresponding to the given row in the compound model.

Parameters **row** (*int*) –

Returns MinimalTableModel

refresh (*self*)
Refreshes the layout by computing a new row map.

do_refresh (*self*)
Recomputes the row and inverse row maps.

_append_row_map (*self*, *row_map*)
Appends given row map to the tail of the model.

Parameters **row_map** (*list*) – tuples (model, row number)

static _row_map_for_model (*model*)
Returns row map for given model. The base class implementation just returns all model rows.

Parameters **model** (MinimalTableModel) –

Returns tuples (model, row number)

Return type list

canFetchMore (*self*, *parent=QModelIndex()*)
Returns True if any of the submodels that haven't been fetched yet can fetch more.

fetchMore (*self*, *parent=QModelIndex()*)
Fetches the next sub model and increments the fetched counter.

flags (*self*, *index*)
Return index flags.

data (*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the item referred to by the index.

Parameters

- **index** (*QModelIndex*) – Index of item
- **role** (*int*) – Data role

Returns Item data for given role.

rowCount (*self*, *parent=QModelIndex()*)
Returns the sum of rows in all models.

batch_set_data (*self*, *indexes*, *data*)
Sets data for indexes in batch. Distributes indexes and values among the different submodels and calls batch_set_data on each of them.

insertRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Inserts count rows after the given row under the given parent. Localizes the appropriate submodel and calls insertRows on it.

removeRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Removes count rows starting with the given row under parent. Localizes the appropriate submodels and calls removeRows on it.

class spinetoolbox.mvcmodels.compound_table_model.**CompoundWithEmptyTableModel** (*parent=None*, *header=None*)
Bases: *spinetoolbox.mvcmodels.compound_table_model.CompoundTableModel*

A compound parameter table model where the last model is an empty row model.

Initializes model.

Parameters **parent** (*QObject*) – the parent object

single_models

empty_model

_create_single_models(*self*)
Returns a list of single models.

_create_empty_model(*self*)
Returns an empty model.

init_model(*self*)
Initializes the compound model. Basically populates the sub_models list attribute with the result of _create_single_models and _create_empty_model.

connect_model_signals(*self*)
Connects signals so changes in the submodels are acknowledge by the compound.

_recompute_empty_row_map(*self*)
Recomputeds the part of the row map corresponding to the empty model.

_handle_empty_rows_removed(*self*, *parent*, *empty_first*, *empty_last*)
Runs when rows are removed from the empty model. Updates row_map, then emits rowsRemoved so the removed rows are no longer visible.

_handle_empty_rows_inserted(*self*, *parent*, *empty_first*, *empty_last*)
Runs when rows are inserted to the empty model. Updates row_map, then emits rowsInserted so the new rows become visible.

_handle_single_model_reset(*self*, *single_model*)
Runs when one of the single models is reset. Updates row_map, then emits rowsInserted so the new rows become visible.

_insert_single_row_map(*self*, *single_row_map*)
Inserts given row map just before the empty model's.

clear_model(*self*)
Clears the model.

spinetoolbox.mvcmodels.data_package_models

Classes for models dealing with Data Packages.

authors

M. Marin (KTH)

date 24.6.2018

Module Contents

Classes

DatapackageResourcesModel	A model of datapackage resource data, used by SpineDatapackageWidget.
DatapackageResourceDataModel	A model of datapackage field data, used by SpineDatapackageWidget.
DatapackageFieldsModel	A model of datapackage field data, used by SpineDatapackageWidget.
DatapackageForeignKeysModel	A table model with a last empty row.

class spinetoolbox.mvcmodels.data_package_models.**DatapackageResourcesModel**(*parent*, *data_package*)

Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A model of datapackage resource data, used by SpineDatapackageWidget.

Parameters **parent** (SpineDatapackageWidget) –

refresh_model(*self*)

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the item referred to by the index.

Parameters

- **index** (*QModelIndex*) – Index of item
- **role** (*int*) – Data role

Returns Item data for given role.

update_resource_dirty(*self*, *idx*, *dirty*)

batch_set_data(*self*, *indexes*, *data*)
Batch set data for indexes.

set_data(*self*, *index*, *value*)

_check_resource_name(*self*, *name*)

update_resource_name(*self*, *resource_index*, *new_name*)

flags(*self*, *index*)
Return index flags.

class spinetoolbox.mvcmodels.data_package_models.**DatapackageResourceDataModel**(*parent*, *data_package*)

Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A model of datapackage field data, used by SpineDatapackageWidget.

Parameters **parent** (SpineDatapackageWidget) –

refresh_model(*self*, *resource_index*)

headerData(*self*, *section*, *orientation=Qt.Horizontal*, *role=Qt.DisplayRole*)
Returns headers.

batch_set_data(*self*, *indexes*, *data*)
Batch set data for indexes.

update_resource_data(*self*, *resource_index*, *rows*, *columns*, *new_values*)

class spinetoolbox.mvcmodels.data_package_models.**DatapackageFieldsModel**(*parent*, *data_package*)

Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A model of datapackage field data, used by SpineDatapackageWidget.

Parameters **parent** (SpineDatapackageWidget) –

refresh_model(*self*, *resource_index*)

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the item referred to by the index.

Parameters

- **index** (*QModelIndex*) – Index of item
- **role** (*int*) – Data role

Returns Item data for given role.

flags(*self*, *index*)
Return index flags.

batch_set_data(*self*, *indexes*, *data*)
Batch set data for indexes.

_valid_field_names(*self*, *new_names*)

update_field_names(*self*, *resource_index*, *field_indexes*, *old_names*, *new_names*)

update_primary_keys(*self*, *resource_index*, *field_indexes*, *statuses*)

class spinetoolbox.mvcmodels.data_package_models.**DatapackageForeignKeysModel**(*parent*, *data_package*)

Bases: *spinetoolbox.mvcmodels.empty_row_model.EmptyRowModel*

A table model with a last empty row.

A model of datapackage foreign key data, used by SpineDatapackageWidget.

Parameters **parent** (SpineDatapackageWidget) –

foreign_keys

refresh_model(*self*, *resource_index*)

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the item referred to by the index.

Parameters

- **index** (*QModelIndex*) – Index of item
- **role** (*int*) – Data role

Returns Item data for given role.

_true_data(*self*, *index*)

_check_foreign_key(*self*, *foreign_key*)

batch_set_data(*self*, *indexes*, *data*)
Batch set data for indexes.

set_data(*self*, *index*, *value*)

_append_foreign_key(*self*, *fk_index*)

_update_foreign_key(*self*, *fk_index*)

append_foreign_key(*self*, *resource_index*, *foreign_key*)

update_foreign_key(*self*, *resource_index*, *fk_index*, *foreign_key*)

call_remove_foreign_key(*self*, *fk_index*)

remove_foreign_key(*self*, *resource_index*, *fk_index*)

insert_foreign_key(*self*, *resource_index*, *fk_index*, *foreign_key*)

emit_data_changed(*self*, *roles=None*)
Emits dataChanged for the entire model.

spinetoolbox.mvcmodels.empty_row_model

Contains a table model with an empty last row.

authors

M. Marin (KTH)

date 20.5.2018

Module Contents

Classes

EmptyRowModel	A table model with a last empty row.

class spinetoolbox.mvcmodels.empty_row_model.**EmptyRowModel**(*parent=None*, *header=None*)

Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A table model with a last empty row.

Init class.

canFetchMore(*self*, *parent=QModelIndex()*)
Return True if the model hasn't been fetched.

fetchMore(*self*, *parent=QModelIndex()*)
Fetch data and use it to reset the model.

flags(*self*, *index*)
Return default flags except if forcing defaults.

set_default_row(*self*, ***kwargs*)
Set default row data.

clear(*self*)
Clear all data in model.

reset_model(*self*, *main_data=None*)
Reset model.

_handle_data_changed(*self*, *top_left*, *bottom_right*, *roles=None*)
Insert a new last empty row in case the previous one has been filled with any data other than the defaults.

removeRows(*self*, *row*, *count*, *parent=QModelIndex()*)
Don't remove the last empty row.

_handle_rows_inserted(*self*, *parent*, *first*, *last*)
Handle rowsInserted signal.

set_rows_to_default(*self*, *first*, *last=None*)
Set default data in newly inserted rows.

spinetoolbox.mvcmodels.filter_checkbox_list_model

Provides FilterCheckboxListModel for FilterWidget.

author

P. Vennström (VTT)

date 1.11.2018

Module Contents

Classes

SimpleFilterCheckboxListModel	Init class.
LazyFilterCheckboxListModel	Extends SimpleFilterCheckboxListModel to allow for lazy loading in synch with another model.
DataToValueFilterCheckboxListModel	Extends SimpleFilterCheckboxListModel to allow for translating internal data to a value for display role.

class spinetoolbox.mvcmodels.filter_checkbox_list_model.**SimpleFilterCheckboxListModel**(*parent*, *show_*

Bases: PySide2.QtCore.QAbstractListModel

Init class.

Parameters **parent** (*QWidget*) –

_show_empty

_show_add_to_selection

reset_selection(*self*)

_handle_select_all_clicked(*self*)

_check_all_selected(*self*)

rowCount(*self*, *parent=QModelIndex()*)

data(*self*, *index*, *role=Qt.DisplayRole*)

_handle_index_clicked(*self*, *index*)

set_list(*self*, *data*, *all_selected=True*)

set_selected(*self*, *selected*, *select_empty=None*)

get_selected(*self*)

get_not_selected(*self*)

set_filter(*self*, *filter_expression*)

set_base_filter(*self*, *condition*)
Sets the base filter. The other filter, the one that works by typing in the search bar, should be applied on top of this base filter.

Parameters **condition** (*function*) – Filter acceptance condition.

apply_filter(*self*)

_remove_and_add_filtered(*self*)

_remove_and_replace_filtered(*self*)

remove_filter(*self*)

_do_add_items(*self*, *data*)

add_items(*self*, *data*, *selected=None*)

remove_items(*self*, *data*)

class spinetoolbox.mvcmodels.filter_checkbox_list_model.**LazyFilterCheckboxListModel**(*parent*, *source_m*
show_em

Bases: *spinetoolbox.mvcmodels.filter_checkbox_list_model.SimpleFilterCheckboxListModel*

Extends SimpleFilterCheckboxListModel to allow for lazy loading in synch with another model.

Init class.

Parameters

- **parent** (SpineDBEditor) –
- **source_model** (CompoundParameterModel) – a model to lazily get data from

canFetchMore(*self*, *parent=QModelIndex()*)

fetchMore(*self*, *parent=QModelIndex()*)

_do_add_items(*self*, *data*)
Adds items so the list is always sorted, while assuming that both existing and new items are sorted.

class spinetoolbox.mvcmodels.filter_checkbox_list_model.**DataToValueFilterCheckboxListModel**

Bases: *spinetoolbox.mvcmodels.filter_checkbox_list_model.SimpleFilterCheckboxListModel*

Extends SimpleFilterCheckboxListModel to allow for translating internal data to a value for display role.

Init class.

Parameters

- **parent** (SpineDBEditor) –
- **data_to_value** (*method*) – a method to translate item data to a value for display role

data(*self*, *index*, *role=Qt.DisplayRole*)

spinetoolbox.mvcmodels.indexed_value_table_model

A model for indexed parameter values, used by the parameter_value editors.

authors

A. Soininen (VTT)

date 18.6.2019

Module Contents

Classes

IndexedValueTableModel	A base class for time pattern and time series models.

class spinetoolbox.mvcmodels.indexed_value_table_model.**IndexedValueTableModel**(*value*, *index_header*, *value_header*)

Bases: PySide2.QtCore.QAbstractTableModel

A base class for time pattern and time series models.

Parameters

- **value** (*TimePattern*, *TimeSeriesFixedStep*, *TimeSeriesVariableStep*) – a parameter_value
- **index_header** (*str*) – a header for the index column
- **value_header** (*str*) – a header for the value column

value
Returns the parameter_value associated with the model.

columnCount(*self*, *parent=QModelIndex()*)
Returns the number of columns which is two.

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data at index for given role.

headerData(*self*, *section*, *orientation=Qt.Horizontal*, *role=Qt.DisplayRole*)
Returns a header.

reset(*self*, *value*)
Resets the model.

rowCount(*self*, *parent=QModelIndex()*)
Returns the number of rows.

spinetoolbox.mvcmodels.map_model

A model for maps, used by the parameter_value editors.

authors

A. Soininen (VTT)

date 11.2.2020

Module Contents

Classes

MapModel	A model for Map type parameter values.

Functions

_as_rows(map_value, row_this_far=None)	Converts given Map into list of rows recursively.
_make_square(rows)	Makes a list of rows a 2D table by appending None to the row ends.
_rows_to_dict(rows)	Turns table into nested dictionaries.
_reconstruct_map(tree)	Constructs a `Map` from a nested dictionary.

class spinetoolbox.mvcmodels.map_model.**MapModel**(*map_value*)

Bases: `PySide2.QtCore.QAbstractTableModel`

A model for Map type parameter values.

This model represents the Map as a 2D table. Each row consists of one or more index columns and a value column. The last columns of a row are padded with None.

Example

```
Map {
    "A": 1.0
    "B": Map {"a": -1.0}
    "C": 3.0
}
```

The table corresponding to the above map:

"A"	1.0	None
"B"	"a"	-1.0
"C"	3.0	None

Parameters **map_value** (*Map*) – a map

append_column(*self*)

Appends a new column to the right.

columnCount (*self*, *index=QModelIndex()*)
Returns the number of columns in this model.

convert_leaf_maps (*self*)

data (*self*, *index*, *role=Qt.DisplayRole*)
Returns the data associated with the given role.

flags (*self*, *index*)
Returns flags at index.

headerData (*self*, *section*, *orientation*, *role=Qt.DisplayRole*)
Returns row numbers for vertical headers and column titles for horizontal ones.

insertRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Inserts new rows into the map.

Parameters

- **row** (*int*) – an index where to insert the new data
- **count** (*int*) – number of rows to insert
- **parent** (*QModelIndex*) – an index to a parent model

Returns True if the operation was successful

reset (*self*, *map_value*)
Resets the model to given map_value.

rowCount (*self*, *parent=QModelIndex()*)
Returns the number of rows.

removeRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Removes rows from the map.

Parameters

- **row** (*int*) – an index where to remove the data
- **count** (*int*) – number of rows pairs to remove
- **parent** (*QModelIndex*) – an index to a parent model

Returns True if the operation was successful

setData (*self*, *index*, *value*, *role=Qt.EditRole*)
Sets data in the map.

Parameters

- **index** (*QModelIndex*) – an index to the model
- **value** (*str*) – JSON representation of the value
- **role** (*int*) – a role

Returns True if the operation was successful

trim_columns (*self*)
Removes empty columns from the right.

value (*self*)
Returns the Map.

spinetoolbox.mvcmodels.map_model.**_as_rows** (*map_value*, *row_this_far=None*)
Converts given Map into list of rows recursively.

spinetoolbox.mvcmodels.map_model.**_make_square**(*rows*)
Makes a list of rows a 2D table by appending None to the row ends.

spinetoolbox.mvcmodels.map_model.**_rows_to_dict**(*rows*)
Turns table into nested dictionaries.

Parameters **rows** (*list*) – a list of row data

Returns a nested dictionary

Return type dict

spinetoolbox.mvcmodels.map_model.**_reconstruct_map**(*tree*)
Constructs a Map from a nested dictionary.

Parameters **tree** (*dict*) – a nested dictionary

Returns reconstructed Map

Return type Map

spinetoolbox.mvcmodels.minimal_table_model

Contains a minimal table model.

authors

M. Marin (KTH)

date 20.5.2018

Module Contents

Classes

MinimalTableModel	Table model for outlining simple tabular data.

class spinetoolbox.mvcmodels.minimal_table_model.**MinimalTableModel**(*parent=None*, *header=None*, *lazy=True*)

Bases: PySide2.QtCore.QAbstractTableModel

Table model for outlining simple tabular data.

Parameters **parent** (*QObject*) – the parent object

clear(*self*)
Clear all data in model.

flags(*self*, *index*)
Return index flags.

canFetchMore(*self*, *parent=None*)
Return True if the model hasn't been fetched.

fetchMore(*self*, *parent=None*)
Fetch data and use it to reset the model.

rowCount(*self*, *parent=QModelIndex()*)
Number of rows in the model.

columnCount (*self*, *parent=QModelIndex()*)
Number of columns in the model.

headerData (*self*, *section*, *orientation=Qt.Horizontal*, *role=Qt.DisplayRole*)
Returns headers.

set_horizontal_header_labels (*self*, *labels*)
Set horizontal header labels.

insert_horizontal_header_labels (*self*, *section*, *labels*)
Insert horizontal header labels at the given section.

horizontal_header_labels (*self*)

setHeaderData (*self*, *section*, *orientation*, *value*, *role=Qt.EditRole*)
Sets the data for the given role and section in the header with the specified orientation to the value supplied.

data (*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the item referred to by the index.

Parameters

- **index** (*QModelIndex*) – Index of item
- **role** (*int*) – Data role

Returns Item data for given role.

row_data (*self*, *row*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the given row.

Parameters

- **row** (*int*) – Item row
- **role** (*int*) – Data role

Returns Row data for given role.

setData (*self*, *index*, *value*, *role=Qt.EditRole*)
Set data in model.

batch_set_data (*self*, *indexes*, *data*)
Batch set data for indexes.

insertRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Inserts count rows into the model before the given row. Items in the new row will be children of the item represented by the parent model index.

Parameters

- **row** (*int*) – Row number where new rows are inserted
- **count** (*int*) – Number of inserted rows
- **parent** (*QModelIndex*) – Parent index

Returns True if rows were inserted successfully, False otherwise

insertColumns (*self*, *column*, *count*, *parent=QModelIndex()*)
Inserts count columns into the model before the given column. Items in the new column will be children of the item represented by the parent model index.

Parameters

- **column** (*int*) – Column number where new columns are inserted

- **count** (*int*) – Number of inserted columns
- **parent** (*QModelIndex*) – Parent index

Returns True if columns were inserted successfully, False otherwise

removeRows(*self*, *row*, *count*, *parent=QModelIndex()*)
Removes count rows starting with the given row under parent.

Parameters

- **row** (*int*) – Row number where to start removing rows
- **count** (*int*) – Number of removed rows
- **parent** (*QModelIndex*) – Parent index

Returns True if rows were removed successfully, False otherwise

removeColumns(*self*, *column*, *count*, *parent=QModelIndex()*)
Removes count columns starting with the given column under parent.

Parameters

- **column** (*int*) – Column number where to start removing columns
- **count** (*int*) – Number of removed columns
- **parent** (*QModelIndex*) – Parent index

Returns True if columns were removed successfully, False otherwise

reset_model(*self*, *main_data=None*)
Reset model.

spinetoolbox.mvcmodels.minimal_tree_model

Models to represent items in a tree.

authors

P. Vennström (VTT), M. Marin (KTH)

date 11.3.2019

Module Contents

Classes

TreeItem	A tree item that can fetch its children.
MinimalTreeModel	Base class for all tree models.

class spinetoolbox.mvcmodels.minimal_tree_model.**TreeItem**(*model=None*)
A tree item that can fetch its children.

Initializes item.

Parameters **model** (*MinimalTreeModel*, *NoneType*) – The model where the item belongs.

model

child_item_type
Returns the type of child items. Reimplement in subclasses to return something more meaningful.

children

parent_item

display_data

edit_data

child(*self*, *row*)
Returns the child at given row or None if out of bounds.

last_child(*self*)
Returns the last child.

child_count(*self*)
Returns the number of children.

child_number(*self*)
Returns the rank of this item within its parent or -1 if it's an orphan.

find_children(*self*, *cond=lambda child: True*)
Returns children that meet condition expressed as a lambda function.

find_child(*self*, *cond=lambda child: True*)
Returns first child that meet condition expressed as a lambda function or None.

next_sibling(*self*)
Returns the next sibling or None if it's the last.

previous_sibling(*self*)
Returns the previous sibling or None if it's the first.

index(*self*)

insert_children(*self*, *position*, **children*)
Insert new children at given position. Returns a boolean depending on how it went.

Parameters

- **position** (*int*) – insert new items here
- **children** (*iter*) – insert items from this iterable

append_children(*self*, **children*)
Append children at the end.

remove_children(*self*, *position*, *count*)
Removes count children starting from the given position.

clear_children(*self*)
Clear children list.

flags(*self*, *column*)
Enables the item and makes it selectable.

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

has_children(*self*)
Returns whether or not this item has or could have children.

can_fetch_more(*self*)
Returns whether or not this item can fetch more.

fetch_more(*self*)
Fetches more children.

set_data(*self*, *column*, *value*, *role*)
Sets data for this item.

Parameters

- **column** (*int*) – column index
- **value** (*object*) – a new value
- **role** (*int*) – role of the new value

Returns True if data was set successfully, False otherwise

Return type bool

class spinetoolbox.mvcmodels.minimal_tree_model.**MinimalTreeModel**(*parent=None*)
Bases: PySide2.QtCore.QAbstractItemModel

Base class for all tree models.

Init class.

Parameters **parent** (SpineDBEditor) –

visit_all(*self*, *index=QModelIndex()*)
Iterates all items in the model including and below the given index. Iterative implementation so we don't need to worry about Python recursion limits.

item_from_index(*self*, *index*)
Return the item corresponding to the given index.

index_from_item(*self*, *item*)
Return a model index corresponding to the given item.

index(*self*, *row*, *column*, *parent=QModelIndex()*)
Returns the index of the item in the model specified by the given row, column and parent index.

parent(*self*, *index*)
Returns the parent of the model item with the given index.

columnCount(*self*, *parent=QModelIndex()*)

rowCount(*self*, *parent=QModelIndex()*)

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the index.

setData(*self*, *index*, *value*, *role=Qt.EditRole*)
Sets data for given index and role. Returns True if successful; otherwise returns False.

flags(*self*, *index*)
Returns the item flags for the given index.

hasChildren(*self*, *parent*)

canFetchMore(*self*, *parent*)

fetchMore(*self*, *parent*)

spinetoolbox.mvcmodels.project_item_factory_models

Contains a class for storing Tool specifications.

authors

P. Savolainen (VTT)

date 23.1.2018

Module Contents

Classes

ProjectItemFactoryModel	A model for listing project items in the Item Palette view.
ProjectItemSpecFactoryModel	Class to store specs that are available in a project e.g. GAMS or Julia models.
FilteredSpecFactoryModel	

class spinetoolbox.mvcmodels.project_item_factory_models.**ProjectItemFactoryModel**
Bases: PySide2.QtGui.QStandardItemModel

A model for listing project items in the Item Palette view.

add_item(*self*, *item_type*, *factory*)
Add item to model.

Parameters

- **item_type** (*str*) –
- **factory** (ProjectItemFactory) –

flags(*self*, *index*)

static is_index_draggable(*index*)

get_mime_data_text(*self*, *index*)

class spinetoolbox.mvcmodels.project_item_factory_models.**ProjectItemSpecFactoryModel**(*icons*)
Bases: PySide2.QtCore.QAbstractListModel

Class to store specs that are available in a project e.g. GAMS or Julia models.

rowCount(*self*, *parent=None*)
Returns the number of specs in the model.

Parameters **parent** (*QModelIndex*) – Not used (because this is a list)

Returns Number of rows (available specs) in the model

data(*self*, *index*, *role=None*)
Must be reimplemented when subclassing.

Parameters

- **index** (*QModelIndex*) – Requested index
- **role** (*int*) – Data role

Returns Data according to requested role

flags (*self*, *index*)
Returns enabled flags for the given index.

Parameters **index** (*QModelIndex*) – Index of spec

insertRow (*self*, *spec*, *row=None*, *parent=QModelIndex()*)
Insert row (specification) into model.

Parameters

- **spec** (ProjectItemSpecification) – spec added to the model
- **row** (*str*) – Row to insert spec to
- **parent** (*QModelIndex*) – Parent of child (not used)

Returns Void

removeRow (*self*, *row*, *parent=QModelIndex()*)
Remove row (spec) from model.

Parameters

- **row** (*int*) – Row to remove the spec from
- **parent** (*QModelIndex*) – Parent of spec on row (not used)

Returns Boolean variable

update_specification (*self*, *row*, *spec*)
Updates specification.

Parameters

- **row** (*int*) – Position of the spec to be updated
- **spec** (ProjectItemSpecification) – new spec, to replace the old one

Returns Boolean value depending on the result of the operation

undo_update_specification (*self*, *row*)

specification (*self*, *row*)
Returns spec specification on given row.

Parameters **row** (*int*) – Row of spec specification

Returns ProjectItemSpecification from specification list or None if given row is zero

specifications (*self*)
Yields all specs.

find_specification (*self*, *name*)
Returns specification with the given name.

Parameters **name** (*str*) – Name of specification to find

specification_row (*self*, *name*)
Returns the row on which the given specification is located or -1 if it is not found.

specification_index (*self*, *name*)
Returns the QModelIndex on which a specification with the given name is located or invalid index if it is not found.

static is_index_draggable (*index*)

get_mime_data_text(*self*, *index*)

class spinetoolbox.mvcmodels.project_item_factory_models.**FilteredSpecFactoryModel**(*item_type*)
Bases: PySide2.QtCore.QSortFilterProxyModel

filterAcceptsRow(*self*, *source_row*, *source_parent*)

spinetoolbox.mvcmodels.project_item_model

Contains a class for storing project items.

authors

P. Savolainen (VTT)

date 23.1.2018

Module Contents

Classes

ProjectItemModel	Class to store project tree items and ultimately project items in a tree structure.

class spinetoolbox.mvcmodels.project_item_model.**ProjectItemModel**(*toolbox*, *root*)
Bases: PySide2.QtCore.QAbstractItemModel

Class to store project tree items and ultimately project items in a tree structure.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **root** (RootProjectTreeItem) – Root item for the project item tree

root(*self*)
Returns the root item.

rowCount(*self*, *parent=QModelIndex()*)
Reimplemented rowCount method.

Parameters **parent** (*QModelIndex*) – Index of parent item whose children are counted.

Returns Number of children of given parent

Return type int

columnCount(*self*, *parent=QModelIndex()*)
Returns model column count which is always 1.

flags(*self*, *index*)
Returns flags for the item at given index

Parameters **index** (*QModelIndex*) – Flags of item at this index.

parent(*self*, *index=QModelIndex()*)
Returns index of the parent of given index.

Parameters **index** (*QModelIndex*) – Index of item whose parent is returned

Returns Index of parent item

Return type QModelIndex

index(*self*, *row*, *column*, *parent=QModelIndex()*)
Returns index of item with given row, column, and parent.

Parameters

- **row** (*int*) – Item row
- **column** (*int*) – Item column
- **parent** (*QModelIndex*) – Parent item index

Returns Item index

Return type QModelIndex

data(*self*, *index*, *role=None*)
Returns data in the given index according to requested role.

Parameters

- **index** (*QModelIndex*) – Index to query
- **role** (*int*) – Role to return

Returns Data depending on role.

Return type object

item(*self*, *index*)
Returns item at given index.

Parameters **index** (*QModelIndex*) – Index of item

Returns

Item at given index or root project item if index is not valid

Return type *RootProjectTreeItem*, *CategoryProjectTreeItem* or *LeafProjectTreeItem*

find_category(*self*, *category_name*)
Returns the index of the given category name.

Parameters **category_name** (*str*) – Name of category item to find

Returns index of a category item or None if it was not found

Return type QModelIndex

find_item(*self*, *name*)
Returns the QModelIndex of the leaf item with the given name

Parameters **name** (*str*) – The searched project item (long) name

Returns Index of a project item with the given name or None if not found

Return type QModelIndex

get_item(*self*, *name*)
Returns leaf item with given name or None if it doesn't exist.

Parameters **name** (*str*) – Project item name

Returns LeafProjectTreeItem, NoneType

category_of_item(*self*, *name*)
Returns the category item of the category that contains project item with given name

Parameters **name** (*str*) – Project item name

Returns category item or None if the category was not found

insert_item(*self*, *item*, *parent=QModelIndex()*)
Adds a new item to model. Fails if given parent is not a category item nor a leaf item. New item is inserted as the last item of its branch.

Parameters

- **item** (CategoryProjectTreeItem *or* LeafProjectTreeItem) – Project item to add to model
- **parent** (*QModelIndex*) – Parent project item

Returns True if successful, False otherwise

Return type bool

remove_item(*self*, *item*, *parent=QModelIndex()*)
Removes item from model.

Parameters

- **item** (BaseProjectTreeItem) – Project item to remove
- **parent** (*QModelIndex*) – Parent of item that is to be removed

Returns True if item removed successfully, False if item removing failed

Return type bool

setData(*self*, *index*, *value*, *role=Qt.EditRole*)
Changes the name of the leaf item at given index to given value.

Parameters

- **index** (*QModelIndex*) – Tree item index
- **value** (*str*) – New project item name
- **role** (*int*) – Item data role to set

Returns True or False depending on whether the new name is acceptable and renaming succeeds

Return type bool

items(*self*, *category_name=None*)
Returns a list of leaf items in model according to category name. If no category name given, returns all leaf items in a list.

Parameters **category_name** (*str*) – Item category. Data Connections, Data Stores, Importers, Exporters, Tools or Views permitted.

Returns obj:'list' of :obj:'LeafProjectTreeItem': Depending on category_name argument, returns all items or only items according to category. An empty list is returned if there are no items in the given category or if an unknown category name was given.

n_items(*self*)
Returns the number of all items in the model excluding category items and root.

Returns Number of items

Return type int

item_names(*self*)
Returns all leaf item names in a list.

Returns 'list' of obj:'str': Item names

Return type obj

items_per_category(*self*)
Returns a dict mapping category indexes to a list of items in that category.

Returns dict(QModelIndex,list(LeafProjectTreeItem))

short_name_reserved(*self*, *short_name*)
Checks if the directory name derived from the name of the given item is in use.

Parameters **short_name** (*str*) – Item short name

Returns True if short name is taken, False if it is available.

Return type bool

spinetoolbox.mvcmodels.shared

Contains stuff that is used by more than one model

author

M. Marin (KTH)

date 23.3.2020

Module Contents

spinetoolbox.mvcmodels.shared.**PARSED_ROLE**

spinetoolbox.mvcmodels.time_pattern_model

A model for time patterns, used by the parameter_value editors.

authors

A. Soininen (VTT)

date 4.7.2019

Module Contents

Classes

TimePatternModel	A model for time pattern type parameter values.

class spinetoolbox.mvcmodels.time_pattern_model.**TimePatternModel**(*value*)
Bases: *spinetoolbox.mvcmodels.indexed_value_table_model.IndexedValueTableModel*

A model for time pattern type parameter values.

Parameters **value** (*TimePattern*) – a time pattern value

flags (*self*, *index*)
Returns flags at index.

insertRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Inserts new time period - value pairs into the pattern.

New time periods are initialized to empty strings and the corresponding values to zeros.

Parameters

- **row** (*int*) – an index where to insert the new data
- **count** (*int*) – number of time period - value pairs to insert
- **parent** (*QModelIndex*) – an index to a parent model

Returns True if the operation was successful

removeRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Removes time period - value pairs from the pattern.

Parameters

- **row** (*int*) – an index where to remove the data
- **count** (*int*) – number of time period - value pairs to remove
- **parent** (*QModelIndex*) – an index to a parent model

Returns True if the operation was successful

setData (*self*, *index*, *value*, *role=Qt.EditRole*)
Sets a time period or a value in the pattern.

Column index 0 corresponds to the time periods while 1 corresponds to the values.

Parameters

- **index** (*QModelIndex*) – an index to the model
- **value** (*str, float*) – a new time period or value
- **role** (*int*) – a role

Returns True if the operation was successful

batch_set_data (*self*, *indexes*, *values*)
Sets data for several indexes at once.

Parameters

- **indexes** (*Sequence*) – a sequence of model indexes
- **values** (*Sequence*) – a sequence of time periods/floats corresponding to the indexes

spinetoolbox.mvcmodels.time_series_model_fixed_resolution

A model for fixed resolution time series, used by the parameter_value editors.

authors

A. Soininen (VTT)

date 4.7.2019

Module Contents

Classes

TimeSeriesModelFixedResolution	A model for fixed resolution time series type parameter values.

class spinetoolbox.mvcmodels.time_series_model_fixed_resolution.**TimeSeriesModelFixedResolut**
Bases: *spinetoolbox.mvcmodels.indexed_value_table_model.IndexedValueTableModel*

A model for fixed resolution time series type parameter values.

series
a time series

Type TimeSeriesFixedResolution

A base class for time pattern and time series models.

Parameters

- **value** (*TimePattern*, *TimeSeriesFixedStep*, *TimeSeriesVariableStep*) – a parameter_value
- **index_header** (*str*) – a header for the index column
- **value_header** (*str*) – a header for the value column

indexes
Returns the time stamps as an array.

values
Returns the values of the time series as an array.

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the time stamp or the corresponding value at given model index.

Column index 0 refers to time stamps while index 1 to values.

Parameters

- **index** (*QModelIndex*) – an index to the model
- **role** (*int*) – a role

flags(*self*, *index*)
Returns flags at index.

insertRows(*self*, *row*, *count*, *parent=QModelIndex()*)
Inserts new values to the series.

The new values are set to zero. Start time or resolution are left unchanged.

Parameters

- **row** (*int*) – a numeric index to the first stamp/value to insert
- **count** (*int*) – number of stamps/values to insert
- **parent** (*QModelIndex*) – index to a parent model

Returns True if the operation was successful

removeRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Removes values from the series.

Parameters

- **row** (*int*) – a numeric index to the series where to begin removing
- **count** (*int*) – how many stamps/values to remove
- **parent** (*QModelIndex*) – an index to the parent model

Returns True if the operation was successful.

reset (*self*, *value*)
Resets the model with new time series data.

setData (*self*, *index*, *value*, *role=Qt.EditRole*)
Sets a given value in the series.

Column index 1 refers to values. Note it does not make sense to set the time stamps in fixed resolution series.

Parameters

- **index** (*QModelIndex*) – an index to the model
- **value** (*numpy.datetime64, float*) – a new stamp or value
- **role** (*int*) – a role

Returns True if the operation was successful

batch_set_data (*self*, *indexes*, *values*)
Sets data for several indexes at once.

Only the values of the series are modified as the time stamps are immutable.

Parameters

- **indexes** (*Sequence*) – a sequence of model indexes
- **values** (*Sequence*) – a sequence of floats corresponding to the indexes

set_ignore_year (*self*, *ignore_year*)
Sets the ignore_year option of the time series.

set_repeat (*self*, *repeat*)
Sets the repeat option of the time series.

set_resolution (*self*, *resolution*)
Sets the resolution.

set_start (*self*, *start*)
Sets the start datetime.

spinetoolbox.mvcmodels.time_series_model_variable_resolution

A model for variable resolution time series, used by the parameter_value editors.

authors

A. Soininen (VTT)

date 5.7.2019

Module Contents

Classes

TimeSeriesModelVariableResolution	A model for variable resolution time series type parameter values.

class spinetoolbox.mvcmodels.time_series_model_variable_resolution.**TimeSeriesModelVariableR**

Bases: *spinetoolbox.mvcmodels.indexed_value_table_model.IndexedValueTableModel*

A model for variable resolution time series type parameter values.

series
a time series

Type TimeSeriesVariableResolution

A base class for time pattern and time series models.

Parameters

- **value** (*TimePattern*, *TimeSeriesFixedStep*, *TimeSeriesVariableStep*) – a parameter_value
- **index_header** (*str*) – a header for the index column
- **value_header** (*str*) – a header for the value column

indexes
Returns the time stamps as an array.

values
Returns the values of the time series as an array.

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the time stamp or the corresponding value at given model index.

Column index 0 refers to time stamps while index 1 to values.

Parameters

- **index** (*QModelIndex*) – an index to the model
- **role** (*int*) – a role

flags(*self*, *index*)
Returns the flags for given model index.

insertRows(*self*, *row*, *count*, *parent=QModelIndex()*)
Inserts new time stamps and values to the series.

When inserting in the middle of the series the new time stamps are distributed evenly among the time span between the two time stamps around the insertion point. When inserting at the beginning or at the end of the series the duration between the new time stamps is set equal to the first/last duration in the original series.

The new values are set to zero.

Parameters

- **row** (*int*) – a numeric index to the first stamp/value to insert

- **count** (*int*) – number of stamps/values to insert
- **parent** (*QModelIndex*) – index to a parent model

Returns True if the insertion was successful

removeRows(*self*, *row*, *count*, *parent=QModelIndex()*)
Removes time stamps/values from the series.

Parameters

- **row** (*int*) – a numeric index to the series where to begin removing
- **count** (*int*) – how many stamps/values to remove
- **parent** (*QModelIndex*) – an index to the parent model

Returns True if the operation was successful.

reset(*self*, *value*)
Resets the model with new time series data.

setData(*self*, *index*, *value*, *role=Qt.EditRole*)
Sets a given time stamp or value in the series.

Column index 0 refers to time stamps while index 1 to values.

Parameters

- **index** (*QModelIndex*) – an index to the model
- **value** (*numpy.datetime64, float*) – a new stamp or value
- **role** (*int*) – a role

Returns True if the operation was successful

batch_set_data(*self*, *indexes*, *values*)
Sets data for several indexes at once.

Parameters

- **indexes** (*Sequence*) – a sequence of model indexes
- **values** (*Sequence*) – a sequence of datetimes/floats corresponding to the indexes

set_ignore_year(*self*, *ignore_year*)
Sets the ignore_year option of the time series.

set_repeat(*self*, *repeat*)
Sets the repeat option of the time series.

spinetoolbox.project_items

Standard project item plugins.

author A.Soininen (VTT)

date 27.9.2019

Subpackages

spinetoolbox.project_items.combiner

Combiner plugin.

author

M. Marin (KTH)

date 12.9.2019

Subpackages

spinetoolbox.project_items.combiner.widgets

Widgets for the Combiner project item.

author

M. Marin (KTH)

date 31.5.2020

Submodules

spinetoolbox.project_items.combiner.widgets.add_combiner_widget

Widget shown to user when a new Combiner is created.

author

P. Savolainen (VTT)

date 19.1.2017

Module Contents

Classes

AddCombinerWidget	A widget to query user's preferences for a new item.

class spinetoolbox.project_items.combiner.widgets.add_combiner_widget.**AddCombinerWidget**(*too*
x,
y,
spe

Bases: *spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget*

A widget to query user's preferences for a new item.

toolbox
Parent widget

Type *ToolboxUI*

x
X coordinate of new item

Type int

y
Y coordinate of new item

Type int

Initialize class.

call_add_item(*self*)
Creates new Item according to user's selections.

spinetoolbox.project_items.combiner.widgets.combiner_properties_widget

Combiner properties widget.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 12.9.2019

Module Contents

Classes

CombinerPropertiesWidget	Widget for the Combiner Project Item Properties.

class spinetoolbox.project_items.combiner.widgets.combiner_properties_widget.**CombinerPrope**
Bases: PySide2.QtWidgets.QWidget

Widget for the Combiner Project Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embedded

Init class.

connect_signals(*self*)
Connect signals to slots.

show_combiner_properties_context_menu(*self*, *pos*)
Create and show a context-menu in Combiner properties.

Parameters **pos** (*QPoint*) – Mouse position

spinetoolbox.project_items.combiner.widgets.custom_menus

Classes for custom context menus and pop-up menus.

author

P. Savolainen (VTT)

date 9.1.2018

Module Contents

Classes

CombinerPropertiesContextMenu	Context menu class for the references tree view of the Combiner project item properties.

class spinetoolbox.project_items.combiner.widgets.custom_menus.**CombinerPropertiesContextMe**

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for the references tree view of the Combiner project item properties.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

Submodules

spinetoolbox.project_items.combiner.combiner

Module for view class.

authors

P. Savolainen (VTT), M. Marin (KHT), J. Olauson (KTH)

date 14.07.2018

Module Contents

Classes

Combiner	Class for project items that are not category nor root.

class spinetoolbox.project_items.combiner.combiner.**Combiner**(*toolbox*, *project*, *logger*, *name*, *description*, *x*, *y*, *cancel_on_error=False*)

Bases: *spinetoolbox.project_item.ProjectItem*

Class for project items that are not category nor root. These items can be executed, refreshed, and so on.

x

horizontal position in the screen

Type float

y
vertical position in the screen

Type float

Combiner class.

Parameters

- **toolbox** (ToolboxUI) – a toolbox instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **cancel_on_error** (*bool, optional*) – if True, changes will be reverted on errors

static item_type()
See base class.

static item_category()
See base class.

execution_item(*self*)
Creates project item's execution counterpart.

make_signal_handler_dict(*self*)
Returns a dictionary of all shared signals and their handlers. This is to enable simpler connecting and disconnecting.

_handle_cancel_on_error_changed(*self*, *_state*)

set_cancel_on_error(*self*, *cancel_on_error*)

restore_selections(*self*)
Restore selections into shared widgets when this project item is selected.

save_selections(*self*)
Save selections in shared widgets for this project item into instance variables.

open_db_editor(*self*, *checked=False*)
Opens selected db in the Spine database editor.

populate_reference_list(*self*)
Populates reference list.

update_name_label(*self*)
Update Combiner tab name label. Used only when renaming project items.

handle_execution_successful(*self*, *execution_direction*, *engine_state*)
Notifies Toolbox of successful database import.

_do_handle_dag_changed(*self*, *resources*)
Update the list of references that this item is viewing.

_update_references_list(*self*, *resources_upstream*)
Updates the references list with resources upstream.

Parameters **resources_upstream** (*list*) – ProjectItemResource instances

_selected_indexes(*self*)
Returns selected indexes.

_db_url_codenames(*self*, *indexes*)
Returns a dict mapping url to provider's name for given indexes in the reference model.

item_dict(*self*)
Returns a dictionary corresponding to this item.

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

spinetoolbox.project_items.combiner.combiner_factory

The ViewFactory class.

author

M. Marin (KTH)

date 15.4.2020

Module Contents

Classes

CombinerFactory	Class for project item factories.

class spinetoolbox.project_items.combiner.combiner_factory.**CombinerFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

spinetoolbox.project_items.combiner.combiner_icon

Module for view icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

Classes

CombinerIcon	View icon for the Design View.

class spinetoolbox.project_items.combiner.combiner_icon.**CombinerIcon**(*toolbox*, *x*, *y*, *project_item*, *icon*)

Bases: *spinetoolbox.graphics_items.ProjectItemIcon*

View icon for the Design View.

Parameters

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate
- **project_item** (ProjectItem) – Item
- **icon** (*str*) – icon resource path

_SHAKE_FACTOR = 0.05

_handle_time_line_value_changed(*self*, *value*)

_handle_time_line_state_changed(*self*, *new_state*)

start_animation(*self*)
Start the animation that plays when the Combiner associated to this GraphicsItem is running.

stop_animation (*self*)
Stop animation

spinetoolbox.project_items.combiner.combiner_worker

Contains Combiner program.

authors

M. Marin (KTH)

date 12.5.2020

Module Contents

Classes

CombinerWorker	**param from_urls** list of urls to read data from

class spinetoolbox.project_items.combiner.combiner_worker.**CombinerWorker**(*from_urls*, *to_urls*, *logs_dir*, *cancel_on_error*, *logger*)

Bases: PySide2.QtCore.QObject

Parameters

- **from_urls** (*list (str)*) – list of urls to read data from
- **to_urls** (*list (str)*) – list of urls to write data into
- **logs_dir** (*str*) – path to the directory where logs should be written
- **cancel_on_error** (*bool*) – whether or not to rollback and stop execution if errors
- **logger** (LoggerInterface) – somewhere to log important messages

finished

_get_db_map (*self*, *url*)

do_work (*self*)
Does the work and emits finished when done.

spinetoolbox.project_items.combiner.executable_item

Contains Combiner's executable item as well as support utilities.

authors

A. Soininen (VTT)

date 2.4.2020

Module Contents

Classes

ExecutableItem	The part of a project item that is executed by the Spine Engine.

class spinetoolbox.project_items.combiner.executable_item.**ExecutableItem**(*name*, *logs_dir*, *cancel_on_error*, *logger*)

Bases: *spinetoolbox.executable_item_base.ExecutableItemBase*, PySide2.QtCore.QObject

The part of a project item that is executed by the Spine Engine.

Parameters

- **name** (*str*) – item's name
- **logs_dir** (*str*) – path to the directory where logs should be stored
- **cancel_on_error** (*bool*) – if True, revert changes on error and move on
- **logger** (LoggerInterface) – a logger

static item_type()
Returns Combiner's type identifier string.

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
See base class.

stop_execution(*self*)
Stops execution.

_execute_backward(*self*, *resources*)
See base class.

static _urls_from_resources(*resources*)

_execute_forward(*self*, *resources*)
See base class.

_handle_worker_finished(*self*)
Runs when Combiner worker has finished.

_destroy_current_worker(*self*)
Runs when starting execution and after worker has finished. Destroys current loop, worker and quits thread, if any.

spinetoolbox.project_items.combiner.item_info

Combiner project item info.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

ItemInfo	

class spinetoolbox.project_items.combiner.item_info.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.combiner.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.combiner.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.data_connection

Data connection plugin.

author

M. Marin (KTH)

date 12.9.2019

Subpackages

spinetoolbox.project_items.data_connection.widgets

Widgets for the Data Connection project item.

author A.Soininen (VTT)

date 27.9.2019

Submodules

spinetoolbox.project_items.data_connection.widgets.add_data_connection_widget

Widget shown to user when a new Data Connection is created.

author

P. Savolainen (VTT)

date 19.1.2017

Module Contents

Classes

AddDataConnectionWidget	A widget to query user's preferences for a new item.

class spinetoolbox.project_items.data_connection.widgets.add_data_connection_widget.**AddData**

Bases: *spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget*

A widget to query user's preferences for a new item.

toolbox
Parent widget

Type *ToolboxUI*

x
X coordinate of new item

Type int

y
Y coordinate of new item

Type int

spec
The name of a spec

Type str

Initialize class.

call_add_item(*self*)
Creates new Item according to user's selections.

spinetoolbox.project_items.data_connection.widgets.custom_menus

Classes for custom context menus and pop-up menus.

author

P. Savolainen (VTT)

date 9.1.2018

Module Contents

Classes

DcRefContextMenu	Context menu class for references view in Data Connection properties.

Continued on next page

Table 46 – continued from previous page

DcDataContextMenu	Context menu class for data view in Data Connection properties.

class spinetoolbox.project_items.data_connection.widgets.custom_menus.**DcRefContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for references view in Data Connection properties.

parent
Parent for menu widget (ToolboxUI)

Type QWidget

position
Position on screen

Type QPoint

index
Index of item that requested the context-menu

Type QModelIndex

Class constructor.

class spinetoolbox.project_items.data_connection.widgets.custom_menus.**DcDataContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for data view in Data Connection properties.

parent
Parent for menu widget (ToolboxUI)

Type QWidget

position
Position on screen

Type QPoint

index
Index of item that requested the context-menu

Type QModelIndex

Class constructor.

spinetoolbox.project_items.data_connection.widgets.data_connection_properties_widget

Data connection properties widget.

author

M. Marin (KTH)

date 12.9.2019

Module Contents

Classes

DataConnectionPropertiesWidget	Widget for the Data Connection Item Properties.

class spinetoolbox.project_items.data_connection.widgets.data_connection_properties_widget
Bases: PySide2.QtWidgets.QWidget

Widget for the Data Connection Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embedded

connect_signals(*self*)
Connect signals to slots.

show_references_context_menu(*self*, *pos*)
Create and show a context-menu in data connection properties references view.

Parameters **pos** (*QPoint*) – Mouse position

show_data_context_menu(*self*, *pos*)
Create and show a context-menu in data connection properties data view.

Parameters **pos** (*QPoint*) – Mouse position

Submodules

spinetoolbox.project_items.data_connection.commands

Undo/redo commands for the DataConnection project item.

authors

M. Marin (KTH)

date 5.5.2020

Module Contents

Classes

AddDCReferencesCommand	Command to add DC references.
RemoveDCReferencesCommand	Command to remove DC references.

class spinetoolbox.project_items.data_connection.commands.**AddDCReferencesCommand**(*dc*, *paths*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to add DC references.

Parameters

- **dc** (DataConnection) – the DC
- **paths** (*set(str)*) – set of paths to add

redo(*self*)

undo(*self*)

class spinetoolbox.project_items.data_connection.commands.**RemoveDCReferencesCommand**(*dc*, *paths*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to remove DC references.

Parameters

- **dc** (DataConnection) – the DC
- **paths** (*list(str)*) – list of paths to remove

redo(*self*)

undo(*self*)

spinetoolbox.project_items.data_connection.data_connection

Module for data connection class.

author

P. Savolainen (VTT)

date 19.12.2017

Module Contents

Classes

DataConnection	Class for project items that are not category nor root.

class spinetoolbox.project_items.data_connection.data_connection.**DataConnection**(*toolbox*, *project*, *logger*, *name*, *description*, *x*, *y*, *references=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Class for project items that are not category nor root. These items can be executed, refreshed, and so on.

x
horizontal position in the screen

Type float

y
vertical position in the screen

Type float

Data Connection class.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **references** (*list*) – a list of file paths

set_up(*self*)
Sets up this item. Called when adding the item to the project. Implement in subclasses to eg recreate attributes destroyed by tear_down.

static item_type()
See base class.

static item_category()
See base class.

execution_item(*self*)
Creates DataConnection's execution counterpart.

make_signal_handler_dict(*self*)
Returns a dictionary of all shared signals and their handlers. This is to enable simpler connecting and disconnecting.

restore_selections(*self*)
Restore selections into shared widgets when this project item is selected.

add_files_to_references(*self*, *paths*)
Add multiple file paths to reference list.

Parameters **paths** (*list*) – A list of paths to files

do_add_files_to_references(*self*, *paths*)

receive_files_dropped_on_icon(*self*, *icon*, *file_paths*)
Called when files are dropped onto a data connection graphics item. If the item is this Data Connection's graphics item, add the files to data.

add_files_to_data_dir(*self*, *file_paths*)
Add files to data directory

add_references(*self*, *checked=False*)
Opens a file browser where user can select the files to be added as references for this Data Connection.

remove_references(*self*, *checked=False*)
Pushes a remove references command to undo stack

do_remove_references(*self*, *references*)
Removes given references from this Data Connection. Removes references to file paths that do not exist.

Parameters **references** (*list*) – List of selected paths.

copy_to_project(*self*, *checked=False*)
Copy selected file references to this Data Connection's data directory.

open_reference(*self*, *index*)
Open reference in default program.

open_data_file(*self*, *index*)
Open data file in default program.

show_spine_datapackage_form(*self*)
Show spine_datapackage_form widget.

datapackage_form_destroyed(*self*)
Notify a connection that datapackage form has been destroyed.

make_new_file(*self*)
Create a new blank file to this Data Connections data directory.

remove_files(*self*)
Remove selected files from data directory.

file_references(*self*)
Returns a list of paths to files that are in this item as references.

data_files(*self*)
Returns a list of files that are in the data directory.

refresh(*self*, *_=None*)
Refresh data files in Data Connection Properties. NOTE: Might lead to performance issues.

populate_reference_list(*self*, *items*, *emit_item_changed=True*)
List file references in QTreeView. If items is None or empty list, model is cleared.

populate_data_list(*self*, *items*)
List project internal data (files) in QTreeView. If items is None or empty list, model is cleared.

update_name_label (*self*)
Update Data Connection tab name label. Used only when renaming project items.

resources_for_direct_successors (*self*)
see base class

_do_handle_dag_changed (*self*, *resources*)
See base class.

item_dict (*self*)
Returns a dictionary corresponding to this item.

rename (*self*, *new_name*)
Rename this item.

Parameters **new_name** (*str*) – New name

Returns True if renaming succeeded, False otherwise

Return type bool

tear_down (*self*)
Tears down this item. Called by toolbox just before closing. Closes the SpineDatapackageWidget instances opened.

notify_destination (*self*, *source_item*)
See base class.

static default_name_prefix()
See base class.

spinetoolbox.project_items.data_connection.data_connection_factory

The DataConnectionFactory class.

author

M. Marin (KTH)

date 15.4.2020

Module Contents

Classes

DataConnectionFactory	Class for project item factories.

class spinetoolbox.project_items.data_connection.data_connection_factory.**DataConnectionFact**
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class.

spinetoolbox.project_items.data_connection.data_connection_icon

Module for data connection icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

Classes

DataConnectionIcon	Data Connection icon for the Design View.

class spinetoolbox.project_items.data_connection.data_connection_icon.**DataConnectionIcon**(*to*
x,
y,
pr
ic

Bases: *spinetoolbox.graphics_items.ProjectItemIcon*

Data Connection icon for the Design View.

Parameters

- **toolbox** (*ToolboxUI*) – main window instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate

- **project_item** (ProjectItem) – Item
- **icon** (*str*) – icon resource path

class _SignalHolder
Bases: PySide2.QtCore.QObject

files_dropped_on_icon
A signal that it triggered when files are dragged and dropped on the item.

dragEnterEvent(*self*, *event*)
Drag and drop action enters. Accept file drops from the filesystem.

Parameters **event** (*QGraphicsSceneDragDropEvent*) – Event

dragLeaveEvent(*self*, *event*)
Drag and drop action leaves.

Parameters **event** (*QGraphicsSceneDragDropEvent*) – Event

dragMoveEvent(*self*, *event*)
Accept event.

dropEvent(*self*, *event*)
Emit files_dropped_on_dc signal from scene, with this instance, and a list of files for each dropped url.

select_on_drag_over(*self*)
Called when the timer started in drag_enter_event is elapsed. Select this item if the drag action is still over it.

spinetoolbox.project_items.data_connection.executable_item

Contains Data Connection's executable item as well as support utilities.

authors
A. Soininen (VTT)

date 1.4.2020

Module Contents

Classes

ExecutableItem	The executable parts of Data Connection.

class spinetoolbox.project_items.data_connection.executable_item.**ExecutableItem**(*name*, *file_references*, *data_files*, *logger*)

Bases: *spinetoolbox.executable_item_base.ExecutableItemBase*

The executable parts of Data Connection.

Parameters

- **name** (*str*) – item's name

- **file_references** (*list*) – a list of absolute paths to connected files
- **data_files** (*list*) – a list of absolute paths to files in data connection's data directory

static item_type()
Returns DataConnectionExecutable's type identifier string.

_output_resources_forward(*self*)
See base class.

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
See base class.

spinetoolbox.project_items.data_connection.item_info

Data Connection project item info.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

ItemInfo

class spinetoolbox.project_items.data_connection.item_info.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.data_connection.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class.

class spinetoolbox.project_items.data_connection.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.data_store

Data store plugin.

author

M. Marin (KTH)

date 12.9.2019

Subpackages

spinetoolbox.project_items.data_store.widgets

Widgets for the Data Store project item.

author A.Soininen (VTT)

date 27.9.2019

Submodules

spinetoolbox.project_items.data_store.widgets.add_data_store_widget

Widget shown to user when a new Data Store is created.

author

P. Savolainen (VTT)

date 19.1.2017

Module Contents

Classes

AddDataStoreWidget	A widget to query user's preferences for a new item.

class spinetoolbox.project_items.data_store.widgets.add_data_store_widget.**AddDataStoreWidge**

Bases: *spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget*

A widget to query user's preferences for a new item.

toolbox
Parent widget

Type *ToolboxUI*

x
X coordinate of new item

Type int

y
Y coordinate of new item

Type int

Initialize class.

call_add_item(*self*)
Creates new Item according to user's selections.

spinetoolbox.project_items.data_store.widgets.custom_menus

Classes for custom context menus and pop-up menus.

author

P. Savolainen (VTT)

date 9.1.2018

Module Contents

Classes

DataStoreContextMenu	Context menu for Data Stores in the QTreeView and in the QGraphicsView.

class spinetoolbox.project_items.data_store.widgets.custom_menus.**DataStoreContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.ProjectItemContextMenu*

Context menu for Data Stores in the QTreeView and in the QGraphicsView.

parent
Parent for menu widget (ToolboxUI)

Type QWidget

position
Position on screen

Type QPoint

Class constructor.

spinetoolbox.project_items.data_store.widgets.data_store_properties_widget

Data store properties widget.

author

M. Marin (KTH)

date 12.9.2019

Module Contents

Classes

DataStorePropertiesWidget	Widget for the Data Store Item Properties.

class spinetoolbox.project_items.data_store.widgets.data_store_properties_widget.**DataStoreI**

Bases: PySide2.QtWidgets.QWidget

Widget for the Data Store Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embedded

Submodules

spinetoolbox.project_items.data_store.commands

Undo/redo commands for the DataStore project item.

authors

M. Marin (KTH)

date 5.5.2020

Module Contents

Classes

UpdateDSURLCommand	Command to update DS url.

class spinetoolbox.project_items.data_store.commands.**UpdateDSURLCommand**(*ds*, ***kwargs*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update DS url.

Parameters

- **ds** (DataStore) – the DS
- **kwargs** – url keys and their values

redo(*self*)

undo(*self*)

spinetoolbox.project_items.data_store.data_store

Module for data store class.

authors

P. Savolainen (VTT), M. Marin (KTH)

date 18.12.2017

Module Contents

Classes

DataStore	Class for project items that are not category nor root.

class spinetoolbox.project_items.data_store.data_store.**DataStore**(*toolbox*, *project*, *logger*, *name*, *description*, *x*, *y*, *url=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Class for project items that are not category nor root. These items can be executed, refreshed, and so on.

x
horizontal position in the screen

Type float

y
vertical position in the screen

Type float

Data Store class.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **url** (*str or dict*) – SQLAlchemy url

static item_type()
See base class.

static item_category()
See base class.

execution_item(*self*)
Creates DataStore's execution counterpart.

parse_url(*self*, *url*)
Return a complete url dictionary from the given dict or string

make_signal_handler_dict(*self*)
Returns a dictionary of all shared signals and their handlers. This is to enable simpler connecting and disconnecting.

restore_selections(*self*)
Load url into selections.

url(*self*)
Returns the url attribute.

sql_alchemy_url(*self*)
Returns the URL as an SQLAlchemy URL object or None if no URL is set.

_update_sa_url(*self*, *log_errors=True*)

project (*self*)
Returns current project or None if no project open.

set_path_to_sqlite_file (*self*, *file_path*)
Set path to SQLite file.

open_sqlite_file (*self*, *checked=False*)
Open file browser where user can select the path to an SQLite file that they want to use.

load_url_into_selections (*self*, *url*)
Load given url attribute into shared widget selections.

update_url (*self*, ***kwargs*)
Set url key to value.

do_update_url (*self*, ***kwargs*)

refresh_host (*self*)
Refresh host from selections.

refresh_port (*self*)
Refresh port from selections.

refresh_database (*self*)
Refresh database from selections.

refresh_username (*self*)
Refresh username from selections.

refresh_password (*self*)
Refresh password from selections.

refresh_dialect (*self*, *dialect*)

enable_dialect (*self*, *dialect*)
Enable the given dialect in the item controls.

enable_no_dialect (*self*)
Adjust widget enabled status to default when no dialect is selected.

enable_mssql (*self*)
Adjust controls to mssql connection specification.

enable_sqlite (*self*)
Adjust controls to sqlite connection specification.

enable_common (*self*)
Adjust controls to 'common' connection specification.

open_ds_form (*self*, *checked=False*)
Opens current url in the Spine database editor.

data_files (*self*)
Return a list of files that are in this items data directory.

copy_url (*self*, *checked=False*)
Copy db url to clipboard.

create_new_spine_database (*self*, *checked=False*)
Create new (empty) Spine database.

update_name_label (*self*)
Update Data Store tab name label. Used only when renaming project items.

_do_handle_dag_changed(*self*, *resources*)
See base class.

item_dict(*self*)
Returns a dictionary corresponding to this item.

static upgrade_from_no_version_to_version_1(*item_name*, *old_item_dict*, *old_project_dir*)
See base class.

static custom_context_menu(*parent*, *pos*)
Returns the context menu for this item.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu
- **pos** (*QPoint*) – Position on screen

apply_context_menu_action(*self*, *parent*, *action*)
Applies given action from context menu. Implement in subclasses as needed.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu
- **action** (*str*) – The selected action

rename(*self*, *new_name*)
Rename this item.

Parameters **new_name** (*str*) – New name

Returns True if renaming succeeded, False otherwise

Return type bool

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

resources_for_direct_successors(*self*)
See base class.

spinetoolbox.project_items.data_store.data_store_factory

The DataStoreFactory class.

author

M. Marin (KTH)

date 15.4.2020

Module Contents

Classes

DataStoreFactory	Class for project item factories.

class spinetoolbox.project_items.data_store.data_store_factory.**DataStoreFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class

spinetoolbox.project_items.data_store.data_store_icon

Module for data store icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

Classes

DataStoreIcon	Data Store icon for the Design View.

class spinetoolbox.project_items.data_store.data_store_icon.**DataStoreIcon**(*toolbox*, *x*, *y*, *project_item*, *icon*)

Bases: *spinetoolbox.graphics_items.ProjectItemIcon*

Data Store icon for the Design View.

Parameters

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate
- **project_item** (ProjectItem) – Item
- **icon** (*str*) – icon resource path

mouseDoubleClickEvent(*self*, *e*)

Opens Spine database editor when this Data Store icon is double-clicked.

Parameters **e** (*QGraphicsSceneMouseEvent*) – Event

spinetoolbox.project_items.data_store.executable_item

Contains Data Store's executable item as well as support utilities.

authors

A. Soininen (VTT)

date 1.4.2020

Module Contents

Classes

ExecutableItem	The part of a project item that is executed by the Spine Engine.

class spinetoolbox.project_items.data_store.executable_item.**ExecutableItem**(*name*, *url*, *logger*)

Bases: *spinetoolbox.executable_item_base.ExecutableItemBase*

The part of a project item that is executed by the Spine Engine.

Parameters

- **name** (*str*) – item's name
- **url** (*str*) – database's URL
- **logger** (LoggerInterface) – a logger

static item_type()
Returns the data store executable's type identifier string.

_output_resources_backward(*self*)
See base class.

_output_resources_forward(*self*)
See base class.

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
See base class.

spinetoolbox.project_items.data_store.item_info

Data Store project item info.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

ItemInfo

class spinetoolbox.project_items.data_store.item_info.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.data_store.utils

Contains utility Data Store's utility functions.

authors

A. Soininen (VTT)

date 6.5.2020

Module Contents

Functions

convert_to_sqlalchemy_url(urllib_url, item_name, logger, log_errors)	Returns a sqlalchemy url from the url or None if not valid.

spinetoolbox.project_items.data_store.utils.**convert_to_sqlalchemy_url**(*urllib_url*, *item_name*, *logger*, *log_errors*)

Returns a sqlalchemy url from the url or None if not valid.

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.data_store.**ItemFactory**(*toolbox*)

Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class

class spinetoolbox.project_items.data_store.**ItemInfo**

Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.exporter

Exporter project item plugin.

author

A. Soininen (VTT)

date 25.9.2019

Subpackages

spinetoolbox.project_items.exporter.mvcmodels

This subpackage contains models for Exporter's UI widgets.

author

A. Soininen (VTT)

date 25.8.2020

Submodules

spinetoolbox.project_items.exporter.mvcmodels.indexing_domain_list_model

Contains *IndexingDomainListModel*.

author

A. Soininen (VTT)

date 25.8.2020

Module Contents

Classes

IndexingDomainListItem	Holds additional indexing domain information.
IndexingDomainListModel	A model to manage additional domains needed for indexed parameter expansion.

class spinetoolbox.project_items.exporter.mvcmodels.indexing_domain_list_model.**IndexingDoma**
Holds additional indexing domain information.

name
domain's name

Type str

description
domain's description

Type str

expression
record key generator expression, or None

Type str or NoneType

length
length of the domain

Type int

extract_from
parameter name if the record keys are to be extracted from a parameter

Type str

Parameters **name** (*str*) – domain's name

records(*self*, *db_map*)
Generates Records

Parameters **db_map** (*DatabaseMappingBase*) – a database mapping, needed to extract parameter indexes

Returns domain's records

Return type *Records*

class spinetoolbox.project_items.exporter.mvcmodels.indexing_domain_list_model.**IndexingDoma**
Bases: PySide2.QtCore.QAbstractListModel

A model to manage additional domains needed for indexed parameter expansion.

Parameters **set_settings** (SetSettings) – export settings

domain_renamed
Emitted after a domain has been renamed.

indexes_changed
Emitted when a domain's records change.

create_new_domain(*self*)
Adds a new domain as the last element in the list.

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the domain name at given row on DisplayRole.

Parameters

- **index** (*QModelIndex*) – an index to the list
- **role** (*int*) – data role

item_at(*self*, *row*)
Returns *IndexingDomainListItem* at the given row.

Parameters **row** (*int*) – a row in the list.

Returns item at the given row

Return type *IndexingDomainListItem*

flags (*self*, *index*)
Returns item flags.

Parameters **index** (*QModelIndex*) – list index

Returns item flags

Return type int

gather_domains (*self*, *db_map*)
Returns domain name and records.

Parameters **db_map** (*DatabaseMappingBase*) – a database map

Returns mapping from domain name to records

Return type dict

insertRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Inserts new rows to the list.

Parameters

- **row** (*int*) – first row occupied by the inserted items
- **count** (*int*) – number of inserted items
- **parent** (*QModelIndex*) – ignored

Returns True if the operation was successful

Return type bool

removeRows (*self*, *row*, *count*, *parent=QModelIndex()*)
Removes rows.

Parameters

- **row** (*int*) – first row to remove
- **count** (*int*) – number of rows to remove
- **parent** (*QModelIndex*) – ignored

Returns True if the operations was successful

Return type bool

remove_rows (*self*, *rows*)
Removes non-contiguous set of rows effectively resetting the model.

Parameters **rows** (*list of int*) – rows to remove

rowCount (*self*, *parent=QModelIndex()*)
Returns the list length.

Returns number of rows in the list

Return type int

setData (*self*, *index*, *value*, *role=Qt.EditRole*)
Sets domain name at given index on `EditRole`.

Parameters

- **index** (*QModelIndex*) – an index to the list

- **value** (*str*) – new domain name
- **role** (*int*) – role

Returns True if a domain name was changed

Return type bool

spinetoolbox.project_items.exporter.mvcmodels.indexing_table_model

Contains *IndexingTableModel*.

author

A. Soininen (VTT)

date 25.8.2020

Module Contents

Classes

IndexingTableModel	A table model for parameter_value indexing.

class spinetoolbox.project_items.exporter.mvcmodels.indexing_table_model.**IndexingTableModel**

Bases: PySide2.QtCore.QAbstractTableModel

A table model for parameter_value indexing.

First column contains the proposed new index keys. The rest of the columns contain the parameter values for each set of existing index keys. Only selected new index keys are used for indexing. Unselected rows are left empty.

Parameters **parameter** (Parameter) – a parameter to model

selection_changed
Emitted after the values have been spread over the selected indexes.

manual_selection
Emitted when the selection has been changed by setData().

get_picking(*self*)
Turns the checked record into picking.

Returns picked records

Return type *FixedPicking*

canFetchMore(*self*, *parent*)
Returns True if more rows are available to show.

clear(*self*)
Clears the model.

columnCount(*self*, *parent=QModelIndex()*)
Returns the number of columns.

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns data associated with given model index and role.

headerData (*self*, *section*, *orientation*, *role=Qt.DisplayRole*)
Returns header data.

fetchMore (*self*, *parent*)
Inserts a number of new rows to the table.

flags (*self*, *index*)
Returns flags for given index.

mapped_values_balance (*self*)
Returns the balance between available indexes and parameter values.

Zero means that there is as many indexes available as there are values, i.e. the parameter is 'perfectly' indexed. A positive value means there are more indexes than values while a negative value means there are not enough indexes for all values.

Returns mapped values' balance

Return type int

rowCount (*self*, *parent=QModelIndex()*)
Return the number of rows.

select_all (*self*)
Selects all indexes.

setData (*self*, *index*, *value*, *role=Qt.EditRole*)
Sets the checked state for given index.

set_index_name (*self*, *name*)
Sets the indexing domain name.

set_records (*self*, *records*, *pick_list=None*)
Overwrites all new indexes.

set_picking (*self*, *picking*)
Selects the indexes specified by picking.

Parameters **picking** (Picking) – picking

_spread_values_over_selected_rows (*self*, *first_row*)
Repopulates the table according to selected indexes.

spinetoolbox.project_items.exporter.mvcmodels.record_list_model

Contains *RecordListModel*

author

A. Soininen (VTT)

date 25.8.2020

Module Contents

Classes

RecordListModel	A model to manage record ordering within domains and sets.

class spinetoolbox.project_items.exporter.mvcmodels.record_list_model.**RecordListModel**
Bases: PySide2.QtCore.QAbstractListModel

A model to manage record ordering within domains and sets.

data(*self*, *index*, *role=Qt.DisplayRole*)
With *role == Qt.DisplayRole* returns the record's keys as comma separated string.

flags(*self*, *index*)

headerData(*self*, *section*, *orientation*, *role=Qt.DisplayRole*)
Returns row and column header data.

moveRows(*self*, *sourceParent*, *sourceRow*, *count*, *destinationParent*, *destinationChild*)
Moves the records around.

Parameters

- **sourceParent** (*QModelIndex*) – parent from which the rows are moved
- **sourceRow** (*int*) – index of the first row to be moved
- **count** (*int*) – number of rows to move
- **destinationParent** (*QModelIndex*) – parent to which the rows are moved
- **destinationChild** (*int*) – index where to insert the moved rows

Returns True if the operation was successful, False otherwise

reset(*self*, *records*, *set_name*)
Resets the model's record data.

rowCount(*self*, *parent=QModelIndex()*)
Returns the number of records in the model.

sort_alphabetically(*self*)
Sorts the record alphabetically

spinetoolbox.project_items.exporter.mvcmodels.set_list_model

Contains *SetListModel*

author

A. Soininen (VTT)

date 25.8.2020

Module Contents

Classes

SetListModel	A model to configure the domain and set name lists in gdx export settings.

class spinetoolbox.project_items.exporter.mvcmodels.set_list_model.**SetListModel**(*set_settings*)

Bases: PySide2.QtCore.QAbstractListModel

A model to configure the domain and set name lists in gdx export settings.

This model combines domain and set names into a single list. The two 'parts' are differentiated by different background colors. Items from each part cannot be mixed with the other. Both the ordering of the items within each list as well as their exportability flags are handled here.

Parameters **set_settings** (gdx.SetSettings) – settings whose domain and set name lists should be modelled

add_domain(*self*, *domain_name*, *records*, *origin*)

Adds a new additional domain.

Parameters

- **domain_name** (*str*) – domain's name
- **records** (gdx.Records) – domain's sorted records
- **origin** (gdx.Origin) – domain's origin

drop_domain(*self*, *domain_name*)

Removes a domain.

Parameters **domain_name** (*str*) – name of the domain to remove

update_domain(*self*, *domain_name*, *records*)

Updates the records of an existing domain.

Parameters

- **domain_name** (*str*) – domain's name
- **records** (gdx.Records) – updated records

update_indexing_domains(*self*, *domains*)

Updates additional domains needed for parameter index expansion.

Parameters **domains** (*dict*) – a mapping from domain name to records

data(*self*, *index*, *role=Qt.DisplayRole*)

Returns the value for given role at given index.

Qt.DisplayRole returns the name of the domain or set while Qt.CheckStateRole returns whether the exportable flag has been set or not. Qt.BackgroundRole gives the item's background depending whether it is a domain or a set.

Parameters

- **index** (*QModelIndex*) – an index to the model
- **role** (*int*) – the query's role

Returns the requested value or *None*

flags(*self*, *index*)

Returns an item's flags.

headerData(*self*, *section*, *orientation*, *role=Qt.DisplayRole*)

Returns an empty string for horizontal header and row number for vertical header.

is_domain(*self*, *index*)
Returns True if index points to a domain name, otherwise returns False.

moveRows(*self*, *sourceParent*, *sourceRow*, *count*, *destinationParent*, *destinationChild*)
Moves the domain and set names around.

The names cannot be mixed between domains and sets.

Parameters

- **sourceParent** (*QModelIndex*) – parent from which the rows are moved
- **sourceRow** (*int*) – index of the first row to be moved
- **count** (*int*) – number of rows to move
- **destinationParent** (*QModelIndex*) – parent to which the rows are moved
- **destinationChild** (*int*) – index where to insert the moved rows

Returns True if the operation was successful, False otherwise

rowCount(*self*, *parent=QModelIndex()*)
Returns the number of rows.

setData(*self*, *index*, *value*, *role=Qt.EditRole*)
Sets the exportable flag status for given row.

update_global_parameters_domain(*self*, *domain_name*)

spinetoolbox.project_items.exporter.widgets

Exporter project item widgets.

author

A. Soininen (VTT)

date 3.10.2019

Submodules

spinetoolbox.project_items.exporter.widgets.add_exporter_widget

Widget shown to user when a new Exporter item is created.

author

A. Soininen (VTT)

date 6.9.2019

Module Contents

Classes

AddExporterWidget	A widget to query user's preferences for a new item.

class spinetoolbox.project_items.exporter.widgets.add_exporter_widget.**AddExporterWidget**(*toolbox*, *x*, *y*, *spec*)

Bases: *spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget*

A widget to query user's preferences for a new item.

Parameters

- **toolbox** (ToolboxUI) – Parent widget
- **x** (*int*) – X coordinate of new item
- **y** (*int*) – Y coordinate of new item

call_add_item(*self*)
Creates new Item according to user's selections.

spinetoolbox.project_items.exporter.widgets.export_list_item

A small widget to set up a database export in Gdx Export settings.

author

A. Soininen (VTT)

date 10.9.2019

Module Contents

Classes

ExportListItem	A widget with few controls to select the output file name and open a settings window.

spinetoolbox.project_items.exporter.widgets.export_list_item.**_BASE_ALTERNATIVE_TEXT = Expo**

class spinetoolbox.project_items.exporter.widgets.export_list_item.**ExportListItem**(*url*, *file_name*, *settings_state*, *parent=None*)

Bases: PySide2.QtWidgets.QWidget

A widget with few controls to select the output file name and open a settings window.

Parameters

- **url** (*str*) – database's identifier to be shown on a label
- **file_name** (*str*) – relative path to the exported file name
- **settings_state** (SettingsState) – settings state
- **parent** (*QWidget*) – a parent widget

open_settings_clicked
Emitted when settings window should be opened.

file_name_changed
Emitted when the file name field is changed.

scenario_changed
Emitted when selected scenario has changed.

out_file_name_edit
export file name QLineEdit

url_field
Text in the database URL field.

update_notification_label(*self*, *state*)
Updates the UI and the message label according to settings pack state.

Parameters **state** (SettingsState) – settings state

update_scenarios(*self*, *scenarios*, *selected*)
Updates the scenarios combo box.

Parameters

- **scenarios** (*dict*) – a map from scenario name to boolean active flag
- **selected** (*str, optional*) – currently selected scenario, None for the 'Base' alternative

make_sure_this_scenario_is_shown_in_the_combo_box(*self*, *scenario*)
Makes sure the given scenario is selected in the combo box.

Parameters **scenario** (*str, optional*) – scenario name

_emit_file_name_changed(*self*)
Emits file_name_changed signal.

_emit_open_settings_clicked(*self*, *_*)
Emits open_settings_clicked signal.

_emit_scenario_changed(*self*, *selected*)
Emits scenario_changed signal.

spinetoolbox.project_items.exporter.widgets.exporter_properties

Exporter properties widget.

author

A. Soininen (VTT)

date 25.9.2019

Module Contents

Classes

ExporterProperties	A main window widget to show Gdx Export item's properties.

class spinetoolbox.project_items.exporter.widgets.exporter_properties.**ExporterProperties**(*to*
Bases: PySide2.QtWidgets.QWidget

A main window widget to show Gdx Export item's properties.

Parameters **toolbox** (ToolboxUI) – a main window instance

ui
The UI form of this widget.

spinetoolbox.project_items.exporter.widgets.gdx_export_settings

Export item's settings window for .gdx export.

author

A. Soininen (VTT)

date 9.9.2019

Module Contents

Classes

State	Gdx Export Settings window state
GdxExportSettings	A setting window for exporting .gdx files.

class spinetoolbox.project_items.exporter.widgets.gdx_export_settings.**State**
Bases: enum.Enum

Gdx Export Settings window state

Create and return a new object. See help(type) for accurate signature.

OK
Settings are ok.

BAD_INDEXING
Not all indexed parameters are set up correctly.

class spinetoolbox.project_items.exporter.widgets.gdx_export_settings.**GdxExportSettings**(*set_in-dex ing_ mer ing_ non non sce- nar dat par ent*)

Bases: PySide2.QtWidgets.QWidget

A setting window for exporting .gdx files.

Parameters

- **set_settings** (gdx.SetSettings) – export settings for GAMS sets
- **indexing_settings** (*dict*) – indexing domain information for indexed parameter values
- **merging_settings** (*dict*) – parameter merging settings
- **none_fallback** (NoneFallback) – fallback for None parameter values
- **none_export** (NoneExport) – how to handle None values while exporting
- **scenario** (*str, optional*) – scenario name
- **database_url** (*str*) – database URL
- **parent** (*QWidget*) – a parent widget

reset_requested
Emitted when Reset Defaults button has been clicked.

settings_accepted
Emitted when the OK button has been clicked.

settings_rejected
Emitted when the Cancel button has been clicked.

set_settings
the settings object

indexing_settings
indexing settings dict

merging_settings
dictionary of merging settings

none_fallback

none_export

reset_settings(*self*, *set_settings*, *indexing_settings*, *merging_settings*)
Resets all settings.

