Spine Toolbox Documentation

Release 0.4.0-final.0

Pekka Savolainen, Manuel Marin, Erkka Rinne

Apr 03, 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.

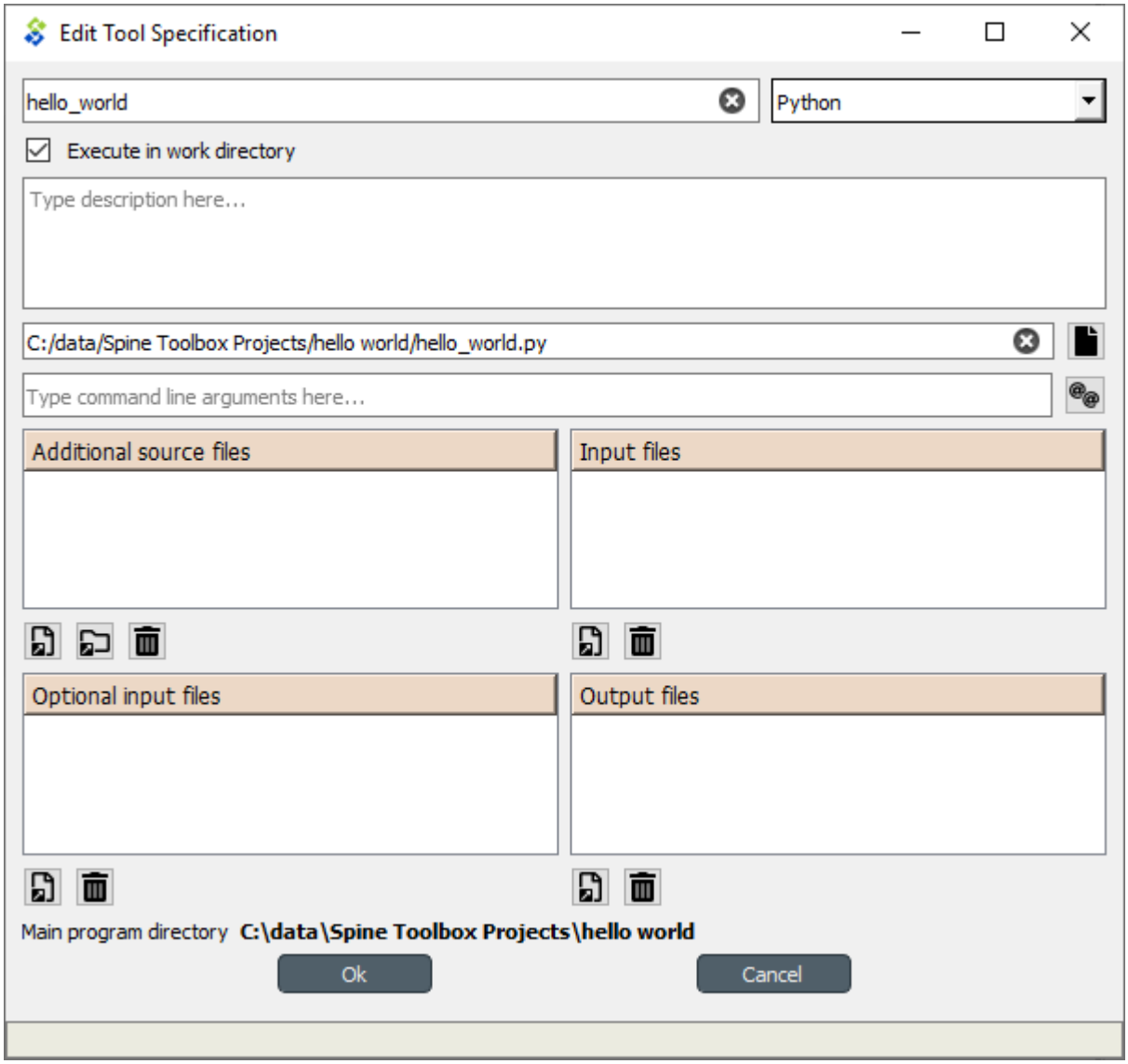
1.3 Creating a Tool specification

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.

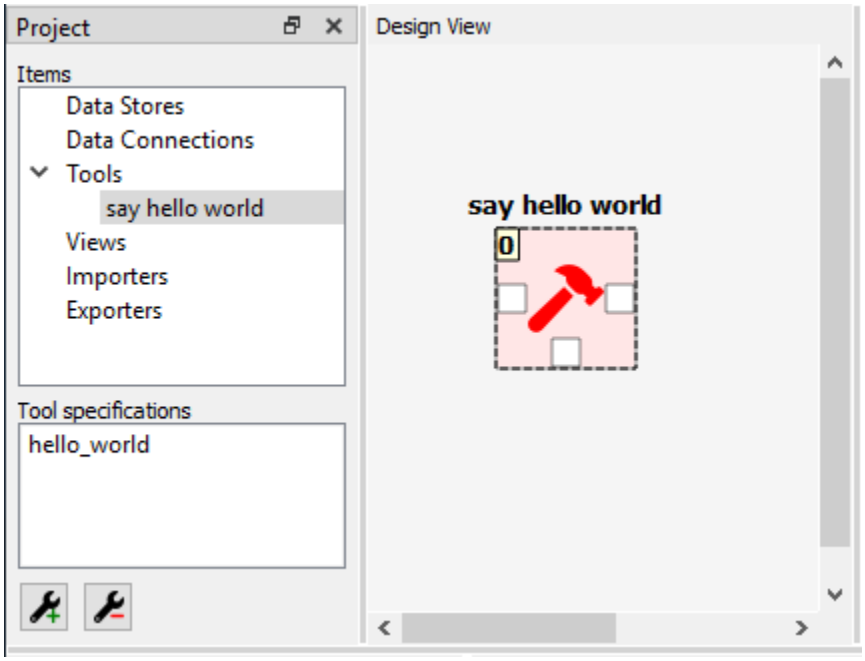
Now, whenever *hello_world.py* is executed, the sentence 'Hello, World!' will be printed to the standard output.