_check_state(*self*)
Checks if there are parameters in need for indexing.

_set_none_fallback (*self*, *option*)
Sets the None fallback option.

Parameters **option** (*str*) – option as a label in the combo box

_init_none_fallback_combo_box (*self*, *fallback*)
Sets the current text in None fallback combo box.

Parameters **fallback** (NoneFallback) – option

_set_none_export (*self*, *option*)
Sets the None export option.

Parameters **option** (*str*) – option as a label in the combo box

_init_none_export_combo_box (*self*, *export*)
Sets the current text in None export combo box

Parameters **export** (NoneExport) – option

_populate_global_parameters_combo_box (*self*, *settings*)
(Re)populates the global parameters combo box.

_set_records_ordering_controls_enabled (*self*, *enabled*)
Sets or unsets the enabled state of buttons that control the record key order.

Parameters **enabled** – True if the buttons should be enabled, False otherwise

handle_settings_state_changed (*self*, *state*)

_accept (*self*)
Emits the settings_accepted signal.

_move_sets_up (*self*, *checked=False*)
Moves selected domains and sets up one position.

_move_sets_down (*self*, *checked=False*)
Moves selected domains and sets down one position.

_move_records_up (*self*, *checked=False*)
Moves selected records up and position.

_move_records_down (*self*, *checked=False*)
Moves selected records down on position.

_reject (*self*)
Closes the window.

closeEvent (*self*, *event*)

_reset_settings (*self*, *button*)
Requests for fresh settings to be read from the database.

_update_global_parameters_domain (*self*, *text*)
Updates the global parameters domain name.

_populate_set_contents (*self*, *selected*, _)
Populates the record list by the selected domain's or set's records.

_sort_records_alphabetically (*self*, _)
Sorts the lists of set records alphabetically.

_show_indexed_parameter_settings (*self*, _)
Shows the indexed parameter settings window.

_show_parameter_merging_settings(*self*, _)
Shows the parameter merging settings window.

_gather_parameter_indexing_settings(*self*)
Gathers settings from the indexed parameters settings window.

_parameter_merging_approved(*self*)
Collects merging settings from the parameter merging window.

_dispose_parameter_indexing_settings_window(*self*)
Removes references to the indexed parameter settings window.

_dispose_parameter_merging_window(*self*)
Removes references to the parameter merging settings window.

_domains_sets_exportable_state_changed(*self*, *top_left*, *bottom_right*, _)

spinetoolbox.project_items.exporter.widgets.merging_error_flag

Error condition flags for Parameter merging.

author

A. Soininen (VTT)

date 2.3.2020

Module Contents

Classes

MergingErrorFlag	Error flags for parameter merging.

class spinetoolbox.project_items.exporter.widgets.merging_error_flag.**MergingErrorFlag**
Bases: enum.Flag

Error flags for parameter merging.

Create and return a new object. See help(type) for accurate signature.

NO_ERRORS = 0

PARAMETER_NAME_MISSING

DOMAIN_NAME_MISSING

NO_PARAMETER_SELECTED

spinetoolbox.project_items.exporter.widgets.parameter_index_settings

Parameter indexing settings window for .gdx export.

author

A. Soininen (VTT)

date 26.11.2019

Module Contents

Classes

IndexSettingsState	An enumeration indicating the state of the settings window.
ParameterIndexSettings	A widget showing setting for a parameter with indexed values.

Functions

_freely_update_domains_combo(widget)	A context manager which temporarily disables the monitoring of current text changes in domains combo box.

class spinetoolbox.project_items.exporter.widgets.parameter_index_settings.**IndexSettingsSta**

Bases: enum.Enum

An enumeration indicating the state of the settings window.

Create and return a new object. See help(type) for accurate signature.

OK

DOMAIN_MISSING_INDEXES

class spinetoolbox.project_items.exporter.widgets.parameter_index_settings.**ParameterIndexSe**

Bases: PySide2.QtWidgets.QWidget

A widget showing setting for a parameter with indexed values.

Parameters

- **parameter_name** (*str*) – parameter's name
- **indexing_setting** (IndexingSetting) – indexing settings for the parameter
- **available_domains** (*dict*) – a dict from existing domain name to Records
- **parent** (*QWidget, optional*) – a parent widget

state
widget's state

indexing_domain_name(*self*)
Returns the selected indexing domain's name

Returns domain name

Return type str

picking(*self*)
Returns picking.

Returns picking

Return type *Picking*

set_domains (*self*, *domains*)
Sets new domains and record keys.

Parameters **domains** (*dict*) – mapping from domain name to records

set_domains_combo_monitoring_enabled (*self*, *enabled*)
Enables or disables monitoring of current text in domains combo box.

Parameters **enabled** (*bool*) – True enables monitoring, False disables

update_domain_name (*self*, *old_name*, *new_name*)
Renames a domain.

Parameters

- **old_name** (*str*) – previous name
- **new_name** (*str*) – new name

update_records (*self*, *domain_name*)
Updates existing domain's records.

Parameters **domain_name** (*str*) – domain's name

notification_message (*self*, *message*)
Shows a notification message on the widget.

warning_message (*self*, *message*)
Shows a warning message on the widget.

error_message (*self*, *message*)
Shows an error message on the widget.

_check_state (*self*)
Updated the widget's state.

_check_errors (*self*, *mapped_values_balance*)
Checks if the parameter is correctly indexed.

_check_warnings (*self*, *mapped_values_balance*)
Checks if there are non-fatal issues with parameter indexing.

_update_indexing_domains_name (*self*)
Updates the model's header and the label showing the indexing domains.

_clear_pick_expression_silently (*self*)
Clears the pick expression line edit.

_change_domain (*self*, *domain_name*)
Change the domain used on the table.

_update_index_list_selection (*self*, *expression*)
Updates selection according to changed selection expression.

_move_indexing_domain_left (*self*, _)
Moves the indexing domain name left on the indexing label.

_move_indexing_domain_right (*self*, _)
Moves the indexing domain name right on the indexing label.

spinetoolbox.project_items.exporter.widgets.parameter_index_settings.**_freely_update_domains**
A context manager which temporarily disables the monitoring of current text changes in domains combo box.

Parameters **widget** (ParameterIndexSettings) – settings widget

spinetoolbox.project_items.exporter.widgets.parameter_index_settings_window

Parameter indexing settings window for .gdx export.

author

A. Soininen (VTT)

date 25.11.2019

Module Contents

Classes

ParameterIndexSettingsWindow	A window which shows a list of ParameterIndexSettings widgets, one for each parameter with indexed values.

Functions

_disable_domain_updates(window)	A context manager which disables updates on the indexing settings widgets.

class spinetoolbox.project_items.exporter.widgets.parameter_index_settings_window.**Parameter**

Bases: PySide2.QtWidgets.QWidget

A window which shows a list of ParameterIndexSettings widgets, one for each parameter with indexed values.

Parameters

- **indexing_settings** (*dict*) – a map from parameter name to IndexingSetting
- **set_settings** (SetSettings) – export settings
- **database_path** (*str*) – a database url
- **scenario** (*str*) – scenario name
- **parent** (*QWidget*) – a parent widget

settings_approved
Emitted when the settings have been approved.

settings_rejected
Emitted when the settings have been rejected.

indexing_settings
indexing settings dictionary

additional_indexing_domains(*self*)

set_domain_updated_enabled(*self*, *enabled*)
Enables or disables updating the indexing settings widgets.

Parameters **enabled** (*bool*) – if True, allow the widgets to update

_switch_additional_domain_widgets_enabled_state(*self*, *using_expression*)
Enabled and disables additional domain widgets.

Parameters **using_expression** (*bool*) – True if expression is used, False if record keys are extracted from existing parameter

_set_additional_domain_widgets_enabled(*self*, *enabled*)

_add_domain(*self*, *_*)
Creates a new additional domain.

_collect_and_hide(*self*)
Collects settings from individual ParameterIndexSettings widgets and hides the window.

_load_additional_domain(*self*, *current*, *previous*)

_reject_and_close(*self*)

_remove_selected_domains(*self*, *_*)

_send_domains_to_indexing_widgets(*self*, *parent*, *first*, *last*)
Updates the available domains combo boxes in indexing widgets.

_update_after_domain_rename(*self*, *old_name*, *new_name*)
Propagates changes in domain names to widgets.

Parameters

- **old_name** (*str*) – domain's previous name
- **new_name** (*str*) – domain's current name

_update_expression(*self*, *expression*)
Updates the domain's record key expression.

Parameters **expression** (*str*) – new expression

_update_length(*self*, *length*)
Updates the number of additional domain's records.

Parameters **length** (*int*) – new record count

_use_expression(*self*, *_*)

_use_extraction(*self*, *_*)

_set_extraction_domain(*self*, *domain_name*)
Sets the domain from which domain's records are extracted.

Parameters **domain_name** (*str*) – domain name

closeEvent(*self*, *event*)
Handles the close event.

spinetoolbox.project_items.exporter.widgets.parameter_index_settings_window.**_disable_domai**
A context manager which disables updates on the indexing settings widgets.

Parameters **window** (ParameterIndexSettingsWindow) – settings window

spinetoolbox.project_items.exporter.widgets.parameter_merging_settings

Parameter merging settings widget.

author

A. Soininen (VTT)

date 19.2.2020

Module Contents

Classes

ParameterMergingSettings	A widget for configure parameter merging.
_DomainNameListModel	Model for domains_list_view.
_ParameterNameListModel	Model for parameter_name_list_view.

spinetoolbox.project_items.exporter.widgets.parameter_merging_settings.**_ERROR_MESSAGE = <sp**

class spinetoolbox.project_items.exporter.widgets.parameter_merging_settings.**ParameterMergi**

Bases: PySide2.QtWidgets.QWidget

A widget for configure parameter merging.

Parameters

- **entity_class_infos** (*list*) – list of EntityClassInfo objects
- **parent** (*QWidget*) – a parent widget
- **parameter_name** (*str*) – merged parameter name of None for widget
- **merging_setting** (MergingSetting) – merging settings or None for empty widget

removal_requested
Emitted when the settings widget wants to get removed from the parent window.

error_flags

parameter_name
Name of the merged parameter.

merging_setting(*self*)
Constructs the MergingSetting object from the widget's contents.

update(*self*, *entity_class_infos*)
Updates the settings after database commit.

_check_state(*self*)
Updates the message label according to widget's error state.

_clear_flag(*self*, *state*)
Clears a state flag.

_set_flag(*self*, *state*)
Sets a state flag.

_reset_indexing_domains_label(*self*, *domain_name=None*, *domain_names=None*)
Rewrites the contents of indexing_domains_label.

_update_parameter_name(*self*, *name*)
Updates the merged parameter name.

_remove_self(*self*, _)
Requests removal from the parent window.

_handle_domain_selection_change(*self*, *selected*, _)
Resets the settings after another item has been selected in domains_list_view.

_update_indexing_domain_name(*self*, *name*)
Resets indexing_domains_label.

_move_domain_left(*self*, _)
Moves the new indexing domain left in indexing_domains_label.

_move_domain_right(*self*, _)
Moves the new indexing domain left in indexing_domains_label.

class spinetoolbox.project_items.exporter.widgets.parameter_merging_settings.**_DomainNameLis**
Bases: PySide2.QtCore.QAbstractListModel

Model for domains_list_view.

Stores EntityClassInfo objects displaying the entity name in domains_list_view.

Parameters **entity_classes** (*list*) – a list of EntityClassObjects

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns model's data for given index.

headerData(*self*, *section*, *orientation*)
Returns None.

index_for(*self*, *set_name*)
Returns the QModelIndex for given set name.

item_at(*self*, *row*)
Returns the EntityClassInfo object at given row.

rowCount(*self*, *parent=QModelIndex()*)
Returns the size of the model.

update(*self*, *entity_classes*)
Updates the model.

class spinetoolbox.project_items.exporter.widgets.parameter_merging_settings.**_ParameterName**
Bases: PySide2.QtCore.QAbstractListModel

Model for parameter_name_list_view.

Parameters **names** (*list*) – list of parameter names to show in the view

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the model's data.

flags (*self*, *index*)
Returns flags for given index.

headerData (*self*, *section*, *orientation*)
Returns None.

reset (*self*, *names*)
Resets the model's contents when a new index is selected in domains_list_view.

rowCount (*self*, *parent=QModelIndex()*)
Returns the number of parameter names.

select (*self*, *names*)
Selects parameters for inclusion in the merged parameter.

selected (*self*)
Returns a list of the selected parameters.

setData (*self*, *index*, *value*, *role=Qt.EditRole*)
Selects or deselects the parameter at given index for inclusion in the merged parameter.

update (*self*, *names*)
Updates the parameter names keeping the previous selection where it makes sense.

spinetoolbox.project_items.exporter.widgets.parameter_merging_settings_window

Parameter merging settings window.

author

A. Soininen (VTT)

date 19.2.2020

Module Contents

Classes

ParameterMergingSettingsWindow	A window which shows a list of ParameterMergingSettings widgets, one for each merged parameter.
EntityClassInfo	Contains information of an entity_class (object or relationship_class) for use in the parameter merging widget.

Functions

_gather_entity_class_infos(db_map)	Collects entity_class infos from database.

class spinetoolbox.project_items.exporter.widgets.parameter_merging_settings_window.**Paramet**

Bases: PySide2.QtWidgets.QWidget

A window which shows a list of ParameterMergingSettings widgets, one for each merged parameter.

Parameters

- **merging_settings** (*dict*) – a map from merged parameter name to merging settings
- **database_path** (*str*) – database URL
- **parent** (*QWidget*) – a parent widget

settings_approved
Emitted when the settings have been approved.

settings_rejected
Emitted when the settings have been rejected.

merging_settings
a dict that maps merged parameter names to their merging settings

update(*self*)
Updates the settings according to changes in the database.

_add_setting(*self*, *parameter_name=None*, *merging_setting=None*)
Inserts a new settings widget to the widget list.

_ok_to_accept(*self*)
Returns True if it is OK to accept the settings, otherwise shows a warning dialog and returns False.

_add_empty_setting(*self*, *_*)
Adds an empty settings widget to the widget list.

_remove_setting(*self*, *settings_widget*)
Removes a setting widget from the widget list.

_collect_and_hide(*self*)
Collects settings from individual ParameterMergingSettings widgets and hides the window.

_reject_and_close(*self*)
Emits settings_rejected and closes the window.

class spinetoolbox.project_items.exporter.widgets.parameter_merging_settings_window.**EntityC**

Contains information of an entity_class (object or relationship_class) for use in the parameter merging widget.

name
entity's name

Type str

class_id
entity's database id

Type int

domain_names
object classes that index the entities in this class; for object classes this list contains the entity's name only, for relationship classes the list contains the relationship's object classes

Type list

parameter_names
entity's defined parameters

Type list

is_object_class
True if the entity is a object_class, False if it is a relationship_class

Type bool

Parameters

- **name** (*str*) – entity's name
- **class_id** (*int*) – entity's database id
- **domain_names** (*list*) – object classes that index the entities in this class; for object classes this list contains the entity's name only, for relationship classes the list contains the relationship's object classes
- **parameter_names** (*list*) – entity's defined parameters
- **is_object_class** (*bool*) – True if the entity is a object_class, False if it is a relationship_class

spinetoolbox.project_items.exporter.widgets.parameter_merging_settings_window.**_gather_entit**
Collects entity_class infos from database.

Parameters **db_map** (*spinedb_api.DatabaseMapping*) – a database map

Returns a list of EntityClassInfo objects

Return type list

Submodules

spinetoolbox.project_items.exporter.commands

Undo/redo commands for the Exporter project item.

authors

A. Soininen (VTT)

date 30.4.2020

Module Contents

Classes

UpdateExporterOutFileName	Command to update Exporter output file name.
UpdateScenario	**param exporter** the Exporter
UpdateExporterSettings	Command to update Exporter settings.

class spinetoolbox.project_items.exporter.commands.**UpdateExporterOutFileName**(*exporter*, *file_name*, *database_path*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Exporter output file name.

Parameters

- **exporter** (Exporter) – the Exporter
- **file_name** (*str*) – the output filename
- **database_path** (*str*) – the associated db path

redo(*self*)

undo(*self*)

class spinetoolbox.project_items.exporter.commands.**UpdateScenario**(*exporter*, *scenario*, *database_url*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Parameters

- **exporter** (Exporter) – the Exporter
- **scenario** (*str, optional*) – new scenario name
- **database_url** (*str*) – database URL

redo(*self*)

undo(*self*)

class spinetoolbox.project_items.exporter.commands.**UpdateExporterSettings**(*exporter*, *settings*, *indexing_settings*, *merging_settings*, *none_fallback*, *none_export*, *database_path*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Exporter settings.

Parameters

- **exporter** (Exporter) – the Exporter
- **settings** (SetSettings) – gdx settings
- **indexing_settings** (*dict*) – parameter indexing settings
- **merging_settings** (*dict*) – parameter merging settings
- **none_fallback** (NoneFallback) – fallback option on None values
- **none_export** (NoneExport) – how to handle Nones while exporting
- **database_path** (*str*) – the db path to update settings for

redo(*self*)

undo(*self*)

spinetoolbox.project_items.exporter.db_utils

Contains utility functions to help with Spine databases.

author

A. Soininen (VTT)

date 5.9.2019

Module Contents

Functions

latest_database_commit_time_stamp(database_map)	Searches the latest commit timestamp from given database
scenario_filtered_database_map(database_url, scenario)	Creates a database mapping and applies scenario filtering to it.

spinetoolbox.project_items.exporter.db_utils.**latest_database_commit_time_stamp**(*database_map*)

Searches the latest commit timestamp from given database

Parameters **database_map** (*DatabaseMappingBase*) – database map

Returns latest time stamp or None if there are no commits.

Return type datetime

spinetoolbox.project_items.exporter.db_utils.**scenario_filtered_database_map**(*database_url*, *scenario*)

Creates a database mapping and applies scenario filtering to it.

Parameters

- **database_url** (*str*) – database URL
- **scenario** (*str, optional*) – scenario name or None for the 'Base' alternative

Returns database mapping

Return type DatabaseMapping

spinetoolbox.project_items.exporter.executable_item

Contains Exporter's executable item as well as support utilities.

authors

A. Soininen (VTT)

date 2.4.2020

Module Contents

Classes

ExecutableItem	The part of a project item that is executed by the Spine Engine.

class spinetoolbox.project_items.exporter.executable_item.**ExecutableItem**(*name*, *settings_packs*, *cancel_on_error*, *data_dir*, *gams_path*, *logger*)

Bases: *spinetoolbox.executable_item_base.ExecutableItemBase*

The part of a project item that is executed by the Spine Engine.

Parameters

- **name** (*str*) – item's name
- **settings_packs** (*dict*) – mapping from database URLs to SettingsPacks
- **cancel_on_error** (*bool*) – True if execution should fail on all errors, False if certain errors can be ignored
- **data_dir** (*str*) – absolute path to exporter's data directory
- **gams_path** (*str*) – GAMS path from Toolbox settings
- **logger** (LoggerInterface) – a logger

static item_type()
Returns Exporter's type identifier string.

_execute_forward(*self*, *resources*)
See base class.

_output_resources_forward(*self*)
See base class.

_resolve_gams_system_directory(*self*)
Returns GAMS system path from Toolbox settings or None if GAMS default is to be used.

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
See base class.

spinetoolbox.project_items.exporter.exporter

Exporter project item.

author

A. Soininen (VTT)

date 5.9.2019

Module Contents

Classes

Exporter	This project item handles all functionality regarding exporting a database to a file.

Functions

_normalize_url(url)	Normalized url's path separators to their OS specific characters.
_latest_database_commit_time_stamp(url)	Returns the latest commit timestamp from database at given URL or None.

class spinetoolbox.project_items.exporter.exporter.**Exporter**(*toolbox*, *project*, *logger*, *name*, *description*, *settings_packs*, *x*, *y*, *cancel_on_export_error=None*, *cancel_on_error=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

This project item handles all functionality regarding exporting a database to a file.

Currently, only .gdx format is supported.

Parameters

- **toolbox** (*ToolboxUI*) – a ToolboxUI instance
- **project** (*SpineToolboxProject*) – the project this item belongs to
- **logger** (*LoggerInterface*) – a logger instance
- **name** (*str*) – item name
- **description** (*str*) – item description
- **settings_packs** (*list*) – dicts mapping database URLs to _SettingsPack objects
- **x** (*float*) – initial X coordinate of item icon
- **y** (*float*) – initial Y coordinate of item icon
- **cancel_on_export_error** (*bool, options*) – legacy `cancel_on_error` option
- **cancel_on_error** (*bool, options*) – True if execution should fail on all export errors, False to ignore certain error cases; optional to provide backwards compatibility

set_up(*self*)
See base class.

static item_type()
See base class.

static item_category()
See base class.

execution_item(*self*)
Creates Exporter's execution counterpart.

settings_pack(*self*, *database_path*)

make_signal_handler_dict(*self*)
Returns a dictionary of all shared signals and their handlers.

restore_selections(*self*)
Restores selections and connects signals.

_connect_signals(*self*)
Connect signals to handlers.

_read_scenarios(*self*, *database_url*)
Reads scenarios from database.

Parameters **database_url** (*str*) – database url

Returns a mapping from scenario name to boolean 'active' flag

Return type dict

_update_properties_tab(*self*)
Updates the database list and scenario combo boxes in the properties tab.

_do_handle_dag_changed(*self*, *resources*)
See base class.

_start_worker(*self*, *database_url*, *update_settings=False*)
Starts fetching settings using a worker in another thread.

_worker_msg(*self*, *database_url*, *text*)

_worker_msg_warning(*self*, *database_url*, *text*)

_worker_msg_error(*self*, *database_url*, *text*)

_worker_finished(*self*, *database_url*, *result*)
Gets and updates and export settings pack from a worker.

_worker_failed(*self*, *database_url*, *exception*)
Clean up after a worker has failed fetching export settings.

_cancel_worker(*self*, *database_url*)
Cleans up after worker has given up fetching export settings.

_check_state(*self*, *clear_before_check=True*)
Checks the status of database export settings.

Updates both the notification message (exclamation icon) and settings states.

_check_missing_file_names(*self*)
Checks the status of output file names.

_check_duplicate_file_names(*self*)
Checks for duplicate output file names.

_check_missing_parameter_indexing(*self*)
Checks the status of parameter indexing settings.

_check_erroneous_databases(*self*)
Checks errors in settings fetching from a database.

_report_notifications(*self*)
Updates the exclamation icon and notifications labels.

_show_settings(*self*, *database_url*)
Opens the item's settings window.

_reset_settings_window(*self*, *database_url*)
Sends new settings to Gdx Export Settings window.

_dispose_settings_window(*self*, *database_url*)
Deletes rejected export settings windows.

_update_out_file_name(*self*, *file_name*, *database_path*)
Pushes a new UpdateExporterOutFileNameCommand to the toolbox undo stack.

_update_scenario(*self*, *scenario*, *database_url*)
Updates the selected scenario.

Parameters

- **scenario** (*str or NoneType*) – selected scenario
- **database_url** (*str*) – database URL

set_scenario(*self*, *scenario*, *database_url*)
Sets the selected scenario in settings pack.

Parameters

- **scenario** (*str or NoneType*) – selected scenario
- **database_url** (*str*) – database URL

_update_settings_from_settings_window(*self*, *database_path*)
Pushes a new UpdateExporterSettingsCommand to the toolbox undo stack.

_cancel_on_error_option_changed(*self*, *checkbox_state*)
Handles changes to the Cancel export on error option.

set_cancel_on_error(*self*, *cancel*)
Sets the Cancel export on error option.

undo_redo_out_file_name(*self*, *file_name*, *database_path*)
Updates the output file name for given database

undo_or_redo_settings(*self*, *settings*, *indexing_settings*, *merging_settings*, *none_fallback*, *none_export*, *database_path*)
Updates the export settings for given database.

item_dict(*self*)
Returns a dictionary corresponding to this item's configuration.

_discard_settings_window(*self*, *database_path*)
Discards the settings window for given database.

_send_settings_to_window(*self*, *database_url*)
Resets settings in given export settings window.

update_name_label(*self*)
See base class.

notify_destination(*self*, *source_item*)
See base class.

_update_settings_after_db_commit(*self*, *committed_db_maps*, *cookie*)
Refreshes export settings for databases after data has been committed to them.

_update_settings_after_db_creation(*self*, *url*)
Triggers settings override.

static default_name_prefix()
See base class.

resources_for_direct_successors(*self*)
See base class.

tear_down(*self*)
See base class.

static upgrade_v1_to_v2(*item_name*, *item_dict*)
Upgrades item's dictionary from v1 to v2.

Changes: - output_file_name and database_url stay the same but state is set to Fetching.

Parameters

- **item_name** (*str*) – item's name
- **item_dict** (*dict*) – Version 1 item dictionary

Returns Version 2 Exporter dictionary

Return type dict

spinetoolbox.project_items.exporter.exporter.**_normalize_url**(*url*)
Normalized url's path separators to their OS specific characters.

This function is needed during the transition period from no-version to version 1 project files. It should be removed once we are using version 1 files.

spinetoolbox.project_items.exporter.exporter.**_latest_database_commit_time_stamp**(*url*)
Returns the latest commit timestamp from database at given URL or None.

spinetoolbox.project_items.exporter.exporter_factory

The ExporterFactory class.

author

M. Marin (KTH)

date 15.4.2020

Module Contents

Classes

ExporterFactory	Class for project item factories.

class spinetoolbox.project_items.exporter.exporter_factory.**ExporterFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class.

spinetoolbox.project_items.exporter.exporter_icon

Icon class for the Exporter project item.

authors

A. Soininen (VTT)

date 25.9.2019

Module Contents

Classes

ExporterIcon	Exporter icon for the Design View.

class spinetoolbox.project_items.exporter.exporter_icon.**ExporterIcon**(*toolbox*, *x*, *y*, *project_item*, *icon*)

Bases: *spinetoolbox.graphics_items.ProjectItemIcon*

Exporter icon for the Design View.

Parameters

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate

- **project_item** (ProjectItem) – Item
- **icon** (*str*) – icon resource path

spinetoolbox.project_items.exporter.item_info

Exporter project item info.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

ItemInfo	

class spinetoolbox.project_items.exporter.item_info.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.exporter.list_utils

Contains list helper functions for list manipulation.

author

A. Soininen (VTT)

date 12.12.2019

Module Contents

Functions

move_list_elements(originals, first, last, target)	Moves elements in a list.
move_selected_elements_by(list_view, delta)	Moves selected items in a QListView by given delta.

spinetoolbox.project_items.exporter.list_utils.**move_list_elements**(*originals*, *first*, *last*, *target*)

Moves elements in a list.

Parameters

- **originals** (*list*) – a list
- **first** (*int*) – index of the first element to move
- **last** (*int*) – index of the last element to move
- **target** (*int*) – index where the elements *[first:last]* should be inserted

Returns a new list with the elements moved

spinetoolbox.project_items.exporter.list_utils.**move_selected_elements_by**(*list_view*, *delta*)

Moves selected items in a QListView by given delta.

Parameters

- **list_view** (*QListView*) – a list view
- **delta** (*int*) – positive values move the items up, negative down

spinetoolbox.project_items.exporter.notifications

Contains the Notifications class.

author

A. Soininen (VTT)

date 6.5.2020

Module Contents

Classes

Notifications	Holds flags for different error conditions.

class spinetoolbox.project_items.exporter.notifications.**Notifications**

Bases: PySide2.QtCore.QObject

Holds flags for different error conditions.

duplicate_output_file_name
if True there are duplicate output file names

Type bool

missing_output_file_name
if True the output file name is missing

Type bool

missing_parameter_indexing
if True there are indexed parameters without indexing domains

Type bool

erroneous_database
if True the database has issues

Type bool

changed_due_to_settings_state
Emitted when notifications have changed due to changes in settings state.

__ior__ (*self*, *other*)
ORs the flags with another notifications.

Parameters **other** (Notifications) – a Notifications object

update_settings_state (*self*, *state*)
Updates the notifications according to settings state.

spinetoolbox.project_items.exporter.settings_pack

Contains the SettingsPack class.

author

A. Soininen (VTT)

date 6.5.2020

Module Contents

Classes

SettingsPack	Keeper of all settings and stuff needed for exporting a database.
_UnsupportedValueTypeLogger	

class spinetoolbox.project_items.exporter.settings_pack.**SettingsPack**(*output_file_name*)
Bases: PySide2.QtCore.QObject

Keeper of all settings and stuff needed for exporting a database.

output_file_name
name of the export file

Type str

settings
export settings

Type *gdx.SetSettings*

indexing_settings
parameter indexing settings

Type dict

merging_settings
parameter merging settings

Type dict

none_fallback
fallback for None parameter values

Type *NoneFallback*

none_export
how to handle None values while exporting

Type *NoneExport*

scenario
name of the scenario to export; None for 'Base' alternative

Type str

last_database_commit
latest database commit time stamp

Type datetime

settings_window
settings editor window

Type *GdxExportSettings*

Parameters **output_file_name** (*str*) – name of the export file

state_changed
Emitted when the pack's state changes.

state
State of the pack.

to_dict(*self*)
Stores the settings pack into a JSON compatible dictionary.

static from_dict(*pack_dict*, *database_url*, *logger*)
Restores the settings pack from a dictionary.

class spinetoolbox.project_items.exporter.settings_pack.**_UnsupportedValueTypeLogger**(*preample*, *real_logg*

Bases: PySide2.QtCore.QObject

msg

msg_warning

msg_error

relay_message(*self*, *text*)

relay_warning(*self*, *text*)

relay_error(*self*, *text*)

spinetoolbox.project_items.exporter.settings_state

Provides the SettingsState enum.

author

A. Soininen (VTT)

date 20.12.2019

Module Contents

Classes

SettingsState	State of export settings.

class spinetoolbox.project_items.exporter.settings_state.**SettingsState**
Bases: enum.Enum

State of export settings.

Create and return a new object. See help(type) for accurate signature.

OK
Settings OK.

FETCHING
Settings are still being fetched/constructed.

INDEXING_PROBLEM
There is a parameter_value indexing issue.

ERROR
An error prevents the creation of export settings.

spinetoolbox.project_items.exporter.worker

A worker based machinery to construct the settings data structures needed for gdx export outside the UI loop.

author
A. Soininen (VTT)

date 19.12.2019

Module Contents

Classes

Worker	A worker to construct export settings for a database.
_Result	Contains fetched export settings.
_Logger	A LoggerInterface compliant logger that relays messages to *Worker*'s signals.

class spinetoolbox.project_items.exporter.worker.**Worker**(*database_url*, *scenario*, *none_fallback*)
Bases: PySide2.QtCore.QObject

A worker to construct export settings for a database.

thread
the thread the worker executes in

Type QThread

Parameters

- **database_url** (*str*) – database's URL
- **scenario** (*str, optional*) – scenario name or None if 'Base' alternative should be used
- **none_fallback** (NoneFallback) – how to handle None parameter values

database_unavailable
Emitted when opening the database fails.

errored
Emitted when an error occurs.

finished
Emitted when the worker has finished.

msg

msg_warning

msg_error

_fetch_settings(*self*)
Constructs settings and parameter index settings.

set_previous_settings(*self*, *previous_settings*, *previous_indexing_settings*, *previous_merging_settings*)
Makes worker update existing settings instead of just making new ones.

Parameters

- **previous_settings** (gdx.SetSettings) – existing set settings
- **previous_indexing_settings** (*dict*) – existing indexing settings
- **previous_merging_settings** (*dict*) – existing merging settings

static _read_scenarios(*database_map*)

_read_settings(*self*)
Reads fresh gdx settings from the database.

_update_indexing_settings(*self*, *updated_settings*, *new_indexing_settings*)
Updates the parameter indexing settings according to changes in the database.

_update_merging_settings(*self*, *updated_settings*)
Updates the parameter merging settings according to changes in the database

class spinetoolbox.project_items.exporter.worker.**_Result**(*time_stamp*, *set_settings*, *indexing_settings*, *scenarios*)
Contains fetched export settings.

commit_time_stamp
time of the database's last commit

Type datetime

set_settings
gdx export settings

Type *gdx.SetSettings*

indexing_settings
parameter indexing settings

Type dict

merging_settings
parameter merging settings

Type dict

scenarios
map from scenario name to boolean 'active' flag

Type dict

Parameters

- **time_stamp** (*datetime*) – time of the database's last commit
- **set_settings** (gdx.SetSettings) – gdx export settings
- **indexing_settings** (*dict*) – parameter indexing settings
- **scenarios** (*dict*) – map from scenario name to boolean 'active' flag

class spinetoolbox.project_items.exporter.worker.**_Logger**(*database_url*, *worker*)
Bases: PySide2.QtCore.QObject

A LoggerInterface compliant logger that relays messages to *Worker*'s signals.

Parameters

- **database_url** (*str*) – a database url
- **worker** (Worker) – a worker

msg

msg_warning

msg_error

relay_message(*self*, *text*)

relay_warning(*self*, *text*)

relay_error(*self*, *text*)

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.exporter.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class.

class spinetoolbox.project_items.exporter.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.gimlet

Gimlet project item plugin.

author

P. Savolainen (VTT)

date 15.4.2020

Subpackages

spinetoolbox.project_items.gimlet.widgets

Widgets for the Gimlet project item.

author

P. Savolainen (VTT)

date 15.4.2020

Submodules

spinetoolbox.project_items.gimlet.widgets.add_gimlet_widget

Widget shown to user when a new Gimlet is created.

author

P. Savolainen (VTT)

date 15.4.2020

Module Contents

Classes

AddGimletWidget	A widget to query user's preferences for a new item.

class spinetoolbox.project_items.gimlet.widgets.add_gimlet_widget.**AddGimletWidget**(*toolbox*, *x*, *y*, *spec=''*)

Bases: *spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget*

A widget to query user's preferences for a new item.

Parameters

- **toolbox** (ToolboxUI) – Parent widget
- **x** (*int*) – X coordinate of new item
- **y** (*int*) – Y coordinate of new item
- **spec** (*str*) – Gimlet specification

call_add_item(*self*)

Creates new project item according to user's selections.

spinetoolbox.project_items.gimlet.widgets.custom_menus

Classes for custom context menus and pop-up menus for Gimlets

author

P. Savolainen (VTT)

date 15.4.2020

Module Contents

Classes

GimletPropertiesContextMenu	Context menu class for a Gimlet project item properties.

class spinetoolbox.project_items.gimlet.widgets.custom_menus.**GimletPropertiesContextMenu**(*pa*
po
si-
tio
in
de

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for a Gimlet project item properties.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

spinetoolbox.project_items.gimlet.widgets.gimlet_properties_widget

Gimlet properties widget.

author

P. Savolainen (VTT)

date 15.4.2020

Module Contents

Classes

GimletPropertiesWidget	Widget for the Gimlet Item Properties.

class spinetoolbox.project_items.gimlet.widgets.gimlet_properties_widget.**GimletPropertiesW**

Bases: PySide2.QtWidgets.QWidget

Widget for the Gimlet Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embedded

Submodules

spinetoolbox.project_items.gimlet.commands

Undo/redo commands for the Gimlet project item.

authors

P. Savolainen (VTT)

date 30.4.2020

Module Contents

Classes

UpdateShellCheckBoxCommand	Command to update Gimlet shell check box state.
UpdateShellComboboxCommand	Command to update Gimlet shell combobox state.
UpdatecmdCommand	Command to update Gimlet command line edit.
UpdateWorkDirModeCommand	Command to update Gimlet work_in_dir setting.

class spinetoolbox.project_items.gimlet.commands.**UpdateShellCheckBoxCommand**(*gimlet*, *use_shell*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Gimlet shell check box state.

Parameters

- **gimlet** (spinetoolbox.project_items.gimlet.gimlet.Gimlet) – The Gimlet issuing the command
- **use_shell** (*bool*) – New check box state

redo(*self*)

undo(*self*)

class spinetoolbox.project_items.gimlet.commands.**UpdateShellComboboxCommand**(*gimlet*, *new_index*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Gimlet shell combobox state.

Parameters

- **gimlet** (spinetoolbox.project_items.gimlet.gimlet.Gimlet) – The Gimlet issuing the command
- **new_index** (*int*) – New combobox index

redo(*self*)

undo(*self*)

class spinetoolbox.project_items.gimlet.commands.**UpdatecmdCommand**(*gimlet*, *txt*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Gimlet command line edit.

Parameters

- **gimlet** (spinetoolbox.project_items.gimlet.gimlet.Gimlet) – The Gimlet issuing the command
- **txt** (*str*) – New text in command line edit after editing is finished

redo(*self*)

undo(*self*)

class spinetoolbox.project_items.gimlet.commands.**UpdateWorkDirModeCommand**(*gimlet*, *work_dir_mode*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Gimlet work_in_dir setting.

Parameters

- **gimlet** (Gimlet) – The Gimlet
- **work_dir_mode** (*bool*) – True or False

redo(*self*)

undo(*self*)

spinetoolbox.project_items.gimlet.executable_item

Contains Gimlet ExecutableItem class.

author

P. Savolainen (VTT)

date 15.4.2020

Module Contents

Classes

ExecutableItem	The part of a project item that is executed by the Spine Engine.

Functions

_file_paths_from_resources(resources)	Pries file paths from resources.
_database_urls_from_resources(resources)	Pries database URLs and their providers' names from resources.

class spinetoolbox.project_items.gimlet.executable_item.**ExecutableItem**(*name*, *logger*, *shell*, *cmd*, *work_dir*, *selected_files*)

Bases: *spinetoolbox.executable_item_base.ExecutableItemBase*, PySide2.QtCore.QObject

The part of a project item that is executed by the Spine Engine.

Parameters

- **name** (*str*) – Project item name

- **logger** (LoggerInterface) – Logger instance
- **shell** (*str*) – Shell name or empty string if no shell should be used
- **cmd** (*list*) – Command to execute
- **work_dir** (*str*) – Full path to work directory
- **selected_files** (*list*) – List of file paths that were selected

gimlet_finished
Emitted after the Gimlet process has finished.

static item_type()
Returns Gimlet's type identifier string.

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
See base class.

stop_execution(*self*)
Stops executing this Gimlet.

_execute_forward(*self*, *resources*)
See base class.

Note: resources given here in args is not used. Files to be copied are given by the Gimlet project item based on user selections made in Gimlet properties.

Parameters **resources** (*list*) – List of resources from direct predecessor items

Returns True if execution succeeded, False otherwise

_execute_backward(*self*, *resources*)
Executes this item in the backward direction.

_output_resources_forward(*self*)
Returns output resources for forward execution.

Returns (list) List of ProjectItemResources.

_output_resources_backward(*self*)
Returns output resources for backward execution. The default implementation returns an empty list.

Returns (list) List of ProjectItemResources. Just an empty list for now.

_handle_gimlet_process_finished(*self*, *ret_code*)
Handles clean up after Gimlet process has finished. After clean up, emits a signal indicating that this project item execution is done.

_copy_files(*self*, *files*, *work_dir*)
Copies selected resources (files) to work directory.

Parameters

- **files** (*list*) – List of full paths to files that will be copied to work dir
- **work_dir** (*str*) – Full path to selected work dir

Returns True when files were copied successfully, False when something went wrong

Return type bool

_expand_gimlet_tags(*self*, *cmd*, *resources*)
Returns Gimlet's command as list with special tags expanded.

Tags that will be replaced:

- @@optional_inputs@@ expands to a space-separated list of Gimlet's optional input files
- @@url:<Data Store name>@@ expands to the URL provided by a named data store
- @@url_inputs@@ expands to a space-separated list of Gimlet's input database URLs
- @@url_outputs@@ expands to a space-separated list of Gimlet's output database URLs

Parameters

- **cmd** (*list*) – Command that may include tags that should be expanded
- **resources** (*list*) – List of resources from direct predecessor items

Returns Expanded command

Return type list

spinetoolbox.project_items.gimlet.executable_item.**_file_paths_from_resources**(*resources*)

Pries file paths from resources.

Parameters **resources** (*list*) – a list of ProjectItemResource objects

Returns List of file paths.

Return type list

spinetoolbox.project_items.gimlet.executable_item.**_database_urls_from_resources**(*resources*)

Pries database URLs and their providers' names from resources.

Parameters **resources** (*list*) – a list of ProjectItemResource objects

Returns a mapping from resource provider's name to a database URL.

Return type dict

spinetoolbox.project_items.gimlet.gimlet

Gimlet class module.

author

P. Savolainen (VTT)

date 15.4.2020

Module Contents

Classes

Gimlet	Gimlet class.

class spinetoolbox.project_items.gimlet.gimlet.**Gimlet**(*toolbox*, *project*, *logger*, *name*, *description*, *x*, *y*, *use_shell=True*, *shell_index=0*, *cmd=''*, *selections=None*, *work_dir_mode=True*)

Bases: *spinetoolbox.project_item.ProjectItem*

Gimlet class.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – Project this item belongs to
- **logger** (LoggerInterface) – Logger instance
- **name** (*str*) – Project item name
- **description** (*str*) – Description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **use_shell** (*bool, optional*) – Use shell flag
- **shell_index** (*int, optional*) – Selected shell as index
- **cmd** (*str, optional*) – Command that this Gimlet executes at run time
- **selections** (*list, optional*) – List of selected files in a serialized format
- **work_dir_mode** (*bool*) – True uses Gimlet's default work dir, False uses a unique work dir on every execution

static item_type()
See base class.

static item_category()
See base class.

execution_item(*self*)
Creates project item's execution counterpart.

_split_gimlet_cmd(*self*, *cmd*)
Splits given string command to a list.

Parameters **cmd** (*str*) – Command to execute as a string

Returns Same command as a list

Return type list

make_signal_handler_dict(*self*)
Returns a dictionary of all shared signals and their handlers. This is to enable simpler connecting and disconnecting.

restore_selections(*self*)
Restores selections into shared widgets when this project item is selected.

save_selections(*self*)
Saves selections in shared widgets for this project item into instance variables.

shell_checkbox_clicked(*self*, *state*)
Pushes a new shell check box command to undo stack. Pushing the command calls the commands redo method.

Parameters **state** (*str*) – New check box state (Qt.CheckState enum)

toggle_shell_state(*self*, *use_shell*)
Sets the use shell check box state. Disables shell combobox when shell check box is unchecked.

Parameters **use_shell** (*bool*) – New check box state

shell_combobox_index_changed(*self*, *ind*)
Pushes a shell combobox selection changed command to undo stack, which in turn calls set_shell_combobox_index() below.

Parameters **ind** (*int*) – New index in combo box

set_shell_combobox_index(*self*, *ind*)
Sets new index to shell combobox.

Parameters **ind** (*int*) – New index in shell combo box

cmd_edited(*self*)
Updates the command instance variable when user has finished editing text in the line edit.

set_command(*self*, *txt*)

_push_file_selection_change_to_undo_stack(*self*, *selected*, *label*)
Makes changes to file selection undoable.

set_file_selected(*self*, *label*, *selected*)
Handles selecting files in Gimlet file list.

push_work_dir_mode_cmd(*self*, *checked*)
Pushes a new UpdateWorkDirModeCommand to the undo stack.

update_work_dir_mode(*self*, *work_dir_mode*)
Updates work_dir_mode setting.

Parameters

- **work_dir_mode** (*bool*) – If True, work dir is set to this Gimlet's data dir,
- **False, a unique work dir is created for every execution.** (*IF*) –

update_work_dir_button_state(*self*)
Sets the work dir radio button check state according to work_dir_mode instance variable.

_do_handle_dag_changed(*self*, *resources*)
Saves a copy of ProjectItemResources for handling changes in the DAG on Design View.

See also base class.

Parameters **resources** (*list*) – ProjectItemResources available from direct predecessors

item_dict(*self*)
Returns a dictionary corresponding to this item.

notify_destination(*self*, *source_item*)
See base class.

_notify_if_duplicate_file_paths(*self*)
Adds a notification if file list contains duplicate entries.

update_name_label(*self*)
Updates the name label in Gimlet properties tab. Used only when a project item is renamed.

static default_name_prefix()
See base class.

resources_for_direct_successors(*self*)
Returns resources for direct successors.

This enables communication of resources between project items in the app.

Returns List of ProjectItemResources

Return type list

spinetoolbox.project_items.gimlet.gimlet_factory

The GimletFactory class.

author

P. Savolainen (VTT)

date 22.4.2020

Module Contents

Classes

GimletFactory	Class for project item factories.

class spinetoolbox.project_items.gimlet.gimlet_factory.**GimletFactory**(*toolbox*)

Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

properties_widget_maker

item_maker

Returns a ProjectItem subclass.

Returns class

icon_maker

Returns a ProjectItemIcon subclass.

Returns class

add_form_maker

Returns an AddProjectItem subclass.

Returns class

specification_form_maker

Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker

Returns an ItemSpecificationMenu subclass.

Returns class

static icon()

Returns the icon resource path.

Returns str

static item_type()

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

spinetoolbox.project_items.gimlet.gimlet_icon

Module for Gimlet icon class.

authors

P. Savolainen (VTT)

date 15.4.2020

Module Contents

Classes

GimletIcon	Gimlet icon for the Design View.

class spinetoolbox.project_items.gimlet.gimlet_icon.**GimletIcon**(*toolbox*, *x*, *y*, *project_item*, *icon*)

Bases: *spinetoolbox.graphics_items.ProjectItemIcon*

Gimlet icon for the Design View.

Parameters

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate
- **project_item** (ProjectItem) – Item
- **icon** (*str*) – Icon resource path

spinetoolbox.project_items.gimlet.item_info

Gimlet project item info.

author

P. Savolainen (VTT)

date 15.5.2020

Module Contents

Classes

ItemInfo	

class spinetoolbox.project_items.gimlet.item_info.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.gimlet.utils

Utility constants and functions for the Gimlet package.

author

P. Savolainen (VTT)

date 20.4.2020

Module Contents

spinetoolbox.project_items.gimlet.utils.**SHELLS = ['cmd.exe', 'powershell.exe', 'bash']**

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.gimlet.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

properties_widget_maker

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static item_type()

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.gimlet.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.importer

Importer plugin.

author

M. Marin (KTH)

date 12.9.2019

Subpackages

spinetoolbox.project_items.importer.widgets

Widgets for the Importer project item.

author A.Soininen (VTT)

date 27.9.2019

Submodules

spinetoolbox.project_items.importer.widgets.add_importer_widget

Widget shown to user when a new Importer is created.

author

P. Savolainen (VTT)

date 19.1.2017

Module Contents

Classes

AddImporterWidget	A widget to query user's preferences for a new item.

class spinetoolbox.project_items.importer.widgets.add_importer_widget.**AddImporterWidget**(*toolbox*, *x*, *y*, *spec*)

Bases: *spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget*

A widget to query user's preferences for a new item.

Parameters

- **toolbox** (*ToolboxUI*) – Parent widget
- **x** (*float*) – X coordinate of new item
- **y** (*float*) – Y coordinate of new item

call_add_item(*self*)
Creates new Item according to user's selections.

spinetoolbox.project_items.importer.widgets.custom_menus

Classes for context menus used alongside the Importer project item.

author

P. Savolainen (VTT)

date 9.1.2018

Module Contents

Classes

FilesContextMenu	Context menu class for source files view in Importer properties tab.

class spinetoolbox.project_items.importer.widgets.custom_menus.**FilesContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for source files view in Importer properties tab.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

spinetoolbox.project_items.importer.widgets.importer_properties_widget

Importer properties widget.

author

M. Marin (KTH)

date 12.9.2019

Module Contents

Classes

ImporterPropertiesWidget	Widget for the Importer Item Properties.

class spinetoolbox.project_items.importer.widgets.importer_properties_widget.**ImporterPropertiesWidget**(*toolbox*)
Bases: PySide2.QtWidgets.QWidget

Widget for the Importer Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embedded

connect_signals(*self*)
Connect signals to slots.

show_files_context_menu(*self*, *pos*)
Create and show a context-menu in Importer properties source files view.

Parameters **pos** (*QPoint*) – Mouse position

Submodules

spinetoolbox.project_items.importer.commands

Undo/redo commands for the Importer project item.

authors

M. Marin (KTH)

date 5.5.2020

Module Contents

Classes

UpdateSettingsCommand	Command to update Importer settings.

class spinetoolbox.project_items.importer.commands.**UpdateSettingsCommand**(*importer*, *settings*, *label*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Importer settings.

Parameters

- **importer** (spinetoolbox.project_items.importer.importer.Importer) – the Importer
- **settings** (*dict*) – the new settings
- **label** (*str*) – settings file label

redo(*self*)

undo(*self*)

spinetoolbox.project_items.importer.executable_item

Contains Importer's executable item as well as support utilities.

authors

A. Soininen (VTT)

date 1.4.2020

Module Contents

Classes

ExecutableItem	The part of a project item that is executed by the Spine Engine.

Functions

_files_from_resources(resources)	Returns a list of files available in given resources.

class spinetoolbox.project_items.importer.executable_item.**ExecutableItem**(*name*, *settings*, *logs_dir*, *gams_path*, *cancel_on_error*, *logger*)

Bases: *spinetoolbox.executable_item_base.ExecutableItemBase*, PySide2.QtCore.QObject

The part of a project item that is executed by the Spine Engine.

Parameters

- **name** (*str*) – Importer's name
- **settings** (*dict*) – import mappings
- **logs_dir** (*str*) – path to the directory where logs should be stored
- **gams_path** (*str*) – path to system's GAMS executable or empty string for the default path
- **cancel_on_error** (*bool*) – if True, revert changes on error and quit
- **logger** (LoggerInterface) – a logger

importing_finished
Emitted after import thread has finished.

static item_type()
Returns ImporterExecutable's type identifier string.

stop_execution(*self*)
Stops execution.

_execute_backward(*self*, *resources*)
See base class.

_execute_forward(*self*, *resources*)
See base class.

_handle_worker_finished(*self*, *exit_code*)

_destroy_current_worker(*self*)
Runs before starting execution and after worker finishes. Destroys current worker and quits thread if present.

_gams_system_directory(*self*)
Returns GAMS system path or None if GAMS default is to be used.

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
See base class.

spinetoolbox.project_items.importer.executable_item.**_files_from_resources**(*resources*)
Returns a list of files available in given resources.

spinetoolbox.project_items.importer.importer

Contains Importer project item class.

authors

P. Savolainen (VTT), P. Vennström (VTT), A. Soininen (VTT)

date 10.6.2019

Module Contents

Classes

Importer	Class for project items that are not category nor root.

Functions

_fix_1d_array_to_array(mappings)	Replaces '1d array' with 'array' for parameter type in mappings.
_fix_csv_connector_settings(settings)	CSVConnector saved the table names as the filepath, change that

spinetoolbox.project_items.importer.importer.**_CONNECTOR_NAME_TO_CLASS**

class spinetoolbox.project_items.importer.importer.**Importer**(*toolbox*, *project*, *logger*, *name*, *description*, *mappings*, *x*, *y*, *cancel_on_error=True*, *mapping_selection=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Class for project items that are not category nor root. These items can be executed, refreshed, and so on.

x

horizontal position in the screen

Type float

y

vertical position in the screen

Type float

Importer class.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **name** (*str*) – Project item name
- **description** (*str*) – Project item description
- **mappings** (*list*) – List where each element contains two dicts (path dict and mapping dict)

- **x** (*float*) – Initial icon scene X coordinate
- **y** (*float*) – Initial icon scene Y coordinate
- **cancel_on_error** (*bool, optional*) – if True the item's execution will stop on import error
- **mapping_selection** (*list, optional*) – serialized checked states for each file item either selected or unselected

static item_type()
See base class.

static item_category()
See base class.

execution_item(*self*)
Creates project item's execution counterpart.

handle_execution_successful(*self*, *execution_direction*, *engine_state*)
Notifies Toolbox of successful database import.

make_signal_handler_dict(*self*)
Returns a dictionary of all shared signals and their handlers. This is to enable simpler connecting and disconnecting.

_handle_cancel_on_error_changed(*self*, *_state*)

set_cancel_on_error(*self*, *cancel_on_error*)

set_file_selected(*self*, *label*, *selected*)

restore_selections(*self*)
Restores selections into shared widgets when this project item is selected.

save_selections(*self*)
Saves selections in shared widgets for this project item into instance variables.

update_name_label(*self*)
Update Importer properties tab name label. Used only when renaming project items.

_handle_import_editor_clicked(*self*, *checked=False*)
Opens Import editor for the file selected in list view.

_handle_files_double_clicked(*self*, *index*)
Opens Import editor for the double clicked index.

open_import_editor(*self*, *index*)
Opens Import editor for the given index.

get_connector(*self*, *importee*)
Shows a QDialog to select a connector for the given source file. Mimics similar routine in *spine_io.widgets.import_widget.ImportDialog*

Parameters **importee** (*str*) – Label of the file acting as an importee

Returns Asynchronous data reader class for the given importee

select_connector_type(*self*, *index*)
Opens dialog to select connector type for the given index.

_connection_failed(*self*, *msg*, *importee*)

get_settings(*self*, *importee*)
Returns the mapping dictionary for the file in given path.

Parameters **importee** (*str*) – Label of the file whose mapping is queried

Returns Mapping dictionary for the requested importee or an empty dict if not found

Return type dict

save_settings(*self*, *settings*, *importee*)

Updates an existing mapping or adds a new mapping (settings) after closing the import preview window.

Parameters

- **settings** (*dict*) – Updated mapping (settings) dictionary
- **importee** (*str*) – Absolute path to a file, whose mapping has been updated

_preview_destroyed(*self*, *importee*)
Destroys preview widget instance for the given importee.

Parameters **importee** (*str*) – Absolute path to a file, whose preview widget is destroyed

_push_file_selection_change_to_undo_stack(*self*, *selected*, *label*)
Makes changes to file selection undoable.

_do_handle_dag_changed(*self*, *resources*)
See base class.

item_dict(*self*)
Returns a dictionary corresponding to this item.

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

tear_down(*self*)
Closes all preview widgets.

_notify_if_duplicate_file_paths(*self*)
Adds a notification if file_list contains duplicate entries.

static upgrade_from_no_version_to_version_1(*item_name*, *old_item_dict*, *old_project_dir*)
Converts mappings to a list, where each element contains two dictionaries, the serialized path dictionary and the mapping dictionary for the file in that path.

static upgrade_v1_to_v2(*item_name*, *item_dict*)
Upgrades item's dictionary from v1 to v2.

Changes: - if mapping_selection does not exist or if it is a list of booleans, reset mapping_selection to an empty list.

Parameters

- **item_name** (*str*) – item's name
- **item_dict** (*dict*) – Version 1 item dictionary

Returns Version 2 Exporter dictionary

Return type dict

static serialize_mappings(*mappings*, *project_path*)
Serializes the importer's mappings.

Returns a list where each element contains two dictionaries: the 'serialized' file label in a dictionary and the mapping dictionary.

Parameters

- **mappings** (*dict*) – Dictionary with mapping specifications
- **project_path** (*str*) – Path to project directory

Returns serialized file item labels and mappings

Return type list

_gams_system_directory(*self*)
Returns GAMS system path from Toolbox settings or None if GAMS default is to be used.

spinetoolbox.project_items.importer.importer.**_fix_1d_array_to_array**(*mappings*)
Replaces '1d array' with 'array' for parameter type in mappings.

With spinedb_api >= 0.3, '1d array' parameter type was replaced by 'array'. Other settings in a mapping are backwards compatible except the name.

spinetoolbox.project_items.importer.importer.**_fix_csv_connector_settings**(*settings*)
CSVConnector saved the table names as the filepath, change that to 'csv' instead. This function will mutate the dictionary.

Parameters **settings** (*dict*) – Mapping settings that should be updated

spinetoolbox.project_items.importer.importer_factory

The ImporterFactory class.

author

M. Marin (KTH)

date 15.4.2020

Module Contents

Classes

ImporterFactory	Class for project item factories.

class spinetoolbox.project_items.importer.importer_factory.**ImporterFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker

Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

spinetoolbox.project_items.importer.importer_icon

Module for Importer icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

Classes

ImporterIcon	Importer icon for the Design View.

class spinetoolbox.project_items.importer.importer_icon.**ImporterIcon**(*toolbox*, *x*, *y*, *project_item*, *icon*)

Bases: *spinetoolbox.graphics_items.ProjectItemIcon*

Importer icon for the Design View.

Parameters

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate

- **project_item** (ProjectItem) – Item
- **icon** (*str*) – icon resource path

spinetoolbox.project_items.importer.importer_worker

Contains importer_program script.

authors

P. Savolainen (VTT), P. Vennström (VTT), A. Soininen (VTT)

date 10.6.2019

Module Contents

Classes

ImporterWorker	**param checked_files** List of paths to checked source files

class spinetoolbox.project_items.importer.importer_worker.**ImporterWorker**(*checked_files*, *all_import_settings*, *all_source_settings*, *urls_downstream*, *logs_dir*, *cancel_on_error*, *logger*)

Bases: PySide2.QtCore.QObject

Parameters

- **checked_files** (*list(str)*) – List of paths to checked source files
- **all_import_settings** (*dict*) – Maps source file to setting for that file
- **all_source_settings** (*dict*) – Maps source type to setting for that type
- **urls_downstream** (*list(str)*) – List of urls to import data into
- **logs_dir** (*str*) – path to the directory where logs should be written
- **cancel_on_error** (*bool*) – whether or not to rollback and stop execution if errors
- **logger** (LoggerInterface) – somewhere to log important messages

import_finished
Emitted when work is finished with 0 if successful, -1 otherwise.

do_work(*self*)
Does the work and emits import_finished when done.

_import(*self*, *all_data*, *url*)

spinetoolbox.project_items.importer.item_info

Importer project item info.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

ItemInfo	

class spinetoolbox.project_items.importer.item_info.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.importer.utils

Contains Importer's utility functions.

authors

A. Soininen (VTT)

date 6.5.2020

Module Contents

Functions

deserialize_mappings(mappings, project_path)	Returns mapping settings as dict with absolute paths as keys.

spinetoolbox.project_items.importer.utils.**deserialize_mappings**(*mappings*, *project_path*)
Returns mapping settings as dict with absolute paths as keys.

Parameters

- **mappings** (*list*) – List where each element contains two dictionaries (path dict and mapping dict)
- **project_path** (*str*) – Path to project directory

Returns Dictionary with absolute paths as keys and mapping settings as values

Return type dict

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.importer.**ItemFactory**(*toolbox*)

Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.importer.**ItemInfo**

Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.shared

Submodules

spinetoolbox.project_items.shared.animations

Animation class for the Exporter and Importer items.

authors

M. Marin (KTH)

date 12.11.2019

Module Contents

Classes

ImporterExporterAnimation	Initializes animation stuff.
ImporterAnimation	Initializes animation stuff.
ExporterAnimation	Initializes animation stuff.

class spinetoolbox.project_items.shared.animations.**ImporterExporterAnimation**(*item*, *duration=2000*, *count=5*, *percentage_size=0.24*, *x_shift=0*)

Initializes animation stuff.

Parameters **item** (*QGraphicsItem*) – The item on top of which the animation should play.

_handle_time_line_value_changed(*self*, *value*)

_handle_time_line_state_changed(*self*, *new_state*)

start(*self*)
Starts the animation.

static percent(*value*)

stop(*self*)
Stops the animation

class spinetoolbox.project_items.shared.animations.**ImporterAnimation**(*item*, *duration=2000*, *count=5*, *percentage_size=0.24*, *x_shift=0*)

Bases: *spinetoolbox.project_items.shared.animations.ImporterExporterAnimation*

Initializes animation stuff.

Parameters **item** (*QGraphicsItem*) – The item on top of which the animation should play.

static percent(*value*)

class spinetoolbox.project_items.shared.animations.**ExporterAnimation**(*item*, *duration=2000*, *count=5*, *percentage_size=0.24*, *x_shift=0*)

Bases: *spinetoolbox.project_items.shared.animations.ImporterExporterAnimation*

Initializes animation stuff.

Parameters **item** (*QGraphicsItem*) – The item on top of which the animation should play.

static percent(*value*)

spinetoolbox.project_items.shared.commands

Undo/redo commands that can be used by multiple project items.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 5.5.2020

Module Contents

Classes

UpdateCancelOnErrorCommand	Command to update Importer, Exporter, and Combiner cancel on error setting.
ChangeItemSelectionCommand	Command to change file item's selection status.

class spinetoolbox.project_items.shared.commands.**UpdateCancelOnErrorCommand**(*project_item*, *cancel_on_error*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Importer, Exporter, and Combiner cancel on error setting.

Parameters

- **project_item** (ProjectItem) – Item
- **cancel_on_error** (*bool*) – New setting

redo(*self*)

undo(*self*)

class spinetoolbox.project_items.shared.commands.**ChangeItemSelectionCommand**(*project_item*, *selected*, *label*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to change file item's selection status. Used by Importers and Gimlets.

Parameters

- **project_item** (ProjectItem) – Item
- **selected** (*bool*) – True if the item is selected, False otherwise
- **label** (*str*) – File label

redo(*self*)

undo(*self*)

spinetoolbox.project_items.shared.helpers

Helper functions and classes.

authors

M. Marin (KTH)

date 12.5.2020

Module Contents

Classes

CmdlineTag	

Functions

create_log_file_timestamp()	Creates a new timestamp string that is used as Combiner and Importer error log file.
serialize_checked_states(files, project_path)	Serializes file paths and adds a boolean value
deserialize_checked_states(serialized, project_path)	Reverse operation for serialize_checked_states above.
split_cmdline_args(arg_string)	Splits a string of command line into a list of tokens.
expand_tags(args, optional_input_files, input_urls, output_urls)	”

spinetoolbox.project_items.shared.helpers.**CMDLINE_TAG_EDGE = @@**

class spinetoolbox.project_items.shared.helpers.**CmdlineTag**

URL

URL_INPUTS

URL_OUTPUTS

OPTIONAL_INPUTS

spinetoolbox.project_items.shared.helpers.**create_log_file_timestamp**()
Creates a new timestamp string that is used as Combiner and Importer error log file.

Returns Timestamp string or empty string if failed.

spinetoolbox.project_items.shared.helpers.**serialize_checked_states**(*files*, *project_path*)
Serializes file paths and adds a boolean value for each, which indicates whether the path is selected or not. Used in saving checked file states to project.json.

Parameters

- **files** (*list*) – List of absolute file paths
- **project_path** (*str*) – Absolute project directory path

Returns List of serialized paths with a boolean value

Return type list

spinetoolbox.project_items.shared.helpers.**deserialize_checked_states**(*serialized*, *project_path*)
Reverse operation for serialize_checked_states above. Returns absolute file paths with their check state as boolean.

Parameters

- **serialized** (*list*) – List of serialized paths with a boolean value
- **project_path** (*str*) – Absolute project directory path

Returns Dictionary with paths as keys and boolean check states as value

Return type dict

spinetoolbox.project_items.shared.helpers.**split_cmdline_args**(*arg_string*)
Splits a string of command line into a list of tokens.

Things in single (‘) and double (“) quotes are kept as single tokens while the quotes themselves are stripped away. Thus, *–file=”a long quoted ‘file’ name.txt* becomes [“–file=a long quoted ‘file’ name.txt”]

Parameters **arg_string** (*str*) – command line arguments as a string

Returns a list of tokens

Return type list

spinetoolbox.project_items.shared.helpers.**expand_tags**(*args*, *optional_input_files*, *input_urls*, *output_urls*)
” Expands first @@ tags found in given list of command line arguments.

Parameters

- **args** (*list*) – a list of command line arguments
- **optional_input_files** (*list*) – a list of Tool’s optional input file names
- **input_urls** (*dict*) – a mapping from URL provider (input Data Store name) to URL string
- **output_urls** (*dict*) – a mapping from URL provider (output Data Store name) to URL string

Returns

a boolean flag, if True, indicates that tags were expanded and a list of expanded command line arguments

Return type tuple

spinetoolbox.project_items.shared.models

Contains a generic File list model and an Item for that model. Used by the Importer project item but this may be handy for other project items as well.

authors

P. Savolainen (VTT), P. Vennström (VTT), A. Soininen (VTT)

date 5.6.2020

Module Contents

Classes

FileListItem	An item for FileListModel.
FileListModel	A model for files to be shown in a file list view.

Functions

_file_label(resource)	Picks a label for given file resource.

spinetoolbox.project_items.shared.models.**_file_label**(*resource*)
Picks a label for given file resource.

class spinetoolbox.project_items.shared.models.**FileListItem**(*label*, *path*, *provider_name*, *is_pattern=False*)
An item for FileListModel.

Parameters

- **label** (*str*) – File label; a full path for 'permanent' files or just the basename for 'transient' files like Tool's output.
- **path** (*str*) – Absolute path to the file, empty if not known
- **provider_name** (*str*) – Name of the project item providing the file
- **is_pattern** (*bool*) – True if the file is actually a file name pattern

Initialize self. See help(type(self)) for accurate signature.

classmethod from_resource(*cls*, *resource*)
Constructs a FileListItem from ProjectItemResource.

Parameters **resource** (ProjectItemResource) – Resource

Returns An item based on given resource

Return type *FileListItem*

Raises `RuntimeError` – If given resource has an unknown type

exists (*self*)
Returns True if the file exists, False otherwise.

update (*self*, *resource*)
Updates item information.

Parameters **resource** (`ProjectItemResource`) – A fresh file resource

class spinetoolbox.project_items.shared.models.**FileListModel**
Bases: `PySide2.QtCore.QAbstractListModel`

A model for files to be shown in a file list view. Used by Importer.

selected_state_changed
Emitted when a file check box state changes.

files
All model's file items.

data (*self*, *index*, *role=Qt.DisplayRole*)
Returns data associated with given role at given index.

flags (*self*, *index*)
Returns item's flags.

headerData (*self*, *section*, *orientation*, *role=Qt.DisplayRole*)
Returns header information.

find_file (*self*, *label*)
Returns a file item with given label.

labels (*self*)
Returns a list of file labels.

mark_as_nonexistent (*self*, *index*)
Marks item at given index as non-existing.

update (*self*, *resources*)
Updates the model according to given list of resources.

rowCount (*self*, *parent=QModelIndex()*)
Returns the number of rows in the model.

set_selected (*self*, *label*, *selected*)
Changes the given item's selection status.

Parameters

- **label** (*str*) – item's label
- **selected** (*bool*) – True to select the item, False to deselect

setData (*self*, *index*, *value*, *role=Qt.EditRole*)
Sets data in the model.

set_initial_state (*self*, *selected_items*)
Fills model with incomplete data; needs a call to *update()* to make the model usable.

spinetoolbox.project_items.tool

Tool plugin.

author

M. Marin (KTH)

date 12.9.2019

Subpackages

spinetoolbox.project_items.tool.widgets

Widgets for the Tool project icon.

author A.Soininen (VTT)

date 27.9.2019

Submodules

spinetoolbox.project_items.tool.widgets.add_tool_widget

Widget shown to user when a new Tool is created.

author

P. Savolainen (VTT)

date 19.1.2017

Module Contents

Classes

AddToolWidget	A widget that queries user's preferences for a new item.

class spinetoolbox.project_items.tool.widgets.add_tool_widget.**AddToolWidget**(*toolbox*, *x*, *y*, *spec=''*)

Bases: *spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget*

A widget that queries user's preferences for a new item.

toolbox
Parent widget

Type *ToolboxUI*

x
X coordinate of new item

Type int

y
Y coordinate of new item

Type int

spec
Tool specification

Type str

Initialize class.

call_add_item(*self*)
Creates new Item according to user's selections.

spinetoolbox.project_items.tool.widgets.custom_menus

Classes for custom context menus and pop-up menus.

author

P. Savolainen (VTT)

date 9.1.2018

Module Contents

Classes

ToolPropertiesContextMenu	Common context menu class for all Tool QTreeViews in Tool properties.
ToolContextMenu	Context menu for Tools in the QTreeView and in the QGraphicsView.
ToolSpecificationMenu	Context menu class for Tool specifications.
AddIncludesPopupMenu	Popup menu class for add includes button in Tool specification editor widget.
CreateMainProgramPopupMenu	Popup menu class for add main program QToolButton in Tool specification editor widget.

class spinetoolbox.project_items.tool.widgets.custom_menus.**ToolPropertiesContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Common context menu class for all Tool QTreeViews in Tool properties.

parent
Parent for menu widget (ToolboxUI)

Type QWidget

position
Position on screen

Type QPoint

index
Index of item that requested the context-menu

Type QModelIndex

Class constructor.

class spinetoolbox.project_items.tool.widgets.custom_menus.**ToolContextMenu**(*parent*, *tool*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.ProjectItemContextMenu*

Context menu for Tools in the QTreeView and in the QGraphicsView.

parent
Parent for menu widget (ToolboxUI)

Type QWidget

position
Position on screen

Type QPoint

Class constructor.

class spinetoolbox.project_items.tool.widgets.custom_menus.**ToolSpecificationMenu**(*parent*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.ItemSpecificationMenu*

Context menu class for Tool specifications.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **index** (*QModelIndex*) – the index from specification model

open_main_program_file(*self*)

open_main_program_dir(*self*)

class spinetoolbox.project_items.tool.widgets.custom_menus.**AddIncludesPopupMenu**(*parent*)

Bases: *spinetoolbox.widgets.custom_menus.CustomPopupMenu*

Popup menu class for add includes button in Tool specification editor widget.

Parameters **parent** (*QWidget*) – Parent widget (ToolSpecificationWidget)

class spinetoolbox.project_items.tool.widgets.custom_menus.**CreateMainProgramPopupMenu**(*parent*

Bases: *spinetoolbox.widgets.custom_menus.CustomPopupMenu*

Popup menu class for add main program QToolButton in Tool specification editor widget.

Parameters **parent** (*QWidget*) – Parent widget (ToolSpecificationWidget)

spinetoolbox.project_items.tool.widgets.tool_properties_widget

Tool properties widget.

author

M. Marin (KTH)

date 12.9.2019

Module Contents

Classes

ToolPropertiesWidget	Widget for the Tool Item Properties.

class spinetoolbox.project_items.tool.widgets.tool_properties_widget.**ToolPropertiesWidget**(*t*

Bases: PySide2.QtWidgets.QWidget

Widget for the Tool Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embedded

Init class.

connect_signals(*self*)
Connect signals to slots.

update_combo_box_tool_model(*self*)

show_tool_properties_context_menu(*self*, *pos*)
Create and show a context-menu in Tool properties if selected Tool has a Tool specification.

Parameters **pos** (*QPoint*) – Mouse position

spinetoolbox.project_items.tool.widgets.tool_specification_widget

QWidget that is used to create or edit Tool specifications. In the former case it is presented empty, but in the latter it is filled with all the information from the specification being edited.

author

M. Marin (KTH), P. Savolainen (VTT)

date 12.4.2018

Module Contents

Classes

ToolSpecificationWidget	A widget to query user's preferences for a new tool specification.

class spinetoolbox.project_items.tool.widgets.tool_specification_widget.**ToolSpecificationW**

Bases: PySide2.QtWidgets.QWidget

A widget to query user's preferences for a new tool specification.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **specification** (ToolSpecification) – If given, the form is pre-filled with this specification

connect_signals (*self*)
Connect signals to slots.

populate_sourcefile_list (*self*, *items*)
List source files in QTreeView. If items is None or empty list, model is cleared.

populate_inputfiles_list (*self*, *items*)
List input files in QTreeView. If items is None or empty list, model is cleared.

populate_inputfiles_opt_list (*self*, *items*)
List optional input files in QTreeView. If items is None or empty list, model is cleared.

populate_outputfiles_list (*self*, *items*)
List output files in QTreeView. If items is None or empty list, model is cleared.

browse_main_program (*self*, *checked=False*)
Open file browser where user can select the path of the main program file.

set_main_program_path (*self*, *file_path*)
Set main program file and folder path.

new_main_program_file (*self*)
Creates a new blank main program file. Let's user decide the file name and path. Alternative version using only one getSaveFileName dialog.

new_source_file (*self*)
Let user create a new source file for this tool specification.

show_add_source_files_dialog (*self*, *checked=False*)
Let user select source files for this tool specification.

show_add_source_dirs_dialog (*self*, *checked=False*)
Let user select a source directory for this tool specification. All files and sub-directories will be added to the source files.

add_dropped_includes (*self*, *file_paths*)
Adds dropped file paths to Source files list.

add_single_include (*self*, *path*)
Add file path to Source files list.

open_includes_file (*self*, *index*)
Open source file in default program.

remove_source_files_with_del (*self*)
Support for deleting items with the Delete key.

remove_source_files (*self*, *checked=False*)
Remove selected source files from include list. Do not remove anything if there are no items selected.

add_inputfiles (*self*, *checked=False*)
Let user select input files for this tool specification.

remove_inputfiles_with_del (*self*)
Support for deleting items with the Delete key.

remove_inputfiles (*self*, *checked=False*)
Remove selected input files from list. Do not remove anything if there are no items selected.

add_inputfiles_opt (*self*, *checked=False*)
Let user select optional input files for this tool specification.

remove_inputfiles_opt_with_del (*self*)
Support for deleting items with the Delete key.

remove_inputfiles_opt (*self*, *checked=False*)
Remove selected optional input files from list. Do not remove anything if there are no items selected.

add_outputfiles (*self*, *checked=False*)
Let user select output files for this tool specification.

remove_outputfiles_with_del (*self*)
Support for deleting items with the Delete key.

remove_outputfiles (*self*, *checked=False*)
Remove selected output files from list. Do not remove anything if there are no items selected.

handle_ok_clicked (*self*)
Checks that everything is valid, creates Tool spec definition dictionary and adds Tool spec to project.

_make_tool_specification (*self*, *def_path*)
Returns a ToolSpecification from current form settings.

Parameters **def_path** (*str*) – path to definition .json file

Returns ToolSpecification

call_add_tool_specification (*self*)
Adds or updates Tool specification according to user's selections. If the name is the same as an existing tool specification, it is updated and auto-saved to the definition file. (User is editing an existing tool specification.) If the name is not in the tool specification model, creates a new tool specification and offers to save the definition file. (User is creating a new tool specification from scratch or spawning from an existing one).

keyPressEvent (*self*, *e*)
Close Setup form when escape key is pressed.

Parameters **e** (*QKeyEvent*) – Received key press event.

closeEvent (*self*, *event=None*)
Handle close window.

Parameters **event** (*QEvent*) – Closing event if 'X' is clicked.

_make_add_cmdline_tag_menu (*self*)
Constructs a popup menu for the '@@' button.

_insert_spaces_around_tag_in_args_edit (*self*, *tag_length*, *restore_cursor_to_tag_end=False*)
Inserts spaces before/after @@ around cursor position/selection

Expects cursor to be at the end of the tag.

_add_cmdline_tag_url_inputs (*self*, *_*)
Inserts @@url_inputs@@ tag to command line arguments.

_add_cmdline_tag_url_outputs (*self*, *_*)
Inserts @@url_outputs@@ tag to command line arguments.

_add_cmdline_tag_data_store_url (*self*, *_*)
Inserts @@url:<data-store-name>@@ tag to command line arguments and selects '<data-store-name>'.

_add_cmdline_tag_optional_inputs (*self*, *_*)
Inserts @@optional_inputs@@ tag to command line arguments.

Submodules

spinetoolbox.project_items.tool.commands

Undo/redo commands for the Tool project item.

authors

M. Marin (KTH)

date 5.5.2020

Module Contents

Classes

UpdateToolExecuteInWorkCommand	Command to update Tool execute_in_work setting.
UpdateToolCmdLineArgsCommand	Command to update Tool command line args.

class spinetoolbox.project_items.tool.commands.**UpdateToolExecuteInWorkCommand**(*tool*, *execute_in_work*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Tool execute_in_work setting.

Parameters

- **tool** (*Tool*) – the Tool
- **execute_in_work** (*bool*) – True or False

redo(*self*)

undo(*self*)

class spinetoolbox.project_items.tool.commands.**UpdateToolCmdLineArgsCommand**(*tool*, *cmd_line_args*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Tool command line args.

Parameters

- **tool** (*Tool*) – the Tool
- **cmd_line_args** (*list*) – list of str args

redo(*self*)

undo(*self*)

spinetoolbox.project_items.tool.executable_item

Contains Tool's executable item and support functionality.

authors

A. Soininen (VTT)

date 30.3.2020

Module Contents

Classes

ExecutableItem	Tool project item's executable parts.
_ExecutionToken	A token that acts as a callback after the tool process has finished execution.

Functions

_count_files_and_dirs(paths)	Counts the number of files and directories in given paths.
_create_output_dir_timestamp()	Creates a new timestamp string that is used as Tool output
_database_urls_from_resources(resources)	Pries database URLs and their providers' names from resources.
_execution_directory(work_dir, tool_specification)	Returns the path to the execution directory, depending on `execute_in_work`.
_find_files_in_pattern(pattern, available_file_paths)	Returns a list of files that match the given pattern.
_unique_dir_name(tool_specification)	Builds a unique name for Tool's work directory.

class spinetoolbox.project_items.tool.executable_item.**ExecutableItem**(*name*, *work_dir*, *output_dir*, *tool_specification*, *cmd_line_args*, *logger*)

Bases: *spinetoolbox.executable_item_base.ExecutableItemBase*

Tool project item's executable parts.

Parameters

- **name** (*str*) – item's name
- **work_dir** (*str*) – an absolute path to Spine Toolbox work directory or None if the Tool should not execute in work directory
- **output_dir** (*str*) – path to the directory where output files should be archived
- **tool_specification** (ToolSpecification) – a tool specification
- **cmd_line_args** (*list*) – a list of command line argument to pass to the tool instance
- **logger** (LoggerInterface) – a logger

static item_type()

Returns the item's type identifier string.

execution_finished(*self*, *execution_token*, *return_code*, *execution_dir*)
Handles things after execution has finished.

stop_execution(*self*)
Stops executing this Tool.

_copy_input_files(*self*, *paths*, *execution_dir*)
Copies input files from given paths to work or source directory, depending on where the Tool specification requires them to be.

Parameters

- **paths** (*dict*) – key is path to destination file, value is path to source file
- **execution_dir** (*str*) – absolute path to the execution directory

Returns True if the operation was successful, False otherwise

Return type bool

_copy_optional_input_files(*self*, *paths*)
Copies optional input files from given paths to work or source directory, depending on where the Tool specification requires them to be.

Parameters **paths** (*dict*) – key is the source path, value is the destination path

_copy_output_files(*self*, *target_dir*, *execution_dir*)
Copies Tool specification output files from work directory to given target directory.

Parameters

- **target_dir** (*str*) – Destination directory for Tool specification output files
- **execution_dir** (*str*) – path to the execution directory

Returns Contains two lists. The first list contains paths to successfully copied files. The second list contains paths (or patterns) of Tool specification output files that were not found.

Return type tuple

Raises OSError – If creating a directory fails.

_copy_program_files(*self*, *execution_dir*)
Copies Tool specification source files to base directory.

_create_input_dirs(*self*, *execution_dir*)
Iterates items in required input files and check if there are any directories to create. Create found directories directly to work or source directory.

Parameters **execution_dir** (*str*) – the execution directory

Returns True if the operation was successful, False otherwiseBoolean variable depending on success

Return type bool

_create_output_dirs(*self*, *execution_dir*)
Makes sure that work directory has the necessary output directories for Tool output files. Checks only "outputfiles" list. Alternatively you can add directories to "inputfiles" list in the tool definition file.

Parameters **execution_dir** (*str*) – a path to the execution directory

Returns True for success, False otherwise.

Return type bool

Raises OSError – If creating an output directory to work fails.

_execute_backward(*self*, *resources*)
Stores resources for forward execution.

_execute_forward(*self*, *resources*)
Executes the Tool according to the Tool specification.

Before launching the tool script in a separate instance, prepares the execution environment by creating all necessary directories and copying input files where needed. After execution archives the output files.

Parameters **resources** (*list*) – a list of resources from direct predecessor items

Returns True if execution succeeded, False otherwise

_find_input_files(*self*, *resources*)
Iterates required input files in tool specification and looks for them in the given resources.

Parameters **resources** (*list*) – resources available

Returns Dictionary mapping required files to path where they are found, or to None if not found

_find_optional_input_files(*self*, *resources*)
Tries to find optional input files from previous project items in the DAG.

Parameters **resources** (*list*) – resources available

Returns

Dictionary of optional input file paths or an empty dictionary if no files found. Key is the optional input item and value is a list of paths that matches the item.

Return type dict

_handle_output_files(*self*, *return_code*, *execution_dir*)
Copies Tool specification output files from work directory to result directory.

Parameters

- **return_code** (*int*) – Tool specification process return value
- **execution_dir** (*str*) – path to the execution directory

_optional_output_destination_paths(*self*, *paths*, *execution_dir*)
Returns a dictionary telling where optional output files should be copied to before execution.

Parameters

- **paths** (*dict*) – key is the optional file name pattern, value is a list of paths to source files
- **execution_dir** (*str*) – a path to the execution directory

Returns a map from source path to destination path

Return type dict

_output_resources_forward(*self*)
Returns a list of resources, i.e. the output files produced by the tool.

Returns the files that were actually created during the execution. The URL points to the archive directory.

Returns a list of Tool's output resources

Return type list

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
See base class.

spinetoolbox.project_items.tool.executable_item.**_count_files_and_dirs**(*paths*)
Counts the number of files and directories in given paths.

Parameters **paths** (*list*) – list of paths

Returns Tuple containing the number of required files and directories.

spinetoolbox.project_items.tool.executable_item.**_create_output_dir_timestamp**()
Creates a new timestamp string that is used as Tool output directory.

Returns Timestamp string or empty string if failed.

spinetoolbox.project_items.tool.executable_item.**_database_urls_from_resources**(*resources*)
Pries database URLs and their providers' names from resources.

Parameters **resources** (*list*) – a list of ProjectItemResource objects

Returns a mapping from resource provider's name to a database URL.

Return type dict

spinetoolbox.project_items.tool.executable_item.**_execution_directory**(*work_dir*, *tool_specification*)
Returns the path to the execution directory, depending on `execute_in_work`.

If `execute_in_work` is `True`, a new unique path will be returned. Otherwise, the main program file path from tool specification is returned.

Returns a full path to next basedir

Return type str

spinetoolbox.project_items.tool.executable_item.**_find_files_in_pattern**(*pattern*, *available_file_paths*)
Returns a list of files that match the given pattern.

Parameters

- **pattern** (*str*) – file pattern
- **available_file_paths** (*list*) – list of available file paths from upstream items

Returns List of (full) paths

Return type list

spinetoolbox.project_items.tool.executable_item.**_unique_dir_name**(*tool_specification*)
Builds a unique name for Tool's work directory.

class spinetoolbox.project_items.tool.executable_item.**_ExecutionToken**(*tool_executable*, *execution_dir*)
A token that acts as a callback after the tool process has finished execution.

Parameters

- **tool_executable** (spinetoolbox.project_items.tool.executable_item.ExecutableItem) – the object that has initiated the execution
- **execution_dir** (*str*) – absolute path to the execution working directory

handle_execution_finished(*self*, *return_code*)
Handles Tool specification execution finished.

Parameters **return_code** (*int*) – Process exit code

spinetoolbox.project_items.tool.item_info

Tool project item info.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

ItemInfo	

class spinetoolbox.project_items.tool.item_info.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.tool.specification_factory

Tool's specification factory.

authors

A. Soininen (VTT)

date 6.5.2020

Module Contents

Classes

SpecificationFactory	A factory to make tool specifications.

class spinetoolbox.project_items.tool.specification_factory.**SpecificationFactory**
Bases: *spinetoolbox.project_item_specification_factory.ProjectItemSpecificationFactory*

A factory to make tool specifications.

static item_type()
See base class.

static make_specification(*definition*, *definition_path*, *app_settings*, *logger*, *embedded_julia_console*, *embedded_python_console*)
Returns a tool specifications.

spinetoolbox.project_items.tool.tool

Tool class.

author

P. Savolainen (VTT)

date 19.12.2017

Module Contents

Classes

Tool	Class for project items that are not category nor root.

class spinetoolbox.project_items.tool.tool.**Tool**(*toolbox*, *project*, *logger*, *name*, *description*, *x*, *y*, *specification=''*, *execute_in_work=True*, *cmd_line_args=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Class for project items that are not category nor root. These items can be executed, refreshed, and so on.

x

horizontal position in the screen

Type float

y

vertical position in the screen

Type float

Tool class.

Parameters

- **toolbox** (*ToolboxUI*) – QMainWindow instance
- **project** (*SpineToolboxProject*) – the project this item belongs to
- **logger** (*LoggerInterface*) – a logger instance
- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **specification** (*str*) – Tool specification name
- **execute_in_work** (*bool*) – Execute associated Tool specification in work (True) or source directory (False)
- **cmd_line_args** (*list*) – Tool command line arguments

static item_type()

See base class.

static item_category()
See base class.

make_signal_handler_dict(*self*)
Returns a dictionary of all shared signals and their handlers. This is to enable simpler connecting and disconnecting.

restore_selections(*self*)
Restore selections into shared widgets when this project item is selected.

update_execution_mode(*self*, *checked*)
Pushes a new UpdateToolExecuteInWorkCommand to the toolbox stack.

do_update_execution_mode(*self*, *execute_in_work*)
Updates execute_in_work setting.

update_execute_in_work_button(*self*)
Sets the execute in work radio button check state according to execute_in_work instance variable.

update_specification(*self*, *text*)
Update Tool specification according to selection in the specification comboBox.

Parameters **text** (*str*) – Tool specification name in the comboBox

tool_args_text_edited(*self*, *txt*)
Calls the editingFinished slot when the line edit is cleared. Needed in order to clear the cmd line args list in case the line edit clear button is clicked.

Parameters **txt** (*str*) – Text in line edit after edit

tool_args_editing_finished(*self*)
Processed when the user has finished editing the cmd line args line edit. Pushes a new command to undo stack if the args were changed.

update_tool_cmd_line_args(*self*, *cmd_line_args*)
Updates instance cmd line args list and sets the list as text to the line edit widget.

Parameters **cmd_line_args** (*list*) – Tool cmd line args

do_set_specification(*self*, *specification*)
Sets Tool specification for this Tool. Removes Tool specification if None given as argument.

Parameters **specification** (ToolSpecification) – Tool specification of this Tool. None removes the specification.

undo_set_specification(*self*)

update_tool_ui(*self*)
Updates Tool UI to show Tool specification details. Used when Tool specification is changed. Overrides execution mode (work or source) with the specification default.

update_tool_models(*self*)
Update Tool models with Tool specification details. Used when Tool specification is changed. Overrides execution mode (work or source) with the specification default.

open_results(*self*, *checked=False*)
Open output directory in file browser.

edit_specification(*self*)
Open Tool specification editor for the Tool specification attached to this Tool.

open_specification_file(*self*)
Open Tool specification file.

open_main_program_file(*self*)
Open Tool specification main program file in an external text edit application.

open_main_directory(*self*)
Open directory where the Tool specification main program is located in file explorer.

specification(*self*)
Returns Tool specification.

populate_source_file_model(*self*, *items*)
Add required source files (includes) into a model. If items is None or an empty list, model is cleared.

populate_input_file_model(*self*, *items*)
Add required Tool input files into a model. If items is None or an empty list, model is cleared.

populate_opt_input_file_model(*self*, *items*)
Add optional Tool specification files into a model. If items is None or an empty list, model is cleared.

populate_output_file_model(*self*, *items*)
Add Tool output files into a model. If items is None or an empty list, model is cleared.

populate_specification_model(*self*, *populate*)
Add all tool specifications to a single QTreeView.

Parameters **populate** (*bool*) – False to clear model, True to populate.

update_name_label(*self*)
Update Tool tab name label. Used only when renaming project items.

resources_for_direct_successors(*self*)
Returns a list of resources, i.e. the outputs defined by the tool specification.

The output files are available only after tool has been executed, therefore the resource type is 'transient_file' or 'file_pattern'. A 'file_pattern' type resource is returned only if the pattern doesn't match any output file. For 'transient_file' resources, the url attribute is set to an empty string if the file doesn't exist yet or it points to a file from most recent execution. The metadata attribute's label key gives the base name or file pattern of the output file.

Returns a list of Tool's output resources

Return type list

execution_item(*self*)
Creates project item's execution counterpart.

_find_input_files(*self*, *resources*)
Iterates files in required input files model and looks for them in the given resources.

Parameters **resources** (*list*) – resources available

Returns Dictionary mapping required files to path where they are found, or to None if not found

_do_handle_dag_changed(*self*, *resources*)
See base class.

item_dict(*self*)
Returns a dictionary corresponding to this item.

custom_context_menu(*self*, *parent*, *pos*)
Returns the context menu for this item.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu

- **pos** (*QPoint*) – Position on screen

apply_context_menu_action(*self*, *parent*, *action*)
Applies given action from context menu. Implement in subclasses as needed.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu
- **action** (*str*) – The selected action

rename(*self*, *new_name*)
Rename this item.

Parameters **new_name** (*str*) – New name

Returns Boolean value depending on success

Return type bool

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

static _file_path_duplicates(*file_paths*)
Returns a list of lists of duplicate items in file_paths.

_notify_if_duplicate_file_paths(*self*, *duplicates*)
Adds a notification if duplicates contains items.

static upgrade_v1_to_v2(*item_name*, *item_dict*)
Upgrades item's dictionary from v1 to v2.

Changes: - 'tool' key is renamed to 'specification'

Parameters

- **item_name** (*str*) – item's name
- **item_dict** (*dict*) – Version 1 item dictionary

Returns Version 2 Tool dictionary

Return type dict

spinetoolbox.project_items.tool.tool_factory

The ToolFactory class.

author

M. Marin (KTH)

date 15.4.2020

Module Contents

Classes

ToolFactory	Class for project item factories.

class spinetoolbox.project_items.tool.tool_factory.**ToolFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static supports_specifications()
Returns whether or not this factory supports specs.

If the subclass implementation returns True, then it must also implement `specification_form_maker`, and `specification_menu_maker`.

Returns bool

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

spinetoolbox.project_items.tool.tool_icon

Module for tool icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

Classes

ToolIcon	Tool icon for the Design View.

class spinetoolbox.project_items.tool.tool_icon.**ToolIcon**(*toolbox*, *x*, *y*, *project_item*, *icon*)

Bases: *spinetoolbox.graphics_items.ProjectItemIcon*

Tool icon for the Design View.

Parameters

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate
- **project_item** (ProjectItem) – Item
- **icon** (*str*) – icon resource path

_handle_time_line_value_changed(*self*, *value*)

_handle_time_line_state_changed(*self*, *new_state*)

start_animation(*self*)
Starts the item execution animation.

stop_animation(*self*)
Stop animation

spinetoolbox.project_items.tool.tool_instance

Contains ToolInstance class.

authors

P. Savolainen (VTT), E. Rinne (VTT)

date 1.2.2018

Module Contents

Classes

ToolInstance	Tool instance base class.
GAMSToolInstance	Class for GAMS Tool instances.
JuliaToolInstance	Class for Julia Tool instances.
PythonToolInstance	Class for Python Tool instances.
ExecutableToolInstance	Class for Executable Tool instances.

class spinetoolbox.project_items.tool.tool_instance.**ToolInstance**(*tool_specification*, *basedir*, *settings*, *logger*)

Bases: PySide2.QtCore.QObject

Tool instance base class.

Parameters

- **tool_specification** (ToolSpecification) – the tool specification for this instance
- **basedir** (*str*) – the path to the directory where this instance should run
- **settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – a logger instance

instance_finished
Signal to emit when a Tool instance has finished processing

is_running(*self*)

terminate_instance(*self*)
Terminates Tool instance execution.

remove(*self*)
[Obsolete] Removes Tool instance files from work directory.

prepare(*self*, *optional_input_files*, *input_database_urls*, *output_database_urls*, *tool_args*)
Prepares this instance for execution.

Implement in subclasses to perform specific initialization.

Parameters

- **optional_input_files** (*list*) – list of tool's optional input files
- **input_database_urls** (*dict*) – a mapping from upstream Data Store name to database URL
- **output_database_urls** (*dict*) – a mapping from downstream Data Store name to database URL
- **tool_args** (*list*) – Tool cmd line args

execute(*self*, ***kwargs*)
Executes a prepared instance. Implement in subclasses.

handle_execution_finished(*self*, *ret*)
Handles execution finished.

Parameters **ret** (*int*) –

append_cmdline_args(*self*, *optional_input_files*, *input_database_urls*, *output_database_urls*, *tool_args*)
Appends Tool specification command line args into instance args list.

Parameters

- **optional_input_files** (*list*) – list of tool's optional input files
- **input_database_urls** (*dict*) – a mapping from upstream Data Store name to database URL

- **output_database_urls** (*dict*) – a mapping from downstream Data Store name to database URL
- **tool_args** (*list*) – List of Tool cmd line args

class spinetoolbox.project_items.tool.tool_instance.**GAMSToolInstance**(*tool_specification*, *basedir*, *settings*, *logger*)

Bases: *spinetoolbox.project_items.tool.tool_instance.ToolInstance*

Class for GAMS Tool instances.

Parameters

- **tool_specification** (ToolSpecification) – the tool specification for this instance
- **basedir** (*str*) – the path to the directory where this instance should run
- **settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – a logger instance

prepare(*self*, *optional_input_files*, *input_database_urls*, *output_database_urls*, *tool_args*)
See base class.

execute(*self*, ***kwargs*)
Executes a prepared instance.

handle_execution_finished(*self*, *ret*)
Handles execution finished.

Parameters **ret** (*int*) –

class spinetoolbox.project_items.tool.tool_instance.**JuliaToolInstance**(*tool_specification*, *basedir*, *settings*, *embedded_julia_console*, *logger*)

Bases: *spinetoolbox.project_items.tool.tool_instance.ToolInstance*

Class for Julia Tool instances.

Parameters

- **tool_specification** (ToolSpecification) – the tool specification for this instance
- **basedir** (*str*) – the path to the directory where this instance should run
- **settings** (*QSettings*) – Toolbox settings
- **embedded_julia_console** (*JuliaREPLWidget*) – a Julia console for execution in the embedded console
- **logger** (LoggerInterface) – a logger instance

prepare(*self*, *optional_input_files*, *input_database_urls*, *output_database_urls*, *tool_args*)
See base class.

execute(*self*, ***kwargs*)
Executes a prepared instance.

handle_repl_execution_finished(*self*, *ret*)
Handles repl-execution finished.

Parameters **ret** (*int*) – Tool specification process return value

handle_execution_finished(*self*, *ret*)
Handles execution finished.

Parameters **ret** (*int*) – Tool specification process return value

class spinetoolbox.project_items.tool.tool_instance.**PythonToolInstance**(*tool_specification*, *basedir*, *settings*, *embedded_python_console*, *logger*)

Bases: *spinetoolbox.project_items.tool.tool_instance.ToolInstance*

Class for Python Tool instances.

Parameters

- **tool_specification** (ToolSpecification) – the tool specification for this instance
- **basedir** (*str*) – the path to the directory where this instance should run
- **settings** (*QSettings*) – Toolbox settings
- **embedded_python_console** (*PythonReplWidget*) – a Python console widget for execution in embedded console
- **logger** (LoggerInterface) – A logger instance

prepare(*self*, *optional_input_files*, *input_database_urls*, *output_database_urls*, *tool_args*)
See base class.

execute(*self*, ***kwargs*)
Executes a prepared instance.

handle_console_execution_finished(*self*, *ret*)
Handles console-execution finished.

Parameters **ret** (*int*) – Tool specification process return value

handle_execution_finished(*self*, *ret*)
Handles execution finished.

Parameters **ret** (*int*) – Tool specification process return value

class spinetoolbox.project_items.tool.tool_instance.**ExecutableToolInstance**(*tool_specification*, *basedir*, *settings*, *logger*)

Bases: *spinetoolbox.project_items.tool.tool_instance.ToolInstance*

Class for Executable Tool instances.

Parameters

- **tool_specification** (ToolSpecification) – the tool specification for this instance
- **basedir** (*str*) – the path to the directory where this instance should run
- **settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – a logger instance

prepare(*self*, *optional_input_files*, *input_database_urls*, *output_database_urls*, *tool_args*)
See base class.

execute(*self*, ***kwargs*)
Executes a prepared instance.

handle_execution_finished(*self*, *ret*)
Handles execution finished.

Parameters **ret** (*int*) – Tool specification process return value

spinetoolbox.project_items.tool.tool_specifications

Contains Tool specification classes.

authors

P. Savolainen (VTT), E. Rinne (VTT), M. Marin (KTH)

date 24.1.2018

Module Contents

Classes

ToolSpecification	Super class for all tool specifications.
GAMSTool	Class for GAMS tool specifications.
JuliaTool	Class for Julia tool specifications.
PythonTool	Class for Python tool specifications.
ExecutableTool	Class for Executable tool specifications.

spinetoolbox.project_items.tool.tool_specifications.**TOOL_TYPES = ['Julia', 'Python', 'GAMS**

spinetoolbox.project_items.tool.tool_specifications.**REQUIRED_KEYS = ['name', 'tooltype', '**

spinetoolbox.project_items.tool.tool_specifications.**OPTIONAL_KEYS = ['description', 'short_**

spinetoolbox.project_items.tool.tool_specifications.**LIST_REQUIRED_KEYS = ['includes', 'inpu**

class spinetoolbox.project_items.tool.tool_specifications.**ToolSpecification**(*name*, *tooltype*, *path*, *includes*, *settings*, *logger*, *description=None*, *inputfiles=None*, *inputfiles_opt=None*, *outputfiles=None*, *cmdline_args=None*, *execute_in_work=True*)

Bases: *spinetoolbox.project_item_specification.ProjectItemSpecification*

Super class for all tool specifications.

Parameters

- **name** (*str*) – Tool specification name
- **tooltype** (*str*) – Type of Tool (e.g. Python, Julia, ..)
- **path** (*str*) – Path to Tool specification
- **includes** (*list*) – List of files belonging to the tool specification (relative to 'path')
- **settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – a logger instance
- **description** (*str*) – Description of the Tool specification
- **inputfiles** (*list*) – List of required data files
- **inputfiles_opt** (*list, optional*) – List of optional data files (wildcards may be used)
- **outputfiles** (*list, optional*) – List of output files (wildcards may be used)
- **cmdline_args** (*str, optional*) – Tool command line arguments (read from tool definition file)
- **execute_in_work** (*bool*) – Execute in work folder

save(*self*)

Saves this specification to a .json file in the definition path.

Returns How it went

Return type bool

is_equivalent (*self*, *definition*)
Checks if this spec is equivalent to the given definition dictionary. Used by the tool spec widget when updating specs.

Parameters **definition** (*dict*) –

Returns True if equivalent

Return type bool

set_return_code (*self*, *code*, *description*)
Sets a return code and an associated text description for the tool specification.

Parameters

- **code** (*int*) – Return code
- **description** (*str*) – Description

static check_definition (*data*, *logger*)
Checks that a tool specification contains the required keys and that it is in correct format.

Parameters

- **data** (*dict*) – Tool specification
- **logger** (LoggerInterface) – A logger instance

Returns Dictionary or None if there was a problem in the tool definition.

get_cmdline_args (*self*, *optional_input_files*, *input_urls*, *output_urls*)
Returns tool specification's command line args as list.

Replaces special tags in arguments:

- @@optional_inputs@@ expands to a space-separated list of Tool's optional input files
- @@url:<Data Store name>@@ expands to the URL provided by a named data store
- @@url_inputs@@ expands to a space-separated list of Tool's input database URLs
- @@url_outputs@@ expands to a space-separated list of Tool's output database URLs

Parameters

- **optional_input_files** (*list*) – a list of Tool's optional input file names
- **input_urls** (*dict*) – a mapping from URL provider (input Data Store name) to URL string
- **output_urls** (*dict*) – a mapping from URL provider (output Data Store name) to URL string

Returns a list of expanded command line arguments

Return type list

create_tool_instance (*self*, *basedir*)
Returns an instance of the tool specification configured to run in the given directory. Needs to be implemented in subclasses.

Parameters **basedir** (*str*) – Path to directory where the instance will run

static toolbox_load(*definition*, *definition_path*, *app_settings*, *logger*, *embedded_julia_console*, *embedded_python_console*)

Deserializes and constructs a tool specification from definition.

Parameters

- **definition** (*dict*) – a dictionary containing the serialized specification.
- **definition_path** (*str*) – path to the definition file
- **app_settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – a logger
- **embedded_julia_console** (*JuliaREPLWidget, optional*) – Julia console widget, required if a Julia tool is to be run in the console
- **embedded_python_console** (*PythonReplWidget, optional*) – Python console widget, required if a Python tool is to be run in the console

Returns

a tool specification constructed from the given definition, or None if there was an error

Return type *ToolSpecification*

open_main_program_file(*self*)

Open this specification's main program file in the default editor.

class spinetoolbox.project_items.tool.tool_specifications.**GAMSTool**(*name*, *tooltype*, *path*, *includes*, *settings*, *logger*, *description=None*, *input-files=None*, *input-files_opt=None*, *output-files=None*, *cmd-line_args=None*, *exe-cute_in_work=True*)

Bases: *spinetoolbox.project_items.tool.tool_specifications.ToolSpecification*

Class for GAMS tool specifications.

Parameters

- **name** (*str*) – GAMS Tool name
- **tooltype** (*str*) – Tool specification type
- **path** (*str*) – Path to model main file
- **includes** (*list*) – List of files belonging to the tool (relative to 'path'). # TODO: Change to src_files
- **settings** (*QSettings*) – Toolbox settings

- **logger** (LoggerInterface) – a logger instance
- **file in the list is the main GAMS program.** (*First*) –
- **description** (*str*) – GAMS Tool description
- **inputfiles** (*list*) – List of required data files
- **inputfiles_opt** (*list, optional*) – List of optional data files (wildcards may be used)
- **outputfiles** (*list, optional*) – List of output files (wildcards may be used)
- **cmdline_args** (*str, optional*) – GAMS tool command line arguments (read from tool definition file)

update_gams_options(*self*, *key*, *value*)
[OBSOLETE?] Updates GAMS command line options. Only 'cerr' and 'logoption' keywords supported.

Parameters

- **key** (*str*) – Option name
- **value** (*int, float*) – Option value

static load(*path*, *data*, *settings*, *logger*)
Creates a GAMSTool according to a tool specification file.

Parameters

- **path** (*str*) – Base path to tool files
- **data** (*dict*) – Dictionary of tool definitions
- **settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – A logger instance

Returns GAMSTool instance or None if there was a problem in the tool specification file.

create_tool_instance(*self*, *basedir*)
Returns an instance of this tool specification that is configured to run in the given directory.

Parameters **basedir** (*str*) – the path to the directory where the instance will run

class spinetoolbox.project_items.tool.tool_specifications.**JuliaTool**(*name*, *tooltype*, *path*, *includes*, *settings*, *embedded_julia_console*, *logger*, *description=None*, *inputfiles=None*, *inputfiles_opt=None*, *outputfiles=None*, *cmdline_args=None*, *execute_in_work=True*)

Bases: *spinetoolbox.project_items.tool.tool_specifications.ToolSpecification*

Class for Julia tool specifications.

Parameters

- **name** (*str*) – Julia Tool name
- **tooltype** (*str*) – Tool specification type
- **path** (*str*) – Path to model main file
- **includes** (*list*) – List of files belonging to the tool (relative to 'path'). # TODO: Change to src_files
- **file in the list is the main Julia program.** (*First*) –
- **settings** (*QSettings*) – Toolbox settings
- **embedded_julia_console** (*JuliaREPLWidget*) – a Julia console widget for execution in the embedded console
- **logger** (LoggerInterface) – A logger instance
- **description** (*str*) – Julia Tool description
- **inputfiles** (*list*) – List of required data files
- **inputfiles_opt** (*list, optional*) – List of optional data files (wildcards may be used)
- **outputfiles** (*list, optional*) – List of output files (wildcards may be used)
- **cmdline_args** (*str, optional*) – Julia tool command line arguments (read from tool definition file)

update_julia_options(*self*, *key*, *value*)

[OBSOLETE?] Updates Julia command line options.

Parameters

- **key** (*str*) – Option name
- **value** (*int, float*) – Option value

static load(*path*, *data*, *settings*, *embedded_julia_console*, *logger*)
Creates a JuliaTool according to a tool specification file.

Parameters

- **path** (*str*) – Base path to tool files
- **data** (*dict*) – Dictionary of tool definitions
- **settings** (*QSetting*) – Toolbox settings
- **embedded_julia_console** (*JuliaREPLWidget*) – a Julia console for execution in the embedded console
- **logger** (LoggerInterface) – A logger instance

Returns JuliaTool instance or None if there was a problem in the tool definition file.

create_tool_instance(*self*, *basedir*)
Returns an instance of this tool specification that is configured to run in the given directory.

Parameters **basedir** (*str*) – the path to the directory where the instance will run

class spinetoolbox.project_items.tool.tool_specifications.**PythonTool**(*name*, *tooltype*, *path*, *includes*, *settings*, *embedded_python_console*, *logger*, *description=None*, *input_files=None*, *input_files_opt=None*, *output_files=None*, *cmdline_args=None*, *execute_in_work=True*)
Bases: *spinetoolbox.project_items.tool.tool_specifications.ToolSpecification*

Class for Python tool specifications.

Parameters

- **name** (*str*) – Python Tool name
- **tooltype** (*str*) – Tool specification type
- **path** (*str*) – Path to model main file
- **includes** (*list*) – List of files belonging to the tool (relative to 'path'). # TODO: Change to src_files
- **settings** (*QSettings*) – Toolbox settings

- **embedded_python_console** (*PythonReplWidget*) – a Python console widget for embedded console execution
- **logger** (LoggerInterface) – A logger instance
- **file in the list is the main Python program.** (*First*) –
- **description** (*str*) – Python Tool description
- **inputfiles** (*list*) – List of required data files
- **inputfiles_opt** (*list, optional*) – List of optional data files (wildcards may be used)
- **outputfiles** (*list, optional*) – List of output files (wildcards may be used)
- **cmdline_args** (*str, optional*) – Python tool command line arguments (read from tool definition file)

update_python_options (*self*, *key*, *value*)
[OBSOLETE?] Updates Python command line options.

Parameters

- **key** (*str*) – Option name
- **value** (*int, float*) – Option value

static load (*path*, *data*, *settings*, *embedded_python_console*, *logger*)
Creates a PythonTool according to a tool specification file.

Parameters

- **path** (*str*) – Base path to tool files
- **data** (*dict*) – Dictionary of tool definitions
- **settings** (*QSettings*) – Toolbox settings
- **embedded_python_console** (*PythonReplWidget*) – Python console widget for execution in the embedded console
- **logger** (LoggerInterface) – A logger instance

Returns PythonTool instance or None if there was a problem in the tool definition file.

create_tool_instance (*self*, *basedir*)
Returns an instance of this tool specification that is configured to run in the given directory.

Parameters **basedir** (*str*) – the path to the directory where the instance will run

class spinetoolbox.project_items.tool.tool_specifications.**ExecutableTool**(*name*, *tooltype*, *path*, *includes*, *settings*, *logger*, *description=None*, *input_files=None*, *input_files_opt=None*, *output_files=None*, *cmdline_args=None*, *execute_in_work=True*)

Bases: *spinetoolbox.project_items.tool.tool_specifications.ToolSpecification*

Class for Executable tool specifications.

Parameters

- **name** (*str*) – Tool name
- **tooltype** (*str*) – Tool specification type
- **path** (*str*) – Path to main script file
- **includes** (*list*) – List of files belonging to the tool (relative to 'path'). # TODO: Change to src_files
- **file in the list is the main script file.** (*First*) –
- **settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – A logger instance
- **description** (*str*) – Tool description
- **inputfiles** (*list*) – List of required data files
- **inputfiles_opt** (*list, optional*) – List of optional data files (wildcards may be used)
- **outputfiles** (*list, optional*) – List of output files (wildcards may be used)
- **cmdline_args** (*str, optional*) – Tool command line arguments (read from tool definition file)

static load(*path*, *data*, *settings*, *logger*)

Creates an ExecutableTool according to a tool specification file.

Parameters

- **path** (*str*) – Base path to tool files
- **data** (*dict*) – Tool specification
- **settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – A logger instance

Returns ExecutableTool instance or None if there was a problem in the tool specification.

create_tool_instance(*self*, *basedir*)

Returns an instance of this tool specification that is configured to run in the given directory.

Parameters **basedir** (*str*) – the path to the directory where the instance will run

spinetoolbox.project_items.tool.utils

Utility functions for the Tool project item.

author

A. Soininen (VTT)

date 1.4.2020

Module Contents

Classes

_LatestOutputFile	A class to hold information on a latest output file.

Functions

flatten_file_path_duplicates(file_paths, logger, log_duplicates=False)	Flattens the extra duplicate dimension in file_paths.
file_paths_from_resources(resources)	Returns file paths from given resources.
find_file(filename, resources)	Returns all occurrences of full paths to given file name in resources available.
find_last_output_files(output_files, output_dir)	Returns latest output files.
is_pattern(file_name)	Returns True if file_name is actually a file pattern.

spinetoolbox.project_items.tool.utils.**flatten_file_path_duplicates**(*file_paths*, *logger*, *log_duplicates=False*)

Flattens the extra duplicate dimension in file_paths.

spinetoolbox.project_items.tool.utils.**file_paths_from_resources**(*resources*)

Returns file paths from given resources.

Parameters **resources** (*list*) – resources available

Returns a list of file paths, possibly including patterns

spinetoolbox.project_items.tool.utils.**find_file**(*filename*, *resources*)
Returns all occurrences of full paths to given file name in resources available.

Parameters

- **filename** (*str*) – Searched file name (no path)
- **resources** (*list*) – list of resources available from upstream items

Returns Full paths to file if found, None if not found

Return type list

spinetoolbox.project_items.tool.utils.**find_last_output_files**(*output_files*, *output_dir*)
Returns latest output files.

Parameters

- **output_files** (*list*) – output file patterns from tool specification
- **output_dir** (*str*) – path to the execution output directory

Returns a mapping from a file name pattern to the path of the most recent files in the results archive.

Return type dict

spinetoolbox.project_items.tool.utils.**is_pattern**(*file_name*)
Returns True if file_name is actually a file pattern.

class spinetoolbox.project_items.tool.utils.**_LatestOutputFile**(*label*, *path*)
A class to hold information on a latest output file.

label
file label, e.g. file pattern or relative path

Type str

path
absolute path to the file

Type str

Initialize self. See help(type(self)) for accurate signature.

static from_paths(*path*, *archive_dir*)
Constructs a _LatestOutputFile object from an absolute path and archive directory.

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.tool.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker

Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static supports_specifications()
Returns whether or not this factory supports specs.

If the subclass implementation returns True, then it must also implement `specification_form_maker`, and `specification_menu_maker`.

Returns bool

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.tool.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.view

View plugin.

author

M. Marin (KTH)

date 12.9.2019

Subpackages

spinetoolbox.project_items.view.widgets

Widgets for the View project item.

author A.Soininen (VTT)

date 27.9.2019

Submodules

spinetoolbox.project_items.view.widgets.add_view_widget

Widget shown to user when a new View is created.

author

P. Savolainen (VTT)

date 19.1.2017

Module Contents

Classes

AddViewWidget	A widget to query user's preferences for a new item.

class spinetoolbox.project_items.view.widgets.add_view_widget.**AddViewWidget**(*toolbox*, *x*, *y*, *spec=''*)

Bases: *spinetoolbox.widgets.add_project_item_widget.AddProjectItemWidget*

A widget to query user's preferences for a new item.

toolbox
Parent widget

Type *ToolboxUI*

x
X coordinate of new item

Type int

y
Y coordinate of new item

Type int

Initialize class.

call_add_item(*self*)
Creates new Item according to user's selections.

spinetoolbox.project_items.view.widgets.custom_menus

Classes for custom context menus and pop-up menus.

author

P. Savolainen (VTT)

date 9.1.2018

Module Contents

Classes

ViewPropertiesContextMenu	Context menu class for the references tree view of the View project item properties.

class spinetoolbox.project_items.view.widgets.custom_menus.**ViewPropertiesContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for the references tree view of the View project item properties.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

spinetoolbox.project_items.view.widgets.view_properties_widget

View properties widget.

author

M. Marin (KTH)

date 12.9.2019

Module Contents

Classes

ViewPropertiesWidget	Widget for the View Item Properties.

class spinetoolbox.project_items.view.widgets.view_properties_widget.**ViewPropertiesWidget**(*t*

Bases: PySide2.QtWidgets.QWidget

Widget for the View Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embedded

Init class.

connect_signals(*self*)
Connect signals to slots.

show_view_properties_context_menu(*self*, *pos*)
Create and show a context-menu in View properties.

Parameters **pos** (*QPoint*) – Mouse position

Submodules

spinetoolbox.project_items.view.executable_item

Contains View's executable item as well as support utilities.

authors

A. Soininen (VTT)

date 2.4.2020

Module Contents

Classes

ExecutableItem	The part of a project item that is executed by the Spine Engine.

class spinetoolbox.project_items.view.executable_item.**ExecutableItem**(*name*, *logger*)

Bases: *spinetoolbox.executable_item_base.ExecutableItemBase*

The part of a project item that is executed by the Spine Engine.

Parameters

- **name** (*str*) – item's name
- **logger** (LoggerInterface) – a logger

static item_type()
Returns View's type identifier string.

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
See base class.

spinetoolbox.project_items.view.item_info

View project item info.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

ItemInfo	

class spinetoolbox.project_items.view.item_info.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.project_items.view.view

Module for view class.

authors

P. Savolainen (VTT), M. Marin (KHT), J. Olauson (KTH)

date 14.07.2018

Module Contents

Classes

View	Class for project items that are not category nor root.

class spinetoolbox.project_items.view.view.**View**(*toolbox*, *project*, *logger*, *name*, *description*, *x*, *y*)
Bases: *spinetoolbox.project_item.ProjectItem*

Class for project items that are not category nor root. These items can be executed, refreshed, and so on.

x
horizontal position in the screen

Type float

y
vertical position in the screen

Type float

View class.

Parameters

- **toolbox** (ToolboxUI) – a toolbox instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon

static item_type()
See base class.

static item_category()
See base class.

execution_item(*self*)
Creates project item's execution counterpart.

make_signal_handler_dict(*self*)
Returns a dictionary of all shared signals and their handlers. This is to enable simpler connecting and disconnecting.

restore_selections(*self*)
Restore selections into shared widgets when this project item is selected.

save_selections(*self*)
Save selections in shared widgets for this project item into instance variables.

open_db_editor(*self*, *checked=False*)
Opens selected db in the Spine database editor.

populate_reference_list(*self*)
Populates reference list.

update_name_label(*self*)
Update View tab name label. Used only when renaming project items.

_do_handle_dag_changed(*self*, *resources*)
Update the list of references that this item is viewing.

_update_references_list(*self*, *resources_upstream*)
Updates the references list with resources upstream.

Parameters **resources_upstream** (*list*) – ProjectItemResource instances

_selected_indexes(*self*)
Returns selected indexes.

_db_url_codenames(*self*, *indexes*)
Returns a dict mapping url to provider's name for given indexes in the reference model.

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

spinetoolbox.project_items.view.view_factory

The ViewFactory class.

author

M. Marin (KTH)

date 15.4.2020

Module Contents

Classes

ViewFactory	Class for project item factories.

class spinetoolbox.project_items.view.view_factory.**ViewFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

spinetoolbox.project_items.view.view_icon

Module for view icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

Classes

ViewIcon	View icon for the Design View.

class spinetoolbox.project_items.view.view_icon.**ViewIcon**(*toolbox*, *x*, *y*, *project_item*, *icon*)

Bases: *spinetoolbox.graphics_items.ProjectItemIcon*

View icon for the Design View.

Parameters

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate
- **project_item** (ProjectItem) – Item
- **icon** (*str*) – icon resource path

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.view.**ItemFactory**(*toolbox*)

Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker

Returns a ProjectItem subclass.

Returns class

icon_maker

Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.view.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

Package Contents

Classes

ItemFactory	Class for project item factories.
ItemInfo	
ItemFactory	Class for project item factories.
ItemInfo	
ItemFactory	Class for project item factories.
ItemInfo	
ItemFactory	Class for project item factories.
ItemInfo	
ItemFactory	Class for project item factories.
ItemInfo	
ItemFactory	Class for project item factories.
ItemInfo	
ItemFactory	Class for project item factories.
ItemInfo	
ItemFactory	Class for project item factories.
ItemInfo	

class spinetoolbox.project_items.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

class spinetoolbox.project_items.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class.

class spinetoolbox.project_items.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

class spinetoolbox.project_items.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class

class spinetoolbox.project_items.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

class spinetoolbox.project_items.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
See base class.

class spinetoolbox.project_items.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

class spinetoolbox.project_items.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

properties_widget_maker

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static item_type()

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

class spinetoolbox.project_items.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

class spinetoolbox.project_items.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static supports_specifications()
Returns whether or not this factory supports specs.

If the subclass implementation returns True, then it must also implement `specification_form_maker`, and `specification_menu_maker`.

Returns bool

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

class spinetoolbox.project_items.**ItemFactory**(*toolbox*)
Bases: *spinetoolbox.project_item.ProjectItemFactory*

Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

class spinetoolbox.project_items.**ItemInfo**
Bases: *spinetoolbox.project_item_info.ProjectItemInfo*

static item_category()
See base class.

static item_type()
See base class.

spinetoolbox.spine_db_editor

This subpackage contains GUI files for the Spine db editor.

authors

M. Marin (KTH)

date 13.5.2020

Subpackages

spinetoolbox.spine_db_editor.mvcmodels

Modules in this package contain classes that represent Spine Toolbox's models (internal data structures) in the Model-View-Controller design pattern. The model classes define an interface that is used by views and delegates to access data in the application.

author

M. Marin (KTH)

date 23.5.2020

Submodules

spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item

Classes to represent alternative and scenario items in a tree.

authors

P. Vennström (VTT)

date 17.6.2020

Module Contents

Classes

RootItem	A root item.

Continued on next page

Table 154 – continued from previous page

LeafItem	A tree item that fetches their children as they are inserted.
AlternativeRootItem	An alternative root item.
ScenarioRootItem	A scenario root item.
AlternativeLeafItem	An alternative leaf item.
ScenarioLeafItem	A scenario leaf item.
ScenarioAlternativeLeafItem	A scenario alternative leaf item.

spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**_ALTERNATIVE_ICON =**

spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**_SCENARIO_ICON =**

class spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**RootItem**(*model=None*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.EmptyChildMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.AllBoldMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.NonLazyTreeItem*

A root item.

Initializes item.

Parameters **model** (*MinimalTreeModel, NoneType*) – The model where the item belongs.

item_type

display_data

icon_code

db_map

display_icon

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

set_data(*self*, *column*, *value*, *role*)
Sets data for this item.

Parameters

- **column** (*int*) – column index
- **value** (*object*) – a new value
- **role** (*int*) – role of the new value

Returns True if data was set successfully, False otherwise

Return type bool

empty_child(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**LeafItem**(*identifier=None*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.NonLazyTreeItem*

A tree item that fetches their children as they are inserted.

Initializes item.

Parameters **model** (*MinimalTreeModel, NoneType*) – The model where the item belongs.

item_type

tool_tip

db_map

id

item_data

name

add_item_to_db(*self*, *db_item*)

update_item_in_db(*self*, *db_item*)

header_data(*self*, *column*)

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

set_data(*self*, *column*, *value*, *role*)
Sets data for this item.

Parameters

- **column** (*int*) – column index
- **value** (*object*) – a new value
- **role** (*int*) – role of the new value

Returns True if data was set successfully, False otherwise

Return type bool

handle_updated_in_db(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**AlternativeRootItem**
Bases: *spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.RootItem*

An alternative root item.

Initializes item.

Parameters **model** (MinimalTreeModel, *NoneType*) – The model where the item belongs.

item_type

display_data

icon_code

empty_child(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**ScenarioRootItem**(*mo*
Bases: *spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.RootItem*

A scenario root item.

Initializes item.

Parameters **model** (MinimalTreeModel, *NoneType*) – The model where the item belongs.

item_type

display_data

icon_code

empty_child(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**AlternativeLeafItem**
Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.LastGrayMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.EditableMixin*, *spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.LeafItem*

An alternative leaf item.

Initializes item.

Parameters **model** (*MinimalTreeModel, NoneType*) – The model where the item belongs.

item_type

tool_tip

add_item_to_db(*self*, *db_item*)

update_item_in_db(*self*, *db_item*)

flags(*self*, *column*)
Makes items editable.

class spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**ScenarioLeafItem**(*ide*
Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.LastGrayMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.EditableMixin*, *spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.LeafItem*

A scenario leaf item.

Initializes item.

Parameters **model** (*MinimalTreeModel, NoneType*) – The model where the item belongs.

item_type

tool_tip

alternative_id_list

add_item_to_db(*self*, *db_item*)

update_item_in_db(*self*, *db_item*)

flags(*self*, *column*)
Makes items editable.

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

set_data(*self*, *column*, *value*, *role*)
Sets data for this item.

Parameters

- **column** (*int*) – column index
- **value** (*object*) – a new value
- **role** (*int*) – role of the new value

Returns True if data was set successfully, False otherwise

Return type bool

fetch_more(*self*)
Fetches more children.

handle_updated_in_db(*self*)

_update_alternative_id_list(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.**ScenarioAlternativeI**
Bases: *spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_item.LeafItem*

A scenario alternative leaf item.

Initializes item.

Parameters **model** (*MinimalTreeModel, NoneType*) – The model where the item belongs.

item_type

tool_tip

id

add_item_to_db(*self*, *db_item*)

update_item_in_db(*self*, *db_item*)

flags(*self*, *column*)
Enables the item and makes it selectable.

spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_model

Models to represent alternatives, scenarios and scenario alternatives in a tree.

authors

P. Vennström (VTT), M. Marin (KTH)

date 17.6.2020

Module Contents

Classes

AlternativeScenarioModel	A model to display parameter_value_list data in a tree view.

class spinetoolbox.spine_db_editor.mvcmodels.alternative_scenario_model.**AlternativeScenarioM**

Bases: *spinetoolbox.mvcmodels.minimal_tree_model.MinimalTreeModel*

A model to display parameter_value_list data in a tree view.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- **db_maps** (*iter*) – DiffDatabaseMapping instances

Initialize class

columnCount(*self*, *parent=QModelIndex()*)
Returns the number of columns under the given parent. Always 1.

headerData(*self*, *section*, *orientation*, *role=Qt.DisplayRole*)

build_tree(*self*)
Builds tree.

_add_leaves(*self*, *db_map_data*, *leaf_type*)

_update_leaves(*self*, *db_map_data*, *leaf_type*)

_remove_leaves(*self*, *db_map_data*, *leaf_type*)

add_alternatives(*self*, *db_map_data*)

add_scenarios(*self*, *db_map_data*)

update_alternatives(*self*, *db_map_data*)

update_scenarios(*self*, *db_map_data*)

remove_alternatives(*self*, *db_map_data*)

remove_scenarios(*self*, *db_map_data*)

static db_item(*item*)

db_row(*self*, *item*)

supportedDropActions(*self*)

mimeData(*self*, *indexes*)
Builds a dict mapping db name to item type to a list of ids.

Returns QMimeData

canDropMimeData(*self*, *data*, *drop_action*, *row*, *column*, *parent*)

dropMimeData(*self*, *data*, *drop_action*, *row*, *column*, *parent*)

spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models

Compound models for object parameter definitions and values. These models concatenate several 'single' models and one 'empty' model.

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

Classes

CompoundParameterModel	A model that concatenates several single parameter models

Continued on next page

Table 156 – continued from previous page

CompoundObjectParameterMixin	Implements the interface for populating and filtering a compound object parameter model.
CompoundRelationshipParameterMixin	Implements the interface for populating and filtering a compound relationship parameter model.
CompoundParameterDefinitionMixin	Handles signals from db mngr for parameter_definition models.
CompoundParameterValueMixin	Handles signals from db mngr for parameter_value models.
CompoundObjectParameterDefinitionModel	A model that concatenates several single object parameter_definition models
CompoundRelationshipParameterDefinitionModel	A model that concatenates several single relationship parameter_definition models
CompoundObjectParameterValueModel	A model that concatenates several single object parameter_value models
CompoundRelationshipParameterValueModel	A model that concatenates several single relationship parameter_value models

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundParameterMod**

Bases: *spinetoolbox.mvcmodels.compound_table_model.CompoundWithEmptyTableModel*

A model that concatenates several single parameter models and one empty parameter model.

Initializes model.

Parameters

- **parent** (SpineDBEditor) – the parent object
- **db_mngr** (SpineDBManager) – the database manager
- ***db_maps** (*DiffDatabaseMapping*) – the database maps included in the model

data_for_single_model_received
Emitted by the fetcher when there's data for another single model.

entity_class_type
Returns the entity_class type, either 'object_class' or 'relationship_class'.

Returns str

item_type
Returns the parameter item type, either 'parameter_definition' or 'parameter_value'.

Returns str

_single_model_type
Returns a constructor for the single models.

Returns SingleParameterModel

_empty_model_type
Returns a constructor for the empty model.

Returns EmptyParameterModel

entity_class_id_key
Returns the key corresponding to the entity_class id (either "object_class_id" or "relationship_class_id")

Returns str

parameter_definition_id_key

_make_header(*self*)

init_model(*self*)
Initializes the model.

_make_auto_filter_menus(*self*)
Makes auto filter menus.

get_auto_filter_menu(*self*, *logical_index*)
Returns auto filter menu for given logical index from header view.

Parameters **logical_index** (*int*) –

Returns ParameterViewFilterMenu

_modify_data_in_filter_menus(*self*, *action*, *db_map*, *db_items*)
Modifies data in filter menus.

Parameters

- **action** (*str*) – either 'add', 'remove', or 'update'
- **db_map** (*DiffDatabaseMapping*) –
- **db_items** (*list(dict)*) –

_do_add_data_to_filter_menus(*self*, *db_map*, *db_items*)

_do_update_data_in_filter_menus(*self*, *db_map*, *db_items*)

_do_remove_data_from_filter_menus(*self*, *db_map*, *db_items*)

headerData(*self*, *section*, *orientation=Qt.Horizontal*, *role=Qt.DisplayRole*)
Returns an italic font in case the given column has an autofilter installed.

_create_single_models(*self*)
Returns a list of single models for this compound model, one for each entity_class in each database.

Returns list

_create_empty_model(*self*)
Returns the empty model for this compound model.

Returns EmptyParameterModel

filter_accepts_model(*self*, *model*)
Returns a boolean indicating whether or not the given model passes the filter for compound model.

Parameters **model** (SingleParameterModel, EmptyParameterModel) –

Returns bool

_class_filter_accepts_model(*self*, *model*)

_auto_filter_accepts_model(*self*, *model*)

accepted_single_models(*self*)
Returns a list of accepted single models by calling filter_accepts_model on each of them, just for convenience.

Returns list

static _settattr_if_different(*obj*, *attr*, *val*)
Sets the given attribute of the given object to the given value if it's different from the one currently stored. Used for updating filters.

Returns True if the attributed was set, False otherwise

Return type bool

_invalidate_filter(*self*)
Sets the filter invalid.

_refresh_if_still_invalid(*self*)

set_filter_class_ids(*self*, *class_ids*)

set_filter_parameter_ids(*self*, *parameter_ids*)

set_auto_filter(*self*, *field*, *auto_filter*)
Updates and applies the auto filter.

Parameters

- **field** (*str*) – the field name
- **auto_filter** (*dict*) – mapping db_map to entity_class id to accepted values for the field

set_compound_auto_filter(*self*, *field*, *auto_filter*)
Sets the auto filter for given column in the compound model.

Parameters

- **field** (*str*) – the field name
- **auto_filter** (*dict*) – maps tuple (database map, entity_class id) to list of accepted ids for the field

set_single_auto_filter(*self*, *model*, *field*)
Sets the auto filter for given column in the given single model.

Parameters

- **model** (SingleParameterModel) – the model
- **field** (*str*) – the field name

Returns True if the auto-filtered values were updated, None otherwise

Return type bool

_row_map_for_model(*self*, *model*)
Returns the row map for the given model. Reimplemented to take filter status into account.

Parameters **model** (SingleParameterModel, EmptyParameterModel) –

Returns tuples (model, row number) for each accepted row

Return type list

_models_with_db_map(*self*, *db_map*)
Returns a collection of single models with given db_map.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns list

receive_entity_classes_removed(*self*, *db_map_data*)
Runs when entity classes are removed from the dbs. Removes sub-models for the given entity classes and dbs.

Parameters **db_map_data** (*dict*) – list of removed dict-items keyed by DiffDatabaseMapping

_items_per_class(*self*, *items*)
Returns a dict mapping entity_class ids to a set of items.

Parameters **items** (*list*) –

Returns dict

receive_parameter_data_added(*self*, *db_map_data*)
Runs when either parameter definitions or values are added to the dbs. Adds necessary sub-models and initializes them with data. Also notifies the empty model so it can remove rows that are already in.

Parameters **db_map_data** (*dict*) – list of removed dict-items keyed by DiffDatabaseMapping

create_and_append_single_model(*self*, *db_map*, *entity_class_id*, *ids*)

receive_parameter_data_updated(*self*, *db_map_data*)
Runs when either parameter definitions or values are updated in the dbs. Emits dataChanged so the parameter_name column is refreshed.

Parameters **db_map_data** (*dict*) – list of updated dict-items keyed by DiffDatabaseMapping

receive_parameter_data_removed(*self*, *db_map_data*)
Runs when either parameter definitions or values are removed from the dbs. Removes the affected rows from the corresponding single models.

Parameters **db_map_data** (*dict*) – list of removed dict-items keyed by DiffDatabaseMapping

_emit_data_changed_for_column(*self*, *field*)
Lazily emits data changed for an entire column.

Parameters **field** (*str*) – the column header

db_item(*self*, *index*)

index_name(*self*, *index*)

get_set_data_delayed(*self*, *index*)
Returns a function that ParameterValueEditor can call to set data for the given index at any later time, even if the model changes.

Parameters **index** (*QModelIndex*) –

Returns function

get_entity_class_id(*self*, *index*, *db_map*)

filter_by(*self*, *rows_per_column*)

filter_excluding(*self*, *rows_per_column*)

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundObjectParame**
Implements the interface for populating and filtering a compound object parameter model.

entity_class_type

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundRelationshi**
Implements the interface for populating and filtering a compound relationship parameter model.

entity_class_type

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundParameterDe**
Handles signals from db mngr for parameter_definition models.

item_type

receive_parameter_definition_tags_set(*self*, *db_map_data*)

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundParameterVal**
Handles signals from db mngr for parameter_value models.

item_type

entity_type
Returns the entity type, either 'object' or 'relationship' Used by update_single_main_filter.

Returns str

set_filter_entity_ids(*self*, *entity_ids*)

set_filter_alternative_ids(*self*, *alternative_ids*)

receive_alternatives_updated(*self*, *db_map_data*)
Updated alternative column

Parameters **db_map_data** (*dict*) – list of updated dict-items keyed by DiffDatabaseMapping

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundObjectParame**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundObjectParameterMixin*, *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundParameterDefinitionMixin*, *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundParameterModel*

A model that concatenates several single object parameter_definition models and one empty object parameter_definition model.

Initializes model.

Parameters

- **parent** (SpineDBEditor) – the parent object
- **db_mngr** (SpineDBManager) – the database manager
- ***db_maps** (*DiffDatabaseMapping*) – the database maps included in the model

_make_header(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundRelationshi**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundRelationshipParameterMixin*, *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundParameterDefinitionMixin*, *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundParameterModel*

A model that concatenates several single relationship parameter_definition models and one empty relationship parameter_definition model.

Initializes model.

Parameters

- **parent** (SpineDBEditor) – the parent object
- **db_mngr** (SpineDBManager) – the database manager

- ***db_maps** (*DiffDatabaseMapping*) – the database maps included in the model

_make_header(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundObjectParame**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundObjectParameterMixin*, *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundParameterValueMixin*, *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundParameterModel*

A model that concatenates several single object parameter_value models and one empty object parameter_value model.

Initializes model.

Parameters

- **parent** (SpineDBEditor) – the parent object
- **db_mngr** (SpineDBManager) – the database manager
- ***db_maps** (*DiffDatabaseMapping*) – the database maps included in the model

entity_type
Returns the entity type, either 'object' or 'relationship' Used by update_single_main_filter.

Returns str

_make_header(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.**CompoundRelationshi**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundRelationshipParameterMixin*, *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundParameterValueMixin*, *spinetoolbox.spine_db_editor.mvcmodels.compound_parameter_models.CompoundParameterModel*

A model that concatenates several single relationship parameter_value models and one empty relationship parameter_value model.

Initializes model.

Parameters

- **parent** (SpineDBEditor) – the parent object
- **db_mngr** (SpineDBManager) – the database manager
- ***db_maps** (*DiffDatabaseMapping*) – the database maps included in the model

entity_type
Returns the entity type, either 'object' or 'relationship' Used by update_single_main_filter.

Returns str

_make_header(*self*)

spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models

Empty models for parameter definitions and values.

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

Classes

EmptyParameterModel	An empty parameter model.
EmptyParameterDefinitionModel	An empty parameter_definition model.
EmptyObjectParameterDefinitionModel	An empty object parameter_definition model.
EmptyRelationshipParameterDefinitionModel	An empty relationship parameter_definition model.
EmptyParameterValueModel	An empty parameter_value model.
EmptyObjectParameterValueModel	An empty object parameter_value model.
EmptyRelationshipParameterValueModel	An empty relationship parameter_value model.

class spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.**EmptyParameterModel**(*par*
hea
*db_*

Bases: *spinetoolbox.mvcmodels.empty_row_model.EmptyRowModel*

An empty parameter model.

Initialize class.

Parameters

- **parent** (*Object*) – the parent object, typically a CompoundParameterModel
- **header** (*list*) – list of field names for the header
- **db_mngr** (SpineDBManager) –

item_type
The item type, either 'parameter_value' or 'parameter_definition', required by the json_fields property.

entity_class_type
Either 'object_class' or 'relationship_class'.

entity_class_id_key

entity_class_name_key

can_be_filtered

json_fields

accepted_rows(*self*)

db_item(*self*, *_index*)

item_id(*self*, *_row*)

flags(*self*, *index*)
Return default flags except if forcing defaults.

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the item referred to by the index.

Parameters

- **index** (*QModelIndex*) – Index of item
- **role** (*int*) – Data role

Returns Item data for given role.

_make_unique_id (*self*, *item*)
Returns a unique id for the given model item (name-based). Used by receive_parameter_data_added.

receive_parameter_data_added (*self*, *db_map_data*)
Runs when parameter definitions or values are added. Finds and removes model items that were successfully added to the db.

batch_set_data (*self*, *indexes*, *data*)
Sets data for indexes in batch. If successful, add items to db.

add_items_to_db (*self*, *rows*)
Add items to db.

Parameters **rows** (*set*) – add data from these rows

_make_db_map_data (*self*, *rows*)
Returns model data grouped by database map.

Parameters **rows** (*set*) – group data from these rows

class spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.**EmptyParameterDefinitio**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInValueListIdMixin*, *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInEntityClassIdMixin*, *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInParameterNameMixin*, *spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.EmptyParameterModel*

An empty parameter_definition model.

Initializes lookup dicts.

item_type
The item type, either 'parameter_value' or 'parameter_definition', required by the json_fields property.

entity_class_type
See base class.

add_items_to_db (*self*, *rows*)
Add items to db.

Parameters **rows** (*set*) – add data from these rows

_check_item (*self*, *item*)
Checks if a db item is ready to be inserted.

class spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.**EmptyObjectParameterDe**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.EmptyParameterDefinitionModel*

An empty object parameter_definition model.

Initializes lookup dicts.

entity_class_type
See base class.

class spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.**EmptyRelationshipParame**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.EmptyParameterDefinitionModel*

An empty relationship parameter_definition model.

Initializes lookup dicts.

entity_class_type
See base class.

flags(*self*, *index*)
Additional hack to make the object_class_name_list column non-editable.

class spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.**EmptyParameterValueMode**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.InferEntityClassIdMixin*, *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInAlternativeIdMixin*, *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInParameterDefinitionIdsMixin*, *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInEntityIdsMixin*, *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInEntityClassIdMixin*, *spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.EmptyParameterModel*

An empty parameter_value model.

Initializes lookup dicts.

item_type
The item type, either 'parameter_value' or 'parameter_definition', required by the json_fields property.

entity_type
Either 'object' or "relationship'.

entity_id_key

entity_name_key

entity_name_key_in_cache

_make_unique_id(*self*, *item*)
Returns a unique id for the given model item (name-based). Used by receive_parameter_data_added.

add_items_to_db(*self*, *rows*)
Add items to db.

Parameters **rows** (*set*) – add data from these rows

_check_item(*self*, *item*)
Checks if a db item is ready to be inserted.

class spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.**EmptyObjectParameterVa**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.EmptyParameterValueModel*

An empty object parameter_value model.

Initializes lookup dicts.

entity_class_type
Either 'object_class' or 'relationship_class'.

entity_type
Either 'object' or "relationship'.

class spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.**EmptyRelationshipParame**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.MakeRelationshipOnTheFlyMixin*, *spinetoolbox.spine_db_editor.mvcmodels.empty_parameter_models.EmptyParameterValueModel*

An empty relationship parameter_value model.

Initializes lookup dicts.

_add_entities_on_the_fly = True

entity_class_type
Either 'object_class' or 'relationship_class'.

entity_type
Either 'object' or "relationship'.

add_items_to_db(*self*, *rows*)
Add items to db.

Parameters **rows** (*set*) – add data from these rows

spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item

Classes to represent entities in a tree.

authors

P. Vennström (VTT), M. Marin (KTH)

date 11.3.2019

Module Contents

Classes

EntityRootItem	A tree item that may belong in multiple databases.
ObjectTreeRootItem	An object tree root item.
RelationshipTreeRootItem	A relationship tree root item.
EntityClassItem	An entity_class item.
ObjectClassItem	An object_class item.
RelationshipClassItem	A relationship_class item.
EntityItem	An entity item.
ObjectItem	An object item.
RelationshipItem	A relationship item.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**EntityRootItem**(*model=None*, *db_map_id=None*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_item.MultiDBTreeItem*

A tree item that may belong in multiple databases.

Init class.

Parameters

- **db_mngr** (SpineDBManager) – a database manager
- **db_map_data** (*dict*) – maps instances of DiffDatabaseMapping to the id of the item in that db

item_type = root

display_id
"See super class.

display_icon
Returns an icon to display next to the name. Reimplement in subclasses to return something nice.

display_data
"See super class.

_get_children_ids(*self*, *db_map*)
See super class.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**ObjectTreeRootItem**(*model=None*, *db_map_id=N*

Bases: *spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.EntityRootItem*

An object tree root item.

Init class.

Parameters

- **db_mngr** (SpineDBManager) – a database manager
- **db_map_data** (*dict*) – maps instances of DiffDatabaseMapping to the id of the item in that db

item_type = root

child_item_type
Returns ObjectClassItem.

set_data(*self*, *column*, *value*, *role*)
See base class.

_get_children_ids(*self*, *db_map*)
Returns a list of object_class ids.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**RelationshipTreeRootItem**(*mode*
db_m

Bases: *spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.EntityRootItem*

A relationship tree root item.

Init class.

Parameters

- **db_mngr** (SpineDBManager) – a database manager
- **db_map_data** (*dict*) – maps instances of DiffDatabaseMapping to the id of the item in that db

item_type = root

child_item_type
Returns RelationshipClassItem.

set_data(*self*, *column*, *value*, *role*)
See base class.

_get_children_ids(*self*, *db_map*)
Returns a list of object_class ids.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**EntityClassItem**(**args*, ***kwargs*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_item.MultiDBTreeItem*

An entity_class item.

Overridden method to declare group_child_count attribute.

display_icon
Returns class icon.

_display_icon(*self*, *for_group=False*)

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

_get_children_ids(*self*, *db_map*)
See base class

fetch_more(*self*)
Fetches children from all associated databases and raises group children.

raise_group_children_by_id(*self*, *db_map_ids*)
Moves group children to the top of the list.

Parameters **db_map_ids** (*dict*) – set of ids corresponding to newly inserted group children, keyed by DiffDatabaseMapping

_raise_group_children_by_row(*self*, *rows*)
Moves group children to the top of the list.

Parameters **rows** (*set, list*) – collection of rows corresponding to newly inserted group children

remove_children(*self*, *position*, *count*)
Overriden method to keep the group child count up to date.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**ObjectClassItem**(**args*, ***kwargs*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.EntityClassItem*

An object_class item.

Overridden method to declare group_child_count attribute.

item_type = object_class

child_item_type
Returns ObjectItem.

default_parameter_data(*self*)
Return data to put as default in a parameter table when this item is selected.

set_data (*self*, *column*, *value*, *role*)
See base class.

_get_children_ids (*self*, *db_map*)
Returns a list of object_class ids.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**RelationshipClassItem**(**args*, ***kwargs*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.EntityClassItem*

A relationship_class item.

Overridden method to declare group_child_count attribute.

visual_key = ['name', 'object_class_name_list']

item_type = relationship_class

child_item_type
Returns RelationshipItem.

default_parameter_data (*self*)
Return data to put as default in a parameter table when this item is selected.

set_data (*self*, *column*, *value*, *role*)
See base class.

_get_children_ids (*self*, *db_map*)
Returns a list of object_class ids.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**EntityItem**(*model=None*, *db_map_id=None*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_item.MultiDBTreeItem*

An entity item.

Init class.

Parameters

- **db_mngr** (SpineDBManager) – a database manager
- **db_map_data** (*dict*) – maps instances of DiffDatabaseMapping to the id of the item in that db

display_icon
Returns corresponding class icon.

member_ids

member_rows

db_map_member_ids (*self*, *db_map*)

db_map_entity_groups (*self*, *db_map*)

is_group (*self*)

data (*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

_get_children_ids (*self*, *db_map*)
See base class.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**ObjectItem**(*model=None*, *db_map_id=None*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.EntityItem*

An object item.

Init class.

Parameters

- **db_mngr** (SpineDBManager) – a database manager
- **db_map_data** (*dict*) – maps instances of DiffDatabaseMapping to the id of the item in that db

item_type = object

child_item_type
Returns RelationshipClassItem.

default_parameter_data(*self*)
Return data to put as default in a parameter table when this item is selected.

set_data(*self*, *column*, *value*, *role*)
See base class.

_get_children_ids(*self*, *db_map*)
See base class.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.**RelationshipItem**(**args*, ***kwargs*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.entity_tree_item.EntityItem*

A relationship item.

Overridden method to make sure we never try to fetch this item.

visual_key = ['name', 'object_name_list']

item_type = relationship

object_name_list

display_data
"Returns the name for display.

edit_data

has_children(*self*)
Returns false, this item never has children.

default_parameter_data(*self*)
Return data to put as default in a parameter table when this item is selected.

can_fetch_more(*self*)
Returns whether or not this item can fetch more.

_get_children_ids(*self*, *db_map*)
See base class

is_valid(*self*)
Checks that the grand parent object is still in the relationship.

spinetoolbox.spine_db_editor.mvcmodels.entity_tree_models

Models to represent entities in a tree.

authors

P. Vennström (VTT), M. Marin (KTH)

date 11.3.2019

Module Contents

Classes

ObjectTreeModel	An 'object-oriented' tree model.
RelationshipTreeModel	A relationship-oriented tree model.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_models.**ObjectTreeModel**(*parent*, *db_mngr*, **db_maps*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_model.MultiDBTreeModel*

An 'object-oriented' tree model.

Init class.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) – A manager for the given db_maps
- **db_maps** (*iter*) – DiffDatabaseMapping instances

root_item_type

Implement in subclasses to create a model specific to any entity type.

_parent_object_data(*self*, *db_map_data*)

Takes given object data and returns the same data keyed by parent tree-item.

Parameters **db_map_data** (*dict*) – maps DiffDatabaseMapping instances to list of items as dict

Returns maps parent tree-items to DiffDatabaseMapping instances to list of item ids

Return type dict

_parent_relationship_class_data(*self*, *db_map_data*)

Takes given relationship_class data and returns the same data keyed by parent tree-item.

Parameters **db_map_data** (*dict*) – maps DiffDatabaseMapping instances to list of items as dict

Returns maps parent tree-items to DiffDatabaseMapping instances to list of item ids

Return type dict

_parent_relationship_data(*self*, *db_map_data*)

Takes given relationship data and returns the same data keyed by parent tree-item.

Parameters **db_map_data** (*dict*) – maps DiffDatabaseMapping instances to list of items as dict

Returns maps parent tree-items to DiffDatabaseMapping instances to list of item ids

Return type dict

_parent_entity_group_data(*self*, *db_map_data*)
Takes given entity_group data and returns the same data keyed by parent tree-item.

Parameters **db_map_data** (*dict*) – maps DiffDatabaseMapping instances to list of items as dict

Returns maps parent tree-items to DiffDatabaseMapping instances to list of item ids

Return type dict

add_object_classes(*self*, *db_map_data*)

add_objects(*self*, *db_map_data*)

add_relationship_classes(*self*, *db_map_data*)

add_relationships(*self*, *db_map_data*)

raise_entity_groups(*self*, *db_map_data*)

remove_object_classes(*self*, *db_map_data*)

remove_objects(*self*, *db_map_data*)

remove_relationship_classes(*self*, *db_map_data*)

remove_relationships(*self*, *db_map_data*)

update_object_classes(*self*, *db_map_data*)

update_objects(*self*, *db_map_data*)

update_relationship_classes(*self*, *db_map_data*)

update_relationships(*self*, *db_map_data*)

find_next_relationship_index(*self*, *index*)
Find and return next ocurrence of relationship item.

class spinetoolbox.spine_db_editor.mvcmodels.entity_tree_models.**RelationshipTreeModel**(*parent*, *db_mn*, **db_m*

Bases: *spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_model.MultiDBTreeModel*

A relationship-oriented tree model.

Init class.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) – A manager for the given db_maps
- **db_maps** (*iter*) – DiffDatabaseMapping instances

root_item_type
Implement in subclasses to create a model specific to any entity type.

_parent_relationship_data(*self*, *db_map_data*)
Takes given relationship data and returns the same data keyed by parent tree-item.

Parameters **db_map_data** (*dict*) – maps DiffDatabaseMapping instances to list of items as dict

Returns maps parent tree-items to DiffDatabaseMapping instances to list of item ids

Return type dict

add_relationship_classes(*self*, *db_map_data*)

add_relationships(*self*, *db_map_data*)

remove_relationship_classes(*self*, *db_map_data*)

remove_relationships(*self*, *db_map_data*)

update_relationship_classes(*self*, *db_map_data*)

update_relationships(*self*, *db_map_data*)

spinetoolbox.spine_db_editor.mvcmodels.frozen_table_model

Contains FrozenTableModel class.

author

P. Vennström (VTT)

date 24.9.2019

Module Contents

Classes

FrozenTableModel	Used by custom_qtableview.FrozenTableView

class spinetoolbox.spine_db_editor.mvcmodels.frozen_table_model.**FrozenTableModel**(*parent*, *headers=None*, *data=None*)

Bases: PySide2.QtCore.QAbstractItemModel

Used by custom_qtableview.FrozenTableView

Parameters **parent** (TabularViewMixin) –

headers

parent(*self*, *child=None*)

index(*self*, *row*, *column*, *parent=QModelIndex()*)

reset_model(*self*, *data*, *headers*)

clear_model(*self*)

rowCount(*self*, *parent=QModelIndex()*)

columnCount(*self*, *parent=QModelIndex()*)

row(*self*, *index*)

data(*self*, *index*, *role*)

spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_item

Base classes to represent items from multiple databases in a tree.

authors

P. Vennström (VTT), M. Marin (KTH)

date 17.6.2020

Module Contents

Classes

MultiDBTreeItem	A tree item that may belong in multiple databases.

class spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_item.**MultiDBTreeItem**(*model=None*, *db_map_id=No*

Bases: *spinetoolbox.mvcmodels.minimal_tree_model.TreeItem*

A tree item that may belong in multiple databases.

Init class.

Parameters

- **db_mngr** (SpineDBManager) – a database manager
- **db_map_data** (*dict*) – maps instances of DiffDatabaseMapping to the id of the item in that db

item_type
Item type identifier string. Should be set to a meaningful value by subclasses.

visual_key = ['name']

db_mngr

child_item_type
Returns the type of child items. Reimplement in subclasses to return something more meaningful.

display_id
"Returns an id for display based on the display key. This id must be the same across all db_maps. If it's not, this property becomes None and measures need to be taken (see update_children_by_id).

display_data
"Returns the name for display.

display_database
"Returns the database for display.

display_icon
Returns an icon to display next to the name. Reimplement in subclasses to return something nice.

first_db_map
Returns the first associated db_map.

last_db_map
Returns the last associated db_map.

db_maps
Returns a list of all associated db_maps.

db_map_ids
Returns dict with db_map as key and id as value

add_db_map_id(*self*, *db_map*, *id_*)
Adds id for this item in the given db_map.

take_db_map(*self*, *db_map*)
Removes the mapping for given db_map and returns it.

_deep_refresh_children(*self*)
Refreshes children after taking db_maps from them. Called after removing and updating children for this item.

deep_remove_db_map(*self*, *db_map*)
Removes given db_map from this item and all its descendants.

deep_take_db_map(*self*, *db_map*)
Removes given db_map from this item and all its descendants, and returns a new item from the db_map's data.

Returns MultiDBTreeItem, NoneType

deep_merge(*self*, *other*)
Merges another item and all its descendants into this one.

db_map_id(*self*, *db_map*)
Returns the id for this item in given db_map or None if not present.

db_map_data(*self*, *db_map*)
Returns data for this item in given db_map or None if not present.

db_map_data_field(*self*, *db_map*, *field*, *default=None*)
Returns field from data for this item in given db_map or None if not found.

_create_new_children(*self*, *db_map*, *children_ids*)
Creates new items from ids associated to a db map.

Parameters

- **db_map** (*DiffDatabaseMapping*) – create children for this db_map
- **children_data** (*iter*) – create childs from these dictionaries

_merge_children(*self*, *new_children*)
Merges new children into this item. Ensures that each children has a valid display id afterwards.

has_children(*self*)
Returns whether or not this item has or could have children.

fetch_more(*self*)
Fetches children from all associated databases.

_get_children_ids(*self*, *db_map*)
Returns a list of children ids. Must be reimplemented in subclasses.

append_children_by_id(*self*, *db_map_ids*)
Appends children by id.

Parameters **db_map_ids** (*dict*) – maps DiffDatabaseMapping instances to list of ids

remove_children_by_id(*self*, *db_map_ids*)
Removes children by id.

Parameters **db_map_ids** (*dict*) – maps DiffDatabaseMapping instances to list of ids

is_valid(*self*)
Checks if the item is still valid after an update operation.

update_children_by_id(*self*, *db_map_ids*)
Updates children by id. Essentially makes sure all children have a valid display id after updating the underlying data. These may require 'splitting' a child into several for different dbs or merging two or more children from different dbs.

Examples of problems:

- The user renames an object_class in one db but not in the others –> we need to split
- The user renames an object_class and the new name is already 'taken' by another object_class in another db_map –> we need to merge

Parameters **db_map_ids** (*dict*) – maps DiffDatabaseMapping instances to list of ids

insert_children(*self*, *position*, **children*)
Insert new children at given position. Returns a boolean depending on how it went.

Parameters

- **position** (*int*) – insert new items here
- **children** (*iter*) – insert items from this iterable

remove_children(*self*, *position*, *count*)
Removes count children starting from the given position.

clear_children(*self*)
Clear children list.

_refresh_child_map(*self*)
Recomputes the child map.

find_children_by_id(*self*, *db_map*, **ids*, *reverse=True*)
Generates children with the given ids in the given db_map. If the first id is True, then generates *all* children with the given db_map.

find_rows_by_id(*self*, *db_map*, **ids*, *reverse=True*)

_find_unsorted_rows_by_id(*self*, *db_map*, **ids*)
Generates rows corresponding to children with the given ids in the given db_map. If the only id given is None, then generates rows corresponding to *all* children with the given db_map.

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

default_parameter_data(*self*)
Returns data to set as default in a parameter table when this item is selected.

spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_model

A base model class to represent items from multiple databases in a tree.

authors

P. Vennström (VTT), M. Marin (KTH)

date 17.6.2020

Module Contents

Classes

MultiDBTreeModel	Base class for all tree models in Spine db editor.

class spinetoolbox.spine_db_editor.mvcmodels.multi_db_tree_model.**MultiDBTreeModel**(*parent*, *db_mngr*, **db_maps*)

Bases: *spinetoolbox.mvcmodels.minimal_tree_model.MinimalTreeModel*

Base class for all tree models in Spine db editor.

Init class.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) – A manager for the given db_maps
- **db_maps** (*iter*) – DiffDatabaseMapping instances

root_item_type
Implement in subclasses to create a model specific to any entity type.

root_item

root_index

build_tree(*self*)
Builds tree.

columnCount(*self*, *parent=QModelIndex()*)

data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data stored under the given role for the index.

headerData(*self*, *section*, *orientation*, *role=Qt.DisplayRole*)

find_items(*self*, *db_map*, *path_prefix*, *parent_items=()*, *fetch=False*)
Returns items at given path prefix.

is_active_member_index(*self*, *index*)

set_active_member_indexes(*self*, *indexes*)

emit_data_changed_for_column(*self*, *column*, *parents*)

spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins

Miscelaneous mixins for parameter models

authors

M. Marin (KTH)

date 4.10.2019

Module Contents

Classes

ConvertToDBMixin	Base class for all mixins that convert model items (name-based) into database items (id-based).
FillInAlternativeIdMixin	Fills in alternative names.
FillInParameterNameMixin	Fills in parameter names.
FillInValueListIdMixin	Fills in value list ids.
MakeParameterTagMixin	Makes parameter_tag items.
FillInEntityClassIdMixin	Fills in entity_class ids.
FillInEntityIdsMixin	Fills in entity ids.
FillInParameterDefinitionIdsMixin	Fills in parameter_definition ids.
InferEntityClassIdMixin	Infers object_class ids.
MakeRelationshipOnTheFlyMixin	Makes relationships on the fly.

Functions

_parse_csv_list(csv_list)	

spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**_parse_csv_list**(*csv_list*)

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**ConvertToDBMixin**

Base class for all mixins that convert model items (name-based) into database items (id-based).

build_lookup_dictionary(*self*, *db_map_data*)

Begins an operation to convert items.

_convert_to_db(*self*, *item*, *db_map*)

Returns a db item (id-based) from the given model item (name-based).

Parameters

- **item** (*dict*) – the model item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db item list: error log

Return type dict

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**FillInAlternativeIdMixin**(**args*, ***kw*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

Fills in alternative names.

Initializes lookup dicts.

build_lookup_dictionary(*self*, *db_map_data*)

Builds a name lookup dictionary for the given data.

Parameters **db_map_data** (*dict*) – lists of model items keyed by DiffDatabaseMapping

_convert_to_db(*self*, *item*, *db_map*)

Returns a db item (id-based) from the given model item (name-based).

Parameters

- **item** (*dict*) – the model item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db item list: error log

Return type dict

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**FillInParameterNameMixin**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

Fills in parameter names.

_convert_to_db(*self*, *item*, *db_map*)

Returns a db item (id-based) from the given model item (name-based).

Parameters

- **item** (*dict*) – the model item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db item list: error log

Return type dict

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**FillInValueListIdMixin**(**args*, ***kwarg*

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

Fills in value list ids.

Initializes lookup dicts.

build_lookup_dictionary(*self*, *db_map_data*)

Builds a name lookup dictionary for the given data.

Parameters **db_map_data** (*dict*) – lists of model items keyed by DiffDatabaseMapping

_convert_to_db(*self*, *item*, *db_map*)

Returns a db item (id-based) from the given model item (name-based).

Parameters

- **item** (*dict*) – the model item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db item list: error log

Return type dict

_fill_in_value_list_id(*self*, *item*, *db_map*)

Fills in the value list id in the given db item.

Parameters

- **item** (*dict*) – the db item
- **db_map** (*DiffDatabaseMapping*) – the database where the given item belongs

Returns error log

Return type list

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**MakeParameterTagMixin**(**args*, ***kwargs*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

Makes parameter_tag items.

Initializes lookup dicts.

build_lookup_dictionary(*self*, *db_map_data*)
Builds a name lookup dictionary for the given data.

Parameters **db_map_data** (*dict*) – lists of model items keyed by DiffDatabaseMapping

_make_parameter_definition_tag(*self*, *item*, *db_map*)
Returns a db parameter_definition tag item (id-based) from the given model parameter_definition item (name-based).

Parameters

- **item** (*dict*) – the model parameter_definition item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db parameter_definition tag item list: error log

Return type dict

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**FillInEntityClassIdMixin**(**args*, ***kw*

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

Fills in entity_class ids.

Initializes lookup dicts.

build_lookup_dictionary(*self*, *db_map_data*)
Builds a name lookup dictionary for the given data.

Parameters **db_map_data** (*dict*) – lists of model items keyed by DiffDatabaseMapping

_fill_in_entity_class_id(*self*, *item*, *db_map*)
Fills in the entity_class id in the given db item.

Parameters

- **item** (*dict*) – the db item
- **db_map** (*DiffDatabaseMapping*) – the database where the given item belongs

Returns error log

Return type list

_convert_to_db(*self*, *item*, *db_map*)
Returns a db item (id-based) from the given model item (name-based).

Parameters

- **item** (*dict*) – the model item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db item list: error log

Return type dict

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**FillInEntityIdsMixin**(**args*, ***kwargs*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

Fills in entity ids.

Initializes lookup dicts.

_add_entities_on_the_fly = False

build_lookup_dictionary(*self*, *db_map_data*)
Builds a name lookup dictionary for the given data.

Parameters **db_map_data** (*dict*) – lists of model items keyed by DiffDatabaseMapping

_fill_in_entity_ids(*self*, *item*, *db_map*)
Fills in all possible entity ids keyed by entity_class id in the given db item (as there can be more than one entity for the same name).

Parameters

- **item** (*dict*) – the db item
- **db_map** (*DiffDatabaseMapping*) – the database where the given item belongs

Returns error log

Return type list

_convert_to_db(*self*, *item*, *db_map*)
Returns a db item (id-based) from the given model item (name-based).

Parameters

- **item** (*dict*) – the model item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db item list: error log

Return type dict

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**FillInParameterDefinitionIds**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

Fills in parameter_definition ids.

Initializes lookup dicts.

build_lookup_dictionary(*self*, *db_map_data*)
Builds a name lookup dictionary for the given data.

Parameters **db_map_data** (*dict*) – lists of model items keyed by DiffDatabaseMapping

_fill_in_parameter_ids(*self*, *item*, *db_map*)
Fills in all possible parameter_definition ids keyed by entity_class id in the given db item (as there can be more than one parameter_definition for the same name).

Parameters

- **item** (*dict*) – the db item
- **db_map** (*DiffDatabaseMapping*) – the database where the given item belongs

Returns error log

Return type list

_convert_to_db(*self*, *item*, *db_map*)
Returns a db item (id-based) from the given model item (name-based).

Parameters

- **item** (*dict*) – the model item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db item list: error log

Return type dict

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**InferEntityClassIdMixin**
Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

Infers object_class ids.

_convert_to_db(*self*, *item*, *db_map*)
Returns a db item (id-based) from the given model item (name-based).

Parameters

- **item** (*dict*) – the model item
- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db item list: error log

Return type dict

_infer_and_fill_in_entity_class_id(*self*, *item*, *db_map*)
Fills the entity_class id in the given db item, by intersecting entity ids and parameter ids. Then picks the correct entity id and parameter_definition id. Also sets the inferred entity_class name in the model.

Parameters

- **item** (*dict*) – the db item
- **db_map** (*DiffDatabaseMapping*) – the database where the given item belongs

Returns error log

Return type list

class spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.**MakeRelationshipOnTheFlyMixi**

Makes relationships on the fly.

Initializes lookup dicts.

static _make_unique_relationship_id(*item*)
Returns a unique name-based identifier for db relationships.

build_lookup_dictionaries(*self*, *db_map_data*)
Builds a name lookup dictionary for the given data.

Parameters **db_map_data** (*dict*) – lists of model items keyed by DiffDatabaseMapping.

_make_relationship_on_the_fly(*self*, *item*, *db_map*)
Returns a database relationship item (id-based) from the given model parameter_value item (name-based).

Parameters

- **item** (*dict*) – the model parameter_value item

- **db_map** (*DiffDatabaseMapping*) – the database where the resulting item belongs

Returns the db relationship item list: error log

Return type dict

spinetoolbox.spine_db_editor.mvcmodels.parameter_tag_model

A tree model for parameter_tags.

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

Classes

DBItem	An item representing a db.
TagItem	Paints the last item gray.
ParameterTagModel	A model to display parameter_tag data in a tree view.

class spinetoolbox.spine_db_editor.mvcmodels.parameter_tag_model.**DBItem**(*db_map*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.EmptyChildMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.NonLazyDBItem*

An item representing a db.

Init class.

Args db_mngr (SpineDBManager) db_map (DiffDatabaseMapping)

empty_child(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.parameter_tag_model.**TagItem**(*identifier=None*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.LastGrayMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.EditableMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.NonLazyTreeItem*

Paints the last item gray.

Initialize self. See help(type(self)) for accurate signature.

item_type

db_map

id

item_data

tag

add_item_to_db(*self*, *db_item*)

update_item_in_db(*self*, *db_item*)

header_data(*self*, *column*)

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

set_data(*self*, *column*, *value*, *role*)
Sets data for this item.

Parameters

- **column** (*int*) – column index
- **value** (*object*) – a new value
- **role** (*int*) – role of the new value

Returns True if data was set successfully, False otherwise

Return type bool

handle_updated_in_db(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.parameter_tag_model.**ParameterTagModel**(*parent*, *db_mngr*, **db_maps*)

Bases: *spinetoolbox.mvcmodels.minimal_tree_model.MinimalTreeModel*

A model to display parameter_tag data in a tree view.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- **db_maps** (*iter*) – DiffDatabaseMapping instances

Initialize class

receive_parameter_tags_added(*self*, *db_map_data*)

receive_parameter_tags_updated(*self*, *db_map_data*)

receive_parameter_tags_removed(*self*, *db_map_data*)

build_tree(*self*)
Initialize the internal data structure of the model.

columnCount(*self*, *parent=QModelIndex()*)
Returns the number of columns under the given parent. Always 2.

headerData(*self*, *section*, *orientation*, *role=Qt.DisplayRole*)

spinetoolbox.spine_db_editor.mvcmodels.parameter_value_list_model

A tree model for parameter_value lists.

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

Classes

DBItem	An item representing a db.
ListItem	A list item.
ValueItem	A value item.
ParameterValueListModel	A model to display parameter_value_list data in a tree view.

class spinetoolbox.spine_db_editor.mvcmodels.parameter_value_list_model.**DBItem**(*db_map*)

Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.EmptyChildMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.NonLazyDBItem*

An item representing a db.

Init class.

Args db_mngr (SpineDBManager) db_map (DiffDatabaseMapping)

empty_child(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.parameter_value_list_model.**ListItem**(*identifier=No*

Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.LastGrayMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.AllBoldMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.EditableMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.NonLazyTreeItem*

A list item.

Initialize self. See help(type(self)) for accurate signature.

db_map

item_type

name

value_list

fetch_more(*self*)
Fetches more children.

empty_child(*self*)

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

set_data(*self*, *column*, *value*, *role*)
Sets data for this item.

Parameters

- **column** (*int*) – column index
- **value** (*object*) – a new value
- **role** (*int*) – role of the new value

Returns True if data was set successfully, False otherwise

Return type bool

set_child_data(*self*, *child*, *value*)

update_name_in_db(*self*, *name*)

_new_value_list(*self*, *child_number*, *value*)

update_value_list_in_db(*self*, *child*, *value*)

add_to_db(*self*, *child*, *value*)
Add item to db.

handle_updated_in_db(*self*)
Runs when an item with this id has been updated in the db.

handle_added_to_db(*self*, *identifier*)
Runs when the item with this name has been added to the db.

update_value_list(*self*)

class spinetoolbox.spine_db_editor.mvcmodels.parameter_value_list_model.**ValueItem**(*is_empty=Fa*

Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.LastGrayMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.EditableMixin*, *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.NonLazyTreeItem*

A value item.

Initialize self. See help(type(self)) for accurate signature.

item_type

db_map

value

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

set_data(*self*, *column*, *value*, *role*)
Sets data for this item.

Parameters

- **column** (*int*) – column index
- **value** (*object*) – a new value
- **role** (*int*) – role of the new value

Returns True if data was set successfully, False otherwise

Return type bool

set_data_in_db(*self*, *db_value*)

class spinetoolbox.spine_db_editor.mvcmodels.parameter_value_list_model.**ParameterValueListM**

Bases: *spinetoolbox.mvcmodels.minimal_tree_model.MinimalTreeModel*

A model to display parameter_value_list data in a tree view.

Parameters

- **parent** (SpineDBEditor) –

- **db_mngr** (SpineDBManager) –
- **db_maps** (*iter*) – DiffDatabaseMapping instances

Initialize class

receive_parameter_value_lists_added(*self*, *db_map_data*)

receive_parameter_value_lists_updated(*self*, *db_map_data*)

receive_parameter_value_lists_removed(*self*, *db_map_data*)

build_tree(*self*)
Initialize the internal data structure of the model.

columnCount(*self*, *parent=QModelIndex()*)
Returns the number of columns under the given parent. Always 1.

index_name(*self*, *index*)

get_set_data_delayed(*self*, *index*)
Returns a function that ParameterValueEditor can call to set data for the given index at any later time, even if the model changes.

Parameters **index** (*QModelIndex*) –

Returns function

spinetoolbox.spine_db_editor.mvcmodels.pivot_model

Provides PivotModel.

author

P. Vennström (VTT)

date 1.11.2018

Module Contents

Classes

PivotModel	Initialize self. See help(type(self)) for accurate signature.

class spinetoolbox.spine_db_editor.mvcmodels.pivot_model.**PivotModel**
Initialize self. See help(type(self)) for accurate signature.

rows

columns

reset_model(*self*, *data*, *index_ids=()*, *rows=()*, *columns=()*, *frozen=()*, *frozen_value=()*)
Resets the model.

clear_model(*self*)

update_model(*self*, *data*)

add_to_model(*self*, *data*)

remove_from_model (*self*, *data*)

_check_pivot (*self*, *rows*, *columns*, *frozen*, *frozen_value*)
Checks if given pivot is valid.

Returns error message or None if no error

Return type str, NoneType

_index_key_getter (*self*, *indexes*)
Returns an itemgetter that always returns tuples from list of indexes

_get_unique_index_values (*self*, *indexes*)
Returns unique indexes that match the frozen condition.

Parameters **indexes** (*list*) –

Returns list

set_pivot (*self*, *rows*, *columns*, *frozen*, *frozen_value*)
Sets pivot.

set_frozen_value (*self*, *value*)
Sets values for the frozen indexes.

get_pivoted_data (*self*, *row_mask*, *column_mask*)
Returns data for indexes in row_mask and column_mask.

Parameters

- **row_mask** (*list*) –
- **column_mask** (*list*) –

Returns list(list)

row_key (*self*, *row*)

column_key (*self*, *column*)

spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models

Provides pivot table models for the Tabular View.

author

P. Vennström (VTT)

date 1.11.2018

Module Contents

Classes

PivotTableModelBase	**param parent**
TopLeftHeaderItem	Base class for all 'top left pivot headers'.

Continued on next page

Table 168 – continued from previous page

TopLeftObjectHeaderItem	A top left header for object_class.
TopLeftParameterHeaderItem	A top left header for parameter_definition.
TopLeftParameterIndexHeaderItem	A top left header for parameter index.
TopLeftAlternativeHeaderItem	A top left header for parameter index.
ParameterValuePivotTableModel	A model for the pivot table in parameter_value input type.
IndexExpansionPivotTableModel	A model for the pivot table in parameter index expansion input type.
RelationshipPivotTableModel	A model for the pivot table in relationship input type.
PivotTableSortFilterProxy	Initialize class.

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**PivotTableModelBase**(*parent*)

Bases: PySide2.QtCore.QAbstractTableModel

Parameters **parent** (SpineDBEditor) –

_V_HEADER_WIDTH = 5

_FETCH_STEP_COUNT = 64

_MIN_FETCH_COUNT = 512

_FETCH_DELAY = 0

item_type

Returns the item type.

plot_x_column

Returns the index of the column designated as Y values for plotting or None.

reset_data_count(*self*)

start_fetching(*self*)

fetch_more_rows(*self*)

fetch_more_columns(*self*)

call_reset_model(*self*, *object_class_ids*, *pivot=None*)

Parameters

- **object_class_names** (*dict*) – mapping disambiguated class names to ids
- **pivot** (*tuple, optional*) – list of rows, list of columns, list of frozen indexes, frozen value

reset_model(*self*, *data*, *index_ids*, *rows=()*, *columns=()*, *frozen=()*, *frozen_value=()*)

clear_model(*self*)

update_model(*self*, *data*)

Update model with new data, but doesn't grow the model.

Parameters **data** (*dict*) –

add_to_model(*self*, *data*)

remove_from_model(*self*, *data*)

set_pivot(*self*, *rows*, *columns*, *frozen*, *frozen_value*)

set_frozen_value(*self*, *frozen_value*)

set_plot_x_column (*self*, *column*, *is_x*)
Sets or clears the X flag on a column

headerRowCount (*self*)
Returns number of rows occupied by header.

headerColumnCount (*self*)
Returns number of columns occupied by header.

dataRowCount (*self*)
Returns number of rows that contain actual data.

dataColumnCount (*self*)
Returns number of columns that contain actual data.

emptyRowCount (*self*)

emptyColumnCount (*self*)

rowCount (*self*, *parent=QModelIndex()*)
Number of rows in table, number of header rows + datarows + 1 empty row

columnCount (*self*, *parent=QModelIndex()*)
Number of columns in table, number of header columns + datacolumns + 1 empty columns

flags (*self*, *index*)
Roles for data

top_left_indexes (*self*)
Returns indexes in the top left area.

Returns list(QModelIndex): top indexes (horizontal headers, associated to rows) list(QModelIndex): left indexes (vertical headers, associated to columns)

index_in_top (*self*, *index*)

index_in_left (*self*, *index*)

index_in_top_left (*self*, *index*)
Returns whether or not the given index is in top left corner, where pivot names are displayed

index_in_column_headers (*self*, *index*)
Returns whether or not the given index is in column headers (horizontal) area

index_in_row_headers (*self*, *index*)
Returns whether or not the given index is in row headers (vertical) area

index_in_headers (*self*, *index*)

index_in_empty_column_headers (*self*, *index*)
Returns whether or not the given index is in empty column headers (vertical) area

index_in_empty_row_headers (*self*, *index*)
Returns whether or not the given index is in empty row headers (vertical) area

index_in_data (*self*, *index*)
Returns whether or not the given index is in data area

column_is_index_column (*self*, *column*)
Returns True if column is the column containing expanded parameter_value indexes.

headerData (*self*, *section*, *orientation*, *role=Qt.DisplayRole*)

map_to_pivot (*self*, *index*)
Returns a tuple of row and column in the pivot model that corresponds to the given model index.

Parameters **index** (*QModelIndex*) –

Returns row int: column

Return type int

_top_left_id(*self*, *index*)
Returns the id of the top left header corresponding to the given header index.

Parameters **index** (*QModelIndex*) –

Returns int, NoneType

_header_id(*self*, *index*)
Returns the id of the given row or column header index.

Parameters **index** (*QModelIndex*) –

Returns int, NoneType

_header_ids(*self*, *row*, *column*)
Returns the ids for the headers at given row *and* column.

Parameters

- **row** (*int*) –
- **column** (*int*) –

Returns tuple(int)

header_name(*self*, *index*)
Returns the name corresponding to the given header index. Used by PivotTableView.

Parameters **index** (*QModelIndex*) –

Returns str

_color_data(*self*, *index*)

_text_alignment_data(*self*, *index*)

_header_data(*self*, *index*, *role=Qt.DisplayRole*)

_header_name(*self*, *top_left_id*, *header_id*)

_data(*self*, *index*, *role*)

data(*self*, *index*, *role=Qt.DisplayRole*)

setData(*self*, *index*, *value*, *role=Qt.EditRole*)

batch_set_data(*self*, *indexes*, *values*)

_batch_set_inner_data(*self*, *inner_data*)

_do_batch_set_inner_data(*self*, *row_map*, *column_map*, *data*, *values*)

_batch_set_header_data(*self*, *header_data*)

_batch_set_empty_header_data(*self*, *header_data*, *get_top_left_id*)

receive_data_added_or_removed(*self*, *data*, *action*)

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**TopLeftHeaderItem**(*model*)
Base class for all 'top left pivot headers'. Represents a header located in the top left area of the pivot table.

Parameters **model** (*PivotTableModel*) –

model

db_mngr

db_map

_get_header_data_from_db(*self*, *item_type*, *header_id*, *field_name*, *role*)

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**TopLeftObjectHeaderItem**(*mod*
clas
clas

Bases: *spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.TopLeftHeaderItem*

A top left header for object_class.

Parameters **model** (*PivotTableModel*) –

header_type

name

header_data(*self*, *header_id*, *role=Qt.DisplayRole*)

update_data(*self*, *data*)

add_data(*self*, *names*)

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**TopLeftParameterHeaderItem**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.TopLeftHeaderItem*

A top left header for parameter_definition.

Parameters **model** (*PivotTableModel*) –

header_type

name

header_data(*self*, *header_id*, *role=Qt.DisplayRole*)

update_data(*self*, *data*)

add_data(*self*, *names*)

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**TopLeftParameterIndexHeade**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.TopLeftHeaderItem*

A top left header for parameter index.

Parameters **model** (*PivotTableModel*) –

header_type

name

header_data(*self*, *header_id*, *role=Qt.DisplayRole*)

update_data(*self*, *_data*)

add_data(*self*, *_names*)

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**TopLeftAlternativeHeaderIte**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.TopLeftHeaderItem*

A top left header for parameter index.

Parameters **model** (*PivotTableModel*) –

header_type

name

header_data(*self*, *header_id*, *role=Qt.DisplayRole*)

update_data(*self*, *data*)

add_data(*self*, *names*)

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**ParameterValuePivotTableMo**
Bases: *spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.PivotTableModelBase*

A model for the pivot table in parameter_value input type.

Parameters **parent** (SpineDBEditor) –

item_type
Returns the item type.

object_and_parameter_ids(*self*, *index*)
Returns the object and parameter ids corresponding to the given data index. Used by PivotTableView.

Parameters **index** (*QModelIndex*) –

Returns object ids int: parameter id

Return type list(int)

object_and_parameter_names(*self*, *index*)
Returns the object and parameter names corresponding to the given data index. Used by PivotTableView.

Parameters **index** (*QModelIndex*) –

Returns object names str: parameter name

Return type list(str)

index_name(*self*, *index*)
Returns a string that concatenates the object and parameter names corresponding to the given data index. Used by plotting and ParameterValueEditor.

Parameters **index** (*QModelIndex*) –

Returns str

column_name(*self*, *column*)
Returns a string that concatenates the object and parameter names corresponding to the given column. Used by plotting.

Parameters **column** (*int*) –

Returns str

call_reset_model(*self*, *object_class_ids*, *pivot=None*)
See base class.

_default_pivot(*self*)

_data(*self*, *index*, *role*)

_do_batch_set_inner_data(*self*, *row_map*, *column_map*, *data*, *values*)

_object_parameter_value_to_add(*self*, *header_ids*, *value*)

_relationship_parameter_value_to_add(*self*, *header_ids*, *value*, *rel_id_lookup*)

_make_parameter_value_to_add(*self*)

static _parameter_value_to_update(*id_*, *header_ids*, *value*)

_batch_set_parameter_value_data(*self*, *row_map*, *column_map*, *data*, *values*)
Sets parameter values in batch.

_checked_parameter_values(*self*, *items*)

_add_parameter_values(*self*, *items*)

_update_parameter_values(*self*, *items*)

get_set_data_delayed(*self*, *index*)
Returns a function that ParameterValueEditor can call to set data for the given index at any later time, even if the model changes.

Parameters **index** (*QModelIndex*) –

Returns function

receive_objects_added_or_removed(*self*, *items*, *action*)

receive_relationships_added_or_removed(*self*, *relationships*, *action*)

receive_parameter_definitions_added_or_removed(*self*, *parameters*, *action*)

receive_parameter_values_added_or_removed(*self*, *parameter_values*, *action*)

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**IndexExpansionPivotTableMo**
Bases: *spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.ParameterValuePivotTableModel*

A model for the pivot table in parameter index expansion input type.

Parameters **parent** (SpineDBEditor) –

call_reset_model(*self*, *object_class_ids*, *pivot=None*)
See base class.

flags(*self*, *index*)
Roles for data

column_is_index_column(*self*, *column*)
Returns True if column is the column containing expanded parameter_value indexes.

_data(*self*, *index*, *role*)

static _parameter_value_to_update(*id_*, *header_ids*, *value*)

_update_parameter_values(*self*, *items*)

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**RelationshipPivotTableMode**
Bases: *spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.PivotTableModelBase*

A model for the pivot table in relationship input type.

Parameters **parent** (SpineDBEditor) –

item_type
Returns the item type.

call_reset_model(*self*, *object_class_ids*, *pivot=None*)
See base class.

_default_pivot(*self*)

_data(*self*, *index*, *role*)

_text_alignment_data(*self*, *index*)

_do_batch_set_inner_data(*self*, *row_map*, *column_map*, *data*, *values*)

_batch_set_relationship_data(*self*, *row_map*, *column_map*, *data*, *values*)

receive_objects_added_or_removed(*self*, *items*, *action*)

receive_relationships_added_or_removed(*self*, *relationships*, *action*)

receive_parameter_definitions_added_or_removed(*self*, *db_map_data*, *action*)
Returns False, this model does not hold parameter data.

receive_parameter_values_added_or_removed(*self*, *db_map_data*, *action*)
Returns False, this model does not hold parameter data.

class spinetoolbox.spine_db_editor.mvcmodels.pivot_table_models.**PivotTableSortFilterProxy**(*p*
Bases: PySide2.QtCore.QSortFilterProxyModel

Initialize class.

set_filter(*self*, *identifier*, *filter_value*)
Sets filter for a given index (object_class) name.

Parameters

- **identifier** (*int*) – index identifier
- **filter_value** (*set, None*) – A set of accepted values, or None if no filter (all pass)

clear_filter(*self*)

accept_index(*self*, *index*, *index_ids*)

filterAcceptsRow(*self*, *source_row*, *source_parent*)
Returns true if the item in the row indicated by the given source_row and source_parent should be included in the model; otherwise returns false.

filterAcceptsColumn(*self*, *source_column*, *source_parent*)
Returns true if the item in the column indicated by the given source_column and source_parent should be included in the model; otherwise returns false.

batch_set_data(*self*, *indexes*, *values*)

spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models

Single models for parameter definitions and values (as 'for a single entity').

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

Classes

SingleParameterModel	A parameter model for a single entity_class to go in a CompoundParameterModel.
SingleObjectParameterMixin	Associates a parameter model with a single object_class.
SingleRelationshipParameterMixin	Associates a parameter model with a single relationship_class.
SingleParameterDefinitionMixin	A parameter_definition model for a single entity_class.
SingleParameterValueMixin	A parameter_value model for a single entity_class.
SingleObjectParameterDefinitionModel	An object parameter_definition model for a single object_class.
SingleRelationshipParameterDefinitionModel	A relationship parameter_definition model for a single relationship_class.
SingleObjectParameterValueModel	An object parameter_value model for a single object_class.
SingleRelationshipParameterValueModel	A relationship parameter_value model for a single relationship_class.

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleParameterModel**(

Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A parameter model for a single entity_class to go in a CompoundParameterModel. Provides methods to associate the model to an entity_class as well as to filter entities within the class.

Init class.

Parameters **header** (*list*) – list of field names for the header

item_type
The item type, either 'parameter_value' or 'parameter_definition', required by the data method.

entity_class_type
The entity_class type, either 'object_class' or 'relationship_class'.

json_fields

fixed_fields

group_fields

parameter_definition_id_key

can_be_filtered

insertRows(*self*, *row*, *count*, *parent=QModelIndex()*)
This model doesn't support row insertion.

item_id(*self*, *row*)

db_item(*self*, *index*)

_db_item(*self*, *row*)

db_item_from_id(*self*, *id_*)

db_items(*self*)

flags (*self*, *index*)
Make fixed indexes non-editable.

get_field_item_data (*self*, *field*)
Returns item data for given field.

Parameters **field** (*str*) – A field from the header

Returns str, str

get_id_key (*self*, *field*)

get_field_item (*self*, *field*, *db_item*)
Returns a db item corresponding to the given field from the table header, or an empty dict if the field doesn't contain db items.

data (*self*, *index*, *role=Qt.DisplayRole*)
Gets the id and database for the row, and reads data from the db manager using the item_type property. Paint the object_class icon next to the name. Also paint background of fixed indexes gray and apply custom format to JSON fields.

batch_set_data (*self*, *indexes*, *data*)
Sets data for indexes in batch. Sets data directly in database using db mngr. If successful, updated data will be automatically seen by the data method.

update_items_in_db (*self*, *items*)
Update items in db. Required by batch_set_data

_filter_accepts_row (*self*, *row*)

_parameter_filter_accepts_row (*self*, *row*)
Returns the result of the parameter filter.

_auto_filter_accepts_row (*self*, *row*)
Returns the result of the auto filter.

accepted_rows (*self*)
Returns a list of accepted rows, for convenience.

_get_field_item (*self*, *field*, *id_*)
Returns a item from the db_mngr.get_item depending on the field. If a field doesn't correspond to a item in the database then an empty dict is returned.

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleObjectParameterM**
Associates a parameter model with a single object_class.

entity_class_type

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleRelationshipPara**
Associates a parameter model with a single relationship_class.

entity_class_type

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleParameterDefinit**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInParameterNameMixin*, *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.FillInValueListIdMixin*, *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.MakeParameterTagMixin*

A parameter_definition model for a single entity_class.

Initializes lookup dicts.

item_type

update_items_in_db(*self*, *items*)
Update items in db.

Parameters **item** (*list*) – dictionary-items

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleParameterValueM**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.parameter_mixins.ConvertToDBMixin*

A parameter_value model for a single entity_class.

Initialize self. See help(type(self)) for accurate signature.

item_type

_filter_accepts_row(*self*, *row*)
Reimplemented to also account for the entity filter.

_entity_filter_accepts_row(*self*, *row*)
Returns the result of the entity filter.

_alternative_filter_accepts_row(*self*, *row*)
Returns the result of the alternative filter.

update_items_in_db(*self*, *items*)
Update items in db.

Parameters **item** (*list*) – dictionary-items

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleObjectParameterD**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleObjectParameterMixin*, *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleParameterDefinitionMixin*, *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleParameterModel*

An object parameter_definition model for a single object_class.

Initializes lookup dicts.

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleRelationshipPara**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleRelationshipParameterMixin*, *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleParameterDefinitionMixin*, *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleParameterModel*

A relationship parameter_definition model for a single relationship_class.

Initializes lookup dicts.

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleObjectParameterV**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleObjectParameterMixin*, *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleParameterValueMixin*, *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleParameterModel*

An object parameter_value model for a single object_class.

Initialize self. See help(type(self)) for accurate signature.

class spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.**SingleRelationshipPara**

Bases: *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleRelationshipParameterMixin*, *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleParameterValueMixin*, *spinetoolbox.spine_db_editor.mvcmodels.single_parameter_models.SingleParameterModel*

A relationship parameter_value model for a single relationship_class.

Initialize self. See help(type(self)) for accurate signature.

spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility

A tree model for parameter_value lists.

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

Classes

NonLazyTreeItem	A tree item that fetches their children as they are inserted.
EditableMixin	
LastGrayMixin	Paints the last item gray.
AllBoldMixin	Bolds text.
EmptyChildMixin	Guarantess there's always an empty child.
NonLazyDBItem	An item representing a db.

class spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.**NonLazyTreeItem**(*model=None*)

Bases: *spinetoolbox.mvcmodels.minimal_tree_model.TreeItem*

A tree item that fetches their children as they are inserted.

Initializes item.

Parameters **model** (*MinimalTreeModel, NoneType*) – The model where the item belongs.

item_type

db_mngr

can_fetch_more(*self*)
Disables lazy loading by returning False.

insert_children(*self*, *position*, **children*)
Fetches the children as they become parented.

class spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.**EditableMixin**

flags(*self*, *column*)
Makes items editable.

class spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.**LastGrayMixin**
Paints the last item gray.

data(*self*, *column*, *role=Qt.DisplayRole*)

class spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.**AllBoldMixin**
Bolds text.

data(*self*, *column*, *role=Qt.DisplayRole*)

class spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.**EmptyChildMixin**
Guarantess there's always an empty child.

empty_child(*self*)

fetch_more(*self*)

append_empty_child(*self*, *row*)
Appends empty child if the row is the last one.

class spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.**NonLazyDBItem**(*db_map*)
Bases: *spinetoolbox.spine_db_editor.mvcmodels.tree_item_utility.NonLazyTreeItem*

An item representing a db.

Init class.

Args db_mngr (SpineDBManager) db_map (DiffDatabaseMapping)

item_type

data(*self*, *column*, *role=Qt.DisplayRole*)
Shows Spine icon for fun.

set_data(*self*, *column*, *value*, *role*)
See base class.

spinetoolbox.spine_db_editor.ui

Automatically generated UI modules for Spine db editor.

authors

M. Marin (KTH)

date 13.5.2020

Submodules

spinetoolbox.spine_db_editor.ui.spine_db_editor_window

Module Contents

Classes

Ui_MainWindow

class spinetoolbox.spine_db_editor.ui.spine_db_editor_window.**Ui_MainWindow**
Bases: object

setupUi(*self*, *MainWindow*)

retranslateUi(*self*, *MainWindow*)

spinetoolbox.spine_db_editor.widgets

Interface logic for Spine db editor.

authors

M. Marin (KTH)

date 13.5.2020

Submodules

spinetoolbox.spine_db_editor.widgets.add_items_dialogs

Classes for custom QDialogs to add items to databases.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

Classes

AddReadyRelationshipsDialog	A dialog to let the user add new 'ready' relationships.
AddItemsDialog	A dialog to query user's preferences for new db items.
AddObjectClassesDialog	A dialog to query user's preferences for new object classes.
AddObjectsDialog	A dialog to query user's preferences for new objects.
AddRelationshipClassesDialog	A dialog to query user's preferences for new relationship classes.
AddOrManageRelationshipsDialog	A dialog to query user's preferences for new relationships.
AddRelationshipsDialog	A dialog to query user's preferences for new relationships.
ManageRelationshipsDialog	A dialog to query user's preferences for managing relationships.
AddOrManageObjectGroupDialog	**param parent** data store widget
AddObjectGroupDialog	**param parent** data store widget

Continued on next page

Table 172 – continued from previous page

ManageObjectGroupDialog	**param parent** data store widget

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddReadyRelationshipsDialog**(*pa*
re
la
tic
sh
re
la
tic
sh
db
**a*

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.ManageItemsDialogBase*

A dialog to let the user add new 'ready' relationships.

Parameters

- **parent** (SpineDBEditor) –
- **relationships_class** (*dict*) –
- **relationships** (*list(list(str))*) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – DiffDatabaseMapping instances

make_table_view(*self*)

populate_table_view(*self*)

connect_signals(*self*)
Connect signals to slots.

_handle_table_view_cell_clicked(*self*, *row*, *column*)

_handle_table_view_current_changed(*self*, *current*, *_previous*)

accept(*self*)

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddItemsDialog**(*parent*, *db_mngr*, **db_maps*)

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.ManageItemsDialog*

A dialog to query user's preferences for new db items.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – DiffDatabaseMapping instances

connect_signals(*self*)
Connect signals to slots.

remove_selected_rows(*self*, *checked=True*)

all_databases(*self*, *row*)
Returns a list of db names available for a given row. Used by delegates.

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddObjectClassesDialog**(*parent*, *db_mngr*, **db_maps*)

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.ShowIconColorEditorMixin*, *spinetoolbox.spine_db_editor.widgets.add_items_dialogs.AddItemsDialog*

A dialog to query user's preferences for new object classes.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – DiffDatabaseMapping instances

connect_signals(*self*)
Connect signals to slots.

all_db_maps(*self*, *row*)
Returns a list of db maps available for a given row. Used by ShowIconColorEditorMixin.

accept(*self*)
Collect info from dialog and try to add items.

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddObjectsDialog**(*parent*, *db_mngr*, **db_maps*, *class_name=None*, *force_default=False*)

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.GetObjectClassesMixin*, *spinetoolbox.spine_db_editor.widgets.add_items_dialogs.AddItemsDialog*

A dialog to query user's preferences for new objects.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – DiffDatabaseMapping instances
- **class_name** (*str*) – default object_class name
- **force_default** (*bool*) – if True, defaults are non-editable

accept(*self*)
Collect info from dialog and try to add items.

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddRelationshipClassesDialog**(*p*, *d*, ***, *d*, *j*, *f*)

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.*

GetObjectClassesMixin, *spinetoolbox.spine_db_editor.widgets.add_items_dialogs.AddItemsDialog*

A dialog to query user's preferences for new relationship classes.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – DiffDatabaseMapping instances
- **object_class_one_name** (*str*) – default object_class name
- **force_default** (*bool*) – if True, defaults are non-editable

connect_signals(*self*)
Connect signals to slots.

_handle_spin_box_value_changed(*self*, *i*)

insert_column(*self*)

remove_column(*self*)

_handle_model_data_changed(*self*, *top_left*, *bottom_right*, *roles*)
Reimplement in subclasses to handle changes in model data.

accept(*self*)
Collect info from dialog and try to add items.

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddOrManageRelationshipsDialog**

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.GetRelationshipClassesMixin*, *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.GetObjectsMixin*, *spinetoolbox.spine_db_editor.widgets.add_items_dialogs.AddItemsDialog*

A dialog to query user's preferences for new relationships.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – DiffDatabaseMapping instances

make_model(*self*)

connect_signals(*self*)
Connect signals to slots.

reset_model(*self*, *index*)
Called when relationship_class's combobox's index changes. Update relationship_class attribute accordingly and reset model.

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddRelationshipsDialog**(*parent*, *db_mngr*, **db_maps*, *relationship_class_key*, *object_names_by_class_name*, *force_default*)

Bases: *spinetoolbox.spine_db_editor.widgets.add_items_dialogs.AddOrManageRelationshipsDialog*

A dialog to query user's preferences for new relationships.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – DiffDatabaseMapping instances
- **relationship_class_key** (*tuple(str,str)*) – relationships class name, object_class name list string
- **object_names_by_class_name** (*dict*) – mapping object_class names to default object names
- **force_default** (*bool*) – if True, defaults are non-editable

make_model(*self*)

reset_model(*self*, *index*)
Setup model according to current relationship_class selected in combobox.

_handle_model_data_changed(*self*, *top_left*, *bottom_right*, *roles*)
Reimplement in subclasses to handle changes in model data.

accept(*self*)
Collect info from dialog and try to add items.

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**ManageRelationshipsDialog**(*parent*, *db_mngr*, **db_maps*, *relationship_class_key*, *object_names_by_class_name*, *force_default*)

Bases: *spinetoolbox.spine_db_editor.widgets.add_items_dialogs.AddOrManageRelationshipsDialog*

A dialog to query user's preferences for managing relationships.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) – the manager to do the removal
- ***db_maps** – DiffDatabaseMapping instances
- **relationship_class_key** (*str, optional*) – relationships class name, object_class name list string.

make_model(*self*)

splitter_widgets(*self*)

connect_signals(*self*)
Connect signals to slots.

reset_relationship_class_combo_box(*self*, *database*, *relationship_class_key=None*)

add_relationships(*self*, *checked=True*)

reset_model(*self*, *index*)
Setup model according to current relationship_class selected in combobox.

resize_window_to_columns(*self*, *height=None*)

accept(*self*)
Collect info from dialog and try to add items.

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddOrManageObjectGroupDialog**(*p*
o
j
d
=

Bases: PySide2.QtWidgets.QDialog

Parameters

- **parent** (SpineDBEditor) – data store widget
- **object_class_item** (ObjectClassItem) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – database mappings

connect_signals(*self*)
Connect signals to slots.

reset_list_widgets(*self*, *database*)

initial_member_ids(*self*)

initial_entity_id(*self*)

add_members(*self*, *checked=False*)

remove_members(*self*, *checked=False*)

_check_validity(*self*)

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**AddObjectGroupDialog**(*parent*, *ob-*
ject_class_it
db_mngr,
**db_maps*)

Bases: *spinetoolbox.spine_db_editor.widgets.add_items_dialogs.AddOrManageObjectGroupDialog*

Parameters

- **parent** (SpineDBEditor) – data store widget
- **object_class_item** (ObjectClassItem) –
- **db_mngr** (SpineDBManager) –

- ***db_maps** – database mappings

initial_member_ids(*self*)

initial_entity_id(*self*)

_check_validity(*self*)

accept(*self*)

class spinetoolbox.spine_db_editor.widgets.add_items_dialogs.**ManageObjectGroupDialog**(*parent*, *object_item*, *db_mngr*, **db_maps*)

Bases: *spinetoolbox.spine_db_editor.widgets.add_items_dialogs.AddOrManageObjectGroupDialog*

Parameters

- **parent** (SpineDBEditor) – data store widget
- **object_item** (entity_tree_item.ObjectItem) –
- **db_mngr** (SpineDBManager) –
- ***db_maps** – database mappings

initial_member_ids(*self*)

initial_entity_id(*self*)

accept(*self*)

spinetoolbox.spine_db_editor.widgets.custom_delegates

Custom item delegates.

author

M. Marin (KTH)

date 1.9.2018

Module Contents

Classes

RelationshipPivotTableDelegate	A delegate that places a fully functioning QCheckBox.
ParameterPivotTableDelegate	**param parent**
ParameterDelegate	Base class for all custom parameter delegates.
DatabaseNameDelegate	A delegate for the database name.
ParameterValueOrDefaultValueDelegate	A delegate for the either the value or the default value.
ParameterDefaultValueDelegate	A delegate for the either the default value.
ParameterValueDelegate	A delegate for the parameter_value.
TagListDelegate	A delegate for the parameter_tag list.

Continued on next page

Table 173 – continued from previous page

ValueListDelegate	A delegate for the parameter_value-list.
ObjectClassNameDelegate	A delegate for the object_class name.
RelationshipClassNameDelegate	A delegate for the relationship_class name.
ParameterNameDelegate	A delegate for the object parameter name.
ObjectNameDelegate	A delegate for the object name.
AlternativeNameDelegate	A delegate for the object name.
ObjectNameListDelegate	A delegate for the object name list.
ManageItemsDelegate	A custom delegate for the model in {Add/Edit}ItemDialogs.
ManageObjectClassesDelegate	A delegate for the model and view in {Add/Edit}ObjectClassesDialog.
ManageObjectsDelegate	A delegate for the model and view in {Add/Edit}ObjectsDialog.
ManageRelationshipClassesDelegate	A delegate for the model and view in {Add/Edit}RelationshipClassesDialog.
ManageRelationshipsDelegate	A delegate for the model and view in {Add/Edit}RelationshipsDialog.
RemoveEntitiesDelegate	A delegate for the model and view in RemoveEntities-Dialog.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**RelationshipPivotTableDelegate**

Bases: *spinetoolbox.widgets.custom_delegates.CheckBoxDelegate*

A delegate that places a fully functioning QCheckBox.

parent
either toolbox or spine datapackage widget

Type QMainWindow

centered
whether or not the checkbox should be center-aligned in the widget

Type bool

Parameters **parent** (SpineDBEditor) –

data_committed

static _is_relationship_index(*index*)
Checks whether or not the given index corresponds to a relationship, in which case we need to use the check box delegate.

Returns bool

setModelData(*self*, *editor*, *model*, *index*)
Send signal.

setEditorData(*self*, *editor*, *index*)
Do nothing. We're setting editor data right away in createEditor.

paint(*self*, *painter*, *option*, *index*)
Paint a checkbox without the label.

editorEvent(*self*, *event*, *model*, *option*, *index*)
Change the data in the model and the state of the checkbox when user presses left mouse button and this cell is editable. Otherwise do nothing.

createEditor(*self*, *parent*, *option*, *index*)
Important, otherwise an editor is created if the user clicks in this cell. ** Need to hook up a signal to the model.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ParameterPivotTableDelegate**(*par*
Bases: PySide2.QtWidgets.QStyledItemDelegate

Parameters **parent** (SpineDBEditor) –

parameter_value_editor_requested

data_committed

setModelData(*self*, *editor*, *model*, *index*)
Send signal.

setEditorData(*self*, *editor*, *index*)
Do nothing. We're setting editor data right away in createEditor.

createEditor(*self*, *parent*, *option*, *index*)

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ParameterDelegate**(*parent*, *db_mngr*)
Bases: PySide2.QtWidgets.QStyledItemDelegate

Base class for all custom parameter delegates.

parent
tree or graph view form

Type *SpineDBEditor*

db_mngr

Type *SpineDBManager*

data_committed

setModelData(*self*, *editor*, *model*, *index*)
Send signal.

setEditorData(*self*, *editor*, *index*)
Do nothing. We're setting editor data right away in createEditor.

updateEditorGeometry(*self*, *editor*, *option*, *index*)

_close_editor(*self*, *editor*, *index*)
Closes editor. Needed by SearchBarEditor.

_get_db_map(*self*, *index*)
Returns the db_map for the database at given index or None if not set yet.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**DatabaseNameDelegate**(*parent*, *db_mngr*)
Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the database name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ParameterValueOrDefaultValueDel**

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the either the value or the default value.

parameter_value_editor_requested

setModelData(*self*, *editor*, *model*, *index*)
Sends signal.

_create_or_request_parameter_value_editor(*self*, *parent*, *option*, *index*, *db_map*)
Emits the signal to request a standalone *ParameterValueEditor* from parent widget.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ParameterDefaultValueDelegate**(*p*
d

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterValueOrDefaultValueDelegate*

A delegate for the either the default value.

createEditor(*self*, *parent*, *option*, *index*)
Returns or requests a parameter_value editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ParameterValueDelegate**(*parent*, *db_mngr*)

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterValueOrDefaultValueDelegate*

A delegate for the parameter_value.

setModelData(*self*, *editor*, *model*, *index*)
Send signal.

_get_value_list_id(*self*, *index*, *db_map*)
Returns a value list item for the given index and db_map.

createEditor(*self*, *parent*, *option*, *index*)
If the parameter has associated a value list, returns a SearchBarEditor. Otherwise returns or requests a dedicated parameter_value editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**TagListDelegate**(*parent*, *db_mngr*)

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the parameter_tag list.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ValueListDelegate**(*parent*, *db_mngr*)

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the parameter_value-list.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ObjectClassNameDelegate**(*parent*, *db_mngr*

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the object_class name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**RelationshipClassNameDelegate**(*p*
c

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the relationship_class name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ParameterNameDelegate**(*parent*, *db_mngr*)

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the object parameter name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ObjectNameDelegate**(*parent*, *db_mngr*)

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the object name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**AlternativeNameDelegate**(*parent*, *db_mngr*)

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the object name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ObjectNameListDelegate**(*parent*, *db_mngr*)

Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ParameterDelegate*

A delegate for the object name list.

object_name_list_editor_requested

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ManageItemsDelegate**
Bases: `PySide2.QtWidgets.QStyledItemDelegate`

A custom delegate for the model in {Add/Edit}ItemDialogs.

parent
parent dialog

Type *ManageItemsDialog*

data_committed

setModelData(*self*, *editor*, *model*, *index*)
Send signal.

close_editor(*self*, *editor*, *index*, *model*)

updateEditorGeometry(*self*, *editor*, *option*, *index*)

connect_editor_signals(*self*, *editor*, *index*)
Connect editor signals if necessary.

_create_database_editor(*self*, *parent*, *option*, *index*)

createEditor(*self*, *parent*, *option*, *index*)
Returns an editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ManageObjectClassesDelegate**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ManageItemsDelegate*

A delegate for the model and view in {Add/Edit}ObjectClassesDialog.

parent
parent dialog

Type *ManageItemsDialog*

icon_color_editor_requested

createEditor(*self*, *parent*, *option*, *index*)
Return editor.

paint(*self*, *painter*, *option*, *index*)
Get a pixmap from the index data and paint it in the middle of the cell.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ManageObjectsDelegate**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ManageItemsDelegate*

A delegate for the model and view in {Add/Edit}ObjectsDialog.

parent
parent dialog

Type *ManageItemsDialog*

createEditor(*self*, *parent*, *option*, *index*)
Return editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ManageRelationshipClassesDelega**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ManageItemsDelegate*

A delegate for the model and view in {Add/Edit}RelationshipClassesDialog.

parent
parent dialog

Type *ManageItemsDialog*

createEditor(*self*, *parent*, *option*, *index*)
Return editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**ManageRelationshipsDelegate**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ManageItemsDelegate*

A delegate for the model and view in {Add/Edit}RelationshipsDialog.

parent
parent dialog

Type *ManageItemsDialog*

createEditor(*self*, *parent*, *option*, *index*)
Return editor.

class spinetoolbox.spine_db_editor.widgets.custom_delegates.**RemoveEntitiesDelegate**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_delegates.ManageItemsDelegate*

A delegate for the model and view in RemoveEntitiesDialog.

parent
parent dialog

Type *ManageItemsDialog*

createEditor(*self*, *parent*, *option*, *index*)
Return editor.

spinetoolbox.spine_db_editor.widgets.custom_menus

Classes for custom context menus and pop-up menus.

author

M. Marin (KTH)

date 13.5.2020

Module Contents

Classes

ParameterViewFilterMenu	Filter menu.
TabularViewFilterMenu	Filter menu to use together with FilterWidget in TabularViewMixin.

class spinetoolbox.spine_db_editor.widgets.custom_menus.**ParameterViewFilterMenu**(*parent*, *source_model*, *field*, *show_empty=T*

Bases: *spinetoolbox.widgets.custom_menus.FilterMenuBase*

Filter menu.

Parameters

- **parent** (SpineDBEditor) –
- **source_model** (CompoundParameterModel) – a model to lazily get data from

- **field** (*str*) – the field name

filterChanged

_handle_source_model_refreshed(*self*)
Updates the menu to only present values that are actually shown in the source model.

set_filter_accepted_values(*self*, *accepted_values*)

set_filter_rejected_values(*self*, *rejected_values*)

_get_value_to_remove(*self*, *action*, *db_map*, *db_item*)

_get_value_to_add(*self*, *action*, *db_map*, *db_item*)

modify_menu_data(*self*, *action*, *db_map*, *db_items*)
Modifies data in the menu.

Parameters

- **action** (*str*) – either 'add', 'remove', or 'update'
- **db_map** (*DiffDatabaseMapping*) –
- **db_items** (*list(dict)*) –

_build_auto_filter(*self*, *valid_values*)
Builds the auto filter given valid values.

Parameters **valid_values** (*Sequence*) – Values accepted by the filter.

Returns mapping db_map, to entity_class_id, to set of accepted parameter_value/definition ids

Return type dict

emit_filter_changed(*self*, *valid_values*)
Builds auto filter and emits signal.

Parameters **valid_values** (*Sequence*) – Values accepted by the filter.

class spinetoolbox.spine_db_editor.widgets.custom_menus.**TabularViewFilterMenu**(*parent*, *identifier*, *data_to_value*, *show_empty=True*)

Bases: *spinetoolbox.widgets.custom_menus.FilterMenuBase*

Filter menu to use together with FilterWidget in TabularViewMixin.

Parameters

- **parent** (SpineDBEditor) –
- **identifier** (*int*) – index identifier
- **data_to_value** (*method*) – a method to translate item data to a value for display role

filterChanged

emit_filter_changed(*self*, *valid_values*)

event(*self*, *event*)

spinetoolbox.spine_db_editor.widgets.custom_qgraphicsviews

Classes for custom QGraphicsViews for the Entity graph view.

authors

P. Savolainen (VTT), M. Marin (KTH)

date 6.2.2018

Module Contents

Classes

EntityQGraphicsView	QGraphicsView for the Entity Graph View.

class spinetoolbox.spine_db_editor.widgets.custom_qgraphicsviews.**EntityQGraphicsView**(*parent*)

Bases: *spinetoolbox.widgets.custom_qgraphicsviews.CustomQGraphicsView*

QGraphicsView for the Entity Graph View.

Parameters **parent** (*QWidget*) – Graph View Form's (QMainWindow) central widget (self.centralwidget)

set_cross_hairs_items(*self*, *relationship_class*, *cross_hairs_items*)

Sets 'cross_hairs' items for relationship creation.

Parameters

- **relationship_class** (*dict*) –
- **cross_hairs_items** (*list(QGraphicsItems)*) –

clear_cross_hairs_items(*self*)

connect_data_store_form(*self*, *spine_db_editor*)

create_context_menu(*self*)

edit_selected(*self*)

Edits selected items using the connected Spine db editor.

remove_selected(*self*)

Removes selected items using the connected Spine db editor.

mousePressEvent(*self*, *event*)

Handles relationship creation if one it's in process.

mouseMoveEvent(*self*, *event*)

Updates the hovered object item if we're in relationship creation mode.

_update_cross_hairs_pos(*self*, *pos*)

Updates the hovered object item and sets the 'cross_hairs' icon accordingly.

Parameters **pos** (*QPoint*) – the desired position in view coordinates

mouseReleaseEvent(*self*, *event*)

Reestablish scroll hand drag mode.

keyPressEvent(*self*, *event*)

Aborts relationship creation if user presses ESC.

contextMenuEvent(*self*, *e*)
Shows context menu.

Parameters **e** (*QContextMenuEvent*) – Context menu event

_use_smooth_zoom(*self*)

_zoom(*self*, *factor*)

apply_zoom(*self*)

wheelEvent(*self*, *event*)
Zooms in/out. If user has pressed the shift key, rotates instead.

Parameters **event** (*QWheelEvent*) – Mouse wheel event

_handle_rotation_time_line_advanced(*self*, *pos*)
Performs rotation whenever the smooth rotation time line advances.

_rotate(*self*, *angle*)

rotate_clockwise(*self*)
Performs a rotate clockwise with fixed angle.

rotate_anticlockwise(*self*)
Performs a rotate anticlockwise with fixed angle.

spinetoolbox.spine_db_editor.widgets.custom_qtableview

Custom QTableView classes that support copy-paste and the like.

author

M. Marin (KTH)

date 18.5.2018

Module Contents

Classes

ParameterTableView	Custom QTableView class with autofilter functionality.
ObjectParameterTableMixin	
RelationshipParameterTableMixin	
ParameterDefinitionTableView	Custom QTableView class with autofilter functionality.
ParameterValueTableView	Custom QTableView class with autofilter functionality.
ObjectParameterDefinitionTableView	A custom QTableView for the object parameter_definition pane in Spine db editor.
RelationshipParameterDefinitionTableView	A custom QTableView for the relationship parameter_definition pane in Spine db editor.
ObjectParameterValueTableView	A custom QTableView for the object parameter_value pane in Spine db editor.
RelationshipParameterValueTableView	A custom QTableView for the relationship parameter_value pane in Spine db editor.
PivotTableView	Custom QTableView class with pivot capabilities.
FrozenTableView	

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**ParameterTableView**(*parent*)

Bases: *spinetoolbox.widgets.custom_qtableview.AutoFilterCopyPasteTableView*

Custom QTableView class with autofilter functionality.

parent
The parent of this view

Type QWidget

Initialize the view.

value_column_header
Either "default value" or "value". Used to identifiy the value column for advanced editting and plotting.

connect_data_store_form(*self*, *spine_db_editor*)
Connects a Spine db editor to work with this view.

Parameters **spine_db_editor** (SpineDBEditor) –

_make_delegate(*self*, *column_name*, *delegate_class*)
Creates a delegate for the given column and returns it.

Parameters

- **column_name** (*str*) –
- **delegate_class** (ParameterDelegate) –

Returns ParameterDelegate

create_delegates(*self*)
Creates delegates for this view

open_in_editor(*self*)
Opens the current index in a parameter_value editor using the connected Spine db editor.

plot(*self*, *checked=False*)
Plots current index.

plot_in_window(*self*, *action*)
Plots current index in the window given by action's name.

create_context_menu(*self*)
Creates a context menu for this view.

contextMenuEvent(*self*, *event*)
Shows context menu.

Parameters **event** (*QContextMenuEvent*) –

_selected_rows_per_column(*self*)
Computes selected rows per column.

Returns Mapping columns to selected rows in that column.

Return type dict

filter_by_selection(*self*, *checked=False*)

filter_excluding_selection(*self*, *checked=False*)

remove_selected(*self*)
Removes selected indexes.

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**ObjectParameterTableMixin**

create_delegates(*self*)

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**RelationshipParameterTableMix**

create_delegates(*self*)

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**ParameterDefinitionTableView**(*p*
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtableview.ParameterTableView*

Custom QTableView class with autofilter functionality.

parent
The parent of this view

Type QWidget

Initialize the view.

value_column_header
Either "default value" or "value". Used to identifiy the value column for advanced editting and plotting.

create_delegates(*self*)
Creates delegates for this view

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**ParameterValueTableView**(*parent*)
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtableview.ParameterTableView*

Custom QTableView class with autofilter functionality.

parent
The parent of this view

Type QWidget

Initialize the view.

value_column_header
Either "default value" or "value". Used to identifiy the value column for advanced editting and plotting.

create_delegates(*self*)
Creates delegates for this view

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**ObjectParameterDefinitionTable**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtableview.ObjectParameterTableMixin*, *spinetoolbox.spine_db_editor.widgets.custom_qtableview.ParameterDefinitionTableView*

A custom QTableView for the object parameter_definition pane in Spine db editor.

Initialize the view.

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**RelationshipParameterDefiniti**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtableview.RelationshipParameterTableMixin*, *spinetoolbox.spine_db_editor.widgets.custom_qtableview.ParameterDefinitionTableView*

A custom QTableView for the relationship parameter_definition pane in Spine db editor.

Initialize the view.

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**ObjectParameterValueTableView**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtableview.ObjectParameterTableMixin*, *spinetoolbox.spine_db_editor.widgets.custom_qtableview.ParameterValueTableView*

A custom QTableView for the object parameter_value pane in Spine db editor.

Initialize the view.

create_delegates(*self*)
Creates delegates for this view

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**RelationshipParameterValueTabl**
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtableview.RelationshipParameterTableMixin*, *spinetoolbox.spine_db_editor.widgets.custom_qtableview.ParameterValueTableView*

A custom QTableView for the relationship parameter_value pane in Spine db editor.

Initialize the view.

create_delegates(*self*)
Creates delegates for this view

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**PivotTableView**(*parent=None*)
Bases: *spinetoolbox.widgets.custom_qtableview.CopyPasteTableView*

Custom QTableView class with pivot capabilities.

Initialize the class.

_REMOVE_OBJECT = Remove selected objects

_REMOVE_RELATIONSHIP = Remove selected relationships

_REMOVE_PARAMETER = Remove selected parameter definitions

source_model

db_mngr

db_map

connect_data_store_form(*self*, *spine_db_editor*)

create_context_menu(*self*)

remove_selected(*self*)

remove_values(*self*)

remove_objects(*self*)

remove_relationships(*self*)

remove_parameters(*self*)

open_in_editor(*self*)
Opens the parameter_value editor for the first selected cell.

plot(*self*)
Plots the selected cells in the pivot table.

contextMenuEvent(*self*, *event*)
Shows context menu.

Parameters **event** (*QContextMenuEvent*) –

_find_selected_indexes(*self*)

_update_actions_availability(*self*)

_update_actions_text(*self*)

_plot_in_window(*self*, *action*)

class spinetoolbox.spine_db_editor.widgets.custom_qtableview.**FrozenTableView**(*parent=None*)
Bases: PySide2.QtWidgets.QTableView

header_dropped

area

dragEnterEvent(*self*, *event*)

dragMoveEvent(*self*, *event*)

dropEvent(*self*, *event*)

spinetoolbox.spine_db_editor.widgets.custom_qtreeview

Classes for custom QTreeView.

author

M. Marin (KTH)

date 25.4.2018

Module Contents

Classes

EntityTreeView	Tree view base class for object and relationship tree views.
ObjectTreeView	Custom QTreeView class for the object tree in SpineDBEditor.
RelationshipTreeView	Custom QTreeView class for the relationship tree in SpineDBEditor.
ItemTreeView	Custom QTreeView class for parameter_value_list in SpineDBEditor.
AlternativeScenarioTreeView	Custom QTreeView class for the alternative scenario tree in SpineDBEditor.
ParameterValueListTreeView	Custom QTreeView class for parameter_value_list in SpineDBEditor.
ParameterTagTreeView	Custom QTreeView class for the parameter_tag tree in SpineDBEditor.

class spinetoolbox.spine_db_editor.widgets.custom_qtreeview.**EntityTreeView**(*parent*)
Bases: *spinetoolbox.widgets.custom_qtreeview.CopyTreeView*

Tree view base class for object and relationship tree views.

Initialize the view.

tree_selection_changed

connect_data_store_form(*self*, *spine_db_editor*)
Connects a Spine db editor to work with this view.

Parameters **spine_db_editor** (SpineDBEditor) –

_add_middle_actions(*self*)
Adds action at the middle of the context menu. Subclasses can reimplement at will.

_create_context_menu(*self*)
Creates a context menu for this view.

connect_signals(*self*)
Connects signals.

rowsInserted(*self*, *parent*, *start*, *end*)

rowsRemoved(*self*, *parent*, *start*, *end*)

_resize_first_column_to_contents(*self*, *_index=None*)

_handle_selection_changed(*self*, *selected*, *deselected*)
Classifies selection by item type and emits signal.

_refresh_selected_indexes(*self*)

refresh_active_member_indexes(*self*)

edit(*self*, *index*, *trigger*, *event*)
Edit all selected items.

clear_any_selections(*self*)
Clears the selection if any.

fully_expand(*self*)
Expands selected indexes and all their children.

fully_collapse(*self*)
Collapses selected indexes and all their children.

export_selected(*self*)
Exports data from selected indexes using the connected Spine db editor.

remove_selected(*self*)
Removes selected indexes using the connected Spine db editor.

manage_relationships(*self*)

contextMenuEvent(*self*, *event*)
Shows context menu.

Parameters **event** (*QContextMenuEvent*) –

mousePressEvent(*self*, *event*)
Overrides selection behaviour if the user has selected sticky selection in Settings. If sticky selection is enabled, multiple-selection is enabled when selecting items in the Object tree. Pressing the Ctrl-button down, enables single selection.

Parameters **event** (*QMouseEvent*) –

_add_relationship_actions(*self*)

update_actions_visibility(*self*, *item*)
Updates the visible property of actions according to whether or not they apply to given item.

edit_selected(*self*)
Edits all selected indexes using the connected Spine db editor.

class spinetoolbox.spine_db_editor.widgets.custom_qtreeview.**ObjectTreeView**(*parent*)
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtreeview.EntityTreeView*

Custom QTreeView class for the object tree in SpineDBEditor.

Initialize the view.

update_actions_visibility(*self*, *item*)
Updates the visible property of actions according to whether or not they apply to given item.

_add_middle_actions(*self*)
Adds action at the middle of the context menu. Subclasses can reimplement at will.

connect_signals(*self*)
Connects signals.

add_object_classes(*self*)

add_objects(*self*)

add_relationship_classes(*self*)

add_relationships(*self*)

find_next_relationship(*self*)
Finds the next occurrence of the relationship at the current index and expands it.

duplicate_object(*self*)
Duplicate the object at the current index using the connected Spine db editor.

add_object_group(*self*)

manage_object_group(*self*)

class spinetoolbox.spine_db_editor.widgets.custom_qtreeview.**RelationshipTreeView**(*parent*)
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtreeview.EntityTreeView*

Custom QTreeView class for the relationship tree in SpineDBEditor.

Initialize the view.

_add_middle_actions(*self*)
Adds action at the middle of the context menu. Subclasses can reimplement at will.

update_actions_visibility(*self*, *item*)
Updates the visible property of actions according to whether or not they apply to given item.

add_relationship_classes(*self*)

add_relationships(*self*)

class spinetoolbox.spine_db_editor.widgets.custom_qtreeview.**ItemTreeView**(*parent*)
Bases: *spinetoolbox.widgets.custom_qtreeview.CopyTreeView*

Custom QTreeView class for parameter_value_list in SpineDBEditor.

Initialize the view.

connect_signals(*self*)
Connects signals.

_resize_first_column_to_contents(*self*, *_index=None*)

remove_selected(*self*)
Removes items selected in the view.

update_actions_visibility(*self*, *item*)
Updates the visible property of actions according to whether or not they apply to given item.

connect_data_store_form(*self*, *spine_db_editor*)

create_context_menu(*self*)
Creates a context menu for this view.

contextMenuEvent(*self*, *event*)
Shows context menu.

Parameters **event** (*QContextMenuEvent*) –

class spinetoolbox.spine_db_editor.widgets.custom_qtreeview.**AlternativeScenarioTreeView**(*par*
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtreeview.ItemTreeView*

Custom QTreeView class for the alternative scenario tree in SpineDBEditor.

Initialize the view.

alternative_selection_changed

mouseMoveEvent(*self*, *e*)

connect_signals(*self*)
Connects signals.

_db_map_alt_ids_from_selection(*self*, *selection*)

_db_map_scen_alt_ids_from_selection(*self*, *selection*)

_handle_selection_changed(*self*, *selected*, *deselected*)
Emits alternative_selection_changed with the current selection.

remove_selected(*self*)
See base class.

update_actions_visibility(*self*, *item*)
See base class.

class spinetoolbox.spine_db_editor.widgets.custom_qtreeview.**ParameterValueListTreeView**(*pare*
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtreeview.ItemTreeView*

Custom QTreeView class for parameter_value_list in SpineDBEditor.

Initialize the view.

create_context_menu(*self*)
Creates a context menu for this view.

update_actions_visibility(*self*, *item*)
See base class.

open_in_editor(*self*)
Opens the parameter_value editor for the first selected cell.

remove_selected(*self*)
See base class.

class spinetoolbox.spine_db_editor.widgets.custom_qtreeview.**ParameterTagTreeView**(*parent*)
Bases: *spinetoolbox.spine_db_editor.widgets.custom_qtreeview.ItemTreeView*

Custom QTreeView class for the parameter_tag tree in SpineDBEditor.

Initialize the view.

tag_selection_changed

connect_signals(*self*)
Connects signals.

remove_selected(*self*)
See base class.

update_actions_visibility(*self*, *item*)
See base class.

_handle_selection_changed(*self*, *selected*, *deselected*)
Emits tag_selection_changed with the current selection.

_db_map_tag_ids_from_selection(*self*, *selection*)

spinetoolbox.spine_db_editor.widgets.custom_qwidgets

Custom QWidgets.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

Classes

DataToValueFilterWidget	Filter widget class.
LazyFilterWidget	Filter widget class.
OpenFileButton	A button to open files or show them in the folder.
OpenSQLiteFileButton	A button to open sqlite files, show them in the folder, or add them to the project.
ShootingLabel	
CustomInputDialog	

class spinetoolbox.spine_db_editor.widgets.custom_qwidgets.**DataToValueFilterWidget**(*parent*, *data_to_va*
show_empt

Bases: *spinetoolbox.widgets.custom_qwidgets.FilterWidgetBase*

Filter widget class.

Init class.

Parameters

- **parent** (*QWidget*) –
- **data_to_value** (*method*) – a method to translate item data to a value for display role

class spinetoolbox.spine_db_editor.widgets.custom_qwidgets.**LazyFilterWidget**(*parent*, *source_model*, *show_empty=True*)

Bases: *spinetoolbox.widgets.custom_qwidgets.FilterWidgetBase*

Filter widget class.

Init class.

Parameters

- **parent** (SpineDBEditor) –
- **source_model** (CompoundParameterModel, *optional*) – a model to lazily get data from

set_model(*self*)

class spinetoolbox.spine_db_editor.widgets.custom_qwidgets.**OpenFileButton**(*file_path*, *ds_form*)

Bases: PySide2.QtWidgets.QToolButton

A button to open files or show them in the folder.

open_file(*self*, *checked=False*)

open_containing_folder(*self*, *checked=False*)

class spinetoolbox.spine_db_editor.widgets.custom_qwidgets.**OpenSQLiteFileButton**(*file_path*, *ds_form*)

Bases: *spinetoolbox.spine_db_editor.widgets.custom_qwidgets.OpenFileButton*

A button to open sqlite files, show them in the folder, or add them to the project.

open_file(*self*, *checked=False*)

add_to_project(*self*, *checked=False*)

class spinetoolbox.spine_db_editor.widgets.custom_qwidgets.**ShootingLabel**(*origin*, *destination*, *parent=None*, *duration=1200*)

Bases: PySide2.QtWidgets.QLabel

_handle_value_changed(*self*, *value*)

show(*self*)

class spinetoolbox.spine_db_editor.widgets.custom_qwidgets.**CustomInputDialog**(*parent*, *title*)

Bases: PySide2.QtWidgets.QDialog

accept(*self*, *item=None*)

reject(*self*)

_handle_item_double_clicked(*self*, *item*)

_handle_item_changed(*self*, *item*)

classmethod get_item(*cls*, *parent*, *title*, *label*, *items*, *icons=None*, *editable_text=''*)

spinetoolbox.spine_db_editor.widgets.db_session_history_dialog

Classes to show db session history

author

M. Marin (KTH)

date 5.2.2020

Module Contents

Classes

DBSessionHistoryModel	
DBSessionHistoryView	
DBSessionHistoryDialog	Initialize class

class spinetoolbox.spine_db_editor.widgets.db_session_history_dialog.**DBSessionHistoryModel**

Bases: PySide2.QtGui.QStandardItemModel

build(*self*)

class spinetoolbox.spine_db_editor.widgets.db_session_history_dialog.**DBSessionHistoryView**(

Bases: PySide2.QtWidgets.QColumnView

class spinetoolbox.spine_db_editor.widgets.db_session_history_dialog.**DBSessionHistoryDialo**

Bases: PySide2.QtWidgets.QDialog

Initialize class

spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs

Classes for custom QDialogs to edit items in databases.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

Classes

EditOrRemoveItemsDialog	A dialog with a CopyPasteTableView and a QDialogButtonBox. Base class for all
EditObjectClassesDialog	A dialog to query user's preferences for updating object classes.
EditObjectsDialog	A dialog to query user's preferences for updating objects.
EditRelationshipClassesDialog	A dialog to query user's preferences for updating relationship classes.
EditRelationshipsDialog	A dialog to query user's preferences for updating relationships.
RemoveEntitiesDialog	A dialog to query user's preferences for removing tree items.

class spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.**EditOrRemoveItemsDi**

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.ManageItemsDialog*

A dialog with a CopyPasteTableView and a QDialogButtonBox. Base class for all dialogs to query user's preferences for adding/editing/managing data items.

Init class.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) –

all_databases(*self*, *row*)
Returns a list of db names available for a given row. Used by delegates.

class spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.**EditObjectClassesDi**

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.ShowIconColorEditorMixin*, *spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for updating object classes.

Init class.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) – the manager to do the update
- **selected** (*set*) – set of ObjectClassItem instances to edit

connect_signals(*self*)
Connect signals to slots.

all_db_maps(*self*, *row*)
Returns a list of db maps available for a given row. Used by ShowIconColorEditorMixin.

accept(*self*)
Collect info from dialog and try to update items.

class spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.**EditObjectsDialog**(

Bases: *spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for updating objects.

Init class.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) – the manager to do the update
- **selected** (*set*) – set of ObjectItem instances to edit

accept(*self*)
Collect info from dialog and try to update items.

class spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.**EditRelationshipCla**

Bases: *spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for updating relationship classes.

Init class.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) – the manager to do the update
- **selected** (*set*) – set of RelationshipClassItem instances to edit

accept(*self*)
Collect info from dialog and try to update items.

class spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.**EditRelationshipsDi**

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.GetRelationshipClassesMixin*, *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.GetObjectsMixin*, *spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for updating relationships.

Init class.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) – the manager to do the update
- **selected** (*set*) – set of RelationshipItem instances to edit

- **class_key** (*tuple*) – (class_name, object_class_name_list) for identifying the relationship_class

accept(*self*)
Collect info from dialog and try to update items.

class spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.**RemoveEntitiesDialo**

Bases: *spinetoolbox.spine_db_editor.widgets.edit_or_remove_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for removing tree items.

Init class.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) – the manager to do the removal
- **selected** (*dict*) – maps item type (class) to instances

accept(*self*)
Collect info from dialog and try to remove items.

spinetoolbox.spine_db_editor.widgets.graph_layout_generator

Contains the GraphViewMixin class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

Classes

_State	State of GraphLayoutGenerator.
ProgressBarWidget	
GraphLayoutGenerator	Computes the layout for the Entity Graph View.

Functions

make_heat_map(x, y, values)	

spinetoolbox.spine_db_editor.widgets.graph_layout_generator.**make_heat_map**(*x*, *y*, *values*)

class spinetoolbox.spine_db_editor.widgets.graph_layout_generator.**_State**

Bases: enum.Enum

State of GraphLayoutGenerator.

Create and return a new object. See help(type) for accurate signature.

ACTIVE

STOPPED

CANCELLED

class spinetoolbox.spine_db_editor.widgets.graph_layout_generator.**ProgressBarWidget**(*layout_ge*
par-
ent)

Bases: PySide2.QtWidgets.QWidget

paintEvent(*self*, *event*)

class spinetoolbox.spine_db_editor.widgets.graph_layout_generator.**GraphLayoutGenerator**(*verte*
src_i
dst_i
spread
heavy
it-
er-
a-
tions
weig
2)

Bases: PySide2.QtCore.QObject

Computes the layout for the Entity Graph View.

finished

started

progressed

done

show_progress_widget(*self*, *parent*)

clean_up(*self*)

stop(*self*)

cancel(*self*, *checked=False*)

start(*self*)

shortest_path_matrix(*self*)
Returns the shortest-path matrix.

sets(*self*)
Returns sets of vertex pairs indices.

emit_finished(*self*, *x*, *y*)

get_coordinates(*self*)
Computes and returns x and y coordinates for each vertex in the graph, using VSGD-MS.

spinetoolbox.spine_db_editor.widgets.graph_view_mixin

Contains the GraphViewMixin class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

Classes

GraphViewMixin	Provides the graph view for the DS form.

class spinetoolbox.spine_db_editor.widgets.graph_view_mixin.**GraphViewMixin**(**args*, ***kwargs*)

Provides the graph view for the DS form.

Initialize self. See help(type(self)) for accurate signature.

_POS_PARAM_NAME = entity_graph_position

_VERTEX_EXTENT = 64

_ARC_WIDTH

_ARC_LENGTH_HINT

graph_selection_changed

graph_build_finished

populate_graph_db_action_group(*self*)

_update_graph_db_map(*self*, *action*)

add_menu_actions(*self*)
Adds toggle view actions to View menu.

init_models(*self*)

connect_signals(*self*)
Connects signals.

_handle_scene_selection_changed(*self*)
Filters parameters by selected objects in the graph.

set_show_cascading_relationships(*self*, *checked*)

setup_widget_actions(*self*)
Setups zoom and rotate widget action in view menu.

receive_objects_added(*self*, *db_map_data*)
Runs when objects are added to the db. Adds the new objects to the graph if needed.

Parameters **db_map_data** (*dict*) – list of dictionary-items keyed by DiffDatabaseMapping instance.

receive_relationships_added(*self*, *db_map_data*)
Runs when relationships are added to the db. Adds the new relationships to the graph if needed.

Parameters **db_map_data** (*dict*) – list of dictionary-items keyed by DiffDatabaseMapping instance.

receive_object_classes_updated(*self*, *db_map_data*)

receive_relationship_classes_updated(*self*, *db_map_data*)

receive_objects_updated(*self*, *db_map_data*)
Runs when objects are updated in the db. Refreshes names of objects in graph.

Parameters **db_map_data** (*dict*) – list of dictionary-items keyed by DiffDatabaseMapping instance.

receive_objects_removed(*self*, *db_map_data*)
Runs when objects are removed from the db. Rebuilds graph if needed.

Parameters **db_map_data** (*dict*) – list of dictionary-items keyed by DiffDatabaseMapping instance.

receive_relationships_removed(*self*, *db_map_data*)
Runs when relationships are removed from the db. Rebuilds graph if needed.

Parameters **db_map_data** (*dict*) – list of dictionary-items keyed by DiffDatabaseMapping instance.

restore_removed_entities(*self*, *added_ids*)
Restores any entities that have been previously removed and returns their ids. This happens in the context of undo/redo.

Parameters **added_ids** (*set(int)*) – Set of newly added ids.

Returns set(int)

hide_removed_entities(*self*, *db_map_data*)
Hides removed entities while saving them into a list attribute. This allows entities to be restored in case the user undoes the operation.

refresh_icons(*self*, *db_map_data*)
Runs when entity classes are updated in the db. Refreshes icons of entities in graph.

Parameters **db_map_data** (*dict*) – list of dictionary-items keyed by DiffDatabaseMapping instance.

_handle_menu_graph_about_to_show(*self*)
Enables or disables actions according to current selection in the graph.

_handle_entity_graph_visibility_changed(*self*, *visible*)

rebuild_graph(*self*, *selected*)
Stores the given selection of entity tree indexes and builds graph.

build_graph(*self*, *persistent=False*)
Builds the graph.

Parameters **persistent** (*bool, optional*) – If True, builds the graph on top of the current one.

_complete_graph(*self*, *x*, *y*)

Parameters

- **x** (*list*) – Horizontal coordinates
- **y** (*list*) – Vertical coordinates

_get_selected_entity_ids(*self*)
Returns a set of ids corresponding to selected entities in the trees.

Returns selected object ids set: selected relationship ids

Return type set

_get_all_relationships_for_graph(*self*, *object_ids*, *relationship_ids*)

_update_graph_data(*self*)
Updates data for graph according to selection in trees.

_update_src_dst_inds(*self*, *object_id_lists*)

_make_layout_generator(*self*)
Returns a layout generator for the current graph.

Returns GraphLayoutGenerator

_make_new_items(*self*, *x*, *y*)
Returns new items for the graph.

Parameters

- **x** (*list*) –
- **y** (*list*) –

_add_new_items(*self*)

hide_selected_items(*self*, *checked=False*)
Hides selected items.

show_hidden_items(*self*, *checked=False*)
Shows hidden items.

_get_selected_entity_names(*self*)

_get_selected_class_names(*self*)

prune_selected_entities(*self*, *checked=False*)
Prunes selected items.

prune_selected_classes(*self*, *checked=False*)
Prunes selected items.

restore_all_pruned_items(*self*, *checked=False*)
Reinstates all pruned items.

restore_pruned_items(*self*, *key*)
Reinstates last pruned items.

edit_entity_graph_items(*self*)
Starts editing given indexes.

remove_entity_graph_items(*self*, *checked=False*)
Removes all selected items in the graph.

save_positions(*self*, *checked=False*)

clear_saved_positions(*self*, *checked=False*)

export_as_pdf(*self*, *checked=False*)

start_relationship(*self*, *relationship_class*, *obj_item*)
Starts a relationship from the given object item.

Parameters

- **relationship_class** (*dict*) –
- **obj_item** (*.graphics_items.ObjectItem*) –

finalize_relationship(*self*, *relationship_class*, **object_items*)
Tries to add relationships between the given object items.

Parameters

- **relationship_class** (*dict*) –
- **object_items** (*.graphics_items.ObjectItem*) –

_begin_add_relationships(*self*)

_end_add_relationships(*self*)

_populate_menu_add_parameter_heat_map(*self*)
Populates the menu 'Add parameter heat map' with parameters for currently shown items in the graph.

add_heat_map(*self*, *action*)
Adds heat map for the parameter in the action text.

_clean_up_heat_map_items(*self*)

closeEvent(*self*, *event*)
Handle close window.

Parameters **event** (*QCloseEvent*) – Closing event

spinetoolbox.spine_db_editor.widgets.manage_items_dialogs

Classes for custom QDialogs to add edit and remove database items.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

Classes

ManageItemsDialogBase	Init class.
ManageItemsDialog	A dialog with a CopyPasteTableView and a QDialogButtonBox. Base class for all
GetObjectClassesMixin	Provides a method to retrieve object classes for AddObjectsDialog and AddRelationshipClassesDialog.
GetObjectsMixin	Provides a method to retrieve objects for AddRelationshipsDialog and EditRelationshipsDialog.
GetRelationshipClassesMixin	Provides a method to retrieve relationships for AddRelationshipsDialog and EditRelationshipsDialog.
ShowIconColorEditorMixin	Provides methods to show an *IconColorEditor* upon request.

class spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.**ManageItemsDialogBase**(*parent*, *db_mn*

Bases: PySide2.QtWidgets.QDialog

Init class.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) –

make_table_view(*self*)

connect_signals(*self*)
Connect signals to slots.

resize_window_to_columns(*self*, *height=None*)

class spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.**ManageItemsDialog**(*parent*, *db_mngr*)

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.ManageItemsDialogBase*

A dialog with a CopyPasteTableView and a QDialogButtonBox. Base class for all dialogs to query user's preferences for adding/editing/managing data items.

Init class.

Parameters

- **parent** (SpineDBEditor) – data store widget
- **db_mngr** (SpineDBManager) –

connect_signals(*self*)
Connect signals to slots.

_handle_model_data_changed(*self*, *top_left*, *bottom_right*, *roles*)
Reimplement in subclasses to handle changes in model data.

set_model_data(*self*, *index*, *data*)
Update model data.

_handle_model_reset(*self*)
Resize columns and form.

class spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.**GetObjectClassesMixin**
Provides a method to retrieve object classes for AddObjectsDialog and AddRelationshipClassesDialog.

make_db_map_obj_cls_lookup(*self*)

object_class_name_list(*self*, *row*)
Return a list of object_class names present in all databases selected for given row. Used by *ManageObjectsDelegate*.

class spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.**GetObjectsMixin**
Provides a method to retrieve objects for AddRelationshipsDialog and EditRelationshipsDialog.

make_db_map_obj_lookup(*self*)

object_name_list(*self*, *row*, *column*)
Return a list of object names present in all databases selected for given row. Used by *ManageRelationshipsDelegate*.

class spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.**GetRelationshipClassesMixin**
Provides a method to retrieve relationships for AddRelationshipsDialog and EditRelationshipsDialog.

make_db_map_rel_cls_lookup(*self*)

class spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.**ShowIconColorEditorMixin**
Provides methods to show an *IconColorEditor* upon request.

show_icon_color_editor(*self*, *index*)

create_object_pixmap(*self*, *index*)

spinetoolbox.spine_db_editor.widgets.object_name_list_editor

Contains the ObjectNameListEditor class.

author

M. Marin (KTH)

date 27.11.2019

Module Contents

Classes

SearchBarDelegate	A custom delegate to use with ObjectNameListEditor.
ObjectNameListEditor	A dialog to select the object name list for a relationship using Google-like search bars.

class spinetoolbox.spine_db_editor.widgets.object_name_list_editor.**SearchBarDelegate**
Bases: PySide2.QtWidgets.QItemDelegate

A custom delegate to use with ObjectNameListEditor.

data_committed

setModelData(*self*, *editor*, *model*, *index*)

createEditor(*self*, *parent*, *option*, *index*)

updateEditorGeometry(*self*, *editor*, *option*, *index*)

close_editor(*self*, *editor*, *index*, *model*)

eventFilter(*self*, *editor*, *event*)

class spinetoolbox.spine_db_editor.widgets.object_name_list_editor.**ObjectNameListEditor**(*par*
in-
dex
ob-
ject
ob-
ject
cur-
ren

Bases: *spinetoolbox.spine_db_editor.widgets.manage_items_dialogs.ManageItemsDialog*

A dialog to select the object name list for a relationship using Google-like search bars.

Initializes widget.

Parameters

- **parent** (SpineDBEditor) –
- **index** (*QModelIndex*) –
- **object_class_names** (*list*) – string object_class names
- **object_names_lists** (*list*) – lists of string object names
- **current_object_names** (*list*) –

init_model(*self*, *object_class_names*, *object_names_lists*, *current_object_names*)

accept(*self*)

spinetoolbox.spine_db_editor.widgets.parameter_view_mixin

Contains the ParameterViewMixin class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

Classes

ParameterViewMixin	Provides stacked parameter tables for the Spine db editor.

class spinetoolbox.spine_db_editor.widgets.parameter_view_mixin.**ParameterViewMixin**(**args*, ***kwargs*)

Provides stacked parameter tables for the Spine db editor.

Initialize self. See help(type(self)) for accurate signature.

add_menu_actions(*self*)
Adds toggle view actions to View menu.

connect_signals(*self*)
Connects signals to slots.

init_models(*self*)
Initializes models.

set_parameter_data(*self*, *index*, *new_value*)
Updates (object or relationship) parameter_definition or value with newly edited data.

show_object_name_list_editor(*self*, *index*, *rel_cls_id*, *db_map*)
Shows the object names list editor.

Parameters

- **index** (*QModelIndex*) –

- **rel_cls_id** (*int*) –
- **db_map** (*DiffDatabaseMapping*) –

set_default_parameter_data (*self*, *index=None*)
Sets default rows for parameter models according to given index.

Parameters **index** (*QModelIndex*) – and index of the object or relationship tree

_get_filter_class_ids (*self*)
Returns filter class ids by combining filter class ids from entity tree *and* parameter_tag toolbar.

Returns mapping db maps to sets of ids, or None if no filter (none shall pass)

Return type dict, NoneType

reset_filters (*self*)
Resets filters.

_handle_graph_selection_changed (*self*, *selected_items*)
Resets filter according to graph selection.

_handle_object_tree_selection_changed (*self*, *selected_indexes*)
Resets filter according to object tree selection.

_handle_relationship_tree_selection_changed (*self*, *selected_indexes*)
Resets filter according to relationship tree selection.

_handle_alternative_selection_changed (*self*, *selected_db_map_alt_ids*)
Resets filter according to selection in alternative tree view.

_handle_tag_selection_changed (*self*, *selected_db_map_tag_ids*)
Resets filter according to selection in parameter_tag tree view.

restore_ui (*self*)
Restores UI state from previous session.

save_window_state (*self*)
Saves window state parameters (size, position, state) via QSettings.

receive_alternatives_updated (*self*, *db_map_data*)

receive_parameter_tags_fetched (*self*, *db_map_data*)

receive_parameter_definitions_fetched (*self*, *db_map_data*)

receive_parameter_values_fetched (*self*, *db_map_data*)

receive_parameter_tags_added (*self*, *db_map_data*)

receive_parameter_definitions_added (*self*, *db_map_data*)

receive_parameter_values_added (*self*, *db_map_data*)

receive_parameter_tags_updated (*self*, *db_map_data*)

receive_parameter_definitions_updated (*self*, *db_map_data*)

receive_parameter_values_updated (*self*, *db_map_data*)

receive_parameter_definition_tags_set (*self*, *db_map_data*)

receive_object_classes_removed (*self*, *db_map_data*)

receive_relationship_classes_removed (*self*, *db_map_data*)

receive_parameter_tags_removed (*self*, *db_map_data*)

receive_parameter_definitions_removed(*self*, *db_map_data*)

receive_parameter_values_removed(*self*, *db_map_data*)

spinetoolbox.spine_db_editor.widgets.pivot_table_header_view

Contains custom QHeaderView for the pivot table.

author

M. Marin (KTH)

date 2.12.2019

Module Contents

Classes

PivotTableHeaderView

class spinetoolbox.spine_db_editor.widgets.pivot_table_header_view.**PivotTableHeaderView**(*orie*
are
pivo

Bases: PySide2.QtWidgets.QHeaderView

header_dropped

area

dragEnterEvent(*self*, *event*)

dragMoveEvent(*self*, *event*)

dropEvent(*self*, *event*)

contextMenuEvent(*self*, *event*)
Shows context menu.

Parameters **event** (*QContextMenuEvent*) –

_add_column_to_plot(*self*, *action*)
Adds a single column to existing plot window.

_plot_column(*self*)
Plots a single column not the selection.

_set_x_flag(*self*)
Sets the X flag for a column.

spinetoolbox.spine_db_editor.widgets.select_db_items_dialogs

Classes for custom QDialogs to add edit and remove database items.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

Classes

SelectDBItemsDialog	A dialog to query a selection of dbs and items from the user.
MassRemoveItemsDialog	A dialog to query user's preferences for mass removing db items.
MassExportItemsDialog	A dialog to let users chose items for JSON export.

class spinetoolbox.spine_db_editor.widgets.select_db_items_dialogs.**SelectDBItemsDialog**(*parer*
db_m
**db_*

Bases: PySide2.QtWidgets.QDialog

A dialog to query a selection of dbs and items from the user.

Initialize class.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- **db_maps** (*DiffDatabaseMapping*) – the dbs to select items from

_MARGIN = 3

_ITEM_TYPES = ['object_class', 'relationship_class', 'parameter_value_list', 'paramete

_COLUMN_COUNT = 2

_set_item_check_box_enabled(*self*)
Set the enabled property on item check boxes depending on the state of db_map check boxes.

class spinetoolbox.spine_db_editor.widgets.select_db_items_dialogs.**MassRemoveItemsDialog**(*pa*
db
**a*

Bases: *spinetoolbox.spine_db_editor.widgets.select_db_items_dialogs.SelectDBItemsDialog*

A dialog to query user's preferences for mass removing db items.

Initialize class.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- **db_maps** (*DiffDatabaseMapping*) – the dbs to select items from

accept(*self*)

class spinetoolbox.spine_db_editor.widgets.select_db_items_dialogs.**MassExportItemsDialog**(*pa*
db
**a*

Bases: *spinetoolbox.spine_db_editor.widgets.select_db_items_dialogs.SelectDBItemsDialog*

A dialog to let users chose items for JSON export.

Initialize class.

Parameters

- **parent** (SpineDBEditor) –
- **db_mngr** (SpineDBManager) –
- **db_maps** (*DiffDatabaseMapping*) – the dbs to select items from

data_submitted

accept(*self*)

spinetoolbox.spine_db_editor.widgets.spine_db_editor

Contains the SpineDBEditor class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

Classes

SpineDBEditorBase	Base class for SpineDBEditor (i.e. Spine database editor).
SpineDBEditor	A widget to visualize Spine dbs.

class spinetoolbox.spine_db_editor.widgets.spine_db_editor.**SpineDBEditorBase**(*db_mngr*, **db_maps*)

Bases: PySide2.QtWidgets.QMainWindow

Base class for SpineDBEditor (i.e. Spine database editor).

Initializes form.

Parameters

- **db_mngr** (SpineDBManager) – The manager to use
- ***db_maps** (*DiffDatabaseMapping*) – The db map to visualize.

msg

link_msg

msg_error

error_box

first_db_map

_make_db_menu(*self*)

add_menu_actions(*self*)
Adds actions to View and Edit menu.

connect_signals(*self*)
Connects signals to slots.

update_undo_redo_actions(*self*, *index*)

update_commit_enabled(*self*, *_clean=False*)

show_history_dialog(*self*, *checked=False*)

init_models(*self*)
Initializes models.

add_message(*self*, *msg*)
Pushes message to notification stack.

Parameters **msg** (*str*) – String to show in the notification

add_link_msg(*self*, *msg*, *open_link=None*)
Pushes link message to notification stack.

Parameters **msg** (*str*) – String to show in notification

restore_dock_widgets(*self*)
Docks all floating and or hidden QDockWidgets back to the window.

_handle_menu_edit_about_to_show(*self*)
Runs when the edit menu from the main menubar is about to show. Enables or disables actions according to selection status.

remove_selected(*self*, *checked=False*)
Removes selected items.

edit_selected(*self*, *checked=False*)
Edits selected items.

copy(*self*, *checked=False*)
Copies data to clipboard.

paste(*self*, *checked=False*)
Pastes data from clipboard.

import_data(*self*, *data*)

import_file(*self*, *checked=False*)
Import file. It supports SQLite, JSON, and Excel.

import_from_json(*self*, *file_path*)

import_from_sqlite(*self*, *file_path*)

import_from_excel(*self*, *file_path*)

static _make_data_for_export(*db_map_item_ids*)

show_mass_export_items_dialog(*self*, *checked=False*)
Shows dialog for user to select dbs and items for export.

export_session(*self*, *checked=False*)
Exports changes made in the current session as reported by DiffDatabaseMapping.

mass_export_items(*self*, *db_map_item_types*)

export_data(*self*, *db_map_ids_for_export*)
Exports data from given dictionary into a file.

Parameters **db_map_ids_for_export** – Dictionary mapping db maps to keyword arguments for spinedb_api.export_data

export_to_sqlite(*self*, *file_path*, *data_for_export*)
Exports given data into SQLite file.

_open_sqlite_url(*self*, *url*, *codename*)
Opens sqlite url.

_add_sqlite_url_to_project(*self*, *url*)
Adds sqlite url to project.

_undo_add_sqlite_url_to_project(*self*, *_*, *stack*, *index*)

export_to_json(*self*, *file_path*, *data_for_export*)
Exports given data into JSON file.

export_to_excel(*self*, *file_path*, *data_for_export*)
Exports given data into Excel file.

_insert_open_file_button(*self*, *file_path*)

_insert_open_sqlite_file_button(*self*, *file_path*)

_insert_button_to_exports_widget(*self*, *button*)
Inserts given button to the 'beginning' of the status bar and decorates the thing with a shooting label.

_handle_exports_visibility_changed(*self*, *visible*)
Remove all buttons when exports dock is closed.

reload_session(*self*, *db_maps*)
Reloads data from given db_maps.

refresh_session(*self*, *checked=False*)

commit_session(*self*, *checked=False*)
Commits session.

rollback_session(*self*, *checked=False*)

receive_session_committed(*self*, *db_maps*, *cookie*)

receive_session_rolled_back(*self*, *db_maps*)

receive_session_refreshed(*self*, *db_maps*)

show_mass_remove_items_form(*self*, *checked=False*)

show_parameter_value_editor(*self*, *index*)
Shows the parameter_value editor for the given index of given table view.

show_user_guide(*self*, *checked=False*)
Opens Spine Toolbox documentation Spine db editor page in browser.

notify_items_changed(*self*, *action*, *item_type*, *db_map_data*)
Enables or disables actions and informs the user about what just happened.

receive_scenarios_fetched(*self*, *db_map_data*)

receive_alternatives_fetched(*self*, *db_map_data*)

receive_object_classes_fetched(*self*, *db_map_data*)

receive_objects_fetched(*self*, *db_map_data*)

receive_relationship_classes_fetched(*self*, *db_map_data*)

receive_relationships_fetched(*self*, *db_map_data*)

receive_entity_groups_fetched(*self*, *db_map_data*)

receive_parameter_definitions_fetched(*self*, *db_map_data*)

receive_parameter_values_fetched(*self*, *db_map_data*)

receive_parameter_value_lists_fetched(*self*, *db_map_data*)

receive_parameter_tags_fetched(*self*, *db_map_data*)

receive_scenarios_added(*self*, *db_map_data*)

receive_alternatives_added(*self*, *db_map_data*)

receive_object_classes_added(*self*, *db_map_data*)

receive_objects_added(*self*, *db_map_data*)

receive_relationship_classes_added(*self*, *db_map_data*)

receive_relationships_added(*self*, *db_map_data*)

receive_entity_groups_added(*self*, *db_map_data*)

receive_parameter_definitions_added(*self*, *db_map_data*)

receive_parameter_values_added(*self*, *db_map_data*)

receive_parameter_value_lists_added(*self*, *db_map_data*)

receive_parameter_tags_added(*self*, *db_map_data*)

receive_scenarios_updated(*self*, *db_map_data*)

receive_alternatives_updated(*self*, *db_map_data*)

receive_object_classes_updated(*self*, *db_map_data*)

receive_objects_updated(*self*, *db_map_data*)

receive_relationship_classes_updated(*self*, *db_map_data*)

receive_relationships_updated(*self*, *db_map_data*)

receive_parameter_definitions_updated(*self*, *db_map_data*)

receive_parameter_values_updated(*self*, *db_map_data*)

receive_parameter_value_lists_updated(*self*, *db_map_data*)

receive_parameter_tags_updated(*self*, *db_map_data*)

receive_parameter_definition_tags_set(*self*, *db_map_data*)

receive_scenarios_removed(*self*, *db_map_data*)

receive_alternatives_removed(*self*, *db_map_data*)

receive_object_classes_removed(*self*, *db_map_data*)

receive_objects_removed(*self*, *db_map_data*)

receive_relationship_classes_removed(*self*, *db_map_data*)

receive_relationships_removed(*self*, *db_map_data*)

receive_entity_groups_removed(*self*, *db_map_data*)

receive_parameter_definitions_removed(*self*, *db_map_data*)

receive_parameter_values_removed(*self*, *db_map_data*)

receive_parameter_value_lists_removed(*self*, *db_map_data*)

receive_parameter_tags_removed(*self*, *db_map_data*)

restore_ui(*self*)
Restore UI state from previous session.

save_window_state(*self*)
Save window state parameters (size, position, state) via QSettings.

closeEvent(*self*, *event*)
Handle close window.

Parameters **event** (*QCloseEvent*) – Closing event

_focused_widget_has_callable(*self*, *callable_name*)
Returns True if the currently focused widget or one of its ancestors has the given callable.

_call_on_focused_widget(*self*, *callable_name*)
Calls the given callable on the currently focused widget or one of its ancestors.

class spinetoolbox.spine_db_editor.widgets.spine_db_editor.**SpineDBEditor**(*db_mngr*, **db_urls*)

Bases: *spinetoolbox.spine_db_editor.widgets.tabular_view_mixin.TabularViewMixin*, *spinetoolbox.spine_db_editor.widgets.graph_view_mixin.GraphViewMixin*, *spinetoolbox.spine_db_editor.widgets.parameter_view_mixin.ParameterViewMixin*, *spinetoolbox.spine_db_editor.widgets.tree_view_mixin.TreeViewMixin*, *spinetoolbox.spine_db_editor.widgets.spine_db_editor.SpineDBEditorBase*

A widget to visualize Spine dbs.

Initializes everything.

Parameters

- **db_mngr** (SpineDBManager) – The manager to use
- ***db_urls** (*tuple*) – Database url, codename.

connect_signals(*self*)
Connects signals to slots.

add_menu_actions(*self*)
Adds toggle view actions to View menu.

tabify_and_raise(*self*, *docks*)
Tabifies docks in given list, then raises the first.

Parameters **docks** (*list*) –

begin_style_change(*self*)
Begins a style change operation.

end_style_change(*self*)
Ends a style change operation.

apply_stacked_style(*self*, *checked=False*)
Applies the stacked style, inspired in the former tree view.

apply_pivot_style(*self*, *checked=False*)
Applies the pivot style, inspired in the former tabular view.

apply_graph_style(*self*, *checked=False*)
Applies the graph style, inspired in the former graph view.

_get_base_dir(*self*)

spinetoolbox.spine_db_editor.widgets.tabular_view_header_widget

Contains TabularViewHeaderWidget class.

authors

P. Vennström (VTT), M. Marin (KTH)

date 2.12.2019

Module Contents

Classes

TabularViewHeaderWidget	A draggable QWidget.

class spinetoolbox.spine_db_editor.widgets.tabular_view_header_widget.**TabularViewHeaderWidg**

Bases: PySide2.QtWidgets.QFrame

A draggable QWidget.

Parameters

- **identifier** (*str*) –
- **area** (*str*) – either “rows”, “columns”, or “frozen”
- **menu** (*FilterMenu, optional*) –
- **parent** (*QWidget, optional*) – Parent widget

header_dropped

_H_MARGIN = 3

_SPACING = 16

identifier

area

mousePressEvent(*self*, *event*)
Register drag start position

mouseMoveEvent(*self*, *event*)
Start dragging action if needed

mouseReleaseEvent(*self*, *event*)
Forget drag start position

dragEnterEvent(*self*, *event*)

dropEvent(*self*, *event*)

spinetoolbox.spine_db_editor.widgets.tabular_view_mixin

Contains TabularViewMixin class.

author

P. Vennström (VTT)

date 1.11.2018

Module Contents

Classes

TabularViewMixin	Provides the pivot table and its frozen table for the DS form.

class spinetoolbox.spine_db_editor.widgets.tabular_view_mixin.**TabularViewMixin**(**args*, ***kwargs*)

Provides the pivot table and its frozen table for the DS form.

Initialize self. See help(type(self)) for accurate signature.

_PARAMETER_VALUE = Parameter value

_INDEX_EXPANSION = Index expansion

_RELATIONSHIP = Relationship

_PARAMETER = parameter

_ALTERNATIVE = alternative

_INDEX = index

current_object_class_id_list

current_object_class_name_list

populate_pivot_db_action_group(*self*)

populate_input_type_action_group(*self*)

_update_pivot_db_map(*self*, *action*)

add_menu_actions(*self*)
Adds toggle view actions to View menu.

connect_signals(*self*)
Connects signals to slots.

init_models(*self*)
Initializes models.

_set_model_data(*self*, *index*, *value*)

static _is_class_index(*index*)
Returns whether or not the given tree index is a class index.

Parameters **index** (*QModelIndex*) – index from object or relationship tree

Returns bool

_handle_pivot_table_visibility_changed(*self*, *visible*)

_handle_frozen_table_visibility_changed(*self*, *visible*)

_handle_entity_tree_current_changed(*self*, *current*, *previous*)

_get_entities(*self*, *class_id=None*, *class_type=None*)
Returns a list of dict items from the object or relationship tree model corresponding to the given class id.

Parameters

- **class_id** (*int*) –
- **class_type** (*str*) –

Returns list(dict)

load_empty_relationship_data(*self*, *objects_per_class=None*)
Returns a dict containing all possible relationships in the current class.

Parameters **objects_per_class** (*dict*) –

Returns Key is object id tuple, value is None.

Return type dict

load_full_relationship_data(*self*, *relationships=None*, *action='add'*)
Returns a dict of relationships in the current class.

Returns Key is object id tuple, value is relationship id.

Return type dict

load_relationship_data(*self*)
Returns a dict that merges empty and full relationship data.

Returns Key is object id tuple, value is True if a relationship exists, False otherwise.

Return type dict

_get_parameter_value_or_def_ids(*self*, *item_type*)
Returns a list of integer ids from the parameter model corresponding to the currently selected class and the given item type.

Parameters **item_type** (*str*) – either "parameter_value" or "parameter_definition"

Returns list(int)

_get_parameter_values_or_defs(*self*, *item_type*)
Returns a list of dict items from the parameter model corresponding to the currently selected class and the given item type.

Parameters **item_type** (*str*) – either "parameter_value" or "parameter_definition"

Returns list(dict)

load_empty_parameter_value_data(*self*, *entities=None*, *parameter_ids=None*, *alternative_ids=None*)
Returns a dict containing all possible combinations of entities and parameters for the current class.

Parameters

- **entities** (*list, optional*) – if given, only load data for these entities

- **parameter_ids** (*set, optional*) – if given, only load data for these parameter definitions

Returns Key is a tuple object_id, ..., parameter_id, value is None.

Return type dict

load_full_parameter_value_data(*self*, *parameter_values=None*, *action='add'*)
Returns a dict of parameter values for the current class.

Parameters

- **parameter_values** (*list, optional*) –
- **action** (*str*) –

Returns Key is a tuple object_id, ..., parameter_id, value is the parameter_value.

Return type dict

load_parameter_value_data(*self*)
Returns a dict that merges empty and full parameter_value data.

Returns Key is a tuple object_id, ..., parameter_id, value is the parameter_value or None if not specified.

Return type dict

load_expanded_parameter_value_data(*self*)
Returns all permutations of entities as well as parameter indexes and values for the current class.

Returns Key is a tuple object_id, ..., index, while value is None.

Return type dict

get_pivot_preferences(*self*)
Returns saved pivot preferences.

Returns pivot tuple, or None if no preference stored

Return type tuple, NoneType

reload_pivot_table(*self*, *current=None*)
Updates current class (type and id) and reloads pivot table for it.

do_reload_pivot_table(*self*, *action=None*)
Reloads pivot table.

clear_pivot_table(*self*)

wipe_out_filter_menus(*self*)

make_pivot_headers(*self*)
Turns top left indexes in the pivot table into TabularViewHeaderWidget.

_resize_pivot_header_columns(*self*)

make_frozen_headers(*self*)
Turns indexes in the first row of the frozen table into TabularViewHeaderWidget.

create_filter_menu(*self*, *identifier*)
Returns a filter menu for given given object_class identifier.

Parameters **identifier** (*int*) –

Returns TabularViewFilterMenu

create_header_widget(*self*, *identifier*, *area*, *with_menu=True*)
Returns a TabularViewHeaderWidget for given object_class identifier.

Parameters

- **identifier** (*str*) –
- **area** (*str*) –
- **with_menu** (*bool*) –

Returns TabularViewHeaderWidget

static _get_insert_index(*pivot_list*, *catcher*, *position*)
Returns an index for inserting a new element in the given pivot list.

Returns int

handle_header_dropped(*self*, *dropped*, *catcher*, *position=''*)
Updates pivots when a header is dropped.

Parameters

- **dropped** (TabularViewHeaderWidget) –
- **catcher** (TabularViewHeaderWidget, PivotTableHeaderView, FrozenTableView) –
- **position** (*str*) – either "before", "after", or ""

get_frozen_value(*self*, *index*)
Returns the value in the frozen table corresponding to the given index.

Parameters **index** (*QModelIndex*) –

Returns tuple

change_frozen_value(*self*, *current*, *previous*)
Sets the frozen value from selection in frozen table.

change_filter(*self*, *identifier*, *valid_values*, *has_filter*)

reload_frozen_table(*self*)
Resets the frozen model according to new selection in entity trees.

find_frozen_values(*self*, *frozen*)
Returns a list of tuples containing unique values (object ids) for the frozen indexes (object_class ids).

Parameters **frozen** (*tuple(int)*) – A tuple of currently frozen indexes

Returns list(tuple(list(int)))

static refresh_table_view(*table_view*)

update_filter_menus(*self*, *action*)

receive_objects_added_or_removed(*self*, *db_map_data*, *action*)

receive_relationships_added_or_removed(*self*, *db_map_data*, *action*)

receive_parameter_definitions_added_or_removed(*self*, *db_map_data*, *action*)

receive_parameter_values_added_or_removed(*self*, *db_map_data*, *action*)

receive_db_map_data_updated(*self*, *db_map_data*, *get_class_id*)

receive_alternatives_updates(*self*, *db_map_data*)

receive_alternatives_added_or_removed(*self*, *db_map_data*, *action*)

receive_classes_removed(*self*, *db_map_data*)

receive_alternatives_added(*self*, *db_map_data*)
Reacts to alternatives added event.

receive_objects_added(*self*, *db_map_data*)
Reacts to objects added event.

receive_relationships_added(*self*, *db_map_data*)
Reacts to relationships added event.

receive_parameter_definitions_added(*self*, *db_map_data*)
Reacts to parameter definitions added event.

receive_parameter_values_added(*self*, *db_map_data*)
Reacts to parameter values added event.

receive_alternatives_updated(*self*, *db_map_data*)
Reacts to object classes updated event.

receive_object_classes_updated(*self*, *db_map_data*)
Reacts to object classes updated event.

receive_objects_updated(*self*, *db_map_data*)
Reacts to objects updated event.

receive_relationship_classes_updated(*self*, *db_map_data*)
Reacts to relationship classes updated event.

receive_relationships_updated(*self*, *db_map_data*)
Reacts to relationships updated event.

receive_parameter_values_updated(*self*, *db_map_data*)
Reacts to parameter values added event.

receive_parameter_definitions_updated(*self*, *db_map_data*)
Reacts to parameter definitions updated event.

receive_alternatives_removed(*self*, *db_map_data*)
Reacts to object classes removed event.

receive_object_classes_removed(*self*, *db_map_data*)
Reacts to object classes removed event.

receive_objects_removed(*self*, *db_map_data*)
Reacts to objects removed event.

receive_relationship_classes_removed(*self*, *db_map_data*)
Reacts to relationship classes remove event.

receive_relationships_removed(*self*, *db_map_data*)
Reacts to relationships removed event.

receive_parameter_definitions_removed(*self*, *db_map_data*)
Reacts to parameter definitions removed event.

receive_parameter_values_removed(*self*, *db_map_data*)
Reacts to parameter values removed event.

receive_session_rolled_back(*self*, *db_maps*)
Reacts to session rolled back event.

spinetoolbox.spine_db_editor.widgets.tree_view_mixin

Contains the TreeViewMixin class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

Classes

TreeViewMixin	Provides object and relationship trees for the Spine db editor.

class spinetoolbox.spine_db_editor.widgets.tree_view_mixin.**TreeViewMixin**(**args*, ***kwargs*)

Provides object and relationship trees for the Spine db editor.

Initialize self. See help(type(self)) for accurate signature.

_object_classes_added

_relationship_classes_added

_object_classes_fetched

_relationship_classes_fetched
Emitted from fetcher thread, connected to Slots in GUI thread.

add_menu_actions(*self*)
Adds toggle view actions to View menu.

connect_signals(*self*)
Connects signals to slots.

init_models(*self*)
Initializes models.

_handle_object_tree_selection_changed(*self*, *selected*, *deselected*)
Updates object filter and sets default rows.

_handle_relationship_tree_selection_changed(*self*, *selected*, *deselected*)
Updates relationship filter and sets default rows.

static _clear_tree_selections_silently(*tree_view*)
Clears the selections on a given abstract item view without emitting any signals.

static _db_map_items(*indexes*)
Groups items from given tree indexes by db map.

Returns lists of dictionary items keyed by DiffDatabaseMapping

Return type dict

_db_map_ids(*self*, *indexes*)

_db_map_class_ids(*self*, *indexes*)

export_selected(*self*, *selected_indexes*)
Exports data from given indexes in the entity tree.

duplicate_object(*self*, *index*)
Duplicates the object at the given object tree model index.

Parameters **index** (*QModelIndex*) –

show_add_object_classes_form(*self*, *checked=False*)
Shows dialog to add new object classes.

show_add_objects_form(*self*, *checked=False*, *class_name=''*)
Shows dialog to add new objects.

show_add_object_group_form(*self*, *object_class_item*)
Shows dialog to add new object group.

show_manage_object_group_form(*self*, *object_item*)
Shows dialog to manage an object group.

show_add_relationship_classes_form(*self*, *checked=False*, *object_class_one_name=None*)
Shows dialog to add new relationship_class.

show_add_relationships_form(*self*, *checked=False*, *relationship_class_key=None*, *object_names_by_class_name=None*)
Shows dialog to add new relationships.

show_manage_relationships_form(*self*, *checked=False*, *relationship_class_key=None*)

edit_entity_tree_items(*self*, *selected_indexes*)
Starts editing given indexes.

show_edit_object_classes_form(*self*, *items*)

show_edit_objects_form(*self*, *items*)

show_edit_relationship_classes_form(*self*, *items*)

show_remove_alternative_tree_items_form(*self*)
Shows form to remove items from object treeview.

show_edit_relationships_form(*self*, *items*)

show_remove_entity_tree_items_form(*self*, *selected_indexes*)
Shows form to remove items from object treeview.

notify_items_changed(*self*, *action*, *item_type*, *db_map_data*)
Enables or disables actions and informs the user about what just happened.

receive_alternatives_fetched(*self*, *db_map_data*)

receive_scenarios_fetched(*self*, *db_map_data*)

receive_object_classes_fetched(*self*, *db_map_data*)

receive_relationship_classes_fetched(*self*, *db_map_data*)

receive_alternatives_added(*self*, *db_map_data*)

receive_scenarios_added(*self*, *db_map_data*)

receive_object_classes_added(*self*, *db_map_data*)

receive_objects_added(*self*, *db_map_data*)

receive_relationship_classes_added(*self*, *db_map_data*)

receive_relationships_added(*self*, *db_map_data*)

receive_alternatives_updated(*self*, *db_map_data*)

receive_scenarios_updated(*self*, *db_map_data*)

receive_entity_groups_added(*self*, *db_map_data*)

receive_object_classes_updated(*self*, *db_map_data*)

receive_objects_updated(*self*, *db_map_data*)

receive_relationship_classes_updated(*self*, *db_map_data*)

receive_relationships_updated(*self*, *db_map_data*)

receive_alternatives_removed(*self*, *db_map_data*)

receive_scenarios_removed(*self*, *db_map_data*)

receive_object_classes_removed(*self*, *db_map_data*)

receive_objects_removed(*self*, *db_map_data*)

receive_relationship_classes_removed(*self*, *db_map_data*)

receive_relationships_removed(*self*, *db_map_data*)

receive_entity_groups_removed(*self*, *db_map_data*)

Submodules

spinetoolbox.spine_db_editor.graphics_items

Classes for drawing graphics items on graph view's QGraphicsScene.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

Classes

EntityItem	Base class for ObjectItem and RelationshipItem.
RelationshipItem	Represents a relationship in the Entity graph.
ObjectItem	Represents an object in the Entity graph.
ArcItem	Connects a RelationshipItem to an ObjectItem.
CrossHairsItem	Creates new relationships directly in the graph.
CrossHairsRelationshipItem	Represents the relationship that's being created using the CrossHairsItem.
CrossHairsArcItem	Connects a CrossHairsRelationshipItem with the CrossHairsItem,
ObjectLabelItem	Provides a label for ObjectItem's.

Functions

make_figure_graphics_item(scene, z=0, static=True)	Creates a FigureCanvas and adds it to the given scene.

spinetoolbox.spine_db_editor.graphics_items.**make_figure_graphics_item**(*scene*, *z=0*, *static=True*)

Creates a FigureCanvas and adds it to the given scene. Used for creating heatmaps and associated colorbars.

Parameters

- **scene** (*QGraphicsScene*) –
- **z** (*int, optional*) – z value. Defaults to 0.
- **static** (*bool, optional*) – if True (the default) the figure canvas is not movable

Returns the graphics item that represents the canvas Figure: the figure in the canvas

Return type QGraphicsProxyWidget

class spinetoolbox.spine_db_editor.graphics_items.**EntityItem**(*spine_db_editor*, *x*, *y*, *extent*, *entity_id=None*)

Bases: PySide2.QtWidgets.QGraphicsPixmapItem

Base class for ObjectItem and RelationshipItem.

Initializes item

Parameters

- **spine_db_editor** (SpineDBEditor) – 'owner'
- **x** (*float*) – x-coordinate of central point
- **y** (*float*) – y-coordinate of central point
- **extent** (*int*) – Preferred extent
- **entity_id** (*int*) – The entity id

entity_type

entity_name

entity_class_type

entity_class_id

entity_class_name

first_db_map

display_data

display_database

db_maps

db_map_data(*self*, *_db_map*)

db_map_id(*self*, *_db_map*)

boundingRect(*self*)

moveBy(*self*, *dx*, *dy*)

_init_bg(*self*)

refresh_icon(*self*)
Refreshes the icon.

shape(*self*)
Returns a shape containing the entire bounding rect, to work better with icon transparency.

paint(*self*, *painter*, *option*, *widget=None*)
Shows or hides the selection halo.

_paint_as_selected(*self*)

_paint_as_deselected(*self*)

add_arc_item(*self*, *arc_item*)
Adds an item to the list of arcs.

Parameters **arc_item** (ArcItem) –

apply_zoom(*self*, *factor*)
Applies zoom.

Parameters **factor** (*float*) – The zoom factor.

apply_rotation(*self*, *angle*, *center*)
Applies rotation.

Parameters

- **angle** (*float*) – The angle in degrees.
- **center** (*QPoint*) – Rotates around this point.

block_move_by(*self*, *dx*, *dy*)

mouseMoveEvent(*self*, *event*)
Moves the item and all connected arcs. Also checks for a merge target and sets an appropriate mouse cursor.

Parameters **event** (*QGraphicsSceneMouseEvent*) –

update_arcs_line(*self*)
Moves arc items.

itemChange(*self*, *change*, *value*)
Keeps track of item's movements on the scene.

Parameters

- **change** (*GraphicsItemChange*) – a flag signalling the type of the change
- **value** – a value related to the change

Returns the same value given as input

set_all_visible(*self*, *on*)
Sets visibility status for this item and all arc items.

Parameters **on** (*bool*) –

_make_menu(*self*)

contextMenuEvent(*self*, *e*)
Shows context menu.

Parameters **e** (*QGraphicsSceneMouseEvent*) – Mouse event

class spinetoolbox.spine_db_editor.graphics_items.**RelationshipItem**(*spine_db_editor*, *x*, *y*, *extent*, *entity_id=None*)

Bases: *spinetoolbox.spine_db_editor.graphics_items.EntityItem*

Represents a relationship in the Entity graph.

Initializes the item.

Parameters

- **spine_db_editor** (*GraphViewForm*) – 'owner'
- **x** (*float*) – x-coordinate of central point
- **y** (*float*) – y-coordinate of central point
- **extent** (*int*) – preferred extent
- **entity_id** (*int*) – object id

entity_type

object_class_id_list

object_name_list

object_id_list

entity_class_name

db_representation

_make_tool_tip(*self*)

_init_bg(*self*)

follow_object_by(*self*, *dx*, *dy*)

class spinetoolbox.spine_db_editor.graphics_items.**ObjectItem**(*spine_db_editor*, *x*, *y*, *extent*, *entity_id=None*)

Bases: *spinetoolbox.spine_db_editor.graphics_items.EntityItem*

Represents an object in the Entity graph.

Initializes the item.

Parameters

- **spine_db_editor** (*GraphViewForm*) – 'owner'
- **x** (*float*) – x-coordinate of central point
- **y** (*float*) – y-coordinate of central point
- **extent** (*int*) – preferred extent
- **entity_id** (*int*) – object id

entity_type

db_representation

shape(*self*)

Returns a shape containing the entire bounding rect, to work better with icon transparency.

update_name(*self*, *name*)
Refreshes the name.

update_description(*self*, *description*)

block_move_by(*self*, *dx*, *dy*)

_make_menu(*self*)

_populate_add_relationships_menu(*self*, *add_title=False*)
Populates the 'Add relationships' menu.

contextMenuEvent(*self*, *e*)
Shows context menu.

Parameters **e** (*QGraphicsSceneMouseEvent*) – Mouse event

mouseDoubleClickEvent(*self*, *e*)

_start_relationship(*self*, *action*)

class spinetoolbox.spine_db_editor.graphics_items.**ArcItem**(*rel_item*, *obj_item*, *width*)

Bases: PySide2.QtWidgets.QGraphicsPathItem

Connects a RelationshipItem to an ObjectItem.

Initializes item.

Parameters

- **rel_item** (*spinetoolbox.widgets.graph_view_graphics_items.RelationshipItem*) – relationship item
- **obj_item** (*spinetoolbox.widgets.graph_view_graphics_items.ObjectItem*) – object item
- **width** (*float*) – Preferred line width

_make_pen(*self*)

moveBy(*self*, *dx*, *dy*)
Does nothing. This item is not moved the regular way, but follows the EntityItems it connects.

update_line(*self*)

mousePressEvent(*self*, *event*)
Accepts the event so it's not propagated.

other_item(*self*, *item*)

apply_zoom(*self*, *factor*)
Applies zoom.

Parameters **factor** (*float*) – The zoom factor.

class spinetoolbox.spine_db_editor.graphics_items.**CrossHairsItem**(*spine_db_editor*, *x*, *y*, *extent*)

Bases: *spinetoolbox.spine_db_editor.graphics_items.RelationshipItem*

Creates new relationships directly in the graph.

Initializes the item.

Parameters

- **spine_db_editor** (*GraphViewForm*) – 'owner'

- **x** (*float*) – x-coordinate of central point
- **y** (*float*) – y-coordinate of central point
- **extent** (*int*) – preferred extent
- **entity_id** (*int*) – object id

entity_class_name

entity_name

_make_tool_tip(*self*)

refresh_icon(*self*)
Refreshes the icon.

set_plus_icon(*self*)

set_check_icon(*self*)

set_normal_icon(*self*)

set_ban_icon(*self*)

set_icon(*self*, *unicode*, *color=0*)
Refreshes the icon.

mouseMoveEvent(*self*, *event*)
Moves the item and all connected arcs. Also checks for a merge target and sets an appropriate mouse cursor.

Parameters **event** (*QGraphicsSceneMouseEvent*) –

block_move_by(*self*, *dx*, *dy*)

contextMenuEvent(*self*, *e*)
Shows context menu.

Parameters **e** (*QGraphicsSceneMouseEvent*) – Mouse event

class spinetoolbox.spine_db_editor.graphics_items.**CrossHairsRelationshipItem**(**args*, ***kwargs*)

Bases: *spinetoolbox.spine_db_editor.graphics_items.RelationshipItem*

Represents the relationship that's being created using the CrossHairsItem.

Initializes the item.

Parameters

- **spine_db_editor** (*GraphViewForm*) – 'owner'
- **x** (*float*) – x-coordinate of central point
- **y** (*float*) – y-coordinate of central point
- **extent** (*int*) – preferred extent
- **entity_id** (*int*) – object id

_make_tool_tip(*self*)

refresh_icon(*self*)
Refreshes the icon.

contextMenuEvent(*self*, *e*)
Shows context menu.

Parameters **e** (*QGraphicsSceneMouseEvent*) – Mouse event

class spinetoolbox.spine_db_editor.graphics_items.**CrossHairsArcItem**(*rel_item*, *obj_item*, *width*)

Bases: *spinetoolbox.spine_db_editor.graphics_items.ArcItem*

Connects a CrossHairsRelationshipItem with the CrossHairsItem, and with all the ObjectItem's in the relationship so far.

Initializes item.

Parameters

- **rel_item** (*spinetoolbox.widgets.graph_view_graphics_items.RelationshipItem*) – relationship item
- **obj_item** (*spinetoolbox.widgets.graph_view_graphics_items.ObjectItem*) – object item
- **width** (*float*) – Preferred line width

_make_pen(*self*)

class spinetoolbox.spine_db_editor.graphics_items.**ObjectLabelItem**(*entity_item*)

Bases: PySide2.QtWidgets.QGraphicsTextItem

Provides a label for ObjectItem's.

Initializes item.

Parameters **entity_item** (*spinetoolbox.widgets.graph_view_graphics_items.EntityItem*) – The parent item.

entity_name_edited

setPlainText(*self*, *text*)

Set texts and resets position.

Parameters **text** (*str*) –

reset_position(*self*)

Adapts item geometry so text is always centered.

spinetoolbox.spine_io

Init file for spine_io package. Intentionally empty.

author

P. Vennström (VTT)

date 1.6.2019

Subpackages

spinetoolbox.spine_io.exporters

Init file for spine_io.exporters package. Intentionally empty.

author

A. Soininen (VTT)

date 30.8.2019

Submodules

spinetoolbox.spine_io.exporters.excel

Framework for exporting a database to Excel file.

author

P. Vennström (VTT), A. Soininen (VTT)

date 31.1.2020

Module Contents

Functions

_get_objects_and_parameters(db)	Exports all object data from spine database into unstacked list of lists
_get_relationships_and_parameters(db)	Exports all relationship data from spine database into unstacked list of lists
_unstack_list_of_tuples(data, headers, key_cols, value_name_col, value_col)	Unstacks list of lists or list of tuples and creates a list of namedtuples
_get_unstacked_relationships(db)	Gets all data for relationships in a unstacked list of list
_get_unstacked_objects(db)	Gets all data for objects in a unstacked list of list
_write_relationships_to_xlsx(wb, relationship_data)	Writes Classes, parameter and parameter values for relationships.
_write_json_array_to_xlsx(wb, data, sheet_type)	Writes json array data for object classes and relationship classes.
_write_TimeSeries_to_xlsx(wb, data, sheet_type, data_type)	Writes spinedb_api TimeSeries data for object classes and relationship classes.
_write_objects_to_xlsx(wb, object_data)	Writes Classes, parameter and parameter values for objects.
_get_object_groups(db)	Exports all group data from spine database into a dict.
_write_object_groups_to_xlsx(wb, group_data)	Writes classes, groups and members for object groups.
_write_alternatives_to_xlsx(wb, alternative_data)	Writes names, and description for alternatives.
_write_scenarios_to_xlsx(wb, scenario_data)	Writes names, active flag, and description for scenarios.
_write_scenario_alternatives_to_xlsx(wb, scenario_alternative_data)	Writes scenario names, alternative names, and before alternative names for scenario alternatives.
export_spine_database_to_xlsx(db, filepath)	Writes all data in a spine database into an excel file.

spinetoolbox.spine_io.exporters.excel.**_get_objects_and_parameters**(*db*)

Exports all object data from spine database into unstacked list of lists

Parameters **db** (*spinedb_api.DatabaseMapping*) – database mapping for database

Returns (List, List) First list contains parameter data, second one json data

spinetoolbox.spine_io.exporters.excel.**_get_relationships_and_parameters**(*db*)
Exports all relationship data from spine database into unstacked list of lists

Parameters **db** (*spinedb_api.DatabaseMapping*) – database mapping for database

Returns (List, List) First list contains parameter data, second one json data

spinetoolbox.spine_io.exporters.excel.**_unstack_list_of_tuples**(*data*, *headers*, *key_cols*, *value_name_col*, *value_col*)
Unstacks list of lists or list of tuples and creates a list of namedtuples with unstacked data (pivoted data)

Parameters

- **data** (*List[List]*) – List of lists with data to unstack
- **headers** (*List[str]*) – List of header names for data
- **key_cols** (*List[Int]*) – List of index for column that are keys, columns to not unstack
- **value_name_col** (*Int*) – index to column containing name of data to unstack
- **value_col** (*Int*) – index to column containing value to value_name_col

Returns List of list with headers in headers list (List): List of header names for each item in inner list

Return type (List[List])

spinetoolbox.spine_io.exporters.excel.**_get_unstacked_relationships**(*db*)
Gets all data for relationships in a unstacked list of list

Parameters **db** (*spinedb_api.DatabaseMapping*) – database mapping for database

Returns stacked relationships, stacked JSON, stacked time series and stacked time patterns

Return type (list, list, list, list)

spinetoolbox.spine_io.exporters.excel.**_get_unstacked_objects**(*db*)
Gets all data for objects in a unstacked list of list

Parameters **db** (*spinedb_api.DatabaseMapping*) – database mapping for database

Returns stacked objects, parsed JSON, parsed time series and parsed time patterns

Return type (list, list, list, list)

spinetoolbox.spine_io.exporters.excel.**_write_relationships_to_xlsx**(*wb*, *relationship_data*)
Writes Classes, parameter and parameter values for relationships. Writes one sheet per relationship_class.

Parameters

- **wb** (*openpyxl.Workbook*) – excel workbook to write too.
- **relationship_data** (*List[List]*) – List of lists containing relationship
- **give by function get_unstacked_relationships** (*data*) –

spinetoolbox.spine_io.exporters.excel.**_write_json_array_to_xlsx**(*wb*, *data*, *sheet_type*)
Writes json array data for object classes and relationship classes. Writes one sheet per relationship/object_class.

Parameters

- **wb** (*openpyxl.Workbook*) – excel workbook to write too.

- **data** (*List[List]*) – List of lists containing json data give by function
- **and get_unstacked_relationships** (*get_unstacked_objects*) –
- **sheet_type** (*str*) – str with value "relationship" or "object" telling if data is for a relationship or object

spinetoolbox.spine_io.exporters.excel.**_write_TimeSeries_to_xlsx**(*wb*, *data*, *sheet_type*, *data_type*)

Writes spinedb_api TimeSeries data for object classes and relationship classes. Writes one sheet per relationship/object_class.

Parameters

- **wb** (*openpyxl.Workbook*) – excel workbook to write too.
- **data** (*List[List]*) – List of lists containing json data give by function
- **and get_unstacked_relationships** (*get_unstacked_objects*) –
- **sheet_type** (*str*) – str with value "relationship" or "object" telling if data is for a relationship or object

spinetoolbox.spine_io.exporters.excel.**_write_objects_to_xlsx**(*wb*, *object_data*)

Writes Classes, parameter and parameter values for objects. Writes one sheet per relationship/object_class.

Parameters

- **wb** (*openpyxl.Workbook*) – excel workbook to write too.
- **object_data** (*List[List]*) – List of lists containing object data give by function get_unstacked_objects

spinetoolbox.spine_io.exporters.excel.**_get_object_groups**(*db*)

Exports all group data from spine database into a dict.

Parameters **db** (*spinedb_api.DatabaseMapping*) – database mapping for database

Returns mapping class_name, to a list of (group_name, member_name) tuples sorted by group_name

Return type dict

spinetoolbox.spine_io.exporters.excel.**_write_object_groups_to_xlsx**(*wb*, *group_data*)

Writes classes, groups and members for object groups. Writes one sheet per object_class.

Parameters

- **wb** (*openpyxl.Workbook*) – excel workbook to write too.
- **group_data** (*dict*) – containing group data as given by function _get_object_groups

spinetoolbox.spine_io.exporters.excel.**_write_alternatives_to_xlsx**(*wb*, *alternative_data*)

Writes names, and description for alternatives. Writes one sheet.

Parameters

- **wb** (*openpyxl.Workbook*) – excel workbook to write too.
- **object_data** (*List[List]*) – List of lists containing object data give by function get_unstacked_objects

spinetoolbox.spine_io.exporters.excel.**_write_scenarios_to_xlsx**(*wb*, *scenario_data*)

Writes names, active flag, and description for scenarios. Writes one sheet.

Parameters

- **wb** (*openpyxl.Workbook*) – excel workbook to write too.
- **object_data** (*List[List]*) – List of lists containing object data give by function get_unstacked_objects

spinetoolbox.spine_io.exporters.excel.**_write_scenario_alternatives_to_xlsx**(*wb*, *scenario_alternative_data*)

Writes scenario names, alternative names, and before alternative names for scenario alternatives. Writes one sheet.

Parameters

- **wb** (*openpyxl.Workbook*) – excel workbook to write too.
- **object_data** (*List[List]*) – List of lists containing object data give by function get_unstacked_objects

spinetoolbox.spine_io.exporters.excel.**export_spine_database_to_xlsx**(*db*, *filepath*)

Writes all data in a spine database into an excel file.

Parameters

- **db** (*spinedb_api.DatabaseMapping*) – database mapping for database.
- **filepath** (*str*) – str with filepath to save excel file to.

spinetoolbox.spine_io.exporters.gdx

For exporting a database to GAMS .gdx file.

Currently, this module supports databases that are "GAMS-like", that is, they follow the EAV model but the object classes, objects, relationship classes etc. directly reflect the GAMS data structures. Conversions e.g. from Spine model to TIMES are not supported at the moment.

This module contains low level functions for reading a database into an intermediate format and for writing that intermediate format into a .gdx file. A higher lever function *to_gdx_file()* that does basically everything needed for exporting is provided for convenience.

author

A. Soininen (VTT)

date 30.8.2019

Module Contents

Classes

NoneExport	Options to export None values.
NoneFallback	Options load None values from the database.

Continued on next page

Table 196 – continued from previous page

Set	Represents a GAMS domain, set or a subset.
Record	Represents a GAMS set element in a *Set*.
Parameter	Represents a GAMS parameter.
Picking	An interface for picking objects.
FixedPicking	Picking from a fixed boolean array.
GeneratedPicking	Picking using a Python expression.
Records	An interface for records used in *SetSettings*.
LiteralRecords	Shufflable records with fixed keys.
GeneratedRecords	Non-shuffleable records where keys are generated by a Python expression.
ExtractedRecords	Records that are extracted from an indexed parameter.
MergingSetting	Holds settings needed to merge a single parameter.
IndexingSetting	Settings for indexed value expansion for a single Parameter.
SetSettings	This class holds the settings for domains, sets and records needed by *to_gdx_file()* for .gdx export.
ExportFlag	Options for exporting Set objects.
Origin	Domain or set origin.
SetMetadata	This class holds some additional configuration for Sets.

Functions

_picking_from_dict(picking_dict)	Deserializes pickings from dictionary.
_update_records(old, new)	Updates records where appropriate.
_records_from_dict(record_dict)	Deserializes records from a dict.
_python_interpreter_bitness()	Returns 64 for 64bit Python interpreter or 32 for 32bit interpreter.
_read_value(value_in_database)	Converts a parameter from its database representation to a value object.
_windows_dlls_exist(gams_path)	Returns True if required DLL files exist in given GAMS installation path.
find_gams_directory()	Returns GAMS installation directory or None if not found.
expand_indexed_parameter_values(parameters, indexing_settings, sets)	Expands the dimensions of indexed parameter values.
update_merging_settings(merging_settings, set_settings, db_map)	Returns parameter merging settings updated according to new export settings.
merging_records(merging_setting)	Constructs records which contain the merged parameters' names.
merge_parameters(parameters, merging_settings)	Merges multiple parameters into a single parameter.
sets_to_gams(gdx_file, sets, omitted_set=None)	Writes Set objects to .gdx file as GAMS sets.
parameters_to_gams(gdx_file, parameters, none_export)	Writes parameters to .gdx file as GAMS parameters.
domain_parameters_to_gams_scalars(gdx_file, parameters, domain_name)	Adds the parameter from given domain as a scalar to .gdx file.
object_classes_to_domains(db_map, domain_names)	Converts object classes and objects from a database to the intermediate format.
relationship_classes_to_sets(db_map, domain_names, set_names)	Converts relationship classes and relationships from a database to the intermediate format.

Continued on next page

Table 197 – continued from previous page

object_parameters(db_map, domains_with_ids, fallback_on_none, logger)	Converts object parameters from database to *Parameter* objects.
relationship_parameters(db_map, sets_with_ids, fallback_on_none, logger)	Converts relationship parameters from database to *Parameter* objects.
_default_values(db_map, subquery, sets_with_ids, classes_with_ignored_parameters)	Reads default parameter values from the database.
_update_using_existing_relationship_parameter_values(relationships, db_map, sets_with_ids, classes_with_ignored_parameters)	Updates an existing relationship parameter dict using actual parameter values.
domain_names_and_records(db_map)	Returns a list of domain names and a map from a name to list of record keys.
set_names_and_records(db_map)	Returns a list of set names and a map from a name to list of record keys.
make_indexing_settings(db_map, none_fallback, logger)	Constructs skeleton indexing settings for parameter indexed value expansion.
_object_indexing_settings(db_map, none_fallback, logger)	Constructs skeleton indexing settings from object parameters.
_relationship_indexing_settings(db_map, none_fallback, logger)	Constructs skeleton indexing settings from relationship parameters.
_add_to_indexing_settings(settings, parameter_name, entity_class_name, dimensions, parsed_value, index_keys, classes_with_unsupported_value_types)	Adds parameter to indexing settings.
update_indexing_settings(old_indexing_settings, new_indexing_settings, set_settings)	Returns new indexing settings merged from old and new ones.
indexing_settings_to_dict(settings)	Stores indexing settings to a JSON compatible dictionary.
indexing_settings_from_dict(settings_dict, db_map, none_fallback, logger)	Restores indexing settings from a json compatible dictionary.
_find_indexed_parameter(parameter_name, db_map, none_fallback, logger=None)	Searches for parameter_name in db_map and returns Parameter and its entity_class name.
_create_additional_domains(set_settings)	Generates additional domains found in the settings.
_exported_set_names(names, set_settings)	Returns a set of names of the domains that are marked for exporting.
sort_sets(sets, order)	Sorts a list of sets according to `sorted_names`
sort_records_inplace(sets, set_settings)	Sorts the record lists of given domains according to the order given in settings.
extract_domain(domains, name_to_extract)	Extracts the domain with given name from a list of domains.
to_gdx_file(database_map, file_name, set_settings, indexing_settings, merging_settings, none_fallback, none_export, gams_system_directory=None, logger=None)	Exports given database map into .gdx file.
make_set_settings(database_map)	Builds a *SetSettings* object from given database.

class spinetoolbox.spine_io.exporters.gdx.**NoneExport**

Bases: `enum.Enum`

Options to export None values.

Create and return a new object. See help(type) for accurate signature.

DO_NOT_EXPORT = 0

Does not export Nones.

EXPORT_AS_NAN = 1
Replace Nones with NaNs while exporting.

class spinetoolbox.spine_io.exporters.gdx.**NoneFallback**
Bases: enum.Enum

Options load None values from the database.

Create and return a new object. See help(type) for accurate signature.

USE_IT = 0
Keep using the value.

USE_DEFAULT_VALUE = 1
Replace None by the default value.

exception spinetoolbox.spine_io.exporters.gdx.**GdxExportException**(*message*)
Bases: Exception

An exception raised when something goes wrong within the gdx module.

Parameters **message** (*str*) – a message detailing the cause of the exception

message
A message detailing the cause of the exception.

__str__(*self*)
Returns the message detailing the cause of the exception.

exception spinetoolbox.spine_io.exporters.gdx.**GdxUnsupportedValueTypeException**(*message*)
Bases: *spinetoolbox.spine_io.exporters.gdx.GdxExportException*

An exception raised when an unsupported parameter type is read from the database.

Parameters **message** (*str*) – a message detailing the cause of the exception

class spinetoolbox.spine_io.exporters.gdx.**Set**(*name*, *description=''*, *domain_names=None*)
Represents a GAMS domain, set or a subset.

description
set's explanatory text

Type str

domain_names
a list of superset (domain) names, None if the Set is a domain

Type list of str

name
set's name

Type str

records
set's elements as a list of Record objects

Type list of Record

Parameters

- **name** (*str*) – set's name
- **description** (*str*) – set's explanatory text
- **domain_names** (*list of str*) – a list of indexing domain names

dimensions
Number of dimensions of this Set.

is_domain(*self*)
Returns True if this set is a domain set.

to_dict(*self*)
Stores Set to a dictionary.

static from_dict(*set_dict*)
Restores Set from a dictionary.

class spinetoolbox.spine_io.exporters.gdx.**Record**(*keys*)
Represents a GAMS set element in a *Set*.

keys
a tuple of record's keys

Type tuple

Parameters **keys** (*tuple*) – a tuple of record's keys

name
Record's 'name' as a comma separated list of its keys.

__eq__(*self*, *other*)
Returns True if other is equal to self.

Parameters **other** (Record) – a record to compare to

to_dict(*self*)
Stores Record to a dictionary.

static from_dict(*record_dict*)
Restores Record from a dictionary.

class spinetoolbox.spine_io.exporters.gdx.**Parameter**(*domain_names*, *indexes*, *values*)
Represents a GAMS parameter.

domain_names
indexing domain names (currently Parameters can be indexed by domains only)

Type list

data
a map from index tuples to parsed values

Type dict

Parameters

- **domain_names** (*list*) – indexing domain names (currently Parameters can be indexed by domains only)
- **indexes** (*list*) – parameter's indexes
- **values** (*list*) – parameter's values

indexes
indexing key tuples

Type list

values
parsed values

Type list

__eq__ (*self*, *other*)
Compares two *Parameter* objects for equality.

Parameters **other** (Parameter) – a parameter

Returns True if the parameters are equal, False otherwise

Return type bool

is_consistent (*self*)
Checks that all values are IndexedValue objects or scalars.

slurp (*self*, *parameter*)
Appends the indexes and values from another parameter.

Parameters **parameter** (Parameter) – a parameter to append from

is_scalar (*self*)
Returns True if this parameter seems to contain scalars.

is_indexed (*self*)
Returns True if this parameter seems to contain indexed values.

expand_indexes (*self*, *indexing_setting*, *sets*)
Expands indexed values to scalars in place by adding a new dimension (index).

The indexes and values attributes are resized to accommodate all scalars in the indexed values. A new indexing domain is inserted to domain_names and the corresponding keys into indexes. Effectively, this increases parameter's dimensions by one.

Parameters

- **indexing_setting** (IndexingSetting) – description of how the expansion should be done
- **sets** (*dict*) – mapping from set name to *Set*

class spinetoolbox.spine_io.exporters.gdx.**Picking**
An interface for picking objects.

Picking object are used to select indexes from an indexing domain when performing parameter index expansion.

pick (*self*, *i*)
Returns pick for given indexing domain record.

Parameters **i** (*int*) – record index

Returns True if the record is picked, False otherwise

Return type bool

to_dict (*self*)
Serializes the picking to a dict.

Returns serialized picking

Return type dict

static from_dict (*picking_dict*)
Deseriealizes the picking from a dict.

Parameters **picking_dict** (*dict*) – serialized picking

Returns deserialized picking

Return type *Picking*

class spinetoolbox.spine_io.exporters.gdx.**FixedPicking**(*picked*)
Bases: *spinetoolbox.spine_io.exporters.gdx.Picking*

Picking from a fixed boolean array.

Parameters **picked** (*list of bool*) – a list of booleans, where True picks and False drops a record

__eq__(*self*, *other*)
Compared pickings for equality.

Parameters **other** (FixedPicking) – another picking

Returns True if the pickings are equal, False otherwise

Return type bool

pick(*self*, *i*)
See base class.

to_dict(*self*)
See base class.

static from_dict(*picking_dict*)
See base class.

class spinetoolbox.spine_io.exporters.gdx.**GeneratedPicking**(*expression*)
Bases: *spinetoolbox.spine_io.exporters.gdx.Picking*

Picking using a Python expression.

The expression should return a value that can be cast to bool. It has a single parameter, `i`, at its disposal. This is a one-based index to the pick list.

Parameters **expression** (*str*) – the expression used for picking

expression
the picking expression

pick(*self*, *i*)
See base class.

to_dict(*self*)
See base class.

static from_dict(*picking_dict*)
See base class.

spinetoolbox.spine_io.exporters.gdx.**_picking_from_dict**(*picking_dict*)
Deserializes pickings from dictionary.

Parameters **picking_dict** (*dict*) – a serialized picking

Returns a *FixedPicking* or *GeneratedPicking*

Return type *Picking*

class spinetoolbox.spine_io.exporters.gdx.**Records**
An interface for records used in *SetSettings*.

records
stored records as a list of key tuples

__eq__(*self*, *other*)
Tests for equality.

Returns True if the records are equal, False otherwise.

Return type bool

__len__(*self*)
Gives the number of records

Returns number of records

Return type int

shuffle(*self*, *new_order*)
Reorders the records if the order is not fixed, otherwise raises `NotImplementedError`.

Parameters **new_order** (*list of tuple*) – new records

is_shufflable(*self*)
Tells if the records can be shuffled.

Returns True if the records can be shuffled, False otherwise

Return type bool

static update(*old*, *new*)
Merges two records.

Parameters

- **old** (Records) – the 'original' records
- **new** (Records) – 'new' records

Returns merged records

Return type *Records*

to_dict(*self*)
Serializes the records to a dict.

Returns serialized records.

Return type dict

static from_dict(*record_dict*)
Deserializes records from a dict.

Parameters **record_dict** – serialized records

Returns deserialized records

Return type *Records*

class spinetoolbox.spine_io.exporters.gdx.**LiteralRecords**(*records*)
Bases: *spinetoolbox.spine_io.exporters.gdx.Records*

Shufflable records with fixed keys.

Parameters **records** (*list of tuple*) – list of key tuples

records
See base class.

__eq__(*self*, *other*)
Compares two *LiteralRecords* for equality.

Parameters **other** (LiteralRecords) – records to compare to

Returns True if the key lists are equal, False otherwise

Return type bool

__len__(*self*)
See base class.

shuffle(*self*, *new_order*)
See base class.

is_shufflable(*self*)
Returns True; *LiteralRecords* is shufflable.

static update(*old*, *new*)
Updates the keys from another *LiteralRecords*.

Common keys are kept in their old order while new keys are added last. Keys present only in old records are dropped.

Parameters

- **old** (LiteralRecords) – original records
- **new** (LiteralRecords) – new records

Returns updated records

Return type *LiteralRecords*

to_dict(*self*)
See base class.

static from_dict(*record_dict*)
See base class.

class spinetoolbox.spine_io.exporters.gdx.**GeneratedRecords**(*expression*, *length*)
Bases: *spinetoolbox.spine_io.exporters.gdx.Records*

Non-shuffleable records where keys are generated by a Python expression.

The expression should return a string.The expression has a single parameter, `i`, at it disposal. `i` is a one-based index to the pick list.

Parameters

- **expression** (*str*) – key generator expression
- **length** (*int*) – number of records to generate

expression
the expression used to generate the records

records
See base class.

__eq__(*self*, *other*)
Compares to another *GeneratedRecords* for equality

Parameters **other** (GeneratedRecords) – records

Returns True if the record expressions and lengths are equal, False otherwise

Return type bool

__len__ (*self*)
See base class.

shuffle (*self*, *new_order*)
See base class.

is_shufflable (*self*)
Returns False; *GeneratedRecords* is not shuffleable.

static update (*old*, *new*)
Updating is not supported by *GeneratedRecords*.

to_dict (*self*)
See base class.

static from_dict (*record_dict*)
See base class.

_record_list (*self*)
Generates records according to given Python expression.

Returns generated records

Return type list

class spinetoolbox.spine_io.exporters.gdx.**ExtractedRecords** (*parameter_name*, *indexes*)

Bases: *spinetoolbox.spine_io.exporters.gdx.Records*

Records that are extracted from an indexed parameter.

Parameters

- **parameter_name** (*str*) – name of the parameter from which the records were extracted
- **indexes** (*list of tuple*) – records

parameter_name
name of the parameter from which the records were extracted

records
See base class.

__eq__ (*self*, *other*)
Compares two *ExtractedRecords* for equality.

Parameters **other** (ExtractedRecords) – records to compare to

Returns True if the records and paramter name are equal, False otherwise

Return type bool

__len__ (*self*)
See base class.

shuffle (*self*, *new_order*)
See base class.

is_shufflable (*self*)
Returns False; *ExtractedRecords* is never shufflable.

static extract (*parameter_name*, *db_map*)
Gets the record keys from a given indexed parameter.

Parameters

- **parameter_name** (*str*) – parameter's name
- **db_map** (*DatabaseMappingBase*) – a database map

Returns extracted records

Return type *ExtractedRecords*

static update(*old*, *new*)
Takes the parameter name from old and the records from new.

Parameters

- **old** (ExtractedRecords) – original records
- **new** (ExtractedRecords) – new records

Returns merged records

Return type *ExtractedRecords*

to_dict(*self*)
See base class.

static from_dict(*record_dict*)
See base class.

spinetoolbox.spine_io.exporters.gdx.**_update_records**(*old*, *new*)
Updates records where appropriate.

Parameters

- **old** (Records) – original records
- **new** (Records) – new records

Returns updated records

Return type *Records*

spinetoolbox.spine_io.exporters.gdx.**_records_from_dict**(*record_dict*)
Deserializes records from a dict.

Parameters **record_dict** (*dict*) – serialized records

Returns deserialized records

Return type *Records*

spinetoolbox.spine_io.exporters.gdx.**_python_interpreter_bitness**()
Returns 64 for 64bit Python interpreter or 32 for 32bit interpreter.

spinetoolbox.spine_io.exporters.gdx.**_read_value**(*value_in_database*)
Converts a parameter from its database representation to a value object.

spinetoolbox.spine_io.exporters.gdx.**_windows_dlls_exist**(*gams_path*)
Returns True if required DLL files exist in given GAMS installation path.

spinetoolbox.spine_io.exporters.gdx.**find_gams_directory**()
Returns GAMS installation directory or None if not found.

On Windows systems, this function looks for *gams.location* in registry; on other systems the *PATH* environment variable is checked.

Returns a path to GAMS installation directory or None if not found.

spinetoolbox.spine_io.exporters.gdx.**expand_indexed_parameter_values**(*parameters*, *indexing_settings*, *sets*)

Expands the dimensions of indexed parameter values.

Parameters

- **parameters** (*dict*) – a map from parameter names to Parameters
- **indexing_settings** (*dict*) – mapping from parameter name to *IndexingSetting*
- **sets** (*dict*) – mapping from domain name to *Set*

class spinetoolbox.spine_io.exporters.gdx.**MergingSetting**(*parameter_names*, *new_domain_name*, *new_domain_description*, *previous_set*, *previous_domain_names*)

Holds settings needed to merge a single parameter.

parameter_names
parameters to merge

Type list

new_domain_name
name of the additional domain that contains the parameter names

Type str

new_domain_description
explanatory text for the additional domain

Type str

previous_set
name of the set containing the parameters before merging; not needed for the actual merging but included here to make the parameters' origing traceable

Type str

Parameters

- **parameter_names** (*list*) – parameters to merge
- **new_domain_name** (*str*) – name of the additional domain that contains the parameter names
- **new_domain_description** (*str*) – explanatory text for the additional domain
- **previous_set** (*str*) – name of the set containing the parameters before merging
- **previous_domain_names** (*list*) – list of parameters' original indexing domains

domain_names(*self*)
Composes a list of merged parameter's indexing domains.

Returns a list of indexing domains including the new domain containing the merged parameters' names

Return type list

to_dict(*self*)
Stores the settings to a dictionary.

static from_dict(*setting_dict*)
Restores settings from a dictionary.

spinetoolbox.spine_io.exporters.gdx.**update_merging_settings**(*merging_settings*, *set_settings*, *db_map*)
Returns parameter merging settings updated according to new export settings.

Parameters

- **merging_settings** (*dict*) – old merging settings
- **set_settings** (SetSettings) – new set settings
- **db_map** (*spinedb_api.DatabaseMapping or spinedb_api.DiffDatabaseMapping*) – a database map

Returns updated merging settings

Return type dict

spinetoolbox.spine_io.exporters.gdx.**merging_records**(*merging_setting*)
Constructs records which contain the merged parameters' names.

Parameters **merging_setting** (MergingSetting) – settings

Returns records needed to index merged parameters

Return type *Records*

spinetoolbox.spine_io.exporters.gdx.**merge_parameters**(*parameters*, *merging_settings*)
Merges multiple parameters into a single parameter.

Note, that the merged parameters will be removed from the parameters dictionary.

Parameters

- **parameters** (*dict*) – a mapping from existing parameter name to its Parameter object
- **merging_settings** (*dict*) – a mapping from the merged parameter name to its merging settings

Returns a mapping from merged parameter name to its Parameter object

Return type dict

spinetoolbox.spine_io.exporters.gdx.**sets_to_gams**(*gdx_file*, *sets*, *omitted_set=None*)
Writes Set objects to .gdx file as GAMS sets.

Records and Parameters contained within the Sets are written as well.

Parameters

- **gdx_file** (*GdxFile*) – a target file
- **sets** (*list*) – a list of Set objects
- **omitted_set** (Set) – prevents writing this set even if it is included in given sets

spinetoolbox.spine_io.exporters.gdx.**parameters_to_gams**(*gdx_file*, *parameters*, *none_export*)
Writes parameters to .gdx file as GAMS parameters.

Parameters

- **gdx_file** (*GdxFile*) – a target file
- **parameters** (*dict*) – a list of Parameter objects
- **none_export** (NoneExport) – option how to handle None values

spinetoolbox.spine_io.exporters.gdx.**domain_parameters_to_gams_scalars**(*gdx_file*, *parameters*, *domain_name*)

Adds the parameter from given domain as a scalar to .gdx file.

The added parameters are erased from parameters.

Parameters

- **gdx_file** (*GdxFile*) – a target file
- **parameters** (*dict*) – a map from parameter name to Parameter object
- **domain_name** (*str*) – name of domain whose parameters to add

Returns a list of non-scalar parameters

spinetoolbox.spine_io.exporters.gdx.**object_classes_to_domains**(*db_map*, *domain_names*)

Converts object classes and objects from a database to the intermediate format.

Object classes get converted to *Set* objects while objects are stored as *Record* objects in the *Set* objects.

Parameters

- **db_map** (*DatabaseMapping or DiffDatabaseMapping*) – a database map
- **domain_names** (*set*) – names of domains to convert

Returns a map from object_class id to corresponding *Set*.

Return type dict

spinetoolbox.spine_io.exporters.gdx.**relationship_classes_to_sets**(*db_map*, *domain_names*, *set_names*)

Converts relationship classes and relationships from a database to the intermediate format.

Relationship classes get converted to *Set* objects while relationships are stored as *Record* objects in corresponding *Set* objects.

Parameters

- **db_map** (*DatabaseMapping or DiffDatabaseMapping*) – a database map
- **domain_names** (*set*) – names of domains (a.k.a object classes) the relationships connect
- **set_names** (*set*) – names of sets to convert

Returns a map from relationship_class ids to the corresponding *Set* objects

Return type dict

spinetoolbox.spine_io.exporters.gdx.**object_parameters**(*db_map*, *domains_with_ids*, *fallback_on_none*, *logger*)

Converts object parameters from database to *Parameter* objects.

Parameters

- **db_map** (*DatabaseMapping or DiffDatabaseMapping*) – a database map
- **domains_with_ids** (*dict*) – mapping from object_class ids to corresponding *Set* objects
- **fallback_on_none** (NoneFallback) – fallback when encountering Nones
- **logger** (*LoggingInterface, optional*) – a logger; if not None, some errors are logged and ignored instead of raising an exception

Returns a map from parameter name to corresponding *Parameter*

Return type dict

spinetoolbox.spine_io.exporters.gdx.**relationship_parameters**(*db_map*, *sets_with_ids*, *fallback_on_none*, *logger*)

Converts relationship parameters from database to *Parameter* objects.

Parameters

- **db_map** (*DatabaseMapping or DiffDatabaseMapping*) – a database map
- **sets_with_ids** (*dict*) – mapping from relationship_class ids to corresponding *Set* objects
- **fallback_on_none** (NoneFallback) – fallback when encountering Nones
- **logger** (*LoggingInterface, optional*) – a logger; if not None, some errors are logged and ignored instead of raising an exception

Returns a map from parameter name to corresponding *Parameter*

Return type dict

spinetoolbox.spine_io.exporters.gdx.**_default_values**(*db_map*, *subquery*, *sets_with_ids*, *classes_with_ignored_parameters*)

Reads default parameter values from the database.

Parameters

- **db_map** (*DatabaseMapping or DiffDatabaseMapping*) – a database map
- **subquery** (*Alias*) – object_parameter_definition_sq or relationship_parameter_definition_sq
- **sets_with_ids** (*dict*) – mapping from relationship_class ids to corresponding *Set* objects
- **classes_with_ignored_parameters** (*set, optional*) – a set of problematic relationship_class names; if not None, relationship_class names are added to this set in case of errors instead of raising an exception

Returns a map from parameter name to the parsed default value

Return type dict

spinetoolbox.spine_io.exporters.gdx.**_update_using_existing_relationship_parameter_values**(*pa*
db
se
cl

Updates an existing relationship parameter dict using actual parameter values.

Parameters

- **parameters** (*dict*) – a mapping from relationship parameter names to *Parameter* objects to update
- **db_map** (*DatabaseMapping or DiffDatabaseMapping*) – a database map
- **sets_with_ids** (*dict*) – mapping from relationship_class ids to corresponding *Set* objects
- **classes_with_ignored_parameters** (*set, optional*) – a set of problematic relationship_class names; if not None, class names are added to this set in case of errors instead of raising an exception

spinetoolbox.spine_io.exporters.gdx.**domain_names_and_records**(*db_map*)

Returns a list of domain names and a map from a name to list of record keys.

Parameters **db_map** (*DatabaseMapping or DiffDatabaseMapping*) – a database map

Returns a tuple containing set of domain names and a dict from domain name to its records

Return type tuple

spinetoolbox.spine_io.exporters.gdx.**set_names_and_records**(*db_map*)

Returns a list of set names and a map from a name to list of record keys.

Parameters **db_map** (*spinedb_api.DatabaseMapping or spinedb_api.DiffDatabaseMapping*) – a database map

Returns a tuple containing a set of set names and a dict from set name to its records

Return type tuple

class spinetoolbox.spine_io.exporters.gdx.**IndexingSetting**(*indexed_parameter*, *set_name*)

Settings for indexed value expansion for a single Parameter.

parameter
a parameter containing indexed values

Type *Parameter*

indexing_domain_name
indexing domain's name

Type str

picking
index picking

Type *FixedPicking* or GeneratePicking

index_position
where to insert the new index when expanding a parameter

Type int

set_name
name of the domain or set to which this parameter belongs

Type str

Parameters

- **indexed_parameter** (Parameter) – a parameter containing indexed values
- **set_name** (*str*) – name of the original entity_class to which this parameter belongs

append_parameter(*self*, *parameter*)
Adds indexes and values from another parameter.

Parameters **parameter** (Parameter) – parameter to slurp

to_dict(*self*)
Serializes settings to dict.

Returns serialized settings

Return type dict

static from_dict(*setting_dict*, *parameter*, *set_name*)
Restores serialized setting from dict.

Parameters

- **setting_dict** (*dict*) – serialized settings
- **parameter** (Parameter) – indexed parameter
- **set_name** (*str*) – name of the set containing the parameter

Returns restored setting

Return type *IndexingSetting*

spinetoolbox.spine_io.exporters.gdx.**make_indexing_settings**(*db_map*, *none_fallback*, *logger*)
Constructs skeleton indexing settings for parameter indexed value expansion.

Parameters

- **db_map** (*spinedb_api.DatabaseMapping or spinedb_api.DiffDatabaseMapping*) – a database mapping
- **none_fallback** (NoneFallback) – how to handle None values
- **logger** (LoggerInterface, *optional*) – a logger

Returns a mapping from parameter name to IndexingSetting

Return type dict

spinetoolbox.spine_io.exporters.gdx.**_object_indexing_settings**(*db_map*, *none_fallback*, *logger*)
Constructs skeleton indexing settings from object parameters.

Parameters

- **db_map** (*spinedb_api.DatabaseMapping or spinedb_api.DiffDatabaseMapping*) – a database mapping
- **none_fallback** – how to handle Nones
- **logger** (*LoggingInterface, optional*) – a logger

Returns a mapping from parameter name to IndexingSetting

Return type dict

spinetoolbox.spine_io.exporters.gdx.**_relationship_indexing_settings**(*db_map*, *none_fallback*, *logger*)
Constructs skeleton indexing settings from relationship parameters.

Parameters

- **db_map** (*spinedb_api.DatabaseMapping or spinedb_api.DiffDatabaseMapping*) – a database mapping
- **none_fallback** (NoneFallback) – how to handle Nones
- **logger** (*LoggingInterface, optional*) – a logger

Returns a mapping from parameter name to IndexingSetting

Return type dict

spinetoolbox.spine_io.exporters.gdx.**_add_to_indexing_settings**(*settings*, *parameter_name*, *entity_class_name*, *dimensions*, *parsed_value*, *index_keys*, *classes_with_unsupported_value_types*)

Adds parameter to indexing settings.

Parameters

- **settings** (*dict*) – indexing settings
- **parameter_name** (*str*) – parameter's name
- **entity_class_name** (*str*) – name of the object or relationship_class the parameter belongs to
- **dimensions** (*list*) – a list of parameter's domain names
- **parsed_value** (*IndexedValue*) – parsed parameter_value
- **index_keys** (*tuple*) – parameter's keys
- **classes_with_unsupported_value_types** (*set, optional*) – entity_class names with unsupported value types

spinetoolbox.spine_io.exporters.gdx.**update_indexing_settings**(*old_indexing_settings*, *new_indexing_settings*, *set_settings*)

Returns new indexing settings merged from old and new ones.

Entries that do not exist in old settings will be removed. If entries exist in both settings the old one will be chosen if both entries are 'equal', otherwise the new entry will override the old one. Entries existing in new settings only will be added.

Parameters

- **old_indexing_settings** (*dict*) – settings to be updated
- **new_indexing_settings** (*dict*) – settings used for updating
- **set_settings** (SetSettings) – new set settings

Returns merged old and new indexing settings

Return type dict

spinetoolbox.spine_io.exporters.gdx.**indexing_settings_to_dict**(*settings*)

Stores indexing settings to a JSON compatible dictionary.

Parameters **settings** (*dict*) – a mapping from parameter name to IndexingSetting.

Returns a JSON serializable dictionary

Return type dict

spinetoolbox.spine_io.exporters.gdx.**indexing_settings_from_dict**(*settings_dict*, *db_map*, *none_fallback*, *logger*)

Restores indexing settings from a json compatible dictionary.

Parameters

- **settings_dict** (*dict*) – a JSON compatible dictionary representing parameter indexing settings.
- **db_map** (*DatabaseMapping*) – database mapping
- **none_fallback** (NoneFallback) – how to handle None parameter values
- **logger** (LoggerInterface, *optional*) – a logger

Returns a dictionary mapping parameter name to IndexingSetting.

Return type dict

spinetoolbox.spine_io.exporters.gdx.**_find_indexed_parameter**(*parameter_name*, *db_map*, *none_fallback*, *logger=None*)

Searches for parameter_name in db_map and returns Parameter and its entity_class name.

spinetoolbox.spine_io.exporters.gdx.**_create_additional_domains**(*set_settings*)

Generates additional domains found in the settings.

Parameters **set_settings** (SetSettings) – settings

Returns a list of additional *Set* objects

Return type list

spinetoolbox.spine_io.exporters.gdx.**_exported_set_names**(*names*, *set_settings*)

Returns a set of names of the domains that are marked for exporting.

Parameters

- **names** (*set*) – list of all domain or set names
- **set_settings** (SetSettings) – settings

Returns names that should be exported

Return type set of str

spinetoolbox.spine_io.exporters.gdx.**sort_sets**(*sets*, *order*)

Sorts a list of sets according to sorted_names

Parameters

- **sets** (*list*) – *Set* objects to be sorted
- **order** (*dict*) – a mapping from set name to index

Returns sorted *Set* objects

Return type list

spinetoolbox.spine_io.exporters.gdx.**sort_records_inplace**(*sets*, *set_settings*)

Sorts the record lists of given domains according to the order given in settings.

Parameters

- **sets** (*list of Set*) – a list of *Set* objects whose records are to be sorted
- **set_settings** (SetSettings) – settings that define the sorting order

spinetoolbox.spine_io.exporters.gdx.**extract_domain**(*domains*, *name_to_extract*)
Extracts the domain with given name from a list of domains.

Parameters

- **domains** (*list*) – a list of Set objects
- **name_to_extract** (*str*) – name of the domain to be extracted

Returns a tuple (list, Set) of the modified domains list and the extracted Set object

spinetoolbox.spine_io.exporters.gdx.**to_gdx_file**(*database_map*, *file_name*, *set_settings*, *indexing_settings*, *merging_settings*, *none_fallback*, *none_export*, *gams_system_directory=None*, *logger=None*)
Exports given database map into .gdx file.

Parameters

- **database_map** (*spinedb_api.DatabaseMapping or spinedb_api.DiffDatabaseMapping*) – a database to export
- **file_name** (*str*) – output file name
- **set_settings** (SetSettings) – export settings
- **indexing_settings** (*dict*) – a dictionary containing settings for indexed parameter expansion
- **merging_settings** (*dict*) – a list of merging settings for parameter merging
- **none_fallback** (NoneFallback) – options how to handle none parameter values on database read
- **none_export** (NoneExport) – option how to handle none parameter values on export
- **gams_system_directory** (*str, optional*) – path to GAMS system directory or None to let GAMS choose one for you
- **logger** (*LoggingInterface, optional*) – a logger; if None given all error conditions raise GdxExportException otherwise some errors are logged and ignored

spinetoolbox.spine_io.exporters.gdx.**make_set_settings**(*database_map*)
Builds a *SetSettings* object from given database.

Parameters **database_map** (*spinedb_api.DatabaseMapping or spinedb_api.DiffDatabaseMapping*) – a database from which domains, sets, records etc are extracted

Returns settings needed for exporting the entities and class from the given `database_map`

Return type *SetSettings*

class spinetoolbox.spine_io.exporters.gdx.**SetSettings**(*domain_names*, *set_names*, *records*, *domain_tiers=None*, *set_tiers=None*, *metadatas=None*, *global_parameters_domain_name=''*)
This class holds the settings for domains, sets and records needed by *to_gdx_file()* for .gdx export.

SetSettings keeps track which domains, sets and records are exported into the .gdx file and in which order they are written to the file. This order is paramount for some models, like TIMES.

Parameters

- **domain_names** (*set of str*) – domain names
- **set_names** (*set of str*) – set names
- **records** (*dict*) – a mapping from domain or set name to *Records*
- **domain_tiers** (*dict, optional*) – a mapping from domain name to tier
- **set_tiers** (*dict, optional*) – a mapping from set name to tier
- **metadatas** (*dict, optional*) – a mapping from domain or set name to *SetMetadata*
- **global_parameters_domain_name** (*str, optional*) – name of the domain whose parameters should be exported as scalars

domain_names
domain names

domain_tiers
a mapping from domain name to tier

set_names
set names

set_tiers
a mapping from set name to tier

global_parameters_domain_name
the name of the domain, parameters of which should be exported as GAMS scalars

metadata (*self*, *name*)
Returns the metadata for given domain/set.

Parameters **name** (*str*) – set/domain name

Returns metadata

Return type Metadata

is_exportable (*self*, *set_name*)
Returns True if the domain or set with the given name is exportable, False otherwise.

Parameters **set_name** (*str*) – domain/set name

add_or_replace_domain (*self*, *domain_name*, *records*, *metadata*)
Adds a new domain or replaces an existing domain's records and metadata.

Parameters

- **domain_name** (*str*) – a domain to add/replace
- **records** (*Records*) – domain's records
- **metadata** (*SetMetadata*) – domain's metadata

Returns True if a new domain was added, False if an existing domain was replaced

Return type bool

remove_domain (*self*, *domain_name*)
Erases domain.

Parameters **domain_name** (*str*) – name of the domain to remove

records (*self*, *name*)
Returns the records of a given domain or set.

Parameters **name** (*str*) – domain or set name

Returns domain's or set's records

Return type *Records*

update_records (*self*, *set_name*, *records*)
Updates the records of given domain or set.

Parameters

- **set_name** (*str*) – domain or set name
- **records** (Records) – updated records

update (*self*, *updating_settings*)
Updates the settings by merging with another one.

All domains, sets and records that are in both settings (common) or in *updating_settings* (new) are retained. Common elements are ordered the same way they were ordered in the original settings. New elements are appended to the common ones in the order they were in *updating_settings*

Parameters **updating_settings** (SetSettings) – settings to merge with

to_dict (*self*)
Serializes the this object to a dict.

Returns serialized settings

Return type dict

static from_dict (*dictionary*)
Deserializes *SetSettings* from a dict.

Parameters **dictionary** (*dict*) – serialized settings

Returns restored settings

Return type *SetSettings*

class spinetoolbox.spine_io.exporters.gdx.**ExportFlag**
Bases: enum.Enum

Options for exporting Set objects.

Create and return a new object. See help(type) for accurate signature.

EXPORTABLE = 1
User has declared that the set should be exported.

NON_EXPORTABLE = 2
User has declared that the set should not be exported.

class spinetoolbox.spine_io.exporters.gdx.**Origin**
Bases: enum.Enum

Domain or set origin.

Create and return a new object. See help(type) for accurate signature.

DATABASE = 1
Set exists in the database.

INDEXING = 2
Set has been generated for indexed parameter indexing.

MERGING = 3
Set has been generated for parameter merging.

class spinetoolbox.spine_io.exporters.gdx.**SetMetadata**(*exportable=ExportFlag.EXPORTABLE*, *origin=Origin.DATABASE*)
This class holds some additional configuration for Sets.

exportable
set's export flag

Type *ExportFlag*

origin
True if the domain does not exist in the database but is supplied separately.

Type bool

description
set's description or None if its origin is from database

Type str

Parameters

- **exportable** (ExportFlag) – set's export flag
- **origin** (Origin) – where the set comes from

__eq__(*self*, *other*)
Returns True if other is equal to this metadata.

is_additional(*self*)
Returns True if Set does not originate from the database.

is_exportable(*self*)
Returns True if Set should be exported.

to_dict(*self*)
Serializes metadata to a dictionary.

static from_dict(*metadata_dict*)
Deserializes metadata from a dictionary.

spinetoolbox.spine_io.importers

Intentionally empty.

author

P. Vennström (VTT)

date 1.6.2019

Submodules

spinetoolbox.spine_io.importers.csv_reader

Contains CSVConnector class and a help function.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

CSVConnector	Template class to read data from another QThread.

Functions

select_csv_file(parent=None)	Launches QFileDialog with no filter

spinetoolbox.spine_io.importers.csv_reader.**select_csv_file**(*parent=None*)
Launches QFileDialog with no filter

class spinetoolbox.spine_io.importers.csv_reader.**CSVConnector**(*settings*)
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

Parameters **settings** (*dict, optional*) – connector specific settings or None

DISPLAY_NAME = Text/CSV
"Text/CSV

Type name of data source, ex

_ENCODINGS = ['utf-8', 'utf-16', 'utf-32', 'ascii', 'iso-8859-1', 'iso-8859-2']
List of available text encodings

OPTIONS
dict with option specification for source.

SELECT_SOURCE_UI
Modal widget that returns source object and action (OK, CANCEL)

connect_to_source(*self*, *source*)
saves filepath

Parameters **source** (*str*) – filepath

disconnect(*self*)
Disconnect from connected source.

get_tables(*self*)
Returns a mapping from file name to options.

Returns dict

static parse_options(*options*)
Parses options dict to dialect and quotechar options for csv.reader

Parameters **options** (*dict*) – dict with options: “encoding”: file text encoding “delimiter”: file delimiter “quotechar”: file quotechar “has_header”: if first row should be treated as a header “skip”: how many rows should be skipped

Returns

tuple dialect for csv.reader, quotechar for csv.reader and number of rows to skip

Return type tuple(dict, bool, integer)

file_iterator(*self*, *options*, *max_rows*)
creates an iterator that reads max_rows number of rows from text file

Parameters

- **options** (*dict*) – dict with options:
- **max_rows** (*integer*) – max number of rows to read, if -1 then read all rows

Returns iterator of csv file

Return type iterator

get_data_iterator(*self*, *table*, *options*, *max_rows=-1*)
Creates an iterator for the file in self.filename

Parameters

- **table** (*string*) – ignored, used in abstract IOWorker class
- **options** (*dict*) – dict with options

Keyword Arguments **max_rows** (*int*) – how many rows of data to read, if -1 read all rows (default: {-1})

Returns

Return type tuple

spinetoolbox.spine_io.importers.excel_reader

Contains ExcelConnector class and a help function.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

ExcelConnector	Template class to read data from another QThread.

Functions

select_excel_file(parent=None)	Launches QFileDialog with .xlsx and friends filter
get_mapped_data_from_xlsx(filepath)	Returns mapped data from given Excel file assuming it has the default Spine Excel format.
create_mapping_from_sheet(worksheet)	Checks if sheet has the default Spine Excel format, if so creates a

spinetoolbox.spine_io.importers.excel_reader.**select_excel_file**(*parent=None*)
Launches QFileDialog with .xlsx and friends filter

class spinetoolbox.spine_io.importers.excel_reader.**ExcelConnector**(*settings*)
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

Parameters **settings** (*dict, optional*) – connector specific settings or None

DISPLAY_NAME = Excel

OPTIONS

SELECT_SOURCE_UI

connect_to_source(*self*, *source*)
saves filepath

Parameters **{str} -- filepath** (*source*) –

disconnect(*self*)
Disconnect from connected source.

get_tables(*self*)
Method that should return Excel sheets as mappings and their options.

Returns Sheets as mappings and options for each sheet or an empty dictionary if no workbook.

Return type dict

Raises Exception – If something goes wrong.

get_data_iterator(*self*, *table*, *options*, *max_rows=-1*)
Return data read from data source table in table. If max_rows is specified only that number of rows.

get_mapped_data(*self*, *tables_mappings*, *options*, *table_types*, *table_row_types*, *max_rows=-1*)
Overrides io_api method to check for some parameter_value types.

spinetoolbox.spine_io.importers.excel_reader.**get_mapped_data_from_xlsx**(*filepath*)
Returns mapped data from given Excel file assuming it has the default Spine Excel format.

Parameters **filepath** (*str*) – path to Excel file

spinetoolbox.spine_io.importers.excel_reader.**create_mapping_from_sheet**(*worksheet*)
Checks if sheet has the default Spine Excel format, if so creates a mapping object for each sheet.

spinetoolbox.spine_io.importers.gdx_connector

Contains GDXConnector class and a help function.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

GdxConnector	Template class to read data from another QThread.

Functions

select_gdx_file(parent=None)	Launches QFileDialog with .gdx filter

spinetoolbox.spine_io.importers.gdx_connector.**select_gdx_file**(*parent=None*)
Launches QFileDialog with .gdx filter

class spinetoolbox.spine_io.importers.gdx_connector.**GdxConnector**(*settings*)
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

Parameters **settings** (*dict*) – a dict from "gams_directory" to GAMS path; if the argument is None or the path is empty or None, a default path is used

DISPLAY_NAME = Gdx
name of data source

OPTIONS
dict with option specification for source

SELECT_SOURCE_UI
Modal widget that returns source object and action (OK, CANCEL).

__exit__(*self*, *exc_type*, *exc_value*, *traceback*)

__del__(*self*)

connect_to_source(*self*, *source*)
Connects to given .gdx file.

Parameters **source** (*str*) – path to .gdx file.

disconnect(*self*)
Disconnects from connected source.

get_tables(*self*)
Returns a list of table names.

GAMS scalars are also regarded as tables.

Returns Table names in list

Return type list(str)

get_data_iterator(*self*, *table*, *options*, *max_rows=-1*)
Creates an iterator for the data source

Parameters

- **table** (*string*) – table name
- **options** (*dict*) – dict with options

Keyword Arguments **max_rows** (*int*) – ignored

Returns data iterator, list of column names, number of columns

Return type tuple

spinetoolbox.spine_io.importers.json_reader

Contains JSONConnector class.

author

M. Marin (KTH)

date 10.2.2020

Module Contents

Classes

JSONConnector	Template class to read data from another QThread.

Functions

select_json_file(parent=None)	Launches QFileDialog with .json filter
_tabulize_json(obj)	
_tabulize_json_object(obj)	
_tabulize_json_array(arr)	

spinetoolbox.spine_io.importers.json_reader.**select_json_file**(*parent=None*)
Launches QFileDialog with .json filter

class spinetoolbox.spine_io.importers.json_reader.**JSONConnector**(*settings*)
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

Parameters **settings** (*dict, optional*) – connector specific settings or None

DISPLAY_NAME = JSON

OPTIONS

SELECT_SOURCE_UI

connect_to_source(*self*, *source*)
saves filepath

Parameters **{str} -- filepath** (*source*) –

disconnect(*self*)
Disconnect from connected source.

get_tables(*self*)
Method that should return a list of table names, list(str)

Raises NotImplementedError – [description]

file_iterator(*self*, *table*, *options*, *max_rows=-1*)

get_data_iterator(*self*, *table*, *options*, *max_rows=-1*)
Return data read from data source table in table. If max_rows is specified only that number of rows.

spinetoolbox.spine_io.importers.json_reader.**_tabulize_json**(*obj*)

spinetoolbox.spine_io.importers.json_reader.**_tabulize_json_object**(*obj*)

spinetoolbox.spine_io.importers.json_reader.**_tabulize_json_array**(*arr*)

spinetoolbox.spine_io.importers.sqlalchemy_connector

Contains SqlAlchemyConnector class and a help function.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

SqlAlchemyConnector	Template class to read data from another QThread.

Functions

select_sa_conn_string(parent=None)	Launches QInputDialog for entering connection string

spinetoolbox.spine_io.importers.sqlalchemy_connector.**select_sa_conn_string**(*parent=None*)
Launches QInputDialog for entering connection string

class spinetoolbox.spine_io.importers.sqlalchemy_connector.**SqlAlchemyConnector**(*settings*)
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

Parameters **settings** (*dict, optional*) – connector specific settings or None

DISPLAY_NAME = SqlAlchemy

OPTIONS

SELECT_SOURCE_UI

connect_to_source(*self*, *source*)
saves filepath

Parameters **{str} -- filepath** (*source*) –

disconnect(*self*)
Disconnect from connected source.

get_tables(*self*)
Method that should return a list of table names, list(str)

Returns Table names in list

Return type list(str)

get_data_iterator(*self*, *table*, *options*, *max_rows=-1*)
Creates a iterator for the file in self.filename

Parameters

- **{string} -- table name**(*table*) –
- **{dict} -- dict with options, not used**(*options*) –

Keyword Arguments **{int} -- how many rows of data to read, if -1 read all rows (default**(*max_rows*) – {-1})

Returns [type] – [description]

Submodules

spinetoolbox.spine_io.connection_manager

Contains ConnectionManager class.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

ConnectionManager	Class to manage data connections in another thread.
ConnectionWorker	A class for delegating SourceConnection operations to another QThread.

class spinetoolbox.spine_io.connection_manager.**ConnectionManager**(*connection*, *connection_settings*, *parent=None*)

Bases: PySide2.QtCore.QObject

Class to manage data connections in another thread.

Parameters **connection**(*class*) – A class derived from *SourceConnection*, e.g. *CSVConnector*

start_table_get

start_data_get

start_mapped_data_get

connection_failed

connection_ready

connection_closed

error

fetching_data

data_ready

tables_ready

mapped_data_ready

current_table_changed
Emitted when the current table has changed.

connection

current_table

is_connected

table_options

table_types

table_row_types

source

source_type

set_table(*self*, *table*)
Sets the current table of the data source.

Parameters **table** (*str*) – table name

request_tables(*self*)
Get tables tables from source, emits two singals, fetchingData: ConnectionManager is busy waiting for data startTableGet: a signal that the worker in another thread is listening to know when to run get a list of table names.

request_data(*self*, *table=None*, *max_rows=-1*)
Request data from emits dataReady to with data

Keyword Arguments

- **{str} -- which table to get data from (default** (*table*) – {None})
- **{int} -- how many rows to read (default** (*max_rows*) – {-1})

request_mapped_data(*self*, *table_mappings*, *max_rows=-1*)
Get mapped data from csv file

Parameters **{dict} -- dict with filename as key and a list of mappings as value** (*table_mappings*) –

Keyword Arguments **{int} -- number of rows to read, if -1 read all rows (default** (*max_rows*) – {-1})

connection_ui(*self*)
launches a modal ui that prompts the user to select source.

ex: fileselect if source is a file.

init_connection(*self*)
Creates a Worker and a new thread to read source data. If there is an existing thread close that one.

_handle_connection_ready(*self*)

_handle_tables_ready(*self*, *table_options*)

update_options(*self*, *options*)

get_current_options(*self*)

get_current_option_value(*self*, *option_key*)
Returns the value for option_key for the current table or the default value.

set_table_options(*self*, *options*)
Sets connection manager options for current connector

Parameters **options** (*dict*) – settings for the tables

set_table_types(*self*, *types*)
Sets connection manager types for current connector

Parameters **types** (*dict*) – dict with types settings, column (int) as key, type as value

set_table_row_types(*self*, *types*)
Sets connection manager types for current connector

Parameters **{dict} -- Dict with types settings, row** (*types*) –

close_connection(*self*)
Closes and deletes thread and worker

class spinetoolbox.spine_io.connection_manager.**ConnectionWorker**(*source*, *connection*, *connection_settings*, *parent=None*)

Bases: PySide2.QtCore.QObject

A class for delegating SourceConnection operations to another QThread.

Parameters

- **source** (*str*) – path of the source file
- **connection** (*class*) – A class derived from *SourceConnection* for connecting to the source file

connectionFailed

error

connectionReady

tablesReady

dataReady

mappedDataReady

init_connection(*self*)
Connect to data source

tables(*self*)

data(*self*, *table*, *options*, *max_rows*)

mapped_data(*self*, *table_mappings*, *options*, *types*, *table_row_types*, *max_rows*)

disconnect(*self*)

spinetoolbox.spine_io.gdx_utils

Utility functions for .gdx import/export.

author

A. Soininen (VTT)

date 7.1.2020

Module Contents

Functions

_python_interpreter_bitness()	Returns 64 for 64bit Python interpreter or 32 for 32bit interpreter.
_windows_dlls_exist(gams_path)	Returns True if requred DLL files exist in given GAMS installation path.
find_gams_directory()	Returns GAMS installation directory or None if not found.

spinetoolbox.spine_io.gdx_utils.**_python_interpreter_bitness**()
Returns 64 for 64bit Python interpreter or 32 for 32bit interpreter.

spinetoolbox.spine_io.gdx_utils.**_windows_dlls_exist**(*gams_path*)
Returns True if requred DLL files exist in given GAMS installation path.

spinetoolbox.spine_io.gdx_utils.**find_gams_directory**()
Returns GAMS installation directory or None if not found.

On Windows systems, this function looks for *gams.location* in registry; on other systems the *PATH* environment variable is checked.

Returns a path to GAMS installation directory or None if not found.

spinetoolbox.spine_io.io_api

Contains a class template for a data source connector used in import ui.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

Classes

SourceConnection	Template class to read data from another QThread.

spinetoolbox.spine_io.io_api.**TYPE_STRING_TO_CLASS**

spinetoolbox.spine_io.io_api.**TYPE_CLASS_TO_STRING**

class spinetoolbox.spine_io.io_api.**SourceConnection**(*settings*)

Template class to read data from another QThread.

Parameters **settings** (*dict, optional*) – connector specific settings or None

DISPLAY_NAME = unnamed source

OPTIONS

SELECT_SOURCE_UI

connect_to_source(*self*, *source*)

Connects to source, ex: connecting to a database where source is a connection string.

Parameters **{} -- object with information on source to be connected to, ex** (*source*) – filepath string for a csv connection

disconnect(*self*)

Disconnect from connected source.

get_tables(*self*)

Method that should return a list of table names, list(str)

Raises NotImplementedError – [description]

get_data_iterator(*self*, *table*, *options*, *max_rows=-1*)

Function that should return a data iterator, data header and number of columns.

get_data(*self*, *table*, *options*, *max_rows=-1*)

Return data read from data source table in table. If max_rows is specified only that number of rows.

get_mapped_data(*self*, *tables_mappings*, *options*, *table_types*, *table_row_types*, *max_rows=-1*)

Reads all mappings in dict tables_mappings, where key is name of table and value is the mappings for that table. emits mapped data when ready.

spinetoolbox.spine_io.type_conversion

Type conversion functions.

author

P. Vennström (VTT)

date 21.11.2019

Module Contents

Classes

NewIntegerSequenceDateTimeConvertSpecDialog	
ConvertSpec	

Continued on next page

Table 211 – continued from previous page

DateTimeConvertSpec	
DurationConvertSpec	
FloatConvertSpec	
StringConvertSpec	
BooleanConvertSpec	
IntegerSequenceDateTimeConvertSpec	Initialize self. See help(type(self)) for accurate signature.

Functions

value_to_convert_spec(value)	

class spinetoolbox.spine_io.type_conversion.**NewIntegerSequenceDateTimeConvertSpecDialog**(**ar*
***k*

Bases: PySide2.QtWidgets.QDialog

_validate(*self*)

get_spec(*self*)

spinetoolbox.spine_io.type_conversion.**value_to_convert_spec**(*value*)

class spinetoolbox.spine_io.type_conversion.**ConvertSpec**

DISPLAY_NAME =

RETURN_TYPE

convert_function(*self*)

to_json_value(*self*)

class spinetoolbox.spine_io.type_conversion.**DateTimeConvertSpec**
Bases: *spinetoolbox.spine_io.type_conversion.ConvertSpec*

DISPLAY_NAME = datetime

RETURN_TYPE

class spinetoolbox.spine_io.type_conversion.**DurationConvertSpec**
Bases: *spinetoolbox.spine_io.type_conversion.ConvertSpec*

DISPLAY_NAME = duration

RETURN_TYPE

class spinetoolbox.spine_io.type_conversion.**FloatConvertSpec**
Bases: *spinetoolbox.spine_io.type_conversion.ConvertSpec*

DISPLAY_NAME = float

RETURN_TYPE

class spinetoolbox.spine_io.type_conversion.**StringConvertSpec**
Bases: *spinetoolbox.spine_io.type_conversion.ConvertSpec*

DISPLAY_NAME = string

RETURN_TYPE

class spinetoolbox.spine_io.type_conversion.**BooleanConvertSpec**
Bases: *spinetoolbox.spine_io.type_conversion.ConvertSpec*

DISPLAY_NAME = boolean

RETURN_TYPE

convert_function(*self*)

class spinetoolbox.spine_io.type_conversion.**IntegerSequenceDateTimeConvertSpec**(*start_datetime*, *start_int*, *duration*)
Bases: *spinetoolbox.spine_io.type_conversion.ConvertSpec*

Initialize self. See help(type(self)) for accurate signature.

DISPLAY_NAME = integer sequence datetime

RETURN_TYPE

convert_function(*self*)

to_json_value(*self*)

spinetoolbox.widgets

Init file for widgets package. Intentionally empty.

author
P. Savolainen (VTT)

date 3.1.2018

Submodules

spinetoolbox.widgets.about_widget

A widget for presenting basic information about the application.

author
P. Savolainen (VTT)

date 14.12.2017

Module Contents

Classes

AboutWidget	About widget class.

class spinetoolbox.widgets.about_widget.**AboutWidget**(*toolbox*)
Bases: PySide2.QtWidgets.QWidget

About widget class.

Parameters **toolbox** (ToolboxUI) – QMainWindow instance

calc_pos(*self*)
Calculate the top-left corner position of this widget in relation to main window position and size in order to show about window in the middle of the main window.

setup_license_text(*self*)
Add license to QTextBrowser.

keyPressEvent(*self*, *e*)
Close form when Escape, Enter, Return, or Space bar keys are pressed.

Parameters **e** (*QKeyEvent*) – Received key press event.

closeEvent(*self*, *event=None*)
Handle close window.

Parameters **event** (*QEvent*) – Closing event if 'X' is clicked.

mousePressEvent(*self*, *e*)
Save mouse position at the start of dragging.

Parameters **e** (*QMouseEvent*) – Mouse event

mouseReleaseEvent(*self*, *e*)
Save mouse position at the end of dragging.

Parameters **e** (*QMouseEvent*) – Mouse event

mouseMoveEvent(*self*, *e*)
Moves the window when mouse button is pressed and mouse cursor is moved.

Parameters **e** (*QMouseEvent*) – Mouse event

spinetoolbox.widgets.add_project_item_widget

Widget shown to user when a new Project Item is created.

author

P. Savolainen (VTT)

date 19.1.2017

Module Contents

Classes

AddProjectItemWidget	A widget to query user's preferences for a new item.

class spinetoolbox.widgets.add_project_item_widget.**AddProjectItemWidget**(*toolbox*, *x*, *y*, *class_*, *spec=''*)

Bases: PySide2.QtWidgets.QWidget

A widget to query user's preferences for a new item.

toolbox
Parent widget

Type *ToolboxUI*

x
X coordinate of new item

Type int

y
Y coordinate of new item

Type int

Initialize class.

connect_signals(*self*)
Connect signals to slots.

handle_name_changed(*self*)
Update label to show upcoming folder name.

handle_ok_clicked(*self*)
Check that given item name is valid and add it to project.

call_add_item(*self*)
Creates new Item according to user's selections.

Must be reimplemented by subclasses.

keyPressEvent(*self*, *e*)
Close Setup form when escape key is pressed.

Parameters **e** (*QKeyEvent*) – Received key press event.

closeEvent(*self*, *event=None*)
Handle close window.

Parameters **event** (*QEvent*) – Closing event if 'X' is clicked.

spinetoolbox.widgets.array_editor

Contains an editor widget array type parameter values.

author

A. Soininen (VTT)

date 25.3.2020

Module Contents

Classes

ArrayEditor

class spinetoolbox.widgets.array_editor.**ArrayEditor**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

set_value(*self*, *value*)
Sets the parameter_value for editing in this widget.

value(*self*)

_check_if_plotting_enabled(*self*, *type_name*)

_change_value_type(*self*, *type_name*)

_show_table_context_menu(*self*, *pos*)
Shows the table's context menu.

_update_plot(*self*, *topLeft=None*, *bottomRight=None*, *roles=None*)
Updates the plot widget.

spinetoolbox.widgets.commit_dialog

Classes for custom QDialogs to add edit and remove database items.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

Classes

CommitDialog	A dialog to query user's preferences for new commit.

class spinetoolbox.widgets.commit_dialog.**CommitDialog**(*parent*, **db_names*)
Bases: PySide2.QtWidgets.QDialog

A dialog to query user's preferences for new commit.

Initialize class.

Parameters

- **parent** (*QWidget*) – the parent widget
- **db_names** (*str*) – database names

receive_text_changed(*self*)
Called when text changes in the commit msg text edit. Enable/disable commit button accordingly.

spinetoolbox.widgets.custom_delegates

Custom item delegates.

author

M. Marin (KTH)

date 1.9.2018

Module Contents

Classes

ComboBoxDelegate	
LineEditDelegate	A delegate that places a fully functioning QLineEdit.
CheckBoxDelegate	A delegate that places a fully functioning QCheckBox.
ForeignKeysDelegate	A QComboBox delegate with checkboxes.

class spinetoolbox.widgets.custom_delegates.**ComboBoxDelegate**(*items*)

Bases: PySide2.QtWidgets.QStyledItemDelegate

createEditor(*self*, *parent*, *option*, *index*)

paint(*self*, *painter*, *option*, *index*)

setEditorData(*self*, *editor*, *index*)

setModelData(*self*, *editor*, *model*, *index*)

updateEditorGeometry(*self*, *editor*, *option*, *index*)

_finalize_editing(*self*, *editor*)

class spinetoolbox.widgets.custom_delegates.**LineEditDelegate**

Bases: PySide2.QtWidgets.QStyledItemDelegate

A delegate that places a fully functioning QLineEdit.

parent
: either data store or spine datapackage widget

  Type QMainWindow

data_committed

createEditor(*self*, *parent*, *option*, *index*)
: Return CustomLineEditor.

setEditorData(*self*, *editor*, *index*)
: Init the line editor with previous data from the index.

setModelData(*self*, *editor*, *model*, *index*)
: Send signal.

class spinetoolbox.widgets.custom_delegates.**CheckBoxDelegate**(*parent*, *centered=True*)

Bases: PySide2.QtWidgets.QStyledItemDelegate

A delegate that places a fully functioning QCheckBox.

parent
: either toolbox or spine datapackage widget

  Type QMainWindow

centered
: whether or not the checkbox should be center-aligned in the widget

  Type bool

data_committed

createEditor(*self*, *parent*, *option*, *index*)
Important, otherwise an editor is created if the user clicks in this cell. ** Need to hook up a signal to the model.

paint(*self*, *painter*, *option*, *index*)
Paint a checkbox without the label.

editorEvent(*self*, *event*, *model*, *option*, *index*)
Change the data in the model and the state of the checkbox when user presses left mouse button and this cell is editable. Otherwise do nothing.

setModelData(*self*, *editor*, *model*, *index*)
Do nothing. Model data is updated by handling the *data_committed* signal.

get_checkbox_rect(*self*, *option*)

class spinetoolbox.widgets.custom_delegates.**ForeignKeysDelegate**
Bases: PySide2.QtWidgets.QStyledItemDelegate

A QComboBox delegate with checkboxes.

parent
spine datapackage widget

Type *SpineDatapackageWidget*

data_committed

_close_editor(*self*, *editor*, *index*)
Closes editor. Needed by SearchBarEditor.

createEditor(*self*, *parent*, *option*, *index*)
Return editor.

updateEditorGeometry(*self*, *editor*, *option*, *index*)

setModelData(*self*, *editor*, *model*, *index*)
Send signal.

spinetoolbox.widgets.custom_editors

Custom editors for model/view programming.

author

M. Marin (KTH)

date 2.9.2018

Module Contents

Classes

CustomLineEditor	A custom QLineEdit to handle data from models.
ParameterValueLineEditor	A custom QLineEdit to handle data from models.
CustomComboEditor	A custom QComboBox to handle data from models.
_CustomLineEditDelegate	A delegate for placing a CustomLineEditor on the first row of SearchBarEditor.

Continued on next page

Table 218 – continued from previous page

SearchBarEditor	A Google-like search bar, implemented as a QTableView with a _CustomLineEditDelegate in the first row.
CheckListEditor	A check list editor.
_IconPainterDelegate	A delegate to highlight decorations in a QListWidget.
IconColorEditor	An editor to let the user select an icon and a color for an object_class.

class spinetoolbox.widgets.custom_editors.**CustomLineEditor**
Bases: PySide2.QtWidgets.QLineEdit

A custom QLineEdit to handle data from models.

set_data(*self*, *data*)

data(*self*)

keyPressEvent(*self*, *event*)
Prevents shift key press to clear the contents.

class spinetoolbox.widgets.custom_editors.**ParameterValueLineEditor**
Bases: *spinetoolbox.widgets.custom_editors.CustomLineEditor*

A custom QLineEdit to handle data from models.

set_data(*self*, *data*)

data(*self*)

class spinetoolbox.widgets.custom_editors.**CustomComboEditor**
Bases: PySide2.QtWidgets.QComboBox

A custom QComboBox to handle data from models.

data_committed

set_data(*self*, *current_text*, *items*)

data(*self*)

class spinetoolbox.widgets.custom_editors.**_CustomLineEditDelegate**
Bases: PySide2.QtWidgets.QItemDelegate

A delegate for placing a CustomLineEditor on the first row of SearchBarEditor.

text_edited

setModelData(*self*, *editor*, *model*, *index*)

createEditor(*self*, *parent*, *option*, *index*)
Create editor and 'forward' *textEdited* signal.

eventFilter(*self*, *editor*, *event*)
Handle all sort of special cases.

class spinetoolbox.widgets.custom_editors.**SearchBarEditor**(*parent*, *tutor=None*)
Bases: PySide2.QtWidgets.QTableView

A Google-like search bar, implemented as a QTableView with a _CustomLineEditDelegate in the first row.

Initializes instance.

Parameters

- **parent** (*QWidget*) – parent widget

- **tutor** (*QWidget, NoneType*) – another widget used for positioning.

data_committed

set_data (*self*, *current*, *items*)
Populates model.

Parameters

- **current** (*str*) –
- **items** (*Sequence(str)*) –

set_base_size (*self*, *size*)

update_geometry (*self*)
Updates geometry.

refit (*self*)

data (*self*)

_handle_delegate_text_edited (*self*, *text*)
Filters model as the first row is being edited.

_proxy_model_filter_accepts_row (*self*, *source_row*, *source_parent*)
Always accept first row.

keyPressEvent (*self*, *event*)
Sets data from current index into first index as the user navigates through the table using the up and down keys.

currentChanged (*self*, *current*, *previous*)

edit_first_index (*self*)
Edits first index if valid and not already being edited.

mouseMoveEvent (*self*, *event*)
Sets the current index to the one hovered by the mouse.

mousePressEvent (*self*, *event*)
Commits data.

class spinetoolbox.widgets.custom_editors.**CheckListEditor** (*parent*, *tutor=None*, *ranked=False*)

Bases: PySide2.QtWidgets.QTableView

A check list editor.

Initialize class.

_make_icon (*self*, *i=None*)

keyPressEvent (*self*, *event*)
Toggles checked state if the user presses space.

toggle_selected (*self*, *index*)
Adds or removes given index from selected items.

Parameters **index** (*QModelIndex*) –

_select_item (*self*, *qitem*, *rank*)

_deselect_item (*self*, *qitem*, *update_ranks=False*)

mouseMoveEvent (*self*, *event*)
Sets the current index to the one under mouse.

mousePressEvent(*self*, *event*)
Toggles checked state of pressed index.

set_data(*self*, *items*, *checked_items*)
Sets data and updates geometry.

Parameters

- **items** (*Sequence(str)*) – All items.
- **checked_items** (*Sequence(str)*) – Initially checked items.

data(*self*)
Returns a comma separated list of checked items.

Returns str

set_base_size(*self*, *size*)

update_geometry(*self*)
Updates geometry.

class spinetoolbox.widgets.custom_editors.**_IconPainterDelegate**
Bases: PySide2.QtWidgets.QItemDelegate

A delegate to highlight decorations in a QListWidget.

paint(*self*, *painter*, *option*, *index*)
Paints selected items using the highlight brush.

class spinetoolbox.widgets.custom_editors.**IconColorEditor**(*parent*)
Bases: PySide2.QtWidgets.QDialog

An editor to let the user select an icon and a color for an object_class.

Init class.

_proxy_model_filter_accepts_row(*self*, *source_row*, *source_parent*)
Overridden method to filter icons according to search terms.

connect_signals(*self*)
Connect signals to slots.

set_data(*self*, *data*)

data(*self*)

spinetoolbox.widgets.custom_menus

Classes for custom context menus and pop-up menus.

author

P. Savolainen (VTT)

date 9.1.2018

Module Contents

Classes

CustomContextMenu	Context menu master class for several context menus.
CategoryProjectItemContextMenu	Context menu for category project items in the QTreeView.
ProjectItemModelContextMenu	Context menu for project item model in the QTreeView.
ProjectItemContextMenu	Context menu for project items in the Project tree widget and in the Design View.
LinkContextMenu	Context menu class for connection links.
OpenProjectDialogComboBoxContextMenu	Context menu master class for several context menus.
CustomPopupMenu	Popup menu master class for several popup menus.
AddSpecificationPopupMenu	Popup menu class for add Tool specification button.
ItemSpecificationMenu	Context menu class for item specifications.
RecentProjectsPopupMenu	Recent projects menu embedded to 'File-Open recent' QAction.
FilterMenuBase	Filter menu.
SimpleFilterMenu	Filter menu.

class spinetoolbox.widgets.custom_menus.**CustomContextMenu**(*parent*, *position*)

Bases: PySide2.QtWidgets.QMenu

Context menu master class for several context menus.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen

add_action(*self*, *text*, *icon=QIcon()*, *enabled=True*)

Adds an action to the context menu.

Parameters

- **text** (*str*) – Text description of the action
- **icon** (*QIcon*) – Icon for menu item
- **enabled** (*bool*) – Is action enabled?

set_action(*self*, *option*)

Sets the action which was clicked.

Parameters **option** (*str*) – string with the text description of the action

get_action(*self*)

Returns the clicked action, a string with a description.

class spinetoolbox.widgets.custom_menus.**CategoryProjectItemContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu for category project items in the QTreeView.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen

class spinetoolbox.widgets.custom_menus.**ProjectItemModelContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu for project item model in the QTreeView.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen

class spinetoolbox.widgets.custom_menus.**ProjectItemContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu for project items in the Project tree widget and in the Design View.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen

class spinetoolbox.widgets.custom_menus.**LinkContextMenu**(*parent*, *position*, *link*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for connection links.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen
- **link** (Link *(QGraphicsPathItem)*) – Link that requested the menu

class spinetoolbox.widgets.custom_menus.**OpenProjectDialogComboBoxContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu master class for several context menus.

Parameters

- **parent** (*QWidget*) – Parent for menu widget
- **position** (*QPoint*) – Position on screen

class spinetoolbox.widgets.custom_menus.**CustomPopupMenu**(*parent*)

Bases: PySide2.QtWidgets.QMenu

Popup menu master class for several popup menus.

Parameters **parent** (*QWidget*) – Parent widget of this pop-up menu

add_action(*self*, *text*, *slot*, *enabled=True*, *tooltip=None*)

Adds an action to the popup menu.

Parameters

- **text** (*str*) – Text description of the action
- **slot** (*method*) – Method to connect to action's triggered signal
- **enabled** (*bool*) – Is action enabled?

- **tooltip** (*str*) – Tool tip for the action

class spinetoolbox.widgets.custom_menus.**AddSpecificationPopupMenu**(*parent*)
Bases: *spinetoolbox.widgets.custom_menus.CustomPopupMenu*

Popup menu class for add Tool specification button.

Parameters **parent** (*QWidget*) – parent widget (ToolboxUI)

class spinetoolbox.widgets.custom_menus.**ItemSpecificationMenu**(*parent*, *index*)
Bases: *spinetoolbox.widgets.custom_menus.CustomPopupMenu*

Context menu class for item specifications.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – the index

class spinetoolbox.widgets.custom_menus.**RecentProjectsPopupMenu**(*parent*)
Bases: *spinetoolbox.widgets.custom_menus.CustomPopupMenu*

Recent projects menu embedded to 'File-Open recent' QAction.

Parameters **parent** (*QWidget*) – Parent widget of this menu (ToolboxUI)

add_recent_projects(*self*)
Reads the previous project names and paths from QSettings. Adds them to the QMenu as QActions.

call_open_project(*self*, *checked*, *p*)
Slot for catching the user selected action from the recent projects menu.

Parameters

- **checked** (*bool*) – Argument sent by triggered signal
- **p** (*str*) – Full path to a project file

class spinetoolbox.widgets.custom_menus.**FilterMenuBase**(*parent*)
Bases: PySide2.QtWidgets.QMenu

Filter menu.

Parameters **parent** (*QWidget*) – a parent widget

connect_signals(*self*)

set_filter_list(*self*, *data*)

add_items_to_filter_list(*self*, *items*)

remove_items_from_filter_list(*self*, *items*)

_clear_filter(*self*)

_check_filter(*self*)

_change_filter(*self*)

emit_filter_changed(*self*, *valid_values*)

wipe_out(*self*)

class spinetoolbox.widgets.custom_menus.**SimpleFilterMenu**(*parent*, *show_empty=True*)

Bases: *spinetoolbox.widgets.custom_menus.FilterMenuBase*

Filter menu.

Parameters **parent** (SpineDBEditor) –

filterChanged

emit_filter_changed(*self*, *valid_values*)

spinetoolbox.widgets.custom_qcombobox

Class for a custom QComboBox.

author

P. Savolainen (VTT)

date 16.10.2020

Module Contents

Classes

CustomQComboBox	A custom QComboBox for showing kernels in Settings->Tools.

class spinetoolbox.widgets.custom_qcombobox.**CustomQComboBox**

Bases: PySide2.QtWidgets.QComboBox

A custom QComboBox for showing kernels in Settings->Tools.

mouseMoveEvent(*self*, *e*)

Catch mouseMoveEvent and accept it because the comboBox popup (QListView) has mouse tracking on as default. This makes sure the comboBox popup appears in correct position and clicking on the combobox repeatedly does not move the Settings window.

spinetoolbox.widgets.custom_qgraphicsscene

Custom QGraphicsScene used in the Design View.

author

P. Savolainen (VTT)

date 13.2.2019

Module Contents

Classes

CustomGraphicsScene	A custom QGraphicsScene. It provides signals to notify about items,
DesignGraphicsScene	A scene for the Design view.

class spinetoolbox.widgets.custom_qgraphicsscene.**CustomGraphicsScene**

Bases: PySide2.QtWidgets.QGraphicsScene

A custom QGraphicsScene. It provides signals to notify about items, and a method to center all items in the scene.

At the moment it's used by DesignGraphicsScene and the GraphViewMixin

item_move_finished
Emitted when an item has finished moving.

item_removed
Emitted when an item has been removed.

center_items(*self*)
Centers toplevel items in the scene.

class spinetoolbox.widgets.custom_qgraphicsscene.**DesignGraphicsScene**(*parent*, *toolbox*)

Bases: *spinetoolbox.widgets.custom_qgraphicsscene.CustomGraphicsScene*

A scene for the Design view.

Mainly, it handles drag and drop events of ProjectItemFactoryModel or ProjectItemSpecFactoryModel sources.

Parameters

- **parent** (*QObject*) – scene's parent object
- **toolbox** (ToolboxUI) – reference to the main window

mouseMoveEvent(*self*, *event*)
Moves link drawer.

mousePressEvent(*self*, *event*)
Puts link drawer to sleep and log message if it looks like the user doesn't know what they're doing.

mouseReleaseEvent(*self*, *event*)
Makes link if drawer is released over a valid connector button.

emit_connection_failed(*self*)

keyPressEvent(*self*, *event*)
Puts link drawer to sleep if user presses ESC.

connect_signals(*self*)
Connect scene signals.

handle_selection_changed(*self*)
Synchronize selection with the project tree.

set_bg_color(*self*, *color*)
Change background color when this is changed in Settings.

Parameters **color** (*QColor*) – Background color

set_bg_choice(*self*, *bg_choice*)
Set background choice when this is changed in Settings.

Parameters **bg** (*str*) – “grid”, “tree”, or “solid”

static _is_project_item_drag(*source*)
Checks whether or not source corresponds to a project item being dragged into the scene.

dragLeaveEvent(*self*, *event*)
Accept event.

dragEnterEvent(*self*, *event*)
Accept event. Then call the super class method only if drag source is not a ProjectItemFactoryModel or ProjectItemSpecFactoryModel.

dragMoveEvent(*self*, *event*)
Accept event. Then call the super class method only if drag source is not a ProjectItemFactoryModel or ProjectItemSpecFactoryModel.

dropEvent(*self*, *event*)
Only accept drops when the source is an instance of ProjectItemFactoryModel or ProjectItemSpecFactoryModel. Capture text from event's mimedata and show the appropriate 'Add Item form.'

event(*self*, *event*)
Accepts GraphicsSceneHelp events without doing anything, to not interfere with our usage of QToolTip.showText in graphics_items.ExclamationIcon.

drawBackground(*self*, *painter*, *rect*)
Reimplemented method to make a custom background.

Parameters

- **painter** (*QPainter*) – Painter that is used to paint background
- **rect** (*QRectF*) – The exposed (viewport) rectangle in scene coordinates

_draw_solid_bg(*self*, *painter*, *rect*)
Draws solid bg.

_draw_grid_bg(*self*, *painter*, *rect*)
Draws grid bg.

_draw_tree_bg(*self*, *painter*, *rect*)
Draws 'tree of life' bg.

spinetoolbox.widgets.custom_qgraphicsviews

Classes for custom QGraphicsViews for the Design and Graph views.

authors

P. Savolainen (VTT), M. Marin (KTH)

date 6.2.2018

Module Contents

Classes

CustomQGraphicsView	Super class for Design and Entity QGraphicsViews.
DesignQGraphicsView	QGraphicsView for the Design View.

class spinetoolbox.widgets.custom_qgraphicsviews.**CustomQGraphicsView**(*parent*)
Bases: PySide2.QtWidgets.QGraphicsView

Super class for Design and Entity QGraphicsViews.

parent
Parent widget

Type QWidget

Init CustomQGraphicsView.

zoom_factor

keyPressEvent(*self*, *event*)
Overridden method. Enable zooming with plus and minus keys (comma resets zoom). Send event downstream to QGraphicsItems if pressed key is not handled here.

Parameters **event** (*QKeyEvent*) – Pressed key

mousePressEvent(*self*, *event*)
Set rubber band selection mode if Control pressed. Enable resetting the zoom factor from the middle mouse button.

mouseReleaseEvent(*self*, *event*)
Reestablish scroll hand drag mode.

_use_smooth_zoom(*self*)

wheelEvent(*self*, *event*)
Zooms in/out.

Parameters **event** (*QWheelEvent*) – Mouse wheel event

resizeEvent(*self*, *event*)
Updates zoom if needed when the view is resized.

Parameters **event** (*QResizeEvent*) – a resize event

setScene(*self*, *scene*)
Sets a new scene to this view.

Parameters **scene** (*ShrinkingScene*) – a new scene

_handle_item_move_finished(*self*, *item*)

_update_zoom_limits(*self*)
Updates the minimum zoom limit and the zoom level with which the view fits all the items in the scene.

_handle_zoom_time_line_advanced(*self*, *pos*)
Performs zoom whenever the smooth zoom time line advances.

_handle_transformation_time_line_finished(*self*)
Cleans up after the smooth transformation time line finishes.

_handle_resize_time_line_finished(*self*)
Cleans up after resizing time line finishes.

zoom_in(*self*)
Perform a zoom in with a fixed scaling.

zoom_out(*self*)
Perform a zoom out with a fixed scaling.

reset_zoom(*self*)
Resets zoom to the default factor.

gentle_zoom(*self*, *factor*, *zoom_focus=None*)
Perform a zoom by a given factor.

Parameters

- **factor** (*float*) – a scaling factor relative to the current scene scaling
- **zoom_focus** (*QPoint*) – focus of the zoom, e.g. mouse pointer position

_zoom(*self*, *factor*)

_get_viewport_scene_rect(*self*)
Returns the viewport rect mapped to the scene.

Returns QRectF

_ensure_item_visible(*self*, *item*)
Resets zoom if item is not visible.

_set_preferred_scene_rect(*self*)
Sets the scene rect to the result of uniting the scene viewport rect and the items bounding rect.

class spinetoolbox.widgets.custom_qgraphicsviews.**DesignQGraphicsView**(*parent*)
Bases: *spinetoolbox.widgets.custom_qgraphicsviews.CustomQGraphicsView*

QGraphicsView for the Design View.

Parameters **parent** (*QWidget*) – Graph View Form's (QMainWindow) central widget (self.centralwidget)

set_ui(*self*, *toolbox*)
Set a new scene into the Design View when app is started.

set_project_item_model(*self*, *model*)
Set project item model.

remove_icon(*self*, *icon*)
Removes icon and all connected links from scene.

links(*self*)
Returns all Links in the scene. Used for saving the project.

add_link(*self*, *src_connector*, *dst_connector*)
Pushes an AddLinkCommand to the toolbox undo stack.

make_link(*self*, *src_connector*, *dst_connector*)
Returns a Link between given connectors.

Parameters

- **src_connector** (ConnectorButton) – Source connector button
- **dst_connector** (ConnectorButton) – Destination connector button

Returns Link

do_add_link(*self*, *src_connector*, *dst_connector*)
Makes a Link between given source and destination connectors and adds it to the project.

Parameters

- **src_connector** (ConnectorButton) – Source connector button
- **dst_connector** (ConnectorButton) – Destination connector button

_add_link(*self*, *link*)
Adds given Link to the project.

Parameters **link** (Link) – the link to add

static _remove_redundant_link(*link*)
Checks if there's a link with the same source and destination as the given one, wipes it out and returns it.

Parameters **link** (Link) – a new link being added to the project.

Returns Link, NoneType

remove_link(*self*, *link*)
Pushes a RemoveLinkCommand to the toolbox undo stack.

do_remove_link(*self*, *link*)
Removes link from the project.

take_link(*self*, *link*)
Remove link, then start drawing another one from the same source connector.

restore_links(*self*, *connections*)
Creates Links from the given connections list.

- List of dicts is accepted, e.g.

Parameters **connections** (*list*) – List of connections.

notify_destination_items(*self*, *src_connector*, *dst_connector*)
Notify destination items that they have been connected to a source item.

_start_animation(*self*, *item_name*, *direction*)
Starts item icon animation when executing forward.

_stop_animation(*self*, *item_name*, *direction*, *_*)
Stops item icon animation when executing forward.

_run_leave_animation(*self*, *item_name*, *direction*, *engine_state*)
Runs the animation that represents execution leaving this item. Blocks until the animation is finished.

_make_execution_leave_animation(*self*, *item_name*)
Returns animation to play when execution leaves this item.

Returns QParallelAnimationGroup

spinetoolbox.widgets.custom_qlineedits

Classes for custom line edits.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 11.10.2018

Module Contents

Classes

PropertyQLineEdit	A custom QLineEdit for Project Item Properties.
CustomQLineEdit	A custom QLineEdit that accepts file drops and displays the path.

class spinetoolbox.widgets.custom_qlineedits.**PropertyQLineEdit**
Bases: PySide2.QtWidgets.QLineEdit

A custom QLineEdit for Project Item Properties.

keyPressEvent(*self*, *e*)
Overridden to catch and pass on the Undo and Redo commands when this line edit has the focus.

Parameters **e** (*QKeyEvent*) – Event

class spinetoolbox.widgets.custom_qlineedits.**CustomQLineEdit**
Bases: PySide2.QtWidgets.QLineEdit

A custom QLineEdit that accepts file drops and displays the path.

parent
Parent for line edit widget

Type QMainWindow

file_dropped

dragEnterEvent(*self*, *event*)
Accept a single file drop from the filesystem.

dragMoveEvent(*self*, *event*)
Accept event.

dropEvent(*self*, *event*)
Emit file_dropped signal with the file for the dropped url.

keyPressEvent(*self*, *e*)
Overridden to catch and pass on the Undo and Redo commands when this line edit has the focus.

Parameters **e** (*QKeyEvent*) – Event

spinetoolbox.widgets.custom_qlistview

Classes for custom QListView.

author

M. Marin (KTH)

date 14.11.2018

Module Contents

Classes

DragListView	Custom QListView class with dragging support.
ProjectItemDragListView	Custom QListView class with dragging support.

class spinetoolbox.widgets.custom_qlistview.**DragListView**(*parent*)
Bases: PySide2.QtWidgets.QListView

Custom QListView class with dragging support.

parent
The parent of this view

Type QWidget

Initialize the view.

mousePressEvent(*self*, *event*)
Register drag start position

mouseMoveEvent(*self*, *event*)
Start dragging action if needed

mouseReleaseEvent(*self*, *event*)
Forget drag start position

class spinetoolbox.widgets.custom_qlistview.**ProjectItemDragListView**(*parent*)
Bases: *spinetoolbox.widgets.custom_qlistview.DragListView*

Custom QListView class with dragging support.

parent
The parent of this view

Type QWidget

Initialize the view.

set_orientation(*self*, *orientation*)

updateGeometries(*self*)
Resize to contents.

spinetoolbox.widgets.custom_qtableview

Custom QTableView classes that support copy-paste and the like.

author

M. Marin (KTH)

date 18.5.2018

Module Contents

Classes

CopyPasteTableView	Custom QTableView class with copy and paste methods.
AutoFilterCopyPasteTableView	Custom QTableView class with autofilter functionality.
IndexedParameterValueTableViewBase	Custom QTableView base class with copy and paste methods for indexed parameter values.
TimeSeriesFixedResolutionTableView	A QTableView for fixed resolution time series table.

Continued on next page

Table 225 – continued from previous page

IndexedValueTableView	A QTableView class with for variable resolution time series and time patterns.
ArrayTableView	Custom QTableView with copy and paste methods for single column tables.

class spinetoolbox.widgets.custom_qtableview.**CopyPasteTableView**

Bases: PySide2.QtWidgets.QTableView

Custom QTableView class with copy and paste methods.

keyPressEvent(*self*, *event*)

Copy and paste to and from clipboard in Excel-like format.

delete_content(*self*)

Delete content from editable indexes in current selection.

copy(*self*)

Copy current selection to clipboard in excel format.

canPaste(*self*)

paste(*self*)

Paste data from clipboard.

static _read_pasted_text(*text*)

Parses a tab separated CSV text table.

Parameters **text** (*str*) – a CSV formatted table

Returns a list of rows

paste_on_selection(*self*)

Paste clipboard data on selection, but not beyond. If data is smaller than selection, repeat data to fit selection.

paste_normal(*self*)

Paste clipboard data, overwriting cells if needed

class spinetoolbox.widgets.custom_qtableview.**AutoFilterCopyPasteTableView**(*parent*)

Bases: *spinetoolbox.widgets.custom_qtableview.CopyPasteTableView*

Custom QTableView class with autofilter functionality.

parent

The parent of this view

Type QWidget

Initializes the view.

Parameters **parent** (*QObject*) –

keyPressEvent(*self*, *event*)

Shows the autofilter menu if the user presses Alt + Down.

Parameters **event** (*QEvent*) –

setModel(*self*, *model*)

Disconnects the sectionPressed signal which seems to be connected by the super method. Otherwise pressing the header just selects the column.

Parameters **model** (*QAbstractItemModel*) –

show_auto_filter_menu (*self*, *logical_index*)
Called when user clicks on a horizontal section header. Shows/hides the auto filter widget.

Parameters **logical_index** (*int*) –

class spinetoolbox.widgets.custom_qtableview.**IndexedParameterValueTableViewBase**
Bases: *spinetoolbox.widgets.custom_qtableview.CopyPasteTableView*

Custom QTableView base class with copy and paste methods for indexed parameter values.

copy (*self*)
Copy current selection to clipboard in CSV format.

static _read_pasted_text (*text*)
Reads CSV formatted table.

paste (*self*)
Pastes data from clipboard to selection.

static _range (*indexes*)
Returns the top left and bottom right corners of selected model indexes.

Parameters **indexes** (*list*) – a list of selected QModelIndex objects

Returns a tuple (top row, bottom row, left column, right column)

_select_pasted (*self*, *indexes*)
Selects the given model indexes.

class spinetoolbox.widgets.custom_qtableview.**TimeSeriesFixedResolutionTableView**
Bases: *spinetoolbox.widgets.custom_qtableview.IndexedParameterValueTableViewBase*

A QTableView for fixed resolution time series table.

paste (*self*)
Pastes data from clipboard.

static _read_pasted_text (*text*)
Parses the given CSV table.

Parsing is locale aware.

Parameters **text** (*str*) – a CSV table containing numbers

Returns A list of floats

_paste_to_values_column (*self*, *values*, *first_row*, *paste_length*)
Pastes data to the Values column.

Parameters

- **values** (*list*) – a list of float values to paste
- **first_row** (*int*) – index of the first row where to paste
- **paste_length** (*int*) – length of the paste selection (can be different from len(values))

Returns A tuple (list(pasted indexes), list(pasted values))

class spinetoolbox.widgets.custom_qtableview.**IndexedValueTableView**
Bases: *spinetoolbox.widgets.custom_qtableview.IndexedParameterValueTableViewBase*

A QTableView class with for variable resolution time series and time patterns.

paste (*self*)
Pastes data from clipboard.

_paste_two_columns(*self*, *data_indexes*, *data_values*, *first_row*, *paste_length*)
Pastes data indexes and values.

Parameters

- **data_indexes** (*list*) – a list of data indexes (time stamps/durations)
- **data_values** (*list*) – a list of data values
- **first_row** (*int*) – first row index
- **paste_length** (*int*) – selection length for pasting

Returns a tuple (modified model indexes, modified model values)

_paste_single_column(*self*, *values*, *first_row*, *first_column*, *paste_length*)
Pastes a single column of data

Parameters

- **values** (*list*) – a list of data to paste (data indexes or values)
- **first_row** (*int*) – first row index
- **paste_length** (*int*) – selection length for pasting

Returns a tuple (modified model indexes, modified model values)

static _read_pasted_text(*text*)
Parses a given CSV table

Parameters **text** (*str*) – a CSV table

Returns a tuple (data indexes, data values)

class spinetoolbox.widgets.custom_qtableview.**ArrayTableView**
Bases: *spinetoolbox.widgets.custom_qtableview.CopyPasteTableView*

Custom QTableView with copy and paste methods for single column tables.

copy(*self*)
Copy current selection to clipboard in CSV format.

spinetoolbox.widgets.custom_qtextbrowser

Class for a custom QTextBrowser for showing the logs and tool output.

author

P. Savolainen (VTT)

date 6.2.2018

Module Contents

Classes

CustomQTextBrowser	Custom QTextBrowser class.

class spinetoolbox.widgets.custom_qtextbrowser.**CustomQTextBrowser**(*parent*)
Bases: PySide2.QtWidgets.QTextBrowser

Custom QTextBrowser class.

Parameters **parent** (*QWidget*) – Parent widget

max_blocks
the upper limit of text blocks that can be appended to the widget.

Type int

append(*self*, *text*)
Appends new text block to the end of the current contents.

If the widget contains more text blocks after the addition than a set limit, blocks will be deleted at the start of the contents.

Parameters **text** (*str*) – text to add

contextMenuEvent(*self*, *event*)
Reimplemented method to add a clear action into the default context menu.

Parameters **event** (*QContextMenuEvent*) – Received event

_open_external_link(*self*, *link*)

spinetoolbox.widgets.custom_qtreeview

Classes for custom QTreeView.

author

M. Marin (KTH)

date 25.4.2018

Module Contents

Classes

CopyTreeView	Custom QTreeView class with copy support.
ReferencesTreeView	Custom QTreeView class for 'References' in Data Connection properties.
DataTreeView	Custom QTreeView class for 'Data' in Data Connection properties.
SourcesTreeView	Custom QTreeView class for 'Sources' in Tool specification editor widget.
CustomTreeView	Custom QTreeView class for Tool specification editor form to enable keyPressEvent.

class spinetoolbox.widgets.custom_qtreeview.**CopyTreeView**(*parent*)
Bases: PySide2.QtWidgets.QTreeView

Custom QTreeView class with copy support.

Initialize the view.

copy(*self*)
Copy current selection to clipboard in excel format.

class spinetoolbox.widgets.custom_qtreeview.**ReferencesTreeView**(*parent*)
Bases: PySide2.QtWidgets.QTreeView

Custom QTreeView class for 'References' in Data Connection properties.

parent
The parent of this view

Type QWidget

Initialize the view.

files_dropped

del_key_pressed

dragEnterEvent(*self*, *event*)
Accept file drops from the filesystem.

dragMoveEvent(*self*, *event*)
Accept event.

dropEvent(*self*, *event*)
Emit files_dropped signal with a list of files for each dropped url.

keyPressEvent(*self*, *event*)
Overridden method to make the view support deleting items with a delete key.

class spinetoolbox.widgets.custom_qtreeview.**DataTreeView**(*parent*)
Bases: PySide2.QtWidgets.QTreeView

Custom QTreeView class for 'Data' in Data Connection properties.

parent
The parent of this view

Type QWidget

Initialize the view.

files_dropped

del_key_pressed

dragEnterEvent(*self*, *event*)
Accept file drops from the filesystem.

dragMoveEvent(*self*, *event*)
Accept event.

dropEvent(*self*, *event*)
Emit files_dropped signal with a list of files for each dropped url.

mousePressEvent(*self*, *event*)
Register drag start position.

mouseMoveEvent(*self*, *event*)
Start dragging action if needed.

mouseReleaseEvent(*self*, *event*)
Forget drag start position

keyPressEvent(*self*, *event*)
Overridden method to make the view support deleting items with a delete key.

class spinetoolbox.widgets.custom_qtreeview.**SourcesTreeView**(*parent*)
Bases: PySide2.QtWidgets.QTreeView

Custom QTreeView class for 'Sources' in Tool specification editor widget.

parent
The parent of this view

Type QWidget

Initialize the view.

files_dropped

del_key_pressed

dragEnterEvent(*self*, *event*)
Accept file and folder drops from the filesystem.

dragMoveEvent(*self*, *event*)
Accept event.

dropEvent(*self*, *event*)
Emit files_dropped signal with a list of files for each dropped url.

keyPressEvent(*self*, *event*)
Overridden method to make the view support deleting items with a delete key.

class spinetoolbox.widgets.custom_qtreeview.**CustomTreeView**(*parent*)
Bases: PySide2.QtWidgets.QTreeView

Custom QTreeView class for Tool specification editor form to enable keyPressEvent.

parent
The parent of this view

Type QWidget

Initialize the view.

del_key_pressed

keyPressEvent(*self*, *event*)
Overridden method to make the view support deleting items with a delete key.

spinetoolbox.widgets.custom_qwidgets

Custom QWidgets for Filtering and Zooming.

author

P. Vennström (VTT)

date 4.12.2018

Module Contents

Classes

FilterWidgetBase	Filter widget class.
SimpleFilterWidget	Filter widget class.
CustomWidgetAction	Class constructor.
TitleWidgetAction	A widget action for adding titled sections to menus.
ZoomWidgetAction	A widget action with plus, minus, and reset buttons.
RotateWidgetAction	A widget action with rotate left and right buttons.
ActionToolbarWidget	Class constructor.

class spinetoolbox.widgets.custom_qwidgets.**FilterWidgetBase**(*parent*)

Bases: PySide2.QtWidgets.QWidget

Filter widget class.

Init class.

Parameters **parent** (*QWidget*) –

okPressed

cancelPressed

connect_signals(*self*)

save_state(*self*)
Saves the state of the FilterCheckboxListModel.

reset_state(*self*)
Sets the state of the FilterCheckboxListModel to saved state.

clear_filter(*self*)
Selects all items in FilterCheckBoxListModel.

has_filter(*self*)
Returns true if any item is filtered in FilterCheckboxListModel false otherwise.

set_filter_list(*self*, *data*)
Sets the list of items to filter.

_apply_filter(*self*)
Apply current filter and save state.

_cancel_filter(*self*)
Cancel current edit of filter and set the state to the stored state.

_filter_list(*self*)
Filter list with current text.

_text_edited(*self*, *new_text*)
Callback for edit text, starts/restarts timer. Start timer after text is edited, restart timer if text is edited before last time out.

class spinetoolbox.widgets.custom_qwidgets.**SimpleFilterWidget**(*parent*, *show_empty=True*)

Bases: *spinetoolbox.widgets.custom_qwidgets.FilterWidgetBase*

Filter widget class.

Init class.

Parameters **parent** (*QWidget*) –

class spinetoolbox.widgets.custom_qwidgets.**CustomWidgetAction**(*parent=None*)
Bases: PySide2.QtWidgets.QWidgetAction

Class constructor.

Parameters **parent** (*QWidget*) – the widget's parent

_handle_hovered(*self*)
Hides other menus that might be shown in the parent widget and repaints it. This is to emulate the behavior of QAction.

class spinetoolbox.widgets.custom_qwidgets.**TitleWidgetAction**(*title*, *parent=None*)
Bases: *spinetoolbox.widgets.custom_qwidgets.CustomWidgetAction*

A widget action for adding titled sections to menus.

Class constructor.

Parameters **parent** (*QWidget*) – the widget's parent

H_MARGIN = 6

V_MARGIN = 2

class spinetoolbox.widgets.custom_qwidgets.**ZoomWidgetAction**(*parent=None*)
Bases: *spinetoolbox.widgets.custom_qwidgets.CustomWidgetAction*

A widget action with plus, minus, and reset buttons. Used to create zoom actions for menus.

Class constructor.

Parameters **parent** (*QWidget*) – the widget's parent

minus_pressed

plus_pressed

reset_pressed

_handle_action_triggered(*self*, *name*)

class spinetoolbox.widgets.custom_qwidgets.**RotateWidgetAction**(*parent=None*)
Bases: *spinetoolbox.widgets.custom_qwidgets.CustomWidgetAction*

A widget action with rotate left and right buttons. Used to create rotate actions for menus.

Class constructor.

Parameters **parent** (*QWidget*) – the widget's parent

clockwise_pressed

anticlockwise_pressed

_handle_action_triggered(*self*, *name*)

class spinetoolbox.widgets.custom_qwidgets.**ActionToolbarWidget**(*text*, *actions*, *parent=None*)
Bases: PySide2.QtWidgets.QWidget

Class constructor.

Parameters **parent** (*QWidget*) – the widget's parent

action_triggered

paintEvent(*self*, *event*)
Overridden method.

spinetoolbox.widgets.datetime_editor

An editor widget for editing datetime database (relationship) parameter values.

author

A. Soininen (VTT)

date 28.6.2019

Module Contents

Classes

DatetimeEditor	An editor widget for DateTime type parameter values.

Functions

_QDateTime_to_datetime(dt)	Converts a QDateTime object to Python's datetime.datetime type.
_datetime_to_QDateTime(dt)	Converts Python's datetime.datetime object to QDateTime.

spinetoolbox.widgets.datetime_editor.**_QDateTime_to_datetime**(*dt*)
Converts a QDateTime object to Python's datetime.datetime type.

spinetoolbox.widgets.datetime_editor.**_datetime_to_QDateTime**(*dt*)
Converts Python's datetime.datetime object to QDateTime.

class spinetoolbox.widgets.datetime_editor.**DatetimeEditor**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

An editor widget for DateTime type parameter values.

parent
a parent widget

Type QWidget

_change_datetime(*self*, *new_datetime*)
Updates the internal DateTime value

set_value(*self*, *value*)
Sets the value to be edited.

value(*self*)
Returns the editor's current value.

spinetoolbox.widgets.duration_editor

An editor widget for editing duration database (relationship) parameter values.

author

A. Soininen (VTT)

date 28.6.2019

Module Contents

Classes

DurationEditor	An editor widget for Duration type parameter values.

class spinetoolbox.widgets.duration_editor.**DurationEditor**(*parent=None*)

Bases: PySide2.QtWidgets.QWidget

An editor widget for Duration type parameter values.

parent
a parent widget

Type QWidget

_change_duration(*self*)
Updates the value being edited.

set_value(*self*, *value*)
Sets the value for editing.

value(*self*)
Returns the current Duration.

spinetoolbox.widgets.indexed_value_table_context_menu

Offers a convenience function for time pattern and time series editor widgets.

author

A. Soininen (VTT)

date 5.7.2019

Module Contents

Functions

handle_table_context_menu(click_pos, table_view, model, parent_widget)	Shows a context menu for parameter_value tables and handles the selection.
_remove_rows(selected_rows, model)	Packs consecutive rows into a single removeRows call.

spinetoolbox.widgets.indexed_value_table_context_menu.**handle_table_context_menu**(*click_pos*, *table_view*, *model*, *parent_widget*)

Shows a context menu for parameter_value tables and handles the selection.

Parameters

- **{QPoint)** (*click_pos*) – position from the context menu event
- **table_view** (*QTableView*) – the table widget
- **model** (TimePatternModel, TimeSeriesModelFixedResolution, TimeSeriesModelVariableResolution) – a model
- **(QWidget** (*parent_widget*) – context menu's parent widget

spinetoolbox.widgets.indexed_value_table_context_menu.**_remove_rows**(*selected_rows*, *model*)

Packs consecutive rows into a single removeRows call.

spinetoolbox.widgets.kernel_editor

Dialog for selecting a kernel or creating a new Julia or Python kernel.

author

P. Savolainen (VTT)

date 7.10.2020

Module Contents

Classes

KernelEditor	Class for a Python and Julia kernel editor.

Functions

find_kernels()	Returns a dictionary mapping kernel names to kernel paths.
find_python_kernels()	Returns a dictionary of Python kernels. Keys are kernel_names, values are kernel paths.
find_julia_kernels()	Returns a dictionary of Julia kernels. Keys are kernel_names, values are kernel paths.
find_unknown_kernels()	Returns a dictionary of kernels that are neither Python nor Julia kernels.
format_log_message(msg_type, message, show_datetime=True)	Formats message for the kernel editor text browser.
format_process_message(msg_type, message)	Formats message for the kernel editor text browser.

class spinetoolbox.widgets.kernel_editor.**KernelEditor**(*parent*, *python*, *julia*, *python_or_julia*, *current_kernel*)

Bases: PySide2.QtWidgets.QDialog

Class for a Python and Julia kernel editor.

Parameters

- **parent** (*QWidget*) – Parent widget (Settings widget)

- **python** (*str*) – Python interpreter, may be empty string
- **julia** (*str*) – Julia executable, may be empty string
- **python_or_julia** (*str*) – Setup KernelEditor according to selected mode
- **current_kernel** (*str*) – Current selected Python or Julia kernel name

setup_dialog_style(*self*)
Sets windows icon and stylesheet. This can be removed when SettingsWidget inherits stylesheet from ToolboxUI.

connect_signals(*self*)
Connect signals to slots.

python_kernel_name_edited(*self*, *txt*)
Updates the display name place holder text and the command QCustomLabel tool tip.

select_julia_clicked(*self*, *checked=False*)
Opens file browser where user can select a Julia executable for the new kernel.

select_julia_project_clicked(*self*, *checked=False*)
Opens file browser where user can select a Julia project path for the new kernel.

select_python_clicked(*self*, *checked=False*)
Opens file browser where user can select the python interpreter for the new kernel.

update_python_cmd_tooltip(*self*)
Updates Python command (CustomQLabel) tooltip according to selections.

update_julia_cmd_tooltip(*self*)
Updates Julia command (CustomQLabel) tooltip according to selections.

set_kernel_selected(*self*, *k_name*)
Finds row index of given kernel name from the model, sets it selected and scrolls the view so that it's visible.

Parameters **k_name** (*str*) – Kernel name to find and select

_check_kernel_is_ok(*self*, *current*, *previous*)
Shows a notification if there are any known problems with selected kernel.

Parameters

- **current** (*QModelIndex*) – Currently selected index
- **previous** (*QModelIndex*) – Previously selected index

find_column(*self*, *label*)
Returns the column number from the kernel model with the given label.

Parameters **label** (*str*) – Header column label

Returns Column number or -1 if label not found

Return type int

make_python_kernel(*self*, *checked=False*)
Makes a new Python kernel. Offers to install ipykernel package if it is missing from the selected Python environment. Overwrites existing kernel with the same name if this is ok by user.

start_kernelspec_install_process(*self*, *prgm*, *k_name*, *d_name*)
Installs kernel specifications for the given Python environment. Runs e.g. this command in QProcess

python -m ipykernel install –user –name python-X.Y –display-name PythonX.Y

Creates new kernel specs into %APPDATA%jupyterkernels. Existing directory will be overwritten.

Note: We cannot use –sys.prefix here because if we have selected to create a kernel for some other python that was used in launching the app, the kernel will be created into a location that is not discoverable by jupyter and hence not by Spine Toolbox. E.g. when sys.executable is C:Python36python.exe, and we have selected that as the python for Spine Toolbox (Settings->Tools->Python interpreter is empty), creating a kernel with –sys-prefix creates kernel specs into C:Python36sharejupyterkernelspython-3.6. This is ok and the kernel spec is discoverable by jupyter and Spine Toolbox.

BUT when sys.executable is C:Python36python.exe, and we have selected another python for Spine Toolbox (Settings->Tools->Python interpreter is C:Python38python.exe), creating a kernel with –sys-prefix creates a kernel into C:Python38sharejupyterkernelspython-3.8-sys-prefix. This is not discoverable by jupyter nor Spine Toolbox. You would need to start the app using C:Python38python.exe to see and use that kernel spec.

Using –user option instead, creates kernel specs that are discoverable by any python that was used in starting Spine Toolbox.

Parameters

- **prgm** (*str*) – Full path to Python interpreter for which the kernel is created
- **k_name** (*str*) – Kernel name
- **d_name** (*str*) – Kernel display name

handle_kernelspec_install_process_finished(*self*, *retval*)
Handles case when the process for installing the kernel has finished.

Parameters **retval** (*int*) – Process return value. 0: success, !0: failure

check_options(*self*, *prgm*, *kernel_name*, *display_name*, *python_or_julia*)
Checks that user options are valid before advancing with kernel making.

Parameters

- **prgm** (*str*) – Full path to Python or Julia program
- **kernel_name** (*str*) – Kernel name
- **display_name** (*str*) – Kernel display name
- **python_or_julia** (*str*) – Either 'python' or 'julia'

Returns True if all user input is valid for making a new kernel, False otherwise

Return type bool

populate_kernel_model(*self*)
Populates the kernel model with kernels found in user's system either with Python or Julia kernels. Unknows, invalid, and unsupported kernels are appended to the end.

static get_kernel_deats(*kernel_path*)
Reads kernel.json from given kernel path and returns the details in a dictionary.

Parameters **kernel_path** (*str*) – Full path to kernel directory

Returns language (str), path to interpreter (str), display name (str), project (str) (NA for Python kernels)

Return type dict

show_kernel_list_context_menu(*self*, *pos*)
Shows the context-menu in the kernel list table view.

_open_kernel_json(*self*, *checked=False*)
Opens kernel.json file using the default application for .json files.

_open_kernel_dir(*self*, *checked=False*)
Opens kernel directory in OS file browser.

_remove_kernel(*self*, *checked=False*)
Removes selected kernel by deleting the kernel directory.

mousePressEvent(*self*, *e*)
Saves mouse position at the start of dragging.

Parameters **e** (*QMouseEvent*) – Mouse event

mouseReleaseEvent(*self*, *e*)
Saves mouse position at the end of dragging.

Parameters **e** (*QMouseEvent*) – Mouse event

mouseMoveEvent(*self*, *e*)
Moves the window when mouse button is pressed and mouse cursor is moved.

Parameters **e** (*QMouseEvent*) – Mouse event

is_package_installed(*self*, *python_path*, *package_name*)
Checks if given package is installed to given Python environment.

Parameters

- **python_path** (*str*) – Full path to selected Python interpreter
- **package_name** (*str*) – Package name

Returns True if installed, False if not

Return type (bool)

start_package_install_process(*self*, *python_path*, *package_name*)
Starts installing the given package using pip.

Parameters

- **python_path** (*str*) – Full path to selected Python interpreter
- **package_name** (*str*) – Package name to install using pip

handle_package_install_process_finished(*self*, *retval*)
Handles installing package finished.

Parameters **retval** (*int*) – Process return value. 0: success, !0: failure

make_julia_kernel(*self*, *checked=False*)
Makes a new Julia kernel. Offers to install IJulia package if it is missing from the selected Julia project. Overwrites existing kernel with the same name if this is ok by user.

is_ijulia_installed(*self*, *program*, *project*)
Checks if IJulia is installed for the given project. Note: Trying command 'using IJulia' does not work since it automatically tries loading it from the LOAD_PATH if not it's not found in the active project.

Returns 0 when process failed to start, 1 when IJulia is installed, 2 when IJulia is not installed.

Return type (int)

start_ijulia_install_process(*self*, *julia*, *project*)
Starts installing IJulia package to given Julia project.

Parameters

- **julia** (*str*) – Full path to selected Julia executable
- **project** (*str*) – Julia project (e.g. dir path or '@.', or '.')

handle_ijulia_install_finished(*self*, *ret*)
Runs when IJulia install process finishes.

Parameters **ret** (*int*) – Process return value. 0: success, !0: failure

start_ijulia_rebuild_process(*self*, *program*, *project*)
Starts rebuilding IJulia.

handle_ijulia_rebuild_finished(*self*, *ret*)
Runs when IJulia rebuild process finishes.

Parameters **ret** (*int*) – Process return value. 0: success, !0: failure

start_ijulia_installkernel_process(*self*, *program*, *project*, *kernel_name*)
Installs the kernel using IJulia.installkernel function. Given kernel_name is actually the new kernel DISPLAY name. IJulia strips the whitespace and uncapitalizes this to make the kernel name automatically. Julia version is concatenated to both names automatically (This cannot be changed).

handle_installkernel_process_finished(*self*, *retval*)
Checks whether or not the IJulia.installkernel process finished successfully.

Parameters **retval** (*int*) – Process return value. 0: success, !0: failure

restore_dialog_dimensions(*self*)
Restore widget location, dimensions, and state from previous session.

add_message(*self*, *msg*)
Append regular message to kernel editor text browser.

Parameters **msg** (*str*) – String written to QTextBrowser

add_success_message(*self*, *msg*)
Append message with green text color to kernel editor text browser.

Parameters **msg** (*str*) – String written to QTextBrowser

add_error_message(*self*, *msg*)
Append message with red color to kernel editor text browser.

Parameters **msg** (*str*) – String written to QTextBrowser

add_warning_message(*self*, *msg*)
Append message with yellow (golden) color to kernel editor text browser.

Parameters **msg** (*str*) – String written to QTextBrowser

add_process_message(*self*, *msg*)
Writes message from stdout to kernel editor text browser.

Parameters **msg** (*str*) – String written to QTextBrowser

add_process_error_message(*self*, *msg*)
Writes message from stderr to kernel editor text browser.

Parameters **msg** (*str*) – String written to QTextBrowser

done(*self*, *r*)
Overridden QDialog method. Sets the selected kernel instance attribute so that it can be read by the SettingsForm after this dialog has been closed.

Parameters **r** (*int*) –

closeEvent(*self*, *event=None*)
Handles dialog closing.

Parameters **event** (*QCloseEvent*) – Close event

spinetoolbox.widgets.kernel_editor.**find_kernels**()
Returns a dictionary mapping kernel names to kernel paths.

spinetoolbox.widgets.kernel_editor.**find_python_kernels**()
Returns a dictionary of Python kernels. Keys are kernel_names, values are kernel paths.

spinetoolbox.widgets.kernel_editor.**find_julia_kernels**()
Returns a dictionary of Julia kernels. Keys are kernel_names, values are kernel paths.

spinetoolbox.widgets.kernel_editor.**find_unknown_kernels**()
Returns a dictionary of kernels that are neither Python nor Julia kernels.

spinetoolbox.widgets.kernel_editor.**format_log_message**(*msg_type*, *message*, *show_datetime=True*)
Formats message for the kernel editor text browser. This is a copy of helpers.format_log_message() but the colors have been edited for a text browser with a white background.

spinetoolbox.widgets.kernel_editor.**format_process_message**(*msg_type*, *message*)
Formats message for the kernel editor text browser. This is a copy of helpers.format_process_message() but the colors have been edited for a text browser with a white background.

spinetoolbox.widgets.map_editor

An editor widget for editing a map type parameter values.

author

A. Soininen (VTT)

date 11.2.2020

Module Contents

Classes

MapEditor	A widget for editing maps.

class spinetoolbox.widgets.map_editor.**MapEditor**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

A widget for editing maps.

parent

Type QWidget

_convert_leaves(*self*, *_*)

_show_table_context_menu(*self*, *pos*)

set_value(*self*, *value*)
Sets the parameter_value to be edited.

value (*self*)
Returns the parameter_value currently being edited.

spinetoolbox.widgets.notification

Contains a notification widget.

author

P. Savolainen (VTT)

date 12.12.2019

Module Contents

Classes

Notification	Custom pop-up notification widget with fade-in and fade-out effect.
LinkNotification	A notification that may have a link.
NotificationStack	

class spinetoolbox.widgets.notification.**Notification**(*parent*, *txt*, *anim_duration=500*, *life_span=None*, *alignment=Qt.AlignCenter*)

Bases: PySide2.QtWidgets.QWidget

Custom pop-up notification widget with fade-in and fade-out effect.

Parameters

- **parent** (*QWidget*) – Parent widget
- **txt** (*str*) – Text to display in notification
- **anim_duration** (*int*) – Duration of the animation in msecs
- **life_span** (*int*) – How long does the notification stays in place in msecs

opacity

show(*self*)

get_opacity(*self*)
opacity getter.

set_opacity(*self*, *op*)
opacity setter.

update_opacity(*self*, *value*)
Updates graphics effect opacity.

start_self_destruction(*self*)
Starts fade-out animation and closing of the notification.

enterEvent(*self*, *e*)

dragEnterEvent(*self*, *e*)

remaining_time(*self*)

class spinetoolbox.widgets.notification.**LinkNotification**(**args*, *open_link=None*, ***kwargs*)

Bases: *spinetoolbox.widgets.notification.Notification*

A notification that may have a link.

Parameters

- **parent** (*QWidget*) – Parent widget
- **txt** (*str*) – Text to display in notification
- **anim_duration** (*int*) – Duration of the animation in msecs
- **life_span** (*int*) – How long does the notification stays in place in msecs

enterEvent(*self*, *e*)
Pauses timer as the mouse hovers the notification.

leaveEvent(*self*, *e*)
Resumes timer after the mouse leaves the notification.

class spinetoolbox.widgets.notification.**NotificationStack**(*parent*, *anim_duration=500*, *life_span=None*)

Bases: PySide2.QtCore.QObject

push_notification(*self*, *notification*)
Pushes a notification to the stack with the given text.

push(*self*, *txt*)

push_link(*self*, *txt*, *open_link=None*)

handle_notification_destroyed(*self*, *notification*, *height*)
Removes from the stack the given notification and move up subsequent ones.

spinetoolbox.widgets.open_project_widget

Contains a class for a widget that represents a 'Open Project Directory' dialog.

author

P. Savolainen (VTT)

date 1.11.2019

Module Contents

Classes

OpenProjectDialog	A dialog that let's user select a project to open either by choosing
CustomQFileSystemModel	Custom file system model.
DirValidator	

class spinetoolbox.widgets.open_project_widget.**OpenProjectDialog**(*toolbox*)
Bases: PySide2.QtWidgets.QDialog

A dialog that let's user select a project to open either by choosing an old .proj file or by choosing a project directory.

Parameters **toolbox** (ToolboxUI) – QMainWindow instance

set_keyboard_shortcuts(*self*)
Creates keyboard shortcuts for the 'Root', 'Home', etc. buttons.

connect_signals(*self*)
Connects signals to slots.

expand_and_resize(*self*, *p*)
Expands, resizes, and scrolls the tree view to the current directory when the file model has finished loading the path. Slot for the file model's directoryLoaded signal. The directoryLoaded signal is emitted only if the directory has not been cached already.

Parameters **p** (*str*) – Directory that has been loaded

combobox_key_press_event(*self*, *e*)
Interrupts Enter and Return key presses when QComboBox is in focus. This is needed to prevent showing the 'Not a valid Spine Toolbox project' Notifier every time Enter is pressed.

Parameters **e** (*QKeyEvent*) – Received key press event.

validator_state_changed(*self*)
Changes the combobox border color according to the current state of the validator.

current_index_changed(*self*, *i*)
Combobox selection changed. This slot is processed when a new item is selected from the drop-down list. This is not processed when new item txt is QValidotor.Intermediate.

Parameters **i** (*int*) – Selected row in combobox

current_changed(*self*, *current*, *previous*)
Processed when the current item in file system tree view has been changed with keyboard or mouse. Updates the text in combobox.

Parameters

- **current** (*QModelIndex*) – Currently selected index
- **previous** (*QModelIndex*) – Previously selected index

set_selected_path(*self*, *index*)
Sets the text in the combobox as the selected path in the file system tree view.

Parameters **index** (*QModelIndex*) – The index which was mouse clicked.

combobox_text_edited(*self*, *text*)
Updates selected path when combobox text is edited. Note: pressing enter in combobox does not trigger this.

selection(*self*)
Returns the selected path from dialog.

go_root(*self*, *checked=False*)
Slot for the 'Root' button. Scrolls the treeview to show and select the user's root directory.

Note: We need to expand and scroll the tree view here after setCurrentIndex just in case the directory has been loaded already.

go_home(*self*, *checked=False*)
Slot for the 'Home' button. Scrolls the treeview to show and select the user's home directory.

go_documents(*self*, *checked=False*)
Slot for the 'Documents' button. Scrolls the treeview to show and select the user's documents directory.

go_desktop(*self*, *checked=False*)
Slot for the 'Desktop' button. Scrolls the treeview to show and select the user's desktop directory.

done(*self*, *r*)
Checks that selected path exists and is a valid Spine Toolbox directory when ok button is clicked or when enter is pressed without the combobox being in focus.

Parameters **r** (*int*) –

static update_recents(*entry*, *qsettings*)
Adds a new entry to QSettings variable that remembers the five most recent project storages.

Parameters

- **entry** (*str*) – Abs. path to a directory that most likely contains other Spine Toolbox Projects as well. First entry is also used as the initial path for File->New Project dialog.
- **qsettings** (*QSettings*) – Toolbox qsettings object

static remove_directory_from_recents(*p*, *qsettings*)
Removes directory from the recent project storages.

Parameters

- **p** (*str*) – Full path to a project directory
- **qsettings** (*QSettings*) – Toolbox qsettings object

show_context_menu(*self*, *pos*)
Shows the context menu for the QCombobox with a 'Clear history' entry.

Parameters **pos** (*QPoint*) – Mouse position

closeEvent(*self*, *event=None*)
Handles dialog closing.

Parameters **event** (*QCloseEvent*) – Close event

class spinetoolbox.widgets.open_project_widget.**CustomQFileSystemModel**
Bases: PySide2.QtWidgets.QFileSystemModel

Custom file system model.

columnCount(*self*, *parent=QModelIndex()*)
Returns one.

class spinetoolbox.widgets.open_project_widget.**DirValidator**(*parent=None*)
Bases: PySide2.QtGui.QValidator

validate(*self*, *txt*, *pos*)
Returns Invalid if input is invalid according to this validator's rules, Intermediate if it is likely that a little more editing will make the input acceptable and Acceptable if the input is valid.

Parameters

- **txt** (*str*) – Text to validate
- **pos** (*int*) – Cursor position

Returns Invalid, Intermediate, or Acceptable

Return type QValidator.State

spinetoolbox.widgets.parameter_value_editor

An editor dialog for editing database (relationship) parameter values.

author

A. Soininen (VTT)

date 28.6.2019

Module Contents

Classes

_Editor	Indexes for the specialized editors corresponding to the selector combo box and editor stack.
ParameterValueEditor	Dialog for editing (relationship) parameter values.

class spinetoolbox.widgets.parameter_value_editor.**_Editor**

Bases: enum.IntEnum

Indexes for the specialized editors corresponding to the selector combo box and editor stack.

Initialize self. See help(type(self)) for accurate signature.

PLAIN_VALUE = 0

MAP = 1

TIME_SERIES_FIXED_RESOLUTION = 2

TIME_SERIES_VARIABLE_RESOLUTION = 3

TIME_PATTERN = 4

ARRAY = 5

DATETIME = 6

DURATION = 7

class spinetoolbox.widgets.parameter_value_editor.**ParameterValueEditor**(*index*, *parent=None*)

Bases: PySide2.QtWidgets.QDialog

Dialog for editing (relationship) parameter values.

The dialog takes an index and shows a specialized editor corresponding to the value type in a stack widget. The user can change the value type by changing the specialized editor using a combo box. When the dialog is closed the value from the currently shown specialized editor is written back to the given index.

Parameters

- **index** (*QModelIndex*) – an index to a parameter_value in parent_model
- **parent** (*QWidget*) – a parent widget

accept (*self*)
Saves the parameter_value shown in the currently selected editor widget to the database manager.

_change_parameter_type (*self*, *selector_index*)
Handles switching between value types.

Does a rude conversion between fixed and variable resolution time series. In other cases, a default 'empty' value is used.

Parameters **selector_index** (*int*) – an index to the selector combo box

_select_editor (*self*, *value*)
Shows the editor widget corresponding to the given value type on the editor stack.

_use_default_editor (*self*, *message=None*)
Opens the default editor widget. Optionally, displays a warning dialog indicating the problem.

Parameters **message** (*str, optional*) –

_use_editor (*self*, *value*, *editor_index*)

_editor_for_index (*self*, *editor_index*)

spinetoolbox.widgets.plain_parameter_value_editor

An editor widget for editing plain number database (relationship) parameter values.

author

A. Soininen (VTT)

date 28.6.2019

Module Contents

Classes

PlainParameterValueEditor	A widget to edit float or boolean type parameter values.

class spinetoolbox.widgets.plain_parameter_value_editor.**PlainParameterValueEditor** (*parent_widg*
Bases: PySide2.QtWidgets.QWidget

A widget to edit float or boolean type parameter values.

parent_widget
a parent widget

Type QWidget

_set_number_or_string_enabled (*self*, *on*)

set_value (*self*, *value*)
Sets the value to be edited in this widget.

value (*self*)
Returns the value currently being edited.

spinetoolbox.widgets.plot_canvas

A Qt widget to use as a matplotlib backend.

author

A. Soininen (VTT)

date 3.6.2019

Module Contents

Classes

PlotCanvas	A widget for plotting with matplotlib.

class spinetoolbox.widgets.plot_canvas.**PlotCanvas**(*parent=None*)
Bases: matplotlib.backends.backend_qt5agg.FigureCanvasQTAgg

A widget for plotting with matplotlib.

Parameters **parent** (*QWidget*) – a parent widget

axes
figure's axes

Type matplotlib.axes.Axes

spinetoolbox.widgets.plot_widget

A Qt widget showing a toolbar and a matplotlib plotting canvas.

author

A. Soininen (VTT)

date 27.6.2019

Module Contents

Classes

PlotWidget	A widget that contains a toolbar and a plotting canvas.

Functions

_prepare_plot_in_window_menu(menu)	Fills a given menu with available plot window names.

class spinetoolbox.widgets.plot_widget.**PlotWidget**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

A widget that contains a toolbar and a plotting canvas.

canvas
the plotting canvas

Type *PlotCanvas*

plot_type
type of currently plotted data or None

Type type

plot_windows
A global list of plot windows.

closeEvent(*self*, *event*)
Removes the window from plot_windows and closes.

infer_plot_type(*self*, *values*)
Decides suitable plot_type according to a list of values.

use_as_window(*self*, *parent_window*, *document_name*)
Prepares the widget to be used as a window and adds it to plot_windows list.

Parameters

- **parent_window** (*QWidget*) – a parent window
- **document_name** (*str*) – a string to add to the window title

static _unique_window_name(*document_name*)
Returns an unique identifier for a new plot window.

spinetoolbox.widgets.plot_widget.**_prepare_plot_in_window_menu**(*menu*)
Fills a given menu with available plot window names.

spinetoolbox.widgets.project_form_widget

Widget shown to user when a new project is created.

authors

P. Savolainen (VTT)

date 10.1.2018

Module Contents

Classes

NewProjectForm	Class for a new project widget.

class spinetoolbox.widgets.project_form_widget.**NewProjectForm**(*toolbox*)
Bases: PySide2.QtWidgets.QWidget

Class for a new project widget.

Parameters **toolbox** (ToolboxUI) – Parent widget.

connect_signals(*self*)
Connect signals to slots.

select_project_dir(*self*, *checked=False*)
Opens a file browser, where user can select a directory for the new project.

ok_clicked(*self*)
Check that project name is valid and create project.

call_create_project(*self*)
Call ToolboxUI method create_project().

keyPressEvent(*self*, *e*)
Close project form when escape key is pressed.

Parameters **e** (*QKeyEvent*) – Received key press event.

closeEvent(*self*, *event=None*)
Handle close window.

Parameters **event** (*QEvent*) – Closing event if 'X' is clicked.

spinetoolbox.widgets.report_plotting_failure

Functions to report failures in plotting to the user.

author

A. Soininen (VTT)

date 10.7.2019

Module Contents

Functions

report_plotting_failure(error, parent_widget)	Reports a PlottingError exception to the user.

spinetoolbox.widgets.report_plotting_failure.**report_plotting_failure**(*error*, *parent_widget*)

Reports a PlottingError exception to the user.

spinetoolbox.widgets.settings_widget

Widget for controlling user settings.

author

P. Savolainen (VTT)

date 17.1.2018

Module Contents

Classes

SettingsWidgetBase	Initialize class.
SpineDBEditorSettingsMixin	
SpineDBEditorSettingsWidget	A widget to change user's preferred settings, but only for the Spine db editor.
SettingsWidget	A widget to change user's preferred settings.

class spinetoolbox.widgets.settings_widget.**SettingsWidgetBase**(*qsettings*)

Bases: PySide2.QtWidgets.QWidget

Initialize class.

connect_signals(*self*)
Connect signals.

keyPressEvent(*self*, *e*)
Close settings form when escape key is pressed.

Parameters **e** (*QKeyEvent*) – Received key press event.

mousePressEvent(*self*, *e*)
Save mouse position at the start of dragging.

Parameters **e** (*QMouseEvent*) – Mouse event

mouseReleaseEvent(*self*, *e*)
Save mouse position at the end of dragging.

Parameters **e** (*QMouseEvent*) – Mouse event

mouseMoveEvent(*self*, *e*)
Moves the window when mouse button is pressed and mouse cursor is moved.

Parameters **e** (*QMouseEvent*) – Mouse event

update_ui(*self*)
Updates UI to reflect current settings. Called when the user choses to cancel their changes. Undoes all temporary UI changes that resulted from the user playing with certain settings.

save_settings(*self*)
Gets selections and saves them to persistent memory.

update_ui_and_close(*self*, *checked=False*)
Updates UI to reflect current settings and close.

save_and_close(*self*, *checked=False*)
Saves settings and close.

class spinetoolbox.widgets.settings_widget.**SpineDBEditorSettingsMixin**

connect_signals(*self*)
Connect signals.

set_show_cascading_relationships(*self*, *checked=False*)

read_settings(*self*)
Read saved settings from app QSettings instance and update UI to display them.

save_settings(*self*)
Get selections and save them to persistent memory.

update_ui(*self*)

class spinetoolbox.widgets.settings_widget.**SpineDBEditorSettingsWidget**(*db_mngr*)
Bases: *spinetoolbox.widgets.settings_widget.SpineDBEditorSettingsMixin*, *spinetoolbox.widgets.settings_widget.SettingsWidgetBase*

A widget to change user's preferred settings, but only for the Spine db editor.

Initialize class.

show(*self*)

class spinetoolbox.widgets.settings_widget.**SettingsWidget**(*toolbox*)
Bases: *spinetoolbox.widgets.settings_widget.SpineDBEditorSettingsMixin*, *spinetoolbox.widgets.settings_widget.SettingsWidgetBase*

A widget to change user's preferred settings.

Parameters **toolbox** (ToolboxUI) – Parent widget.

connect_signals(*self*)
Connect signals.

browse_gams_path(*self*, *checked=False*)
Open file browser where user can select a GAMS program.

browse_julia_button_clicked(*self*, *checked=False*)
Calls static method that shows a file browser for selecting the Julia path.

browse_julia_project_button_clicked(*self*, *checked=False*)
Calls static method that shows a file browser for selecting a Julia project.

browse_python_button_clicked(*self*, *checked=False*)
Calls static method that shows a file browser for selecting Python interpreter.

show_python_kernel_editor(*self*, *checked=False*)
Opens kernel editor, where user can make a kernel for the Python Console.

python_kernel_editor_closed(*self*, *ret_code*)
Catches the selected Python kernel name when the editor is closed.

show_julia_kernel_editor(*self*, *checked=False*)
Opens kernel editor, where user can make a kernel the Julia Console.

julia_kernel_editor_closed(*self*, *ret_code*)
Catches the selected Julia kernel name when the editor is closed.

browse_work_path(*self*, *checked=False*)
Open file browser where user can select the path to wanted work directory.

show_color_dialog(*self*, *checked=False*)
Let user pick the bg color.

Parameters **checked** (*boolean*) – Value emitted with clicked signal

update_bg_color(*self*)
Set tool button icon as the selected color and update Design View scene background color.

update_scene_bg(*self*, *checked=False*)
Draw background on scene depending on radiobutton states.

Parameters **checked** (*boolean*) – Toggle state

update_links_geometry(*self*, *checked=False*)

toggle_julia_execution_mode(*self*, *checked=False*)
Toggles between console and non-console Julia execution modes depending on radiobutton states.

Parameters **checked** (*boolean*) – Toggle state

toggle_python_execution_mode(*self*, *checked=False*)
Toggles between console and non-console Python execution modes depending on radiobutton states.

Parameters **checked** (*boolean*) – Toggle state

read_settings(*self*)
Read saved settings from app QSettings instance and update UI to display them.

read_project_settings(*self*)
Get project name and description and update widgets accordingly.

save_settings(*self*)
Get selections and save them to persistent memory. Note: On Linux, True and False are saved as boolean values into QSettings. On Windows, booleans and integers are saved as strings. To make it consistent, we should use strings.

update_project_settings(*self*)
Update project name and description if these have been changed.

check_if_work_dir_changed(*self*, *new_work_dir*)
Checks if work directory was changed.

Parameters **new_work_dir** (*str*) – Possibly a new work directory

update_ui(*self*)
Updates UI to reflect current settings. Called when the user choses to cancel their changes. Undoes all temporary UI changes that resulted from the user playing with certain settings.

spinetoolbox.widgets.spine_console_widget

Class for a custom RichJupyterWidget that can run Tool instances.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 22.10.2019

Module Contents

Classes

SpineConsoleWidget	Base class for all embedded console widgets that can run tool instances.

class spinetoolbox.widgets.spine_console_widget.**SpineConsoleWidget**(*toolbox*, *name*)

Bases: qtconsole.rich_jupyter_widget.RichJupyterWidget

Base class for all embedded console widgets that can run tool instances.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance

- **name** (*str*) – Console name, e.g. 'Python Console'

ready_to_execute

execution_failed

name(*self*)
Returns console name.

start_menu_action(*self*, *checked=False*)
Starts chosen Python/Julia kernel if available and not already running. Context menu start action handler.

restart_menu_action(*self*, *checked=False*)
Restarts chosen Python/Julia kernel. Starts a new kernel if it is not running or if chosen kernel has been changed in Settings. Context menu restart action handler.

wake_up(*self*, *k_name=None*)
Wakes up the console in preparation for execution. Either ready_to_execute or execution_failed signal must be emitted as a consequence of calling this method.

start_kernel(*self*, *k_name*, *k_path*)
Starts a kernel manager and kernel client and attaches the client to Julia or Python Console.

Parameters

- **k_name** (*str*) – Kernel name
- **k_path** (*str*) – Directory where the the kernel specs are located

shutdown_kernel(*self*)
Shut down Julia/Python kernel.

dragEnterEvent(*self*, *e*)
Don't accept project item drops.

_handle_execute_reply(*self*, *msg*)

_handle_status(*self*, *msg*)
Handles status message.

_handle_error(*self*, *msg*)
Handles error messages.

enterEvent(*self*, *event*)
Sets busy cursor during console (re)starts.

_is_complete(*self*, *source*, *interactive*)
See base class.

_context_menu_make(*self*, *pos*)
Reimplemented to add actions to console context-menus.

copy_input(*self*)
Copies only input.

_setup_client(*self*)
Sets up client.

connect_to_kernel(*self*, *kernel_name*, *connection_file*)
Connects to an existing kernel. Used when Spine Engine is managing the kernel for project execution.

Parameters **connection_file** (*str*) – Path to the connection file of the kernel

interrupt(*self*)
[TODO: Remove?] Sends interrupt signal to kernel.

spinetoolbox.widgets.spine_datapackage_widget

Widget shown to user when opening a 'datapackage.json' file in Data Connection item.

author

M. Marin (KTH)

date 7.7.2018

Module Contents

Classes

SpineDatapackageWidget	A widget to edit CSV files in a Data Connection and create a tabular datapackage.
CustomPackage	Custom datapackage class.

class spinetoolbox.widgets.spine_datapackage_widget.**SpineDatapackageWidget**(*data_connection*)

Bases: PySide2.QtWidgets.QMainWindow

A widget to edit CSV files in a Data Connection and create a tabular datapackage.

Initialize class.

Parameters **data_connection** (DataConnection) – Data Connection associated to this widget

msg

msg_error

undo_stack

add_menu_actions(*self*)
Add extra menu actions.

connect_signals(*self*)
Connect signals to slots.

update_window_modified(*self*, *_clean=None*)
Updates window modified status and save actions depending on the state of the undo stack.

is_resource_dirty(*self*, *resource_index*)

get_undo_stack(*self*, *resource_index*)

showEvent(*self*, *e*)
Called when the form shows. Init datapackage (either from existing datapackage.json or by inferring a new one from sources) and update ui.

load_datapackage(*self*)

_handle_source_dir_changed(*self*, *_path*)

_handle_source_file_changed(*self*, *path*)

append_save_resource_actions(*self*)

_handle_menu_edit_about_to_show(*self*)
Adjusts copy and paste actions depending on which widget has the focus.

add_message(*self*, *msg*)
Prepend regular message to status bar.

Parameters **msg** (*str*) – String to show in QStatusBar

add_error_message(*self*, *msg*)
Show error message.

Parameters **msg** (*str*) – String to show

save_all(*self*, *checked=False*)

_save_datapackage(*self*, *datapackage_path*)

save_resource(*self*, *resource_index*)

_save_resource(*self*, *resource_index*, *filepath*)

get_permission(*self*, **filepaths*)

copy(*self*, *checked=False*)
Copies data to clipboard.

paste(*self*, *checked=False*)
Pastes data from clipboard.

_handle_current_resource_changed(*self*, *current*, *_previous*)
Resets resource data and schema models whenever a new resource is selected.

refresh_models(*self*, *current=None*)

_handle_fields_data_changed(*self*, *top_left*, *bottom_right*, *roles*)

show_foreign_keys_context_menu(*self*, *pos*)

_remove_foreign_key(*self*, *checked=False*)

restore_ui(*self*)
Restore UI state from previous session.

closeEvent(*self*, *event=None*)
Handle close event.

Parameters **event** (*QEvent*) – Closing event if 'X' is clicked.

class spinetoolbox.widgets.spine_datapackage_widget.**CustomPackage**(**args*, ***kwargs*)

Bases: datapackage.Package

Custom datapackage class.

sources

set_resource_data(*self*, *resource_index*, *row*, *column*, *value*)

resource_data(*self*, *resource_index*)

add_resource(*self*, *descriptor*)

difference_infer(*self*, *path*)
Infers only what's *new* in the given path.

Parameters **path** (*str*) –

check_resource_name(*self*, *new_name*)

rename_resource(*self*, *index*, *new*)

valid_field_names(*self*, *resource_index*, *new_names*)

rename_fields (*self*, *resource_index*, *field_indexes*, *old_names*, *new_names*)
Renames fields.

append_to_primary_key (*self*, *resource_index*, *field_index*)
Append field to resources's primary key.

remove_from_primary_key (*self*, *resource_index*, *field_index*)
Remove field from resources's primary key.

check_foreign_key (*self*, *resource_index*, *foreign_key*)
Check foreign key.

append_foreign_key (*self*, *resource_index*, *foreign_key*)

insert_foreign_key (*self*, *resource_index*, *fk_index*, *foreign_key*)

update_foreign_key (*self*, *resource_index*, *fk_index*, *foreign_key*)

remove_foreign_key (*self*, *resource_index*, *fk_index*)

update_descriptor (*self*, *descriptor_filepath*)
Updates this package's schema from other package's.

spinetoolbox.widgets.state_machine_widget

Contains the StateMachineWidget class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

Classes

StateMachineWidget	A widget with a state machine.

class spinetoolbox.widgets.state_machine_widget.**StateMachineWidget** (*window_title*, *parent*)

Bases: PySide2.QtWidgets.QDockWidget

A widget with a state machine.

Initializes class.

Parameters

- **window_title** (*str*) –
- **parent** (*QMainWindow*) –

current_state

is_running (*self*)

show (*self*)

_make_state (*self*, *name*)

_make_welcome(*self*)

set_up_machine(*self*)

get_current_state(*self*)

set_current_state(*self*, *state*)

spinetoolbox.widgets.time_pattern_editor

An editor widget for editing a time pattern type (relationship) parameter values.

author

A. Soininen (VTT)

date 28.6.2019

Module Contents

Classes

TimePatternEditor	A widget for editing time patterns.

class spinetoolbox.widgets.time_pattern_editor.**TimePatternEditor**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

A widget for editing time patterns.

parent

Type QWidget

_show_table_context_menu(*self*, *pos*)

set_value(*self*, *value*)
Sets the parameter_value to be edited.

value(*self*)
Returns the parameter_value currently being edited.

spinetoolbox.widgets.time_series_fixed_resolution_editor

Contains logic for the fixed step time series editor widget.

author

A. Soininen (VTT)

date 14.6.2019

Module Contents

Classes

TimeSeriesFixedResolutionEditor	A widget for editing time series data with a fixed time step.

Functions

_resolution_to_text(resolution)	Converts a list of durations into a string of comma-separated durations.
_text_to_resolution(text)	Converts a comma-separated string of durations into a resolution array.

spinetoolbox.widgets.time_series_fixed_resolution_editor.**_resolution_to_text**(*resolution*)
Converts a list of durations into a string of comma-separated durations.

spinetoolbox.widgets.time_series_fixed_resolution_editor.**_text_to_resolution**(*text*)
Converts a comma-separated string of durations into a resolution array.

class spinetoolbox.widgets.time_series_fixed_resolution_editor.**TimeSeriesFixedResolutionEdi**
Bases: PySide2.QtWidgets.QWidget

A widget for editing time series data with a fixed time step.

parent
a parent widget

Type QWidget

_resolution_changed(*self*)
Updates the models after resolution change.

_show_table_context_menu(*self*, *pos*)
Shows the table's context menu.

_select_date(*self*, *selected_date*)

set_value(*self*, *value*)
Sets the parameter_value for editing in this widget.

_show_calendar(*self*)

_start_time_changed(*self*)
Updates the model due to start time change.

_update_plot(*self*, *topLeft=None*, *bottomRight=None*, *roles=None*)
Updated the plot.

value(*self*)
Returns the parameter_value currently being edited.

spinetoolbox.widgets.time_series_variable_resolution_editor

Contains logic for the variable resolution time series editor widget.

author

A. Soininen (VTT)

date 31.5.2019

Module Contents

Classes

TimeSeriesVariableResolutionEditor	A widget for editing variable resolution time series data.

class spinetoolbox.widgets.time_series_variable_resolution_editor.**TimeSeriesVariableResolut**
Bases: PySide2.QtWidgets.QWidget

A widget for editing variable resolution time series data.

parent
a parent widget

Type QWidget

_show_table_context_menu(*self*, *pos*)
Shows the table's context menu.

set_value(*self*, *value*)
Sets the time series being edited.

_update_plot(*self*, *topLeft=None*, *bottomRight=None*, *roles=None*)
Updates the plot widget.

value(*self*)
Return the time series currently being edited.

spinetoolbox.widgets.toolbars

Functions to make and handle QToolBars.

author

P. Savolainen (VTT)

date 19.1.2018

Module Contents

Classes

MainToolBar	A toolbar to add items using drag and drop actions.

class spinetoolbox.widgets.toolbars.**MainToolBar**(*parent*)
Bases: PySide2.QtWidgets.QToolBar

A toolbar to add items using drag and drop actions.

Parameters **parent** (ToolboxUI) – QMainWindow instance

setup(*self*)

add_project_item_list_view(*self*)

add_project_item_spec_list_view(*self*)

add_execute_buttons(*self*)

add_remove_all_button(*self*)

remove_all(*self*, *checked=False*)
Slot for handling the remove all tool button clicked signal. Calls ToolboxUI remove_all_items() method.

execute_project(*self*, *checked=False*)
Slot for handling the Execute project tool button clicked signal.

execute_selected(*self*, *checked=False*)
Slot for handling the Execute selected tool button clicked signal.

stop_execution(*self*, *checked=False*)
Slot for handling the Stop execution tool button clicked signal.

17.1.2 Submodules

spinetoolbox.__main__

Spine Toolbox application main file.

author

P. Savolainen (VTT)

date 14.12.2017

Module Contents

spinetoolbox.__main__.**return_code**

spinetoolbox.category

This module defines the project item categories available in the Toolbox.

author A.Soininen (VTT)

date 6.5.2020

Module Contents

spinetoolbox.category.**CATEGORIES = ['Data Stores', 'Data Connections', 'Tools', 'Views', 'I**

spinetoolbox.category.**CATEGORY_DESCRIPTIONS**

spinetoolbox.config

Application constants and style sheets

author

P. Savolainen (VTT)

date 2.1.2018

Module Contents

Functions

_executable(name)	Appends a .exe extension to *name* on Windows platform.

spinetoolbox.config.**REQUIRED_SPINE_ENGINE_VERSION = 0.5.0**

spinetoolbox.config.**REQUIRED_SPINEDB_API_VERSION = 0.7.15**

spinetoolbox.config.**LATEST_PROJECT_VERSION = 2**

spinetoolbox.config.**INVALID_CHARS = ['<', '>', ':', '"', '/', '\\', '|', '?', '*', '.']**

spinetoolbox.config.**INVALID_FILENAME_CHARS = ['<', '>', ':', '"', '/', '\\', '|', '?', '***

spinetoolbox.config.**_frozen**

spinetoolbox.config.**_path_to_executable**

spinetoolbox.config.**APPLICATION_PATH**

spinetoolbox.config.**_program_root**

spinetoolbox.config.**DEFAULT_WORK_DIR**

spinetoolbox.config.**DOCUMENTATION_PATH**

spinetoolbox.config.**PLUGINS_PATH**

spinetoolbox.config.**ONLINE_DOCUMENTATION_URL = https://spine-toolbox.readthedocs.io/en/late**

spinetoolbox.config.**TOOL_OUTPUT_DIR = output**

spinetoolbox.config.**GIMLET_WORK_DIR_NAME = work**

spinetoolbox.config.**_on_windows**

spinetoolbox.config.**_executable**(*name*)

Appends a .exe extension to *name* on Windows platform.

spinetoolbox.config.**GAMS_EXECUTABLE**

spinetoolbox.config.**GAMSIDE_EXECUTABLE**

spinetoolbox.config.**JULIA_EXECUTABLE**

spinetoolbox.config.**PYTHON_EXECUTABLE**

spinetoolbox.config.**JUPYTER_KERNEL_TIME_TO_DEAD = 8.0**

spinetoolbox.config.**PROJECT_FILENAME = project.json**

spinetoolbox.config.**STATUSBAR_SS = QStatusBar{background-color: #EBEBE0;border-width: 1px**

spinetoolbox.config.**SETTINGS_SS = #SettingsForm{background-color: ghostwhite;}QLabel{color**

spinetoolbox.config.**ICON_TOOLBAR_SS = QToolBar{spacing: 6px; background: qlineargradient**

spinetoolbox.config.**PARAMETER_TAG_TOOLBAR_SS**

spinetoolbox.config.**TEXTBROWSER_SS = QTextBrowser {background-color: #19232D; border: 1px**

spinetoolbox.config.**MAINWINDOW_SS = QMainWindow::separator{width: 3px; background-color:**

spinetoolbox.config.**TREEVIEW_HEADER_SS = QHeaderView::section{background-color: #ecd8c6;**

spinetoolbox.config.**PIVOT_TABLE_HEADER_COLOR = #efefef**

spinetoolbox.dag_handler

Contains classes for handling DAGs.

author

P. Savolainen (VTT)

date 8.4.2019

Module Contents

Classes

DirectedGraphHandler	Class for manipulating graphs according to user's actions.

class spinetoolbox.dag_handler.**DirectedGraphHandler**
Bases: PySide2.QtCore.QObject

Class for manipulating graphs according to user's actions.

dag_simulation_requested

dags(*self*)
Returns a list of graphs (DiGraph) in the project.

add_dag(*self*, *dag*, *request_simulation=True*)
Add graph to list.

Parameters

- **dag** (*DiGraph*) – Graph to add
- **request_simulation** (*bool*) – if True, emits dag_simulation_requested

remove_dag(*self*, *dag*)
Remove graph from instance variable list.

Parameters **dag** (*DiGraph*) – Graph to remove

add_dag_node(*self*, *node_name*)
Create directed graph with one node and add it to list.

Parameters **node_name** (*str*) – Project item name to add as a node

add_graph_edge(*self*, *src_node*, *dst_node*)
Adds an edge between the src and dst nodes. If nodes are in different graphs, the reference to union graph is saved and the references to the original graphs are removed. If src and dst nodes are already in the same graph, the edge is added to the graph. If src and dst are the same node, a self-loop (feedback) edge is added.

Parameters

- **src_node** (*str*) – Source project item node name
- **dst_node** (*str*) – Destination project item node name

remove_graph_edge(*self*, *src_node*, *dst_node*)
Removes edge from a directed graph.

Parameters

- **src_node** (*str*) – Source project item node name
- **dst_node** (*str*) – Destination project item node name

remove_node_from_graph(*self*, *node_name*)
Removes node from a graph that contains it. Called when project item is removed from project.

Parameters **node_name** (*str*) – Project item name

rename_node(*self*, *old_name*, *new_name*)
Handles renaming the node and edges in a graph when a project item is renamed.

Parameters

- **old_name** (*str*) – Old project item name
- **new_name** (*str*) – New project item name

Returns True if successful, False if renaming failed

Return type bool

dag_with_node(*self*, *node_name*)
Returns directed graph that contains given node.

Parameters **node_name** (*str*) – Node to look for

Returns Directed graph that contains node or None if not found.

Return type (DiGraph)

dag_with_edge(*self*, *src_node*, *dst_node*)
Returns directed graph that contains given edge.

Parameters

- **src_node** (*str*) – Source node name
- **dst_node** (*str*) – Destination node name

Returns Directed graph that contains edge or None if not found.

Return type (DiGraph)

static node_successors(*g*)
Returns a dict mapping nodes in the given graph to a list of its direct successors. The nodes are in topological sort order. Topological sort in the words of networkx: "a nonunique permutation of the nodes, such that an edge from u to v implies that u appears before v in the topological sort order."

Parameters **g** (*DiGraph*) – Directed graph to process

Returns key is the node name, value is list of successor names Empty dict if given graph is not a DAG.

Return type dict

successors_til_node(*self*, *g*, *node*)
Like node_successors but only until the given node, and ignoring all nodes that are not its ancestors.

node_is_isolated(*self*, *node*, *allow_self_loop=False*)
Checks if the project item with the given name has any connections.

Parameters

- **node** (*str*) – Project item name
- **allow_self_loop** (*bool*) – If default (False), Self-loops are considered as an in-neighbor or an out-neighbor so the method returns False. If True, single node with a self-loop is considered isolated.

Returns

True if project item has no in-neighbors nor out-neighbors, False if it does. Single node with a self-loop is NOT isolated (returns False).

Return type bool

static source_nodes(*g*)

Returns a list of source nodes in given graph. A source node has no incoming edges. This is determined by calculating the in-degree of each node in the graph. If nodes in-degree == 0, it is a source node

Parameters **g** (*DiGraph*) – Graph to examine

Returns List of source node names or an empty list is there are none.

Return type list

static edges_causing_loops(*g*)

Returns a list of edges whose removal from g results in it becoming acyclic.

static export_to_graphml(*g*, *path*)

Export given graph to a path in GraphML format.

Parameters

- **g** (*DiGraph*) – Graph to export
- **path** (*str*) – Full output path for GraphML file

Returns Operation success status

Return type bool

spinetoolbox.data_package_commands

Classes for models dealing with Data Packages.

authors

M. Marin (KTH)

date 10.7.2020

Module Contents

Classes

UpdateResourceNameCommand	Command to update a resource's name.
UpdateResourceDataCommand	Command to update resource data.
UpdateFieldNamesCommand	Command to update a resource field's name.
UpdatePrimaryKeysCommand	Command to update a resource's primary key.
AppendForeignKeyCommandCommand	Command to append a foreign key to a resource.
RemoveForeignKeyCommandCommand	Command to remove a foreign key from a resource.

Continued on next page

Table 256 – continued from previous page

UpdateForeignKeyCommandCommand	Command to update a foreign key in a resource.

class spinetoolbox.data_package_commands.**UpdateResourceNameCommand**(*model*, *resource_index*, *old_name*, *new_name*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to update a resource's name.

Args:

redo(*self*)

undo(*self*)

class spinetoolbox.data_package_commands.**UpdateResourceDataCommand**(*model*, *resource_index*, *rows*, *columns*, *old_values*, *new_values*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to update resource data.

Args:

redo(*self*)

undo(*self*)

class spinetoolbox.data_package_commands.**UpdateFieldNamesCommand**(*model*, *resource_index*, *field_indexes*, *old_names*, *new_names*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to update a resource field's name.

Args:

redo(*self*)

undo(*self*)

class spinetoolbox.data_package_commands.**UpdatePrimaryKeysCommand**(*model*, *resource_index*, *field_indexes*, *statuses*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to update a resource's primary key.

Args:

redo(*self*)

undo(*self*)

class spinetoolbox.data_package_commands.**AppendForeignKeyCommandCommand**(*model*, *resource_index*, *foreign_key*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to append a foreign key to a resource.

Args:

redo(*self*)

undo(*self*)

class spinetoolbox.data_package_commands.**RemoveForeignKeyCommandCommand**(*model*, *resource_index*, *fk_index*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to remove a foreign key from a resource.

Args:

redo(*self*)

undo(*self*)

class spinetoolbox.data_package_commands.**UpdateForeignKeyCommandCommand**(*model*, *resource_index*, *fk_index*, *foreign_key*)

Bases: PySide2.QtWidgets.QUndoCommand

Command to update a foreign key in a resource.

Args:

redo(*self*)

undo(*self*)

spinetoolbox.executable_item_base

Contains ExecutableItem, a project item's counterpart in execution as well as support utilities.

authors

A. Soininen (VTT)

date 30.3.2020

Module Contents

Classes

ExecutableItemBase	The part of a project item that is executed by the Spine Engine.

class spinetoolbox.executable_item_base.**ExecutableItemBase**(*name*, *logger*)

The part of a project item that is executed by the Spine Engine.

Parameters

- **name** (*str*) – item's name
- **logger** (LoggerInterface) – a logger

name

Project item's name.

execute(*self*, *resources*, *direction*)

Executes this item in the given direction using the given resources and returns a boolean indicating the outcome.

Subclasses need to implement _execute_forward and _execute_backward to do the appropriate work in each direction.

Parameters

- **resources** (*list*) – a list of ProjectItemResources available for execution
- **direction** (*ExecutionDirection*) – direction of execution

Returns True if execution succeeded, False otherwise

Return type bool

static item_type()

Returns the item's type identifier string.

output_resources(*self*, *direction*)

Returns output resources in the given direction.

Subclasses need to implement _output_resources_backward and/or _output_resources_forward if they want to provide resources in any direction.

Parameters **direction** (*ExecutionDirection*) – Direction where output resources are passed

Returns a list of ProjectItemResources

stop_execution(*self*)

Stops executing this item.

_execute_forward(*self*, *resources*)

Executes this item in the forward direction.

The default implementation just returns True.

Parameters **resources** (*list*) – a list of ProjectItemResources available for execution

Returns True if execution succeeded, False otherwise

Return type bool

_execute_backward(*self*, *resources*)

Executes this item in the backward direction.

The default implementation just returns True.

Parameters **resources** (*list*) – a list of ProjectItemResources available for execution

Returns True if execution succeeded, False otherwise

Return type bool

_output_resources_forward(*self*)
Returns output resources for forward execution.

The default implementation returns an empty list.

Returns a list of ProjectItemResources

_output_resources_backward(*self*)
Returns output resources for backward execution.

The default implementation returns an empty list.

Returns a list of ProjectItemResources

classmethod from_dict(*cls*, *item_dict*, *name*, *project_dir*, *app_settings*, *specifications*, *logger*)
Deserializes an executable item from item dictionary.

Parameters

- **item_dict** (*dict*) – serialized project item
- **name** (*str*) – item's name
- **project_dir** (*str*) – absolute path to the project directory
- **app_settings** (*QSettings*) – Toolbox settings
- **specifications** (*dict*) – mapping from item specification name to ProjectItemSpecification
- **logger** (*LoggingInterface*) – a logger

Returns deserialized executable item

Return type *ExecutableItemBase*

spinetoolbox.execution_managers

Classes to manage tool instance execution in various forms.

author

P. Savolainen (VTT)

date 1.2.2018

Module Contents

Classes

ExecutionManager	Base class for all tool instance execution managers.
ConsoleExecutionManager	Class to manage tool instance execution using a SpineConsoleWidget.
QProcessExecutionManager	Class to manage tool instance execution using a PySide2 QProcess.

class spinetoolbox.execution_managers.**ExecutionManager**(*logger*)
Bases: PySide2.QtCore.QObject

Base class for all tool instance execution managers.

Class constructor.

Parameters **logger** (LoggerInterface) – a logger instance

execution_finished

start_execution(*self*, *workdir=None*)
Starts the execution.

Parameters **workdir** (*str*) – Work directory

stop_execution(*self*)
Stops the execution.

class spinetoolbox.execution_managers.**ConsoleExecutionManager**(*console*, *commands*, *logger*)
Bases: *spinetoolbox.execution_managers.ExecutionManager*

Class to manage tool instance execution using a SpineConsoleWidget.

Class constructor.

Parameters

- **console** (SpineConsoleWidget) – Console widget where execution happens
- **commands** (*list*) – List of commands to execute in the console
- **logger** (LoggerInterface) – a logger instance

start_execution(*self*, *workdir=None*)
See base class.

_start_execution(*self*)
Starts execution.

_execute_next_command(*self*)
Executes next command in the buffer.

stop_execution(*self*)
See base class.

class spinetoolbox.execution_managers.**QProcessExecutionManager**(*logger*, *program=None*, *args=None*, *silent=False*, *semisilent=False*)
Bases: *spinetoolbox.execution_managers.ExecutionManager*

Class to manage tool instance execution using a PySide2 QProcess.

Class constructor.

Parameters

- **logger** (LoggerInterface) – a logger instance
- **program** (*str*) – Path to program to run in the subprocess (e.g. julia.exe)
- **args** (*list*) – List of argument for the program (e.g. path to script file)

- **silent** (*bool*) – Whether or not to emit logger msg signals

program(*self*)
Program getter method.

args(*self*)
Program argument getter method.

start_execution(*self*, *workdir=None*)
Starts the execution of a command in a QProcess.

Parameters **workdir** (*str*) – Work directory

inject_data_to_write_channel(*self*)
Writes data to process write channel and closes it afterwards.

wait_for_process_finished(*self*, *msecs=30000*)
Wait for subprocess to finish.

Returns True if process finished successfully, False otherwise

process_started(*self*)
Run when subprocess has started.

on_state_changed(*self*, *new_state*)
Runs when QProcess state changes.

Parameters **new_state** (*QProcess::ProcessState*) – Process state number

on_process_error(*self*, *process_error*)
Runs if there is an error in the running QProcess.

Parameters **process_error** (*QProcess::ProcessError*) – Process error number

teardown_process(*self*)
Tears down the QProcess in case a QProcess.ProcessError occurred. Emits execution_finished signal.

stop_execution(*self*)
See base class.

on_process_finished(*self*, *exit_code*, *exit_status*)
Runs when subprocess has finished.

Parameters

- **exit_code** (*int*) – Return code from external program (only valid for normal exits)
- **exit_status** (*QProcess.ExitStatus*) – Crash or normal exit

on_ready_stdout(*self*)
Emit data from stdout.

on_ready_stderr(*self*)
Emit data from stderr.

spinetoolbox.graphics_items

Classes for drawing graphics items on QGraphicsScene.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

Classes

ProjectItemIcon	Base class for project item icons drawn in Design View.
ConnectorButton	Connector button graphics item. Used for Link drawing between project items.
ExclamationIcon	Exclamation icon graphics item.
RankIcon	Rank icon graphics item.
LinkBase	Base class for Link and LinkDrawer.
Link	Base class for Link and LinkDrawer.
LinkDrawer	An item for drawing links between project items.

class spinetoolbox.graphics_items.**ProjectItemIcon**(*toolbox*, *x*, *y*, *project_item*, *icon_file*, *icon_color*, *background_color*)

Bases: PySide2.QtWidgets.QGraphicsRectItem

Base class for project item icons drawn in Design View.

Parameters

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **x** (*float*) – Icon x coordinate
- **y** (*float*) – Icon y coordinate
- **project_item** (ProjectItem) – Item
- **icon_file** (*str*) – Path to icon resource
- **icon_color** (*QColor*) – Icon's color
- **background_color** (*QColor*) – Background color

ITEM_EXTENT = 64

activate(*self*)
Adds items to scene and setup graphics effect. Called in the constructor and when re-adding the item to the project in the context of undo/redo.

_setup(*self*, *brush*, *svg*, *svg_color*)
Setup item's attributes.

Parameters

- **brush** (*QBrush*) – Used in filling the background rectangle
- **svg** (*str*) – Path to SVG icon file
- **svg_color** (*QColor*) – Color of SVG icon

name(*self*)
Returns name of the item that is represented by this icon.

update_name_item(*self*, *new_name*)
Set a new text to name item. Used when a project item is renamed.

set_name_attributes(*self*)
Set name QGraphicsSimpleTextItem attributes (font, size, position, etc.)

conn_button(*self*, *position='left'*)
Returns items connector button (QWidget).

outgoing_links(*self*)

incoming_links(*self*)

hoverEnterEvent(*self*, *event*)
Sets a drop shadow effect to icon when mouse enters its boundaries.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

hoverLeaveEvent(*self*, *event*)
Disables the drop shadow when mouse leaves icon boundaries.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

mousePressEvent(*self*, *event*)

mouseMoveEvent(*self*, *event*)
Moves icon(s) while the mouse button is pressed. Update links that are connected to selected icons.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

moveBy(*self*, *dx*, *dy*)

update_links_geometry(*self*)
Updates geometry of connected links to reflect this item's most recent position.

mouseReleaseEvent(*self*, *event*)

notify_item_move(*self*)

contextMenuEvent(*self*, *event*)
Show item context menu.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Mouse event

keyPressEvent(*self*, *event*)
Handles deleting and rotating the selected item when dedicated keys are pressed.

Parameters **event** (*QKeyEvent*) – Key event

itemChange(*self*, *change*, *value*)
Reacts to item removal and position changes.

In particular, destroys the drop shadow effect when the items is removed from a scene and keeps track of item's movements on the scene.

Parameters

- **change** (*GraphicsItemChange*) – a flag signalling the type of the change
- **value** – a value related to the change

Returns Whatever super() does with the value parameter

show_item_info(*self*)
Update GUI to show the details of the selected item.

class spinetoolbox.graphics_items.**ConnectorButton**(*parent*, *toolbox*, *position='left'*)
Bases: PySide2.QtWidgets.QGraphicsRectItem

Connector button graphics item. Used for Link drawing between project items.

Parameters

- **parent** (*QGraphicsItem*) – Project item bg rectangle

- **toolbox** (*ToolBoxUI*) – QMainWindow instance
- **position** (*str*) – Either "top", "left", "bottom", or "right"

brush

hover_brush

outgoing_links(*self*)

incoming_links(*self*)

parent_name(*self*)
Returns project item name owning this connector button.

mousePressEvent(*self*, *event*)
Connector button mouse press event. Either starts or closes a link.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

set_friend_connectors_enabled(*self*, *enabled*)
Enables or disables all connectors in the parent. This is called by LinkDrawer to disable invalid connectors while drawing and reenabling them back when done.

mouseDoubleClickEvent(*self*, *event*)
Connector button mouse double click event. Makes sure the LinkDrawer is hidden.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

hoverEnterEvent(*self*, *event*)
Sets a darker shade to connector button when mouse enters its boundaries.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

hoverLeaveEvent(*self*, *event*)
Restore original brush when mouse leaves connector button boundaries.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

class spinetoolbox.graphics_items.**ExclamationIcon**(*parent*)
Bases: PySide2.QtSvg.QGraphicsSvgItem

Exclamation icon graphics item. Used to notify that a ProjectItem is missing some configuration.

Parameters **parent** (ProjectItemIcon) – the parent item

clear_notifications(*self*)
Clear all notifications.

add_notification(*self*, *text*)
Add a notification.

hoverEnterEvent(*self*, *event*)
Shows notifications as tool tip.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

hoverLeaveEvent(*self*, *event*)
Hides tool tip.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

class spinetoolbox.graphics_items.**RankIcon**(*parent*)
Bases: PySide2.QtWidgets.QGraphicsTextItem

Rank icon graphics item. Used to show the rank of a ProjectItem within its DAG

Parameters **parent** (ProjectItemIcon) – the parent item

set_rank(*self*, *rank*)

class spinetoolbox.graphics_items.**LinkBase**(*toolbox*)
Bases: PySide2.QtWidgets.QGraphicsPathItem

Base class for Link and LinkDrawer.

Mainly provides the update_geometry method for 'drawing' the link on the scene.

Initializes the instance.

Parameters **toolbox** (ToolboxUI) – main UI class instance

magic_number

src_rect
Returns the scene rectangle of the source connector.

src_center
Returns the center point of the source rectangle.

dst_rect
Returns the scene rectangle of the destination connector.

dst_center
Returns the center point of the destination rectangle.

moveBy(*self*, *_dx*, *_dy*)
Does nothing. This item is not moved the regular way, but follows the ConnectorButtons it connects.

update_geometry(*self*)
Updates geometry.

do_update_geometry(*self*, *curved_links*)
Sets the path for this item.

Parameters **curved_links** (*bool*) – Whether the path should follow a curvy line or a straight line

_make_ellipse_path(*self*)
Returns an ellipse path for the link's base.

Returns QPainterPath

_get_src_offset(*self*)

_get_dst_offset(*self*, *c1*)

_make_guide_path(*self*, *curved_links*)
Returns a 'narrow' path connecting this item's source and destination.

Parameters **curved_links** (*bool*) – Whether the path should follow a curved line or just a straight line

Returns QPainterPath

_points_and_angles_from_path(*self*, *path*)
Returns a list of representative points and angles from given path.

Parameters **path** (*QPainterPath*) –

Returns points list(float): angles

Return type list(QPointF)

_make_connecting_path(*self*, *guide_path*)
Returns a 'thick' path connecting source and destination, by following the given 'guide' path.

Parameters **guide_path** (*QPainterPath*) –

Returns QPainterPath

static _follow_points(*curve_path*, *points*)

_radius_from_point_and_angle(*self*, *point*, *angle*)

_make_arrow_path(*self*, *guide_path*)
Returns an arrow path for the link's tip.

Parameters **guide_path** (*QPainterPath*) – A narrow path connecting source and destination, used to determine the arrow orientation.

Returns QPainterPath

_get_joint_line(*self*, *guide_path*)

_get_joint_angle(*self*, *guide_path*)

class spinetoolbox.graphics_items.**Link**(*toolbox*, *src_connector*, *dst_connector*)
Bases: *spinetoolbox.graphics_items.LinkBase*

Base class for Link and LinkDrawer.

Mainly provides the `update_geometry` method for 'drawing' the link on the scene.

A graphics item to represent the connection between two project items.

Parameters

- **toolbox** (ToolboxUI) – main UI class instance
- **src_connector** (ConnectorButton) – Source connector button
- **dst_connector** (ConnectorButton) – Destination connector button

make_execution_animation(*self*)
Returns an animation to play when execution 'passes' through this link.

Returns QVariantAnimation

_handle_execution_animation_value_changed(*self*, *step*)

has_parallel_link(*self*)
Returns whether or not this link entirely overlaps another.

send_to_bottom(*self*)
Stacks this link before the parallel one if any.

mousePressEvent(*self*, *e*)
Ignores event if there's a connector button underneath, to allow creation of new links.

Parameters **e** (*QGraphicsSceneMouseEvent*) – Mouse event

contextMenuEvent(*self*, *e*)
Selects the link and shows context menu.

Parameters **e** (*QGraphicsSceneMouseEvent*) – Mouse event

keyPressEvent(*self*, *event*)
Removes this link if delete is pressed.

paint(*self*, *painter*, *option*, *widget*)
Sets a dashed pen if selected.

itemChange (*self*, *change*, *value*)
Brings selected link to top.

wipe_out (*self*)
Removes any trace of this item from the system.

class spinetoolbox.graphics_items.**LinkDrawer** (*toolbox*)
Bases: *spinetoolbox.graphics_items.LinkBase*

An item for drawing links between project items.

Init class.

Parameters **toolbox** (ToolboxUI) – main UI class instance

src_rect
Returns the scene rectangle of the source connector.

dst_rect
Returns the scene rectangle of the destination connector.

dst_center
Returns the center point of the destination rectangle.

add_link (*self*)
Makes link between source and destination connectors.

wake_up (*self*, *src_connector*)
Sets the source connector, shows this item and adds it to the scene. After calling this, the scene is in link drawing mode.

Parameters **src_connector** (ConnectorButton) –

sleep (*self*)
Removes this drawer from the scene, clears its source and destination connectors, and hides it. After calling this, the scene is no longer in link drawing mode.

spinetoolbox.headless

Contains facilities to open and execute projects without GUI.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

HeadlessLogger	A LoggerInterface compliant logger that uses Python's standard logging facilities.
ExecuteProject	A 'task' which opens and executes a Toolbox project when triggered to do so.
_Status	Status codes returned from headless execution.

Functions

headless_main(args)	Executes a project using QCoreApplication.
open_project(project_dict, project_dir, logger)	Opens a project.
_specifications(project_dict, project_dir, specification_factories, app_settings, logger)	Creates project item specifications.

class spinetoolbox.headless.**HeadlessLogger**

Bases: PySide2.QtCore.QObject

A LoggerInterface compliant logger that uses Python's standard logging facilities.

msg
Emits a notification message.

msg_success
Emits a message on success

msg_warning
Emits a warning message.

msg_error
Emits an error message.

msg_proc
Emits a message originating from a subprocess (usually something printed to stdout).

msg_proc_error
Emits an error message originating from a subprocess (usually something printed to stderr).

information_box
Requests an 'information message box' (e.g. a message window) to be opened with a given title and message.

error_box
Requests an 'error message box' to be opened with a given title and message.

_log_message(*self*, *message*)
Writes an information message to Python's logging system.

_log_warning(*self*, *message*)
Writes a warning message to Python's logging system.

_log_error(*self*, *message*)
Writes an error message to Python's logging system.

_show_information_box(*self*, *title*, *message*)
Writes an information message with a title to Python's logging system.

_show_error_box(*self*, *title*, *message*)
Writes an error message with a title to Python's logging system.

class spinetoolbox.headless.**ExecuteProject**(*args*, *startup_event_type*, *parent*)

Bases: PySide2.QtCore.QObject

A 'task' which opens and executes a Toolbox project when triggered to do so.

The execution of this task is triggered by sending it a 'startup' QEvent using, e.g. QCoreApplication.postEvent()

Parameters

- **args** (*argparse.Namespace*) – parsed command line arguments

- **startup_event_type** (*int*) – expected type id for the event that starts this task
- **parent** (*QObject*) – a parent object

_start
A private signal to actually start execution. Not to be used directly. Post a startup event instead.

_execute(*self*)
Executes this task.

_open_and_execute_project(*self*)
Opens a project and executes all DAGs in that project.

event(*self*, *e*)

spinetoolbox.headless.**headless_main**(*args*)
Executes a project using QCoreApplication.

Parameters **args** (*argparser.Namespace*) – parsed command line arguments.

Returns exit status code; 0 for success, everything else for failure

Return type int

spinetoolbox.headless.**open_project**(*project_dict*, *project_dir*, *logger*)
Opens a project.

Parameters

- **project_dict** (*dict*) – a serialized project dictionary
- **project_dir** (*str*) – path to a directory containing the .spinetoolbox dir
- **logger** (LoggerInterface) – a logger

Returns a list of executable items, a dict of item specifications, and a DagHandler object

Return type tuple

spinetoolbox.headless.**_specifications**(*project_dict*, *project_dir*, *specification_factories*, *app_settings*, *logger*)
Creates project item specifications.

Parameters

- **project_dict** (*dict*) – a serialized project dictionary
- **project_dir** (*str*) – path to a directory containing the .spinetoolbox dir
- **specification_factories** (*dict*) – a mapping from item type to specification factory
- **app_settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – a logger

Returns a mapping from item type and specification name to specification

Return type dict

class spinetoolbox.headless.**_Status**
Bases: enum.IntEnum

Status codes returned from headless execution.

Initialize self. See help(type(self)) for accurate signature.

OK = 0

ERROR = 1

spinetoolbox.load_project_items

Functions to load project item modules.

author

A. Soininen (VTT)

date 29.4.2020

Module Contents

Functions

load_project_items(toolbox)	Loads the standard project item modules included in the Toolbox package.
load_item_specification_factories()	Loads the project item specification factories in the standard Toolbox package.
load_executable_items()	Loads the project item executable classes included in the standard Toolbox package.

spinetoolbox.load_project_items.**load_project_items**(*toolbox*)

Loads the standard project item modules included in the Toolbox package.

Parameters **toolbox** (ToolboxUI) – Toolbox main widget

Returns

two dictionaries; first maps item type to its category while second maps item type to item factory

Return type tuple of dict

spinetoolbox.load_project_items.**load_item_specification_factories**()

Loads the project item specification factories in the standard Toolbox package.

Returns a map from item type to specification factory

Return type dict

spinetoolbox.load_project_items.**load_executable_items**()

Loads the project item executable classes included in the standard Toolbox package.

Returns a map from item type to the executable item class

Return type dict

spinetoolbox.logger_interface

A logger interface.

authors

A. Soininen (VTT)

date 16.1.2020

Module Contents

Classes

LoggerInterface	Placeholder for signals that can be emitted to send messages to an output device.

class spinetoolbox.logger_interface.**LoggerInterface**

Bases: PySide2.QtCore.QObject

Placeholder for signals that can be emitted to send messages to an output device.

The signals should be connected to a concrete logging system.

Currently, this is just a 'model interface'. ToolboxUI contains the same signals so it can be used as a drop-in replacement for this class.

msg
Emits a notification message.

msg_success
Emits a message on success

msg_warning
Emits a warning message.

msg_error
Emits an error message.

msg_proc
Emits a message originating from a subprocess (usually something printed to stdout).

msg_proc_error
Emits an error message originating from a subprocess (usually something printed to stderr).

information_box
Requests an 'information message box' (e.g. a message window) to be opened with a given title and message.

error_box
Requests an 'error message box' to be opened with a given title and message.

spinetoolbox.main

Provides the main() function.

author

A. Soininen (VTT)

date 4.10.2019

Module Contents

Functions

main()	Creates main window GUI and starts main event loop.
_make_argument_parser()	Returns a command line argument parser configured for Toolbox use.

spinetoolbox.main.**main**()
Creates main window GUI and starts main event loop.

spinetoolbox.main.**_make_argument_parser**()
Returns a command line argument parser configured for Toolbox use.

spinetoolbox.metaobject

MetaObject class.

authors

E. Rinne (VTT), P. Savolainen (VTT)

date 18.12.2017

Module Contents

Classes

MetaObject	Class for an object which has a name, type, and some description.

class spinetoolbox.metaobject.**MetaObject**(*name*, *description*)
Bases: PySide2.QtCore.QObject

Class for an object which has a name, type, and some description.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description

set_name(*self*, *name*)
Set object name and short name. Note: Check conflicts (e.g. name already exists) before calling this method.

Parameters **name** (*str*) – New (long) name for this object

set_description(*self*, *description*)
Set object description.

Parameters **description** (*str*) – Object description

spinetoolbox.plotting

Functions for plotting on PlotWidget.

Currently plotting from the table views found in the SpineDBEditor are supported.

The main entrance points to plotting are: - plot_selection() which plots selected cells on a table view returning a PlotWidget object - plot_pivot_column() which is a specialized method for plotting entire columns of a pivot table - add_time_series_plot() which adds a time series plot to an existing PlotWidget - add_map_plot() which adds a map plot to an existing PlotWidget

author

A. Soininen(VTT)

date 9.7.2019

Module Contents

Classes

PlottingHints	A base class for plotting hints.
ParameterTablePlottingHints	Support for plotting data in Parameter table views.
PivotTablePlottingHints	Support for plotting data in Tabular view.

Functions

plot_pivot_column(proxy_model, column, hints, plot_widget=None)	Returns a plot widget with a plot of an entire column in PivotTableModel.
plot_selection(model, indexes, hints, plot_widget=None)	Returns a plot widget with plots of the selected indexes.
add_array_plot(plot_widget, value, label=None)	Adds an array plot to a plot widget.
add_map_plot(plot_widget, map_value, label=None)	Adds a map plot to a plot widget.
add_time_series_plot(plot_widget, value, label=None)	Adds a time series step plot to a plot widget.
_add_plot_to_widget(values, labels, plot_widget)	Adds a new plot to plot_widget.
_raise_if_not_all_indexed_values(values)	Raises an exception if not all values are TimeSeries or Maps.
_filter_name_columns(selections)	Returns a dict with all but the entry with the greatest key removed.
_organize_selection_to_columns(indexes)	Organizes a list of model indexes into a dictionary of {column: (rows)} entries.
_collect_single_column_values(model, column, rows, hints)	Collects selected parameter values from a single column.
_collect_x_column_values(model, column, rows, hints)	Collects selected parameter values from an x column.
_collect_index_column_values(model, column, rows, hints)	Collects selected values from an index column.
_collect_column_values(model, column, rows, hints)	Collects selected parameter values from a single column for plotting.
_filter_and_check(xs, ys)	Filters Nones and empty values from x and y and checks that data types match.
_raise_if_indexed_values_not_plottable(values)	Raises an exception if the indexed values in values contain elements that cannot be plotted.

Continued on next page

Table 267 – continued from previous page

_raise_if_value_types_clash(values, plot_widget)	Raises a PlottingError if values type is incompatible with plot_widget.
_x_values_from_rows(model, rows, hints)	Returns x value array constructed from model rows.

exception spinetoolbox.plotting.**PlottingError**(*message*)

Bases: Exception

An exception signalling failure in plotting.

Parameters **message** (*str*) – an error message

message

the error message.

Type str

spinetoolbox.plotting.**plot_pivot_column**(*proxy_model*, *column*, *hints*, *plot_widget=None*)

Returns a plot widget with a plot of an entire column in PivotTableModel.

Parameters

- **proxy_model** (PivotTableSortFilterProxy) – a pivot table filter
- **column** (*int*) – a column index to the model
- **hints** (PlottingHints) – a helper needed for e.g. plot labels
- **plot_widget** (PlotWidget) – an existing plot widget to draw into or None to create a new widget

Returns a plot widget

Return type *PlotWidget*

spinetoolbox.plotting.**plot_selection**(*model*, *indexes*, *hints*, *plot_widget=None*)

Returns a plot widget with plots of the selected indexes.

Parameters

- **model** (*QAbstractTableModel*) – a model
- **indexes** (*Iterable*) – a list of QModelIndex objects for plotting
- **hints** (PlottingHints) – a helper needed for e.g. plot labels
- **plot_widget** (PlotWidget) – an existing plot widget to draw into or None to create a new widget

Returns a PlotWidget object

spinetoolbox.plotting.**add_array_plot**(*plot_widget*, *value*, *label=None*)

Adds an array plot to a plot widget.

Parameters

- **plot_widget** (PlotWidget) – a plot widget to modify
- **value** (*Array*) – the array to plot
- **label** (*str*) – a label for the array

spinetoolbox.plotting.**add_map_plot**(*plot_widget*, *map_value*, *label=None*)

Adds a map plot to a plot widget.

Parameters

- **plot_widget** (PlotWidget) – a plot widget to modify
- **map_value** (*Map*) – the map to plot
- **label** (*str*) – a label for the map

spinetoolbox.plotting.**add_time_series_plot**(*plot_widget*, *value*, *label=None*)
Adds a time series step plot to a plot widget.

Parameters

- **plot_widget** (PlotWidget) – a plot widget to modify
- **value** (*TimeSeries*) – the time series to plot
- **label** (*str*) – a label for the time series

class spinetoolbox.plotting.**PlottingHints**
A base class for plotting hints.

The functionality in this class allows the plotting functions to work without explicit knowledge of the underlying table model or widget.

cell_label(*self*, *model*, *index*)
Returns a label for the cell given by index in a table.

column_label(*self*, *model*, *column*)
Returns a label for a column.

filter_columns(*self*, *selections*, *model*)
Filters columns and returns the filtered selections.

is_index_in_data(*self*, *model*, *index*)
Returns true if the cell given by index is actually plottable data.

static normalize_row(*row*, *model*)
Returns a 'human understandable' row number

special_x_values(*self*, *model*, *column*, *rows*)
Returns X values if available, otherwise returns None.

x_label(*self*, *model*)
Returns a label for the x axis.

class spinetoolbox.plotting.**ParameterTablePlottingHints**
Bases: *spinetoolbox.plotting.PlottingHints*

Support for plotting data in Parameter table views.

cell_label(*self*, *model*, *index*)
Returns a label build from the columns on the left from the data column.

column_label(*self*, *model*, *column*)
Returns the column header.

filter_columns(*self*, *selections*, *model*)
Returns the 'value' or 'default_value' column only.

is_index_in_data(*self*, *model*, *index*)
Always returns True.

special_x_values(*self*, *model*, *column*, *rows*)
Always returns None.

x_label(*self*, *model*)
Returns an empty string for the x axis label.

class spinetoolbox.plotting.**PivotTablePlottingHints**
Bases: *spinetoolbox.plotting.PlottingHints*

Support for plotting data in Tabular view.

cell_label(*self*, *model*, *index*)
Returns a label for the table cell given by index.

column_label(*self*, *model*, *column*)
Returns a label for a table column.

filter_columns(*self*, *selections*, *model*)
Filters the X column from selections.

is_index_in_data(*self*, *model*, *index*)
Returns True if index is in the data portion of the table.

static normalize_row(*row*, *model*)
See base class.

special_x_values(*self*, *model*, *column*, *rows*)
Returns the values from the X column if one is designated otherwise returns None.

x_label(*self*, *model*)
Returns the label of the X column, if available.

static _map_column_to_source(*proxy_model*, *proxy_column*)
Maps a proxy model column to source model.

static _map_column_from_source(*proxy_model*, *source_column*)
Maps a source model column to proxy model.

spinetoolbox.plotting.**_add_plot_to_widget**(*values*, *labels*, *plot_widget*)
Adds a new plot to plot_widget.

spinetoolbox.plotting.**_raise_if_not_all_indexed_values**(*values*)
Raises an exception if not all values are TimeSeries or Maps.

spinetoolbox.plotting.**_filter_name_columns**(*selections*)
Returns a dict with all but the entry with the greatest key removed.

spinetoolbox.plotting.**_organize_selection_to_columns**(*indexes*)
Organizes a list of model indexes into a dictionary of {column: (rows)} entries.

spinetoolbox.plotting.**_collect_single_column_values**(*model*, *column*, *rows*, *hints*)
Collects selected parameter values from a single column.

The return value of this function depends on what type of data the given column contains. In case of plain numbers, a list of scalars and a single label string are returned. In case of indexed parameters (time series, maps), a list of parameter_value objects is returned, accompanied by a list of labels, each label corresponding to one of the indexed parameters.

Parameters

- **model** (*QAbstractTableModel*) – a table model
- **column** (*int*) – a column index to the model
- **rows** (*Sequence*) – row indexes to plot
- **hints** (PlottingHints) – a plot support object

Returns a tuple of values and label(s)

spinetoolbox.plotting.**_collect_x_column_values**(*model*, *column*, *rows*, *hints*)
Collects selected parameter values from an x column.

Parameters

- **model** (*QAbstractTableModel*) – a table model
- **column** (*int*) – a column index to the model
- **rows** (*Sequence*) – row indexes to plot
- **hints** (PlottingHints) – a plot support object

Returns a tuple of values and label(s)

spinetoolbox.plotting.**_collect_index_column_values**(*model*, *column*, *rows*, *hints*)
Collects selected values from an index column.

Parameters

- **model** (*QAbstractTableModel*) – a table model
- **column** (*int*) – a column index to the model
- **rows** (*Sequence*) – row indexes to plot
- **hints** (PlottingHints) – a plot support object

Returns column's values

Return type list

spinetoolbox.plotting.**_collect_column_values**(*model*, *column*, *rows*, *hints*)
Collects selected parameter values from a single column for plotting.

The return value of this function depends on what type of data the given column contains. In case of plain numbers, a single tuple of two lists of x and y values and a single label string are returned. In case of time series, a list of TimeSeries objects is returned, accompanied by a list of labels, each label corresponding to one of the time series.

Parameters

- **model** (*QAbstractTableModel*) – a table model
- **column** (*int*) – a column index to the model
- **rows** (*Sequence*) – row indexes to plot
- **hints** (PlottingHints) – a support object

Returns a tuple of values and label(s)

spinetoolbox.plotting.**_filter_and_check**(*xs*, *ys*)
Filters Nones and empty values from x and y and checks that data types match.

spinetoolbox.plotting.**_raise_if_indexed_values_not_plottable**(*values*)
Raises an exception if the indexed values in values contain elements that cannot be plotted.

spinetoolbox.plotting.**_raise_if_value_types_clash**(*values*, *plot_widget*)
Raises a PlottingError if values type is incompatible with plot_widget.

spinetoolbox.plotting.**_x_values_from_rows**(*model*, *rows*, *hints*)
Returns x value array constructed from model rows.

spinetoolbox.plugin_loader

Contains a minimal plugin loader infrastructure.

author

P. Savolainen (VTT)

date 11.6.2019

Module Contents

Functions

get_plugins(subpath)	Returns a list of plugin (module) names found in given subpath,
load_plugin(plugin_name)	Loads (imports) a plugin given its name.

spinetoolbox.plugin_loader.**get_plugins**(*subpath*)

Returns a list of plugin (module) names found in given subpath, relative to plugins main directory. Adds the directory to sys.path if any plugins were found.

Parameters **subpath** (*src*) – look for plugins in this subdirectory of the plugins main dir

spinetoolbox.plugin_loader.**load_plugin**(*plugin_name*)

Loads (imports) a plugin given its name.

Parameters **plugin_name** (*str*) – Name of the plugin (module) to load

spinetoolbox.project

Spine Toolbox project class.

authors

P. Savolainen (VTT), E. Rinne (VTT)

date 10.1.2018

Module Contents

Classes

SpineToolboxProject	Class for Spine Toolbox projects.

class spinetoolbox.project.**SpineToolboxProject**(*toolbox*, *name*, *description*, *p_dir*, *project_item_model*, *settings*, *embedded_julia_console*, *embedded_python_console*, *logger*)

Bases: *spinetoolbox.metaobject.MetaObject*

Class for Spine Toolbox projects.

Parameters

- **toolbox** (ToolboxUI) – toolbox of this project
- **name** (*str*) – Project name
- **description** (*str*) – Project description
- **p_dir** (*str*) – Project directory
- **project_item_model** (ProjectItemModel) – project item tree model
- **settings** (*QSettings*) – Toolbox settings
- **embedded_julia_console** (*JuliaConsoleWidget*) – a Julia console widget for execution in the embedded console
- **embedded_python_console** (*PythonConsoleWidget*) – a Python console widget for execution in the embedded console
- **logger** (LoggerInterface) – a logger instance

dag_execution_finished

project_execution_about_to_start
Emitted just before the entire project is executed.

settings

connect_signals(*self*)
Connect signals to slots.

_create_project_structure(*self*, *directory*)
Makes the given directory a Spine Toolbox project directory. Creates directories and files that are common to all projects.

Parameters **directory** (*str*) – Abs. path to a directory that should be made into a project directory

Returns True if project structure was created successfully, False otherwise

Return type bool

call_set_name(*self*, *name*)

call_set_description(*self*, *description*)

set_name(*self*, *name*)
Changes project name.

Parameters **name** (*str*) – New project name

set_description(*self*, *description*)
Set object description.

Parameters **description** (*str*) – Object description

static get_connections(*links*)

save(*self*, *spec_paths*)
Collects project information and objects into a dictionary and writes it to a JSON file.

Parameters **spec_paths** (*list*) – List of absolute paths to specification files

Returns True or False depending on success

Return type bool

load(*self*, *items_dict*)
Populates project item model with items loaded from project file.

Parameters **items_dict** (*dict*) – Dictionary containing all project items in JSON format

Returns True if successful, False otherwise

Return type bool

add_project_items(*self*, *item_type*, **items*, *set_selected=False*, *verbosity=True*)
Pushes an AddProjectItemsCommand to the toolbox undo stack.

make_project_tree_items(*self*, *item_type*, *items*)
Creates and returns a dictionary mapping category indexes to a list of corresponding LeafProjectTreeItem instances.

Parameters

- **item_type** (*str*) – item type
- **items** (*list*) – one or more dicts of items to add

Returns dict(QModelIndex, list(LeafProjectTreeItem))

do_add_project_tree_items(*self*, *category_ind*, **project_tree_items*, *set_selected=False*, *verbosity=True*)
Adds LeafProjectTreeItem instances to project.

Parameters

- **category_ind** (*QModelIndex*) – The category index
- **project_tree_items** (LeafProjectTreeItem) – one or more LeafProjectTreeItem instances to add
- **set_selected** (*bool*) – Whether to set item selected after the item has been added to project
- **verbosity** (*bool*) – If True, prints message

set_item_selected(*self*, *item*)
Selects the given item.

Parameters **item** (LeafProjectTreeItem) –

make_and_add_project_items(*self*, *item_type*, *items*, *set_selected=False*, *verbosity=True*)
Adds items to project at loading.

Parameters

- **item_type** (*str*) – Item type e.g. "Tool"
- **items** (*list*) – one or more dict of items to add
- **set_selected** (*bool*) – Whether to set item selected after the item has been added to project
- **verbosity** (*bool*) – If True, prints message

add_to_dag(*self*, *item_name*)
Add new node (project item) to the directed graph.

remove_all_items(*self*)
Pushes a RemoveAllProjectItemsCommand to the toolbox undo stack.

remove_item(*self*, *name*, *check_dialog=False*)
Pushes a RemoveProjectItemCommand to the toolbox undo stack.

Parameters

- **name** (*str*) – Item's name

- **check_dialog** (*bool*) – If True, shows 'Are you sure?' message box

do_remove_item(*self*, *name*)
Removes item from project given its name. This method is used when closing the existing project for opening a new one.

Parameters **name** (*str*) – Item's name

_remove_item(*self*, *category_ind*, *item*, *delete_data=False*)
Removes LeafProjectTreeItem from project.

Parameters

- **category_ind** (*QModelIndex*) – The category index
- **item** (LeafProjectTreeItem) – the item to remove
- **delete_data** (*bool*) – If set to True, deletes the directories and data associated with the item

execute_dags(*self*, *dags*, *execution_permits*)
Executes given dags.

Parameters

- **dags** (*Sequence(DiGraph)*) –
- **execution_permits** (*Sequence(dict)*) –

execute_next_dag(*self*)
Executes next dag in the execution list.

execute_dag(*self*, *dag*, *execution_permits*, *dag_identifier*)
Executes given dag.

Parameters

- **dag** (*DiGraph*) – Executed DAG
- **execution_permits** (*dict*) – Dictionary, where keys are node names in dag and value is a boolean
- **dag_identifier** (*str*) – Identifier number for printing purposes

execute_selected(*self*)
Executes DAGs corresponding to all selected project items.

execute_project(*self*)
Executes all dags in the project.

stop(*self*)
Stops execution. Slot for the main window Stop tool button in the toolbar.

export_graphs(*self*)
Exports all valid directed acyclic graphs in project to GraphML files.

notify_changes_in_dag(*self*, *dag*)
Notifies the items in given dag that the dag has changed.

notify_changes_in_all_dags(*self*)
Notifies all items of changes in all dags in the project.

notify_changes_in_containing_dag(*self*, *item*)
Notifies items in dag containing the given item that the dag has changed.

_handle_dag_node_execution_finished(*self*, *item_name*, *execution_direction*, *engine_state*)
Handles successful execution of a dag node. Performs post successful execution actions in corresponding project item.

direct_successors(*self*, *item*)
Returns a list of direct successor nodes for given project item.

spinetoolbox.project_commands

QUndoCommand subclasses for modifying the project.

authors

M. Marin (KTH)

date 12.2.2020

Module Contents

Classes

SpineToolboxCommand	
SetProjectNameCommand	Command to set the project name.
SetProjectDescriptionCommand	Command to set the project description.
AddProjectItemsCommand	Command to add items.
RemoveAllProjectItemsCommand	Command to remove all items from project.
RemoveProjectItemCommand	Command to remove items.
RenameProjectItemCommand	Command to rename items.
AddLinkCommand	Command to add link.
RemoveLinkCommand	Command to remove link.
MoveIconCommand	Command to move icons in the Design view.
SetItemSpecificationCommand	Command to set the specification for a Tool.
AddSpecificationCommand	Command to add item specs to a project.
RemoveSpecificationCommand	Command to remove item specs from a project.
UpdateSpecificationCommand	Command to update item specs in a project.

class spinetoolbox.project_commands.**SpineToolboxCommand**
Bases: PySide2.QtWidgets.QUndoCommand

static is_critical()
Returns True if this command needs to be undone before closing the project without saving changes.

class spinetoolbox.project_commands.**SetProjectNameCommand**(*project*, *name*)
Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to set the project name.

Parameters

- **project** (SpineToolboxProject) – the project
- **name** (*str*) – The new name

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**SetProjectDescriptionCommand**(*project*, *description*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to set the project description.

Parameters

- **project** (SpineToolboxProject) – the project
- **description** (*str*) – The new description

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**AddProjectItemsCommand**(*project*, *item_type*, *items*, *set_selected=False*, *verbosity=True*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to add items.

Parameters

- **project** (SpineToolboxProject) – the project
- **item_type** (*str*) – The factory name
- **items** (*Iterable*) – one or more dict of items to add
- **set_selected** (*bool*) – Whether to set item selected after the item has been added to project
- **verbosity** (*bool*) – If True, prints message

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**RemoveAllProjectItemsCommand**(*project*, *items_per_category*, *links*, *delete_data=False*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to remove all items from project.

Parameters

- **project** (SpineToolboxProject) – the project
- **delete_data** (*bool*) – If True, deletes the directories and data associated with the items

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**RemoveProjectItemCommand**(*project*, *name*, *delete_data=False*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to remove items.

Parameters

- **project** (SpineToolboxProject) – the project

- **name** (*str*) – Item's name
- **delete_data** (*bool*) – If True, deletes the directories and data associated with the item

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**RenameProjectItemCommand**(*project_item_model*, *tree_item*, *new_name*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to rename items.

Parameters

- **project_item_model** (ProjectItemModel) – the project
- **tree_item** (LeafProjectTreeItem) – the item to rename
- **new_name** (*str*) – the new name

redo(*self*)

undo(*self*)

static is_critical()

Returns True if this command needs to be undone before closing the project without saving changes.

class spinetoolbox.project_commands.**AddLinkCommand**(*graphics_view*, *src_connector*, *dst_connector*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to add link.

Parameters

- **graphics_view** (DesignQGraphicsView) – the view
- **src_connector** (ConnectorButton) – the source connector
- **dst_connector** (ConnectorButton) – the destination connector

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**RemoveLinkCommand**(*graphics_view*, *link*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to remove link.

Parameters

- **graphics_view** (DesignQGraphicsView) – the view
- **link** (Link) – the link

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**MoveIconCommand**(*graphics_item*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to move icons in the Design view.

Parameters **graphics_item** (ProjectItemIcon) – the icon

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**SetItemSpecificationCommand**(*item*, *specification*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to set the specification for a Tool.

Parameters

- **item** (ProjectItem) – the Item
- **specification** (ProjectItemSpecification) – the new spec

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**AddSpecificationCommand**(*toolbox*, *specification*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to add item specs to a project.

Parameters

- **toolbox** (ToolboxUI) – the toolbox
- **specification** (ProjectItemSpecification) – the spec

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**RemoveSpecificationCommand**(*toolbox*, *row*, *ask_verification*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to remove item specs from a project.

Parameters

- **toolbox** (ToolboxUI) – the toolbox
- **row** (*int*) – the row in the ProjectItemSpecPaletteModel
- **ask_verification** (*bool*) – if True, shows confirmation message the first time

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**UpdateSpecificationCommand**(*toolbox*, *row*, *specification*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update item specs in a project.

Parameters

- **toolbox** (ToolboxUI) – the toolbox
- **row** (*int*) – the row in the ProjectItemSpecPaletteModel of the spec to be replaced
- **specification** (ProjectItemSpecification) – the updated spec

redo(*self*)

undo(*self*)

spinetoolbox.project_item

Contains base classes for project items and item factories.

authors

P. Savolainen (VTT)

date 4.10.2018

Module Contents

Classes

ProjectItem	Class for project items that are not category nor root.
ProjectItemFactory	Class for project item factories.

class spinetoolbox.project_item.**ProjectItem**(*name*, *description*, *x*, *y*, *project*, *logger*)

Bases: *spinetoolbox.metaobject.MetaObject*

Class for project items that are not category nor root. These items can be executed, refreshed, and so on.

x

horizontal position in the screen

Type float

y

vertical position in the screen

Type float

Parameters

- **name** (*str*) – item name
- **description** (*str*) – item description
- **x** (*float*) – horizontal position on the scene
- **y** (*float*) – vertical position on the scene
- **project** (SpineToolboxProject) – project item's project
- **logger** (LoggerInterface) – a logger instance

item_changed

Request DAG update. Emitted when a change affects other items in the DAG.

create_data_dir(*self*)

static item_type()

Item's type identifier string.

static item_category()

Item's category.

make_signal_handler_dict (*self*)
Returns a dictionary of all shared signals and their handlers. This is to enable simpler connecting and disconnecting. Must be implemented in subclasses.

activate (*self*)
Restore selections and connect signals.

deactivate (*self*)
Save selections and disconnect signals.

restore_selections (*self*)
Restore selections into shared widgets when this project item is selected.

save_selections (*self*)
Save selections in shared widgets for this project item into instance variables.

_connect_signals (*self*)
Connect signals to handlers.

_disconnect_signals (*self*)
Disconnect signals from handlers and check for errors.

set_properties_ui (*self*, *properties_ui*)
Sets the properties tab widget for the item.

Note that this method expects the widget that is generated from the .ui files and initialized with the setupUi() method rather than the entire properties tab widget.

Parameters **properties_ui** (*QWidget*) – item's properties UI

specification (*self*)
Returns the specification for this item.

set_specification (*self*, *specification*)
Pushes a new SetToolSpecificationCommand to the toolbox' undo stack.

do_set_specification (*self*, *specification*)
Sets Tool specification for this Tool. Removes Tool specification if None given as argument.

Parameters **specification** (ToolSpecification) – Tool specification of this Tool. None removes the specification.

undo_set_specification (*self*)

set_icon (*self*, *icon*)
Sets the icon for the item.

Parameters **icon** (ProjectItemIcon) – item's icon

get_icon (*self*)
Returns the graphics item representing this item in the scene.

clear_notifications (*self*)
Clear all notifications from the exclamation icon.

add_notification (*self*, *text*)
Add a notification to the exclamation icon.

set_rank (*self*, *rank*)
Set rank of this item for displaying in the design view.

execution_item (*self*)
Creates project item's execution counterpart.

handle_execution_successful(*self*, *execution_direction*, *engine_state*)
Performs item dependent actions after the execution item has finished successfully.

resources_for_direct_successors(*self*)
Returns resources for direct successors.

These resources can include transient files that don't exist yet, or filename patterns. The default implementation returns an empty list.

Returns a list of ProjectItemResources

Return type list

handle_dag_changed(*self*, *rank*, *resources*)
Handles changes in the DAG.

Subclasses should reimplement the _do_handle_dag_changed() method.

Parameters

- **rank** (*int*) – item's execution order
- **resources** (*list*) – resources available from input items

_do_handle_dag_changed(*self*, *resources*)
Handles changes in the DAG.

Usually this entails validating the input resources and populating file references etc. The default implementation does nothing.

Parameters **resources** (*list*) – resources available from input items

invalidate_workflow(*self*, *edges*)
Notifies that this item's workflow is not acyclic.

Parameters **edges** (*list*) – A list of edges that make the graph acyclic after removing them.

item_dict(*self*)
Returns a dictionary corresponding to this item.

static default_name_prefix()
prefix for default item name

rename(*self*, *new_name*)
Renames this item.

If the project item needs any additional steps in renaming, override this method in subclass. See e.g. rename() method in DataStore class.

Parameters **new_name** (*str*) – New name

Returns True if renaming succeeded, False otherwise

Return type bool

open_directory(*self*)
Open this item's data directory in file explorer.

tear_down(*self*)
Tears down this item. Called both before closing the app and when removing the item from the project. Implement in subclasses to eg close all QMainWindows opened by this item.

set_up(*self*)
Sets up this item. Called when adding the item to the project. Implement in subclasses to eg recreate attributes destroyed by tear_down.

update_name_label(*self*)
Updates the name label on the properties widget when renaming an item.

Must be reimplemented by subclasses.

notify_destination(*self*, *source_item*)
Informs an item that it has become the destination of a connection between two items.

The default implementation logs a warning message. Subclasses should reimplement this if they need more specific behavior.

Parameters **source_item** (ProjectItem) – connection source item

static upgrade_from_no_version_to_version_1(*item_name*, *old_item_dict*, *old_project_dir*)
Upgrades item's dictionary from no version to version 1.

Subclasses should reimplement this method if their JSON format changed between no version and version 1 .proj files.

Parameters

- **item_name** (*str*) – item's name
- **old_item_dict** (*dict*) – no version item dictionary
- **old_project_dir** (*str*) – path to the previous project dir. We use old project directory here since the new project directory may be empty at this point and the directories for the new project items have not been created yet.

Returns version 1 item dictionary

static upgrade_v1_to_v2(*item_name*, *item_dict*)
Upgrades item's dictionary from v1 to v2.

Subclasses should reimplement this method if there are changes between version 1 and version 2.

Parameters

- **item_name** (*str*) – item's name
- **item_dict** (*dict*) – Version 1 item dictionary

Returns Version 2 item dictionary

Return type dict

class spinetoolbox.project_item.**ProjectItemFactory**(*toolbox*)
Class for project item factories.

Parameters **toolbox** (ToolboxUI) –

item_maker
Returns a ProjectItem subclass.

Returns class

icon_maker
Returns a ProjectItemIcon subclass.

Returns class

add_form_maker
Returns an AddProjectItem subclass.

Returns class

specification_form_maker
Returns a QWidget subclass to create and edit specifications.

Returns class

specification_menu_maker
Returns an ItemSpecificationMenu subclass.

Returns class

static icon()
Returns the icon resource path.

Returns str

static supports_specifications()
Returns whether or not this factory supports specs.

If the subclass implementation returns True, then it must also implement `specification_form_maker`, and `specification_menu_maker`.

Returns bool

make_icon(*self*, *toolbox*, *x*, *y*, *project_item*)
Returns a ProjectItemIcon to use with given toolbox, for given project item.

Parameters

- **toolbox** (ToolboxUI) –
- **x** (*int*) – Icon X coordinate
- **y** (*int*) – Icon Y coordinate
- **project_item** (ProjectItem) – Project item

Returns ProjectItemIcon

make_item(*self*, **args*, ***kwargs*)
Returns a project item while setting its factory attribute.

Returns ProjectItem

activate_project_item(*self*, *toolbox*, *project_item*)
Activates the given project item so it works with the given toolbox. This is mainly intended to facilitate adding items back with redo.

Parameters

- **toolbox** (ToolboxUI) –
- **project_item** (ProjectItem) –

static _make_properties_widget(*toolbox*)
Creates the item's properties tab widget.

Returns QWidget

spinetoolbox.project_item_info

Provides the ProjectItemInfo class.

authors

A. Soininen (VTT)

date 29.4.2020

Module Contents

Classes

ProjectItemInfo	

class spinetoolbox.project_item_info.**ProjectItemInfo**

static item_category()
Returns the item category string, e.g., "Tools".

Returns str

static item_type()
Returns the item type string, e.g., "Gimlet".

Returns str

spinetoolbox.project_item_resource

Provides the ProjectItemResource class.

authors

M. Marin (KTH)

date 29.4.2020

Module Contents

Classes

ProjectItemResource	Class to hold a resource made available by a project item

class spinetoolbox.project_item_resource.**ProjectItemResource**(*provider*, *type_*, *url=''*, *metadata=None*)
Class to hold a resource made available by a project item and that may be consumed by another project item.

Parameters

- **provider** (*ProjectItem or ExecutionItem*) – The item that provides the resource
- **type** (*str*) – The resource type, currently available types:
  - "file": url points to the file's path
  - "database": url is the databases url
  - "transient_file": a file that may not yet be available or may change its location; url points to latest version or is empty, metadata contains the "label" key and an optional "pattern"

key

- "file_pattern": a file pattern with wildcards that acts as a placeholder; url is empty, metadata contains the "label" key

- **url** (*str*) – The url of the resource
- **metadata** (*dict*) – Some metadata providing extra information about the resource. Currently available keys:
  - label (str): a textual label
  - pattern (str): a file pattern if the file is part of that pattern

path
Returns the resource path in the local syntax, as obtained from parsing the url.

scheme
Returns the resource scheme, as obtained from parsing the url.

__eq__(*self*, *other*)
Return self==value.

__repr__(*self*)
Return repr(self).

spinetoolbox.project_item_specification

Contains project item specification class.

authors

M. Marin (KTH)

date 7.5.2020

Module Contents

Classes

ProjectItemSpecification	Class to hold a project item specification.

class spinetoolbox.project_item_specification.**ProjectItemSpecification**(*name*, *description=None*, *item_type=''*, *item_category=''*)

Bases: *spinetoolbox.metaobject.MetaObject*

Class to hold a project item specification.

item_type
type of the project item the specification is compatible with

Type str

definition_file_path
specification's JSON file path

Type str

Parameters

- **name** (*str*) – specification name
- **description** (*str*) – description
- **item_type** (*str*) – Project item type
- **item_category** (*str*) – Project item category

__eq__(*self*, *other*)
Overrides the default implementation

spinetoolbox.project_item_specification_factory

Contains project item specification factory.

authors

A. Soininen (VTT)

date 6.5.2020

Module Contents

Classes

ProjectItemSpecificationFactory	A factory to make project item specifications.

class spinetoolbox.project_item_specification_factory.**ProjectItemSpecificationFactory**
A factory to make project item specifications.

static item_type()
Returns the project item's type.

static make_specification(*definition*, *definition_path*, *app_settings*, *logger*, *embedded_julia_console*, *embedded_python_console*)
Makes a project item specification.

Parameters

- **definition** (*dict*) – specification's definition dictionary
- **definition_path** (*str*) – path to the definition file
- **app_settings** (*QSettings*) – Toolbox settings
- **logger** (LoggerInterface) – a logger
- **embedded_julia_console** (*JuliaREPLWidget, optional*) – a console widget for specifications that need one
- **embedded_python_console** (*PythonReplWidget, optional*) – a console widget for specifications that need one

Returns a specification built from the given definition

Return type *ProjectItemSpecification*

spinetoolbox.project_tree_item

Project Tree items.

authors

A. Soininen (VTT)

date 17.1.2020

Module Contents

Classes

BaseProjectTreeItem	Base class for all project tree items.
RootProjectTreeItem	Class for the root project tree item.
CategoryProjectTreeItem	Class for category project tree items.
LeafProjectTreeItem	Class for leaf items in the project item tree.

class spinetoolbox.project_tree_item.**BaseProjectTreeItem**(*name*, *description*)

Bases: *spinetoolbox.metaobject.MetaObject*

Base class for all project tree items.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description

flags(*self*)

Returns the item flags.

parent(*self*)

Returns parent project tree item.

child_count(*self*)

Returns the number of child project tree items.

children(*self*)

Returns the children of this project tree item.

child(*self*, *row*)

Returns child BaseProjectTreeItem on given row.

Parameters **row** (*int*) – Row of child to return

Returns item on given row or None if it does not exist

Return type *BaseProjectTreeItem*

row(*self*)

Returns the row on which this item is located.

add_child(*self*, *child_item*)

Base method that shall be overridden in subclasses.

remove_child(*self*, *row*)

Remove the child of this BaseProjectTreeItem from given row. Do not call this method directly. This method is called by LeafProjectItemTreeModel when items are removed.

Parameters **row** (*int*) – Row of child to remove

Returns True if operation succeeded, False otherwise

Return type bool

custom_context_menu(*self*, *parent*, *pos*)
Returns the context menu for this item. Implement in subclasses as needed. :param parent: The widget that is controlling the menu :type parent: QWidget :param pos: Position on screen :type pos: QPoint

apply_context_menu_action(*self*, *parent*, *action*)
Applies given action from context menu. Implement in subclasses as needed.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu
- **action** (*str*) – The selected action

class spinetoolbox.project_tree_item.**RootProjectTreeItem**
Bases: *spinetoolbox.project_tree_item.BaseProjectTreeItem*

Class for the root project tree item.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description

add_child(*self*, *child_item*)
Adds given category item as the child of this root project tree item. New item is added as the last item.

Parameters **child_item** (CategoryProjectTreeItem) – Item to add

Returns True for success, False otherwise

custom_context_menu(*self*, *parent*, *pos*)
See base class.

apply_context_menu_action(*self*, *parent*, *action*)
See base class.

class spinetoolbox.project_tree_item.**CategoryProjectTreeItem**(*name*, *description*)
Bases: *spinetoolbox.project_tree_item.BaseProjectTreeItem*

Class for category project tree items.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description

flags(*self*)
Returns the item flags.

add_child(*self*, *child_item*)
Adds given project tree item as the child of this category item. New item is added as the last item.

Parameters **child_item** (*LeafProjectTreeTreeItem*) – Item to add

Returns True for success, False otherwise

custom_context_menu(*self*, *parent*, *pos*)
Returns the context menu for this item.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu
- **pos** (*QPoint*) – Position on screen

apply_context_menu_action(*self*, *parent*, *action*)
Applies given action from context menu.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu
- **action** (*str*) – The selected action

class spinetoolbox.project_tree_item.**LeafProjectTreeItem**(*project_item*, *toolbox*)
Bases: *spinetoolbox.project_tree_item.BaseProjectTreeItem*

Class for leaf items in the project item tree.

Parameters

- **project_item** (ProjectItem) – the real project item this item represents
- **toolbox** (*ToobloxUI*) – a toolbox instance

project_item
the project item linked to this leaf

toolbox
the toolbox instance

add_child(*self*, *child_item*)
See base class.

flags(*self*)
Returns the item flags.

custom_context_menu(*self*, *parent*, *pos*)
Returns the context menu for this item.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu
- **pos** (*QPoint*) – Position on screen

apply_context_menu_action(*self*, *parent*, *action*)
Applies given action from context menu.

Parameters

- **parent** (*QWidget*) – The widget that is controlling the menu
- **action** (*str*) – The selected action

rename(*self*, *new_name*)
Renames this item.

Parameters **new_name** (*str*) – New name

Returns True if renaming was successful, False if renaming failed

Return type bool

spinetoolbox.project_upgrader

Contains ProjectUpgrader class used in upgrading and converting projects and project dicts from earlier versions to the latest version.

authors

P. Savolainen (VTT)

date 8.11.2019

Module Contents

Classes

ProjectUpgrader	Class to upgrade/convert projects from earlier versions to the current version.

class spinetoolbox.project_upgrader.**ProjectUpgrader**(*toolbox*)

Class to upgrade/convert projects from earlier versions to the current version.

Parameters **toolbox** (ToolboxUI) – toolbox of this project

upgrade_to_v1(*self*, *project_dict*, *old_project_dir*)

Upgrades no version project dict to version 1. This may be removed when we no longer want to support upgrading legacy .proj projects to current ones.

upgrade(*self*, *project_dict*, *project_dir*)

Upgrades the project described in given project dictionary to the latest version.

Parameters

- **project_dict** (*dict*) – Project configuration dictionary
- **project_dir** (*str*) – Path to current project directory

Returns Latest version of the project info dictionary

Return type dict

upgrade_to_latest(*self*, *v*, *project_dict*)

Upgrades the given project dictionary to the latest version.

Parameters

- **v** (*int*) – Current version of the project dictionary
- **project_dict** (*dict*) – Project dictionary (JSON) to be upgraded

Returns Upgraded project dictionary

Return type dict

static upgrade_v1_to_v2(*old*, *item_makers*)

Upgrades version 1 project dictionary to version 2.

Changes: objects -> items, tool_specifications -> specifications store project item dicts under ["items"][<project item name>] instead of using their categories as keys specifications must be a dict instead of a list Add specifications["Tool"] that must be a dict Remove "short name" from all project items

Parameters

- **old** (*dict*) – Version 1 project dictionary
- **item_makers** (*dict*) – Mapping of item type to item class

Returns Version 2 project dictionary

Return type dict

upgrade_from_no_version_to_version_1 (*self*, *old*, *old_project_dir*)
Converts project information dictionaries without 'version' to version 1.

Parameters

- **old** (*dict*) – Project information JSON
- **old_project_dir** (*str*) – Path to old project directory

Returns Project information JSON upgraded to version 1

Return type dict

upgrade_connections (*self*, *item_names*, *connections_old*)
Upgrades connections from old format to the new format.

- Old format. List of lists, e.g.
- New format. List of dicts, e.g.

static upgrade_tool_specification_paths (*spec_paths*, *old_project_dir*)
Upgrades a list of tool specifications paths to new format. Paths in (old) project directory (yes, old is correct) are converted to relative, others as absolute.

open_proj_json (*self*, *proj_file_path*)
Opens an old style project file (.proj) for reading,

Parameters **proj_file_path** (*str*) – Full path to the old .proj project file

Returns Project dictionary or None if the operation fails.

Return type dict

get_project_directory (*self*)
Asks the user to select a new project directory. If the selected directory is already a Spine Toolbox project directory, asks if overwrite is ok. Used when opening a project from an old style project file (.proj).

Returns Path to project directory or an empty string if operation is canceled.

Return type str

copy_data (*self*, *proj_file_path*, *project_dir*)
Copies project item directories from the old project to the new project directory.

Parameters

- **proj_file_path** (*str*) – Path to .proj file
- **project_dir** (*str*) – New project directory

Returns True if copying succeeded, False if it failed

Return type bool

is_valid (*self*, *v*, *p*)
Checks given project dict if it is valid for given version.

is_valid_v1(*self*, *p*)
Checks that the given project JSON dictionary contains a valid version 1 Spine Toolbox project. Valid meaning, that it contains all required keys and values are of the correct type.

Parameters **p** (*dict*) – Project information JSON

Returns True if project is a valid version 1 project, False if it is not

Return type bool

is_valid_v2(*self*, *p*)
Checks that the given project JSON dictionary contains a valid version 2 Spine Toolbox project. Valid meaning, that it contains all required keys and values are of the correct type.

Parameters **p** (*dict*) – Project information JSON

Returns True if project is a valid version 2 project, False if it is not

Return type bool

backup_project_file(*self*, *project_dir*, *v*)
Makes a backup copy of project.json file.

force_save(*self*, *p*, *project_dir*)
Saves given project dictionary to project.json file. Used to force save project.json file when the project dictionary has been upgraded.

spinetoolbox.spine_db_commands

QUndoCommand subclasses for modifying the db.

authors

M. Marin (KTH)

date 31.1.2020

Module Contents

Classes

AgedUndoStack	
AgedUndoCommand	**param parent** The parent command, used for defining macros.
SpineDBCommand	**param db_mngr** SpineDBManager instance
AddItemsCommand	**param db_mngr** SpineDBManager instance

Continued on next page

Table 278 – continued from previous page

AddCheckedParameterValuesCommand	**param db_mngr** SpineDBManager instance
UpdateItemsCommand	**param db_mngr** SpineDBManager instance
UpdateCheckedParameterValuesCommand	**param db_mngr** SpineDBManager instance
RemoveItemsCommand	**param db_mngr** SpineDBManager instance

Functions

_cache_to_db_relationship_class(item)	
_cache_to_db_relationship(item)	
_cache_to_db_parameter_definition(item)	
_cache_to_db_parameter_value(item)	
_cache_to_db_parameter_value_list(item)	
_cache_to_db_item(item_type, item)	
_format_item(item_type, item)	

spinetoolbox.spine_db_commands.**_cache_to_db_relationship_class**(*item*)

spinetoolbox.spine_db_commands.**_cache_to_db_relationship**(*item*)

spinetoolbox.spine_db_commands.**_cache_to_db_parameter_definition**(*item*)

spinetoolbox.spine_db_commands.**_cache_to_db_parameter_value**(*item*)

spinetoolbox.spine_db_commands.**_cache_to_db_parameter_value_list**(*item*)

spinetoolbox.spine_db_commands.**_cache_to_db_item**(*item_type*, *item*)

spinetoolbox.spine_db_commands.**_format_item**(*item_type*, *item*)

class spinetoolbox.spine_db_commands.**AgedUndoStack**

Bases: PySide2.QtWidgets.QUndoStack

redo_age

undo_age

commands(*self*)

class spinetoolbox.spine_db_commands.**AgedUndoCommand**(*parent=None*)

Bases: PySide2.QtWidgets.QUndoCommand

Parameters **parent** (*QUndoCommand, optional*) – The parent command, used for defining macros.

age

class spinetoolbox.spine_db_commands.**SpineDBCommand**(*db_mngr*, *db_map*, *parent=None*)

Bases: *spinetoolbox.spine_db_commands.AgedUndoCommand*

Parameters

- **db_mngr** (SpineDBManager) – SpineDBManager instance
- **db_map** (*DiffDatabaseMapping*) – DiffDatabaseMapping instance
- **parent** (*QUndoCommand, optional*) – The parent command, used for defining macros.

_add_command_name

_update_command_name

_add_method_name

_readd_method_name

_update_method_name

_get_method_name

_added_signal_name

_updated_signal_name

silence_listener(*self*, *func*)
Calls given function while silencing the listener Spine db editors. This is so undo() and subsequent redo() calls don't trigger the same notifications over and over.

static redomethod(*func*)
Returns a new redo method that determines if the command was completed. The command is completed if calling the function triggers the `completed_signal`. Once the command is completed, we don't listen to the signal anymore and we also silence the affected Spine db editors. If the signal is not received, then the command is declared obsolete.

static undomethod(*func*)
Returns a new undo method that silences the affected Spine db editors.

receive_items_changed(*self*, *_db_map_data*)
Marks the command as completed.

data(*self*)
Returns data to present this command in a DBHistoryDialog.

class spinetoolbox.spine_db_commands.**AddItemsCommand**(*db_mngr*, *db_map*, *data*, *item_type*, *parent=None*)

Bases: *spinetoolbox.spine_db_commands.SpineDBCommand*

Parameters

- **db_mngr** (SpineDBManager) – SpineDBManager instance
- **db_map** (*DiffDatabaseMapping*) – DiffDatabaseMapping instance
- **data** (*list*) – list of dict-items to add
- **item_type** (*str*) – the item type
- **parent** (*QUndoCommand, optional*) – The parent command, used for defining macros.

redo(*self*)

undo(*self*)

receive_items_changed(*self*, *db_map_data*)
Marks the command as completed.

data(*self*)
Returns data to present this command in a DBHistoryDialog.

class spinetoolbox.spine_db_commands.**AddCheckedParameterValuesCommand**(*db_mngr*, *db_map*, *data*, *parent=None*)

Bases: *spinetoolbox.spine_db_commands.AddItemsCommand*

Parameters

- **db_mngr** (SpineDBManager) – SpineDBManager instance
- **db_map** (*DiffDatabaseMapping*) – DiffDatabaseMapping instance
- **data** (*list*) – list of dict-items to add
- **item_type** (*str*) – the item type
- **parent** (*QUndoCommand, optional*) – The parent command, used for defining macros.

class spinetoolbox.spine_db_commands.**UpdateItemsCommand**(*db_mngr*, *db_map*, *data*, *item_type*, *parent=None*)

Bases: *spinetoolbox.spine_db_commands.SpineDBCommand*

Parameters

- **db_mngr** (SpineDBManager) – SpineDBManager instance
- **db_map** (*DiffDatabaseMapping*) – DiffDatabaseMapping instance
- **data** (*list*) – list of dict-items to update
- **item_type** (*str*) – the item type
- **parent** (*QUndoCommand, optional*) – The parent command, used for defining macros.

_undo_item(*self*, *db_map*, *redo_item*)

redo(*self*)

undo(*self*)

data(*self*)
Returns data to present this command in a DBHistoryDialog.

class spinetoolbox.spine_db_commands.**UpdateCheckedParameterValuesCommand**(*db_mngr*, *db_map*, *data*, *parent=None*)

Bases: *spinetoolbox.spine_db_commands.UpdateItemsCommand*

Parameters

- **db_mngr** (SpineDBManager) – SpineDBManager instance
- **db_map** (*DiffDatabaseMapping*) – DiffDatabaseMapping instance

- **data** (*list*) – list of dict-items to update
- **item_type** (*str*) – the item type
- **parent** (*QUndoCommand, optional*) – The parent command, used for defining macros.

class spinetoolbox.spine_db_commands.**RemoveItemsCommand**(*db_mngr*, *db_map*, *typed_data*, *parent=None*)

Bases: *spinetoolbox.spine_db_commands.SpineDBCommand*

Parameters

- **db_mngr** (SpineDBManager) – SpineDBManager instance
- **db_map** (*DiffDatabaseMapping*) – DiffDatabaseMapping instance
- **typed_data** (*dict*) – lists of dict-items to remove keyed by string type
- **parent** (*QUndoCommand, optional*) – The parent command, used for defining macros.

redo(*self*)

undo(*self*)

receive_items_changed(*self*, *typed_db_map_data*)

Marks the command as completed.

data(*self*)

Returns data to present this command in a DBHistoryDialog.

spinetoolbox.spine_db_fetcher

SpineDBFetcher class.

authors

M. Marin (KTH)

date 13.3.2020

Module Contents

Classes

SpineDBFetcher	Fetches content from a Spine database and 'sends' them to another thread (via a signal-slot mechanism of course),

class spinetoolbox.spine_db_fetcher.**SpineDBFetcher**(*db_mngr*, *listener*)

Bases: PySide2.QtCore.QObject

Fetches content from a Spine database and 'sends' them to another thread (via a signal-slot mechanism of course), so contents can be processed in that thread without affecting the UI.

Initializes the fetcher object.

Parameters

- **db_mngr** (SpineDBManager) –
- **listener** (SpineDBEditor) –

finished

_ready_to_finish

_alternatives_fetched

_scenarios_fetched

_scenarios_alternatives_fetched

_object_classes_fetched

_objects_fetched

_relationship_classes_fetched

_relationships_fetched

_entity_groups_fetched

_parameter_definitions_fetched

_parameter_definition_tags_fetched

_parameter_values_fetched

_parameter_value_lists_fetched

_parameter_tags_fetched

connect_signals (*self*)
Connects signals.

fetch (*self*, *db_maps*)
Fetches items from the database and emit fetched signals.

clean_up (*self*)

quit (*self*)

_receive_alternatives_fetched (*self*, *db_map_data*)

_receive_scenarios_fetched (*self*, *db_map_data*)

_receive_scenarios_alternatives_fetched (*self*, *db_map_data*)

_receive_object_classes_fetched (*self*, *db_map_data*)

_receive_objects_fetched (*self*, *db_map_data*)

_receive_relationship_classes_fetched (*self*, *db_map_data*)

_receive_relationships_fetched (*self*, *db_map_data*)

_receive_entity_groups_fetched (*self*, *db_map_data*)

_receive_parameter_definitions_fetched (*self*, *db_map_data*)

_receive_parameter_definition_tags_fetched (*self*, *db_map_data*)

_receive_parameter_values_fetched (*self*, *db_map_data*)

_receive_parameter_value_lists_fetched (*self*, *db_map_data*)

_receive_parameter_tags_fetched (*self*, *db_map_data*)

_emit_finished_signal (*self*)

spinetoolbox.spine_db_manager

The SpineDBManager class

authors

P. Vennström (VTT) and M. Marin (KTH)

date 2.10.2019

Module Contents

Classes

SpineDBManager	Class to manage DBs within a project.

Functions

do_create_new_spine_database(url)	Creates a new spine database at the given url.

spinetoolbox.spine_db_manager.**do_create_new_spine_database**(*url*)
Creates a new spine database at the given url.

class spinetoolbox.spine_db_manager.**SpineDBManager**(*settings*, *logger*, *project*)
Bases: PySide2.QtCore.QObject

Class to manage DBs within a project.

TODO: Expand description, how it works, the cache, the signals, etc.

Initializes the instance.

Parameters

- **settings** (*QSettings*) – Toolbox settings
- **logger** (*LoggingInterface*) – a general, non-database-specific logger
- **project** (SpineToolboxProject) –

database_created

session_refreshed

session_committed

session_rolled_back

scenarios_added

alternatives_added

object_classes_added

objects_added

relationship_classes_added

relationships_added

entity_groups_added

parameter_definitions_added

parameter_values_added

parameter_value_lists_added

parameter_tags_added

scenarios_removed

alternatives_removed

object_classes_removed

objects_removed

relationship_classes_removed

relationships_removed

entity_groups_removed

parameter_definitions_removed

parameter_values_removed

parameter_value_lists_removed

parameter_tags_removed

scenarios_updated

alternatives_updated

object_classes_updated

objects_updated

relationship_classes_updated

relationships_updated

parameter_definitions_updated

parameter_values_updated

parameter_value_lists_updated

parameter_tags_updated

parameter_definition_tags_set

items_removed_from_cache

_scenario_alternatives_added

_scenario_alternatives_updated

_scenario_alternatives_removed

_parameter_definition_tags_added

_parameter_definition_tags_removed

_GROUP_SEP =

db_maps

db_editors

create_new_spine_database(*self*, *url*)

close_session(*self*, *url*)
Pops any db map on the given url and closes its connection.

Parameters **url** (*str*) –

close_all_sessions(*self*)
Closes connections to all database mappings.

show_data_store_form(*self*, *db_url_codenames*, *logger*)
Creates a new SpineDBEditor and shows it.

Parameters

- **db_url_codenames** (*dict*) – Mapping db urls to codenames.
- **logger** (*LoggingInterface*) – Where to log SpineDBAPIError

get_db_map(*self*, *url*, *logger*, *upgrade=False*, *codename=None*)
Returns a DiffDatabaseMapping instance from url if possible, None otherwise. If needed, asks the user to upgrade to the latest db version.

Parameters

- **url** (*str, URL*) –
- **logger** (*LoggingInterface*) – Where to log SpineDBAPIError
- **upgrade** (*bool, optional*) –
- **codename** (*str, NoneType, optional*) –

Returns DiffDatabaseMapping, NoneType

_do_get_db_map(*self*, *url*, *upgrade*, *codename*)
Returns a memorized DiffDatabaseMapping instance from url. Called by *get_db_map*.

Parameters

- **url** (*str, URL*) –
- **upgrade** (*bool, optional*) –
- **codename** (*str, NoneType, optional*) –

Returns DiffDatabaseMapping

register_listener(*self*, *ds_form*, **db_maps*)
Register given ds_form as listener for all given db_map's signals.

Parameters

- **ds_form** (SpineDBEditor) –
- **db_maps** (*DiffDatabaseMapping*) –

unregister_listener(*self*, *ds_form*, *db_map*)
Unregisters given ds_form from given db_map signals.

Parameters

- **ds_form** (SpineDBEditor) –
- **db_map** (*DiffDatabaseMapping*) –

set_logger_for_db_map(*self*, *logger*, *db_map*)

unset_logger_for_db_map(*self*, *db_map*)

fetch_db_maps_for_listener(*self*, *listener*, **db_maps*)
Fetches given db_map for given listener.

Parameters

- **listener** (SpineDBEditor) –
- ***db_maps** – database maps to fetch

_clean_up_fetcher(*self*, *fetcher*)
Cleans up things after fetcher has finished working.

Parameters **fetcher** (SpineDBFetcher) – the fetcher to clean up

_stop_fetchers(*self*)
Quits all fetchers and deletes them.

refresh_session(*self*, **db_maps*)

commit_session(*self*, **db_maps*, *rollback_if_no_msg=False*, *cookie=None*)
Commits the current session.

Parameters

- ***db_maps** – database maps to commit
- **rollback_if_no_msg** (*bool*) – if True, the commit will be rolled back if no commit message is provided
- **cookie** (*object, optional*) – a free form identifier which will be forwarded to `session_committed` signal

static _get_commit_msg(*db_map*)

rollback_session(*self*, **db_maps*)

static _get_rollback_confirmation(*db_map*)

_commit_db_map_session(*self*, *db_map*)

_rollback_db_map_session(*self*, *db_map*)

ok_to_close(*self*, *db_map*)
Prompts the user to commit or rollback changes to given database map.

Returns True if successfully committed or rolled back, False otherwise

Return type bool

connect_signals(*self*)
Connects signals.

error_msg(*self*, *db_map_error_log*)

cache_items(*self*, *item_type*, *db_map_data*)
Caches data for a given type. It works for both insert and update operations.

Parameters

- **item_type** (*str*) –
- **db_map_data** (*dict*) – lists of dictionary items keyed by DiffDatabaseMapping

update_icons(*self*, *db_map_data*)
Runs when object classes are added or updated. Setups icons for those classes. :param item_type: :type item_type: str :param db_map_data: lists of dictionary items keyed by DiffDatabaseMapping :type db_map_data: dict

entity_class_icon(*self*, *db_map*, *entity_type*, *entity_class_id*, *for_group=False*)
Returns an appropriate icon for a given entity_class.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **entity_type** (*str*) – either 'object_class' or 'relationship_class'
- **entity_class_id** (*int*) –

Returns QIcon

get_item(*self*, *db_map*, *item_type*, *id_*)
Returns the item of the given type in the given db map that has the given id, or an empty dict if not found.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **item_type** (*str*) –
- **id** (*int*) –

Returns dict

_pop_item(*self*, *db_map*, *item_type*, *id_*)

get_item_by_field(*self*, *db_map*, *item_type*, *field*, *value*)
Returns the first item of the given type in the given db map that has the given value for the given field Returns an empty dictionary if none found.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **item_type** (*str*) –
- **field** (*str*) –
- **value** –

Returns dict

get_items_by_field(*self*, *db_map*, *item_type*, *field*, *value*)
Returns all items of the given type in the given db map that have the given value for the given field. Returns an empty list if none found.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **item_type** (*str*) –
- **field** (*str*) –
- **value** –

Returns list

get_items(*self*, *db_map*, *item_type*)
Returns all the items of the given type in the given db map, or an empty list if none found.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **item_type** (*str*) –

Returns list

get_field(*self*, *db_map*, *item_type*, *id_*, *field*)

static _display_data(*parsed_data*)
Returns the value's database representation formatted for Qt.DisplayRole.

static _tool_tip_data(*parsed_data*)
Returns the value's database representation formatted for Qt.ToolTipRole.

get_value(*self*, *db_map*, *item_type*, *id_*, *role=Qt.DisplayRole*)
Returns the value or default value of a parameter.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **item_type** (*str*) – either "parameter_definition" or "parameter_value"
- **id** (*int*) – The parameter_value or definition id
- **role** (*int, optional*) –

static parse_value(*db_value*)

format_value(*self*, *parsed_value*, *role=Qt.DisplayRole*)
Formats the given value for the given role.

Parameters

- **parsed_value** (*object*) – A python object as returned by spinedb_api.from_database
- **role** (*int, optional*) –

get_value_indexes(*self*, *db_map*, *item_type*, *id_*)
Returns the value or default value indexes of a parameter.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **item_type** (*str*) – either "parameter_definition" or "parameter_value"
- **id** (*int*) – The parameter_value or definition id

get_value_index(*self*, *db_map*, *item_type*, *id_*, *index*, *role=Qt.DisplayRole*)
Returns the value or default value of a parameter for a given index.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **item_type** (*str*) – either "parameter_definition" or "parameter_value"
- **id** (*int*) – The parameter_value or definition id
- **index** – The index to retrieve
- **role** (*int, optional*) –

_split_and_parse_value_list(*self*, *item*)

get_value_list_item(*self*, *db_map*, *id_*, *index*, *role=Qt.DisplayRole*)
Returns one value item of a parameter_value_list.

Parameters

- **db_map** (*DiffDatabaseMapping*) –

- **id** (*int*) – The parameter_value_list id
- **index** (*int*) – The value item index
- **role** (*int, optional*) –

get_parameter_value_list(*self*, *db_map*, *id_*, *role=Qt.DisplayRole*)
Returns a parameter_value_list formatted for the given role.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **id** (*int*) – The parameter_value_list id
- **role** (*int, optional*) –

static get_db_items(*query, key=lambda x: x['id']*)

static _make_query(*db_map*, *sq_name*, *ids=()*)

get_alternatives(*self*, *db_map*, *ids=()*)
Returns alternatives from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_scenarios(*self*, *db_map*, *ids=()*)
Returns scenarios from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_scenario_alternatives(*self*, *db_map*, *ids=()*)
Returns scenario alternatives from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_object_classes(*self*, *db_map*, *ids=()*)
Returns object classes from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_objects(*self*, *db_map*, *ids=()*)
Returns objects from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_relationship_classes(*self*, *db_map*, *ids=()*)
Returns relationship classes from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_relationships(*self*, *db_map*, *ids=()*)
Returns relationships from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_entity_groups(*self*, *db_map*, *ids=()*)
Returns entity groups from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_object_parameter_definitions(*self*, *db_map*, *ids=()*)
Returns object parameter definitions from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_relationship_parameter_definitions(*self*, *db_map*, *ids=()*)
Returns relationship parameter definitions from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_parameter_definitions(*self*, *db_map*, *ids=()*)
Returns both object and relationship parameter definitions.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_parameter_definition_tags(*self*, *db_map*, *ids=()*)
Returns parameter definition tags.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_object_parameter_values(*self*, *db_map*, *ids=()*)
Returns object parameter values from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_relationship_parameter_values (*self*, *db_map*, *ids=()*)
Returns relationship parameter values from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_parameter_values (*self*, *db_map*, *ids=()*)
Returns both object and relationship parameter values.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_parameter_value_lists (*self*, *db_map*, *ids=()*)
Returns parameter_value lists from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_parameter_tags (*self*, *db_map*, *ids=()*)
Get parameter tags from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

import_data (*self*, *db_map_data*, *command_text='Import data'*)
Imports the given data into given db maps using the dedicated import functions from spinedb_api. Condenses all in a single command for undo/redo.

Parameters

- **db_map_data** (*dict(DiffDatabaseMapping, list)*) – Maps dbs to data to be passed as keyword arguments to *get_data_for_import*
- **command_text** (*str, optional*) – What to call the command that condenses the operation.

add_or_update_items (*self*, *db_map_data*, *method_name*, *get_method_name*, *signal_name*)
Adds or updates items in db.

Parameters

- **db_map_data** (*dict*) – lists of items to add or update keyed by DiffDatabaseMapping
- **method_name** (*str*) – attribute of DiffDatabaseMapping to call for performing the operation
- **get_method_name** (*str*) – attribute of SpineDBManager to call for getting affected items
- **signal_name** (*str*) – signal attribute of SpineDBManager to emit if successful

add_alternatives (*self*, *db_map_data*)
Adds alternatives to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_scenarios(*self*, *db_map_data*)
Adds scenarios to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_object_classes(*self*, *db_map_data*)
Adds object classes to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_objects(*self*, *db_map_data*)
Adds objects to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_relationship_classes(*self*, *db_map_data*)
Adds relationship classes to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_relationships(*self*, *db_map_data*)
Adds relationships to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_object_groups(*self*, *db_map_data*)
Adds object groups to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_entity_groups(*self*, *db_map_data*)
Adds entity groups to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_parameter_definitions(*self*, *db_map_data*)
Adds parameter definitions to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_parameter_values(*self*, *db_map_data*)
Adds parameter values to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_checked_parameter_values(*self*, *db_map_data*)
Adds parameter values in db without checking integrity.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_parameter_value_lists(*self*, *db_map_data*)
Adds parameter_value lists to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

add_parameter_tags(*self*, *db_map_data*)
Adds parameter tags to db.

Parameters **db_map_data** (*dict*) – lists of items to add keyed by DiffDatabaseMapping

update_alternatives(*self*, *db_map_data*)
Updates alternatives in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_scenarios(*self*, *db_map_data*)
Updates scenarios in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_object_classes(*self*, *db_map_data*)
Updates object classes in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_objects(*self*, *db_map_data*)
Updates objects in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_relationship_classes(*self*, *db_map_data*)
Updates relationship classes in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_relationships(*self*, *db_map_data*)
Updates relationships in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_parameter_definitions(*self*, *db_map_data*)
Updates parameter definitions in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_parameter_values(*self*, *db_map_data*)
Updates parameter values in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_checked_parameter_values(*self*, *db_map_data*)
Updates parameter values in db without checking integrity.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_expanded_parameter_values(*self*, *db_map_data*)
Updates expanded parameter values in db without checking integrity.

Parameters **db_map_data** (*dict*) – lists of expanded items to update keyed by DiffDatabaseMapping

update_parameter_value_lists(*self*, *db_map_data*)
Updates parameter_value lists in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

update_parameter_tags(*self*, *db_map_data*)
Updates parameter tags in db.

Parameters **db_map_data** (*dict*) – lists of items to update keyed by DiffDatabaseMapping

set_scenario_alternatives(*self*, *db_map_data*)
Sets scenario alternatives in db.

Parameters **db_map_data** (*dict*) – lists of items to set keyed by DiffDatabaseMapping

set_parameter_definition_tags(*self*, *db_map_data*)
Sets parameter_definition tags in db.

Parameters **db_map_data** (*dict*) – lists of items to set keyed by DiffDatabaseMapping

remove_items(*self*, *db_map_typed_ids*)

do_cascade_remove_items(*self*, *db_map_typed_ids*)

do_remove_items(*self*, *db_map_typed_ids*)
Removes items from database.

Parameters **db_map_typed_ids** (*dict*) – lists of items to remove, keyed by item type (str), keyed by DiffDatabaseMapping

_pop_item(*self*, *db_map*, *item_type*, *id_*)

uncache_items(*self*, *db_map_typed_ids*)
Removes data from cache.

Parameters **db_map_typed_ids** –

static db_map_ids(*db_map_data*)

static db_map_class_ids(*db_map_data*)

_refresh_scenario_alternatives(*self*, *db_map_data*)
Refreshes cached scenarios when updating scenario alternatives.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

_refresh_parameter_definitions_by_tag(*self*, *db_map_data*)
Refreshes cached parameter definitions when updating parameter tags.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_relationship_classes(*self*, *db_map_data*)
Refreshes cached relationship classes when updating object classes.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_relationships_by_object(*self*, *db_map_data*)
Refreshed cached relationships in cascade when updating objects.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_parameter_definitions(*self*, *db_map_data*)
Refreshes cached parameter definitions in cascade when updating entity classes.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_parameter_definitions_by_value_list(*self*, *db_map_data*)
Refreshes cached parameter definitions when updating parameter_value lists.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_parameter_values_by_entity_class(*self*, *db_map_data*)
Refreshes cached parameter values in cascade when updating entity classes.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_parameter_values_by_entity(*self*, *db_map_data*)
Refreshes cached parameter values in cascade when updating entities.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_parameter_values_by_alternative(*self*, *db_map_data*)
Refreshes cached parameter values in cascade when updating alternatives.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_parameter_values_by_definition(*self*, *db_map_data*)
Refreshes cached parameter values in cascade when updating parameter definitions.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

cascade_refresh_parameter_definitions_by_tag(*self*, *db_map_data*)
Refreshes cached parameter definitions when updating parameter tags.

Parameters **db_map_data** (*dict*) – lists of updated items keyed by DiffDatabaseMapping

find_cascading_relationship_classes(*self*, *db_map_ids*)
Finds and returns cascading relationship classes for the given object_class ids.

find_cascading_entities(*self*, *db_map_ids*, *item_type*)
Finds and returns cascading entities for the given entity_class ids.

find_cascading_relationships(*self*, *db_map_ids*)
Finds and returns cascading relationships for the given object ids.

find_cascading_parameter_data(*self*, *db_map_ids*, *item_type*)
Finds and returns cascading parameter definitions or values for the given entity_class ids.

find_cascading_parameter_definitions_by_value_list(*self*, *db_map_ids*)
Finds and returns cascading parameter definitions for the given parameter_value_list ids.

find_cascading_parameter_definitions_by_tag(*self*, *db_map_ids*)
Finds and returns cascading parameter definitions for the given parameter_tag ids.

find_cascading_parameter_values_by_entity(*self*, *db_map_ids*)
Finds and returns cascading parameter values for the given entity ids.

find_cascading_parameter_values_by_definition(*self*, *db_map_ids*)
Finds and returns cascading parameter values for the given parameter_definition ids.

find_groups_by_entity(*self*, *db_map_ids*)
Finds and returns groups for the given entity ids.

find_groups_by_member(*self*, *db_map_ids*)
Finds and returns groups for the given entity ids.

find_cascading_parameter_values_by_alternative(*self*, *db_map_ids*)
Finds and returns cascading parameter values for the given parameter alternative ids.

find_cascading_alternative_scenarios_by_alternative(*self*, *db_map_ids*)
Finds and returns cascading parameter values for the given parameter alternative ids.

find_cascading_alternative_scenarios_by_scenario(*self*, *db_map_ids*)
Finds and returns cascading parameter values for the given parameter alternative ids.

spinetoolbox.spine_db_parcel

SpineDBParcel class.

authors

M. Marin (KTH)

date 10.5.2020

Module Contents

Classes

SpineDBParcel	A class to create parcels of data from a Spine db.

class spinetoolbox.spine_db_parcel.**SpineDBParcel**(*db_mngr*)

A class to create parcels of data from a Spine db. Mainly intended for the *Export selection* action in the Spine db editor.

The strategy is the following:

- ***_push* methods (with a leading underscore) push items with everything they need to live in a standalone db.** These are private methods.
- ***push* methods (no leading underscore) call the *_push* methods to get away with pushing some specific content.** These are public methods.

Initializes the parcel object.

Parameters **db_mngr** (SpineDBManager) –

data

_get_fields(*self*, *db_map*, *item_type*, *field*, *ids*)

_push_object_class_ids(*self*, *db_map_ids*)
Pushes object_class ids.

_push_relationship_class_ids(*self*, *db_map_ids*)
Pushes relationship_class ids.

_push_object_ids(*self*, *db_map_ids*)
Pushes object ids.

_push_relationship_ids(*self*, *db_map_ids*)
Pushes relationship ids.

_push_parameter_value_list_ids(*self*, *db_map_ids*)
Pushes parameter_value_list ids.

_push_parameter_definition_ids(*self*, *db_map_ids*, *entity_type*)
Pushes parameter_definition ids.

_push_parameter_value_ids(*self*, *db_map_ids*, *entity_type*)
Pushes parameter_value ids.

_push_object_group_ids(*self*, *db_map_ids*)
Pushes object group ids.

_push_alternative_ids(*self*, *db_map_ids*)
Pushes alternative ids.

_push_scenario_ids(*self*, *db_map_ids*)
Pushes scenario ids.

_push_scenario_alternative_ids(*self*, *db_map_ids*)
Pushes scenario_alternative ids.

push_object_class_ids(*self*, *db_map_ids*)
Pushes parameter definitions associated with given object classes. This essentially pushes the object classes and their parameter definitions.

push_relationship_class_ids(*self*, *db_map_ids*)
Pushes parameter definitions associated with given relationship classes. This essentially pushes the relationships classes, their parameter definitions, and their member object classes.

push_object_ids (*self*, *db_map_ids*)
Pushes parameter values associated with objects and with any relationships involving those objects. This essentially pushes objects, their relationships, all the parameter values, and all the necessary classes, definitions, and lists.

push_relationship_ids (*self*, *db_map_ids*)
Pushes parameter values associated with relationships. This essentially pushes relationships, their parameter values, and all the necessary classes, definitions, and lists.

push_inside_object_ids (*self*, *db_map_ids*)
Pushes object ids, cascading relationship ids, and the associated parameter values, but not any entity classes or parameter definitions. Mainly intended for the *Duplicate object* action.

push_inside_relationship_ids (*self*, *db_map_ids*)
Pushes relationship ids, and the associated parameter values, but not any entity classes or parameter definitions.

push_inside_parameter_value_ids (*self*, *db_map_ids*, *entity_type*)
Pushes parameter_value ids.

_setdefault (*self*, *db_map*)

spinetoolbox.spine_db_signaller

Spine DB Signaller class.

authors

M. Marin (KTH)

date 31.10.2019

Module Contents

Classes

SpineDBSignaller	Handles signals from DB manager and channels them to listeners.

class spinetoolbox.spine_db_signaller.**SpineDBSignaller**(*db_mngr*)
Bases: PySide2.QtCore.QObject

Handles signals from DB manager and channels them to listeners.

Initializes the signaler object.

Parameters **db_mngr** (SpineDBManager) –

add_db_map_listener (*self*, *db_map*, *listener*)
Adds listener for given db_map.

remove_db_map_listener (*self*, *db_map*, *listener*)
Removes db_map from the the maps listener listens to.

db_map_listeners (*self*, *db_map*)

connect_signals (*self*)
Connects signals.

static _shared_db_map_data(*db_map_data*, *db_maps*)

receive_scenarios_added(*self*, *db_map_data*)

receive_alternatives_added(*self*, *db_map_data*)

receive_object_classes_added(*self*, *db_map_data*)

receive_objects_added(*self*, *db_map_data*)

receive_relationship_classes_added(*self*, *db_map_data*)

receive_relationships_added(*self*, *db_map_data*)

receive_entity_groups_added(*self*, *db_map_data*)

receive_parameter_definitions_added(*self*, *db_map_data*)

receive_parameter_values_added(*self*, *db_map_data*)

receive_parameter_value_lists_added(*self*, *db_map_data*)

receive_parameter_tags_added(*self*, *db_map_data*)

receive_scenarios_updated(*self*, *db_map_data*)

receive_alternatives_updated(*self*, *db_map_data*)

receive_object_classes_updated(*self*, *db_map_data*)

receive_objects_updated(*self*, *db_map_data*)

receive_relationship_classes_updated(*self*, *db_map_data*)

receive_relationships_updated(*self*, *db_map_data*)

receive_parameter_definitions_updated(*self*, *db_map_data*)

receive_parameter_values_updated(*self*, *db_map_data*)

receive_parameter_value_lists_updated(*self*, *db_map_data*)

receive_parameter_tags_updated(*self*, *db_map_data*)

receive_parameter_definition_tags_set(*self*, *db_map_data*)

receive_scenarios_removed(*self*, *db_map_data*)

receive_alternatives_removed(*self*, *db_map_data*)

receive_object_classes_removed(*self*, *db_map_data*)

receive_objects_removed(*self*, *db_map_data*)

receive_relationship_classes_removed(*self*, *db_map_data*)

receive_relationships_removed(*self*, *db_map_data*)

receive_entity_groups_removed(*self*, *db_map_data*)

receive_parameter_definitions_removed(*self*, *db_map_data*)

receive_parameter_values_removed(*self*, *db_map_data*)

receive_parameter_value_lists_removed(*self*, *db_map_data*)

receive_parameter_tags_removed(*self*, *db_map_data*)

receive_session_refreshed(*self*, *db_maps*)

receive_session_committed(*self*, *db_maps*, *cookie*)

receive_session_rolled_back(*self*, *db_maps*)

spinetoolbox.spinedb_api_version_check

Contains the spinedb_api_version_check function.

This module should import as few things as possible to avoid accidentally importing anything from spinedb_api that is not available in the current spinedb_api version.

authors

A. Soininen (VTT)

date 30.3.2020

Module Contents

Functions

spinedb_api_version_check()	Check if spinedb_api is the correct version and explain how to upgrade if it is not.

spinetoolbox.spinedb_api_version_check.**spinedb_api_version_check**()
Check if spinedb_api is the correct version and explain how to upgrade if it is not.

spinetoolbox.ui_main

Contains ToolboxUI class.

author

P. Savolainen (VTT)

date 14.12.2017

Module Contents

Classes

ToolboxUI	Class for application main GUI functions.

class spinetoolbox.ui_main.**ToolboxUI**
Bases: PySide2.QtWidgets.QMainWindow

Class for application main GUI functions.

Initializes application and main window.

msg

msg_success

msg_error

msg_warning

msg_proc

information_box

error_box

msg_proc_error

specification_model_changed

connect_signals (*self*)
Connect signals.

update_window_modified (*self*, *clean*)
Updates window modified status and save actions depending on the state of the undo stack.

parse_project_item_modules (*self*)
Collects attributes from project item modules into a dict. This dict is then used to perform all project item related tasks.

init_project_item_factory_model (*self*)

parse_assistant_modules (*self*)
Makes actions to run assistants from assistant modules.

show_assistant (*self*, *module*, *action*)
Creates and shows the assistant for the given module. Disables the given action while the assistant is shown, enables the action back when the assistant is destroyed. This is to make sure we don't open the same assistant twice.

set_work_directory (*self*, *new_work_dir=None*)
Creates a work directory if it does not exist or changes the current work directory to given.

Parameters **new_work_dir** (*str, optional*) – If given, changes the work directory to given and creates the directory if it does not exist.

project (*self*)
Returns current project or None if no project open.

qsettings (*self*)
Returns application preferences object.

update_window_title (*self*)

init_project (*self*, *project_dir*)
Initializes project at application start-up. Opens the last project that was open when app was closed (if enabled in Settings) or starts the app without a project.

new_project (*self*)
Opens a file dialog where user can select a directory where a project is created. Pops up a question box if selected directory is not empty or if it already contains a Spine Toolbox project. Initial project name is the directory name.

create_project (*self*, *name*, *description*, *location*)
Creates new project and sets it active.

Parameters

- **name** (*str*) – Project name
- **description** (*str*) – Project description
- **location** (*str*) – Path to project directory

open_project(*self*, *load_dir=None*, *clear_logs=True*)
Opens project from a selected or given directory.

Parameters

- **load_dir** (*str*) – Path to project base directory. If default value is used,
- **file explorer dialog is opened where the user can select the** (*a*) –
- **to open.** (*project*) –
- **clear_logs** (*bool*) – True clears Event and Process Log, False does not

Returns True when opening the project succeeded, False otherwise

Return type bool

restore_project(*self*, *project_info*, *project_dir*, *clear_logs*)
Initializes UI, Creates project, models, connections, etc., when opening a project.

Parameters

- **project_info** (*dict*) – Project information dictionary
- **project_dir** (*str*) – Project directory
- **clear_logs** (*bool*) – True clears Event and Process Log, False does not

Returns True when restoring project succeeded, False otherwise

Return type bool

show_recent_projects_menu(*self*)
Updates and sets up the recent projects menu to File-Open recent menu item.

save_project(*self*)
Save project.

save_project_as(*self*)
Ask user for a new project name and save. Creates a duplicate of the open project.

upgrade_project(*self*, *checked=False*)
Upgrades an old style project (.proj file) to a new directory based Spine Toolbox project. Note that this method can be removed when we no longer want to support upgrading .proj projects. Project upgrading should happen later automatically when opening a project.

init_project_item_model(*self*)
Initializes project item model. Create root and category items and add them to the model.

init_specification_model(*self*, *specification_paths*)
Initializes Tool specification model.

Parameters **specification_paths** (*list*) – List of tool definition file paths used in this project

load_specification_from_file(*self*, *def_path*)
Returns an Item specification from a definition file.

Parameters **def_path** (*str*) – Path of the specification definition file

Returns item specification or None if reading the file failed

Return type *ProjectItemSpecification*

load_specification(*self*, *definition*, *def_path*)
Returns a Tool specification from a definition dictionary.

Parameters

- **definition** (*dict*) – Dictionary with the tool definition
- **def_path** (*str*) – Path of the specification definition file

Returns ToolSpecification, NoneType

restore_ui (*self*)
Restore UI state from previous session.

clear_ui (*self*)
Clean UI to make room for a new or opened project.

undo_critical_commands (*self*)
Undoes critical commands in the undo stack.

overwrite_check (*self*, *project_dir*)
Checks if given directory is a project directory and/or empty And asks the user what to do in that case.

Parameters **project_dir** (*str*) – Abs. path to a directory

Returns True if user wants to overwrite an existing project or if the directory is not empty and the user wants to make it into a Spine Toolbox project directory anyway. False if user cancels the action.

Return type bool

item_selection_changed (*self*, *selected*, *deselected*)
Synchronize selection with scene. Check if only one item is selected and make it the active item: disconnect signals of previous active item, connect signals of current active item and show correct properties tab for the latter.

activate_no_selection_tab (*self*)
Shows 'No Selection' tab.

activate_item_tab (*self*, *item*)
Shows project item properties tab according to item type. Note: Does not work if a category item is given as argument.

Parameters **item** (ProjectItem) – Instance of a project item

import_specification (*self*)
Opens a file dialog where the user can select an existing specification definition file (.json). If file is valid, calls add_specification().

add_specification (*self*, *specification*)
Pushes a new AddSpecificationCommand to the undo stack.

do_add_specification (*self*, *specification*, *row=None*)
Adds a ProjectItemSpecification instance to project.

Parameters **specification** (ProjectItemSpecification) – specification that is added to project

update_specification (*self*, *row*, *specification*)
Pushes a new UpdateSpecificationCommand to the undo stack.

do_update_specification (*self*, *row*, *specification*)
Updates a specification and refreshes all items that use it.

Parameters

- **row** (*int*) – Row of tool specification in ProjectItemSpecFactoryModel

- **specification** (ProjectItemSpecification) – An updated specification

undo_update_specification(*self*, *row*)
Reverts a specification update and refreshes all items that use it.

Parameters **row** (*int*) – Row of tool specification in ProjectItemSpecFactoryModel

_get_specific_items(*self*, *specification*)
Yields project items with given specification.

Parameters **specification** (ProjectItemSpecification) –

remove_selected_specification(*self*, *checked=False*)
Removes specification selected in QListView.

remove_specification(*self*, *row*, *ask_verification=True*)

do_remove_specification(*self*, *row*, *ask_verification=True*)
Removes specification from ProjectItemSpecFactoryModel. Removes also specifications from all items that use this specification.

Parameters

- **row** (*int*) – Row in ProjectItemSpecFactoryModel
- **ask_verification** (*bool*) – If True, displays a dialog box asking user to verify the removal

remove_all_items(*self*)
Removes all items from project. Slot for Remove All button.

open_anchor(*self*, *qurl*)
Open file explorer in the directory given in qurl.

Parameters **qurl** (*QUrl*) – Directory path or a file to open

show_specification_context_menu(*self*, *pos*)
Context menu for item specifications.

Parameters **pos** (*QPoint*) – Mouse position

edit_specification(*self*, *index*)
Open the tool specification widget for editing an existing tool specification.

Parameters **index** (*QModelIndex*) – Index of the item (from double-click or contex menu signal)

open_specification_file(*self*, *index*)
Open the specification definition file in the default (.json) text-editor.

Parameters **index** (*QModelIndex*) – Index of the item

export_as_graphml(*self*)
Exports all DAGs in project to separate GraphML files.

_handle_zoom_minus_pressed(*self*)
Slot for handling case when '-' button in menu is pressed.

_handle_zoom_plus_pressed(*self*)
Slot for handling case when '+' button in menu is pressed.

_handle_zoom_reset_pressed(*self*)
Slot for handling case when 'reset zoom' button in menu is pressed.

setup_zoom_widget_action(*self*)
Setups zoom widget action in view menu.

restore_dock_widgets(*self*)
Dock all floating and or hidden QDockWidgets back to the main window.

set_debug_qactions(*self*)
Set shortcuts for QActions that may be needed in debugging.

add_menu_actions(*self*)
Add extra actions to View menu.

toggle_properties_tabbar_visibility(*self*)
Shows or hides the tab bar in properties dock widget. For debugging purposes.

update_datetime(*self*)
Returns a boolean, which determines whether date and time is prepended to every Event Log message.

add_message(*self*, *msg*)
Append regular message to Event Log.

Parameters **msg** (*str*) – String written to QTextBrowser

add_success_message(*self*, *msg*)
Append message with green text color to Event Log.

Parameters **msg** (*str*) – String written to QTextBrowser

add_error_message(*self*, *msg*)
Append message with red color to Event Log.

Parameters **msg** (*str*) – String written to QTextBrowser

add_warning_message(*self*, *msg*)
Append message with yellow (golden) color to Event Log.

Parameters **msg** (*str*) – String written to QTextBrowser

add_process_message(*self*, *msg*)
Writes message from stdout to process output QTextBrowser.

Parameters **msg** (*str*) – String written to QTextBrowser

add_process_error_message(*self*, *msg*)
Writes message from stderr to process output QTextBrowser.

Parameters **msg** (*str*) – String written to QTextBrowser

show_add_project_item_form(*self*, *item_type*, *x=0*, *y=0*, *spec=''*)
Show add project item widget.

show_specification_form(*self*, *item_type*, *specification=None*)
Show specification widget.

show_settings(*self*)
Show Settings widget.

show_about(*self*)
Show About Spine Toolbox form.

show_user_guide(*self*)
Open Spine Toolbox User Guide index page. First tries to open the local docs but if they are missing, opens the docs from readthedocs.org.

show_getting_started_guide(*self*)
Open Spine Toolbox Getting Started HTML page in browser.

show_item_context_menu(*self*, *pos*)
Context menu for project items listed in the project QTreeView.

Parameters **pos** (*QPoint*) – Mouse position

show_item_image_context_menu(*self*, *pos*, *name*)
Context menu for project item images on the QGraphicsView.

Parameters

- **pos** (*QPoint*) – Mouse position
- **name** (*str*) – The name of the concerned item

show_project_item_context_menu(*self*, *pos*, *ind*)
Create and show project item context menu.

Parameters

- **pos** (*QPoint*) – Mouse position
- **ind** (*QModelIndex*) – Index of concerned item

show_link_context_menu(*self*, *pos*, *link*)
Context menu for connection links.

Parameters

- **pos** (*QPoint*) – Mouse position
- **link** (Link *(QGraphicsPathItem)*) – The concerned link

tear_down_items(*self*)
Calls the tear_down method on all project items, so they can clean up their mess if needed.

_tasks_before_exit(*self*)
Returns a list of tasks to perform before exiting the application.

Possible tasks are:

- *"prompt exit"*: prompt user if quitting is really desired
- *"prompt save"*: prompt user if project should be saved before quitting
- *"save"*: save project before quitting

Returns a list containing zero or more tasks

_perform_pre_exit_tasks(*self*)
Prompts user to confirm quitting and saves the project if necessary.

Returns True if exit should proceed, False if the process was cancelled

_confirm_exit(*self*)
Confirms exiting from user.

Returns True if exit should proceed, False if user cancelled

_confirm_save_and_exit(*self*)
Confirms exit from user and saves the project if requested.

Returns True if exiting should proceed, False if user cancelled

remove_path_from_recent_projects(*self*, *p*)
Removes entry that contains given path from the recent project files list in QSettings.

Parameters **p** (*str*) – Full path to a project directory

update_recent_projects(*self*)
Adds a new entry to QSettings variable that remembers the five most recent project paths.

closeEvent(*self*, *event*)
Method for handling application exit.

Parameters **event** (*QCloseEvent*) – PySide2 event

_serialize_selected_items(*self*)
Serializes selected project items into a dictionary.

The serialization protocol tries to imitate the format in which projects are saved. The format of the dictionary is following: *{"item_category_1": [{"name": "item_1_name", ... }, ...], ... }*

Returns a dict containing serialized version of selected project items

_deserialized_item_position_shifts(*self*, *serialized_items*)
Calculates horizontal and vertical shifts for project items being deserialized.

If the mouse cursor is on the Design view we try to place the items unders the cursor. Otherwise the items will get a small shift so they don't overlap a possible item below. In case the items don't fit the scene rect we clamp their coordinates within it.

Parameters **serialized_items** (*dict*) – a dictionary of serialized items being deserialized

Returns a tuple of (horizontal shift, vertical shift) in scene's coordinates

static _set_deserialized_item_position(*item_dict*, *shift_x*, *shift_y*, *scene_rect*)
Moves item's position by shift_x and shift_y while keeping it within the limits of scene_rect.

_deserialize_items(*self*, *serialized_items*)
Deserializes project items from a dictionary and adds them to the current project.

Parameters **serialized_items** (*dict*) – serialized project items

project_item_to_clipboard(*self*)
Copies the selected project items to system's clipboard.

project_item_from_clipboard(*self*)
Adds project items in system's clipboard to the current project.

duplicate_project_item(*self*)
Duplicates the selected project items.

propose_item_name(*self*, *prefix*)
Proposes a name for a project item.

The format is *prefix_xx* where *xx* is a counter value [01..99].

Parameters **prefix** (*str*) – a prefix for the name

Returns a name string

_item_edit_actions(*self*)
Creates project item edit actions (copy, paste, duplicate) and adds them to proper places.

_scroll_event_log_to_end(*self*)

_show_message_box(*self*, *title*, *message*)
Shows an information message box.

_show_error_box(*self*, *title*, *message*)

_connect_project_signals(*self*)
Connects signals emitted by project.

spinetoolbox.version

Version info for Spine Toolbox package. Inspired by python sys.version and sys.version_info.

author

P. Savolainen (VTT)

date 8.1.2020

Module Contents

Classes

VersionInfo	A class for a named tuple containing the five components of the version number: major, minor,

class spinetoolbox.version.**VersionInfo**

Bases: typing.NamedTuple

A class for a named tuple containing the five components of the version number: major, minor, micro, releaselevel, and serial. All values except releaselevel are integers; the release level is 'alpha', 'beta', 'candidate', or 'final'.

Create and return a new object. See help(type) for accurate signature.

major :int

minor :int

micro :int

releaselevel :str

serial :int

spinetoolbox.version.**major = 0**

spinetoolbox.version.**minor = 5**

spinetoolbox.version.**micro = 0**

spinetoolbox.version.**releaselevel = beta**

spinetoolbox.version.**serial = 0**

spinetoolbox.version.**__version_info__**

spinetoolbox.version.**__version__**

17.1.3 Package Contents

spinetoolbox.**__version__**

spinetoolbox.**__version_info__**

CHAPTER 18

Indices and tables

- genindex
- modindex
- search

Bibliography

[CB14] Chris Beams. 2014. 'How to Write a Git Commit Message.' https://chris.beams.io/posts/git-commit/

[JF18] Jeff Forcier. 2018. 'Contributing to Open Source Projects.' https://contribution-guide-org.readthedocs.io/

Python Module Index

S

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z