1.4 Adding a Tool item to the project

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

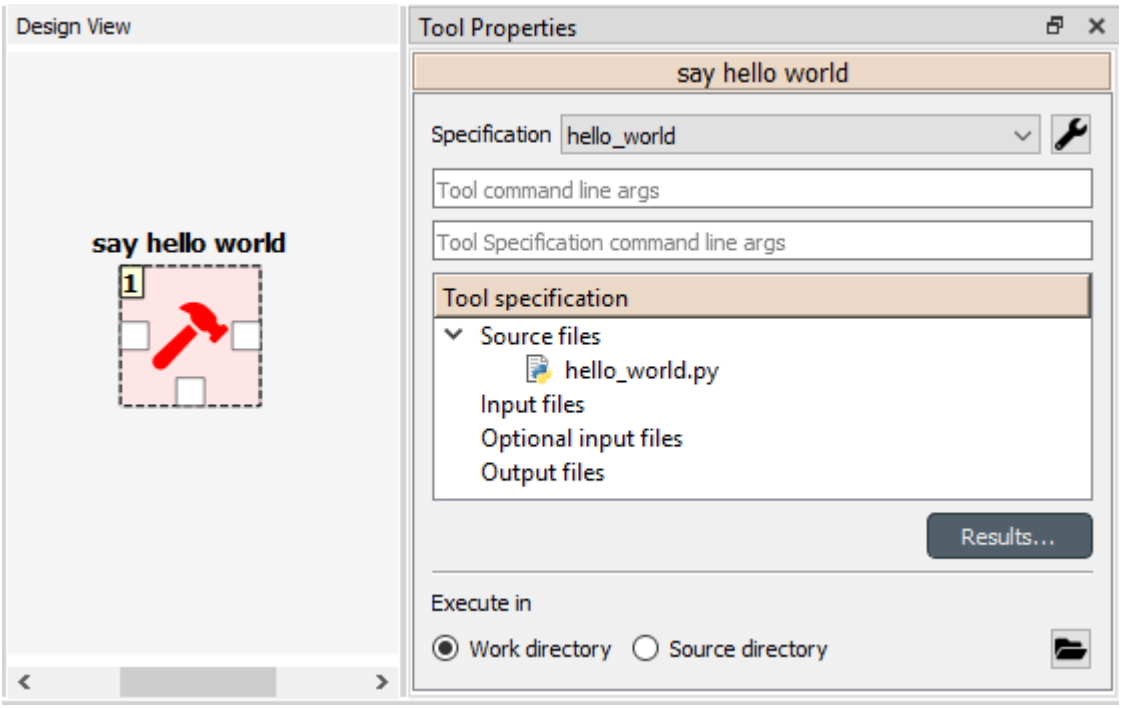
Let's add a Tool item to our project, so that we're able to run the Tool specification we created above. To add a Tool item drag-and-drop the Tool icon () from the *Drag & Drop Icon* toolbar onto the *Design View*.

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:



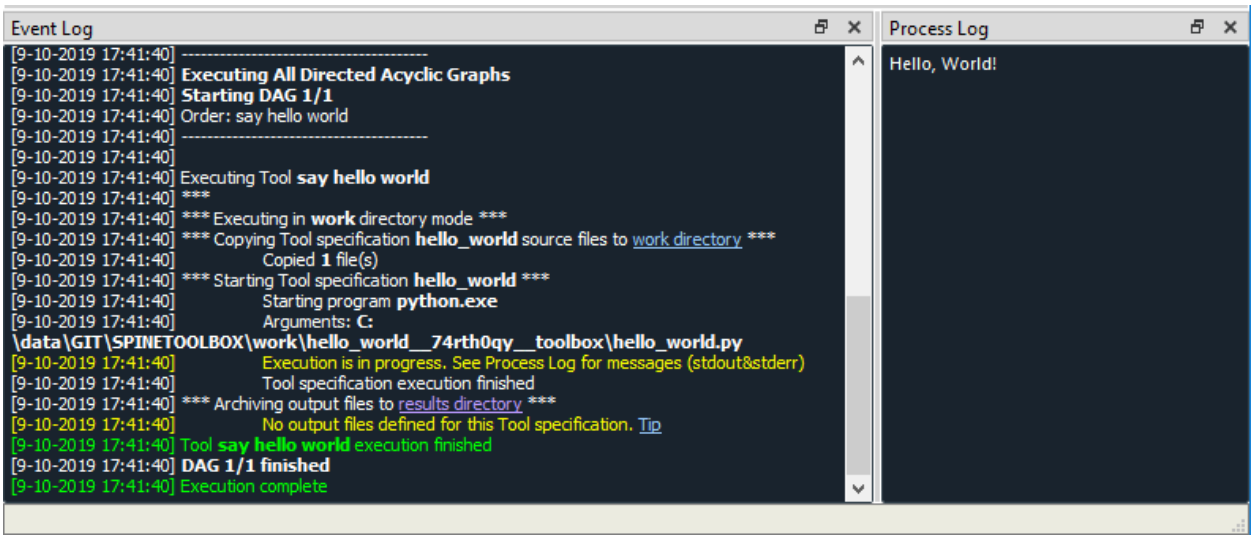
1.5 Executing a Tool

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.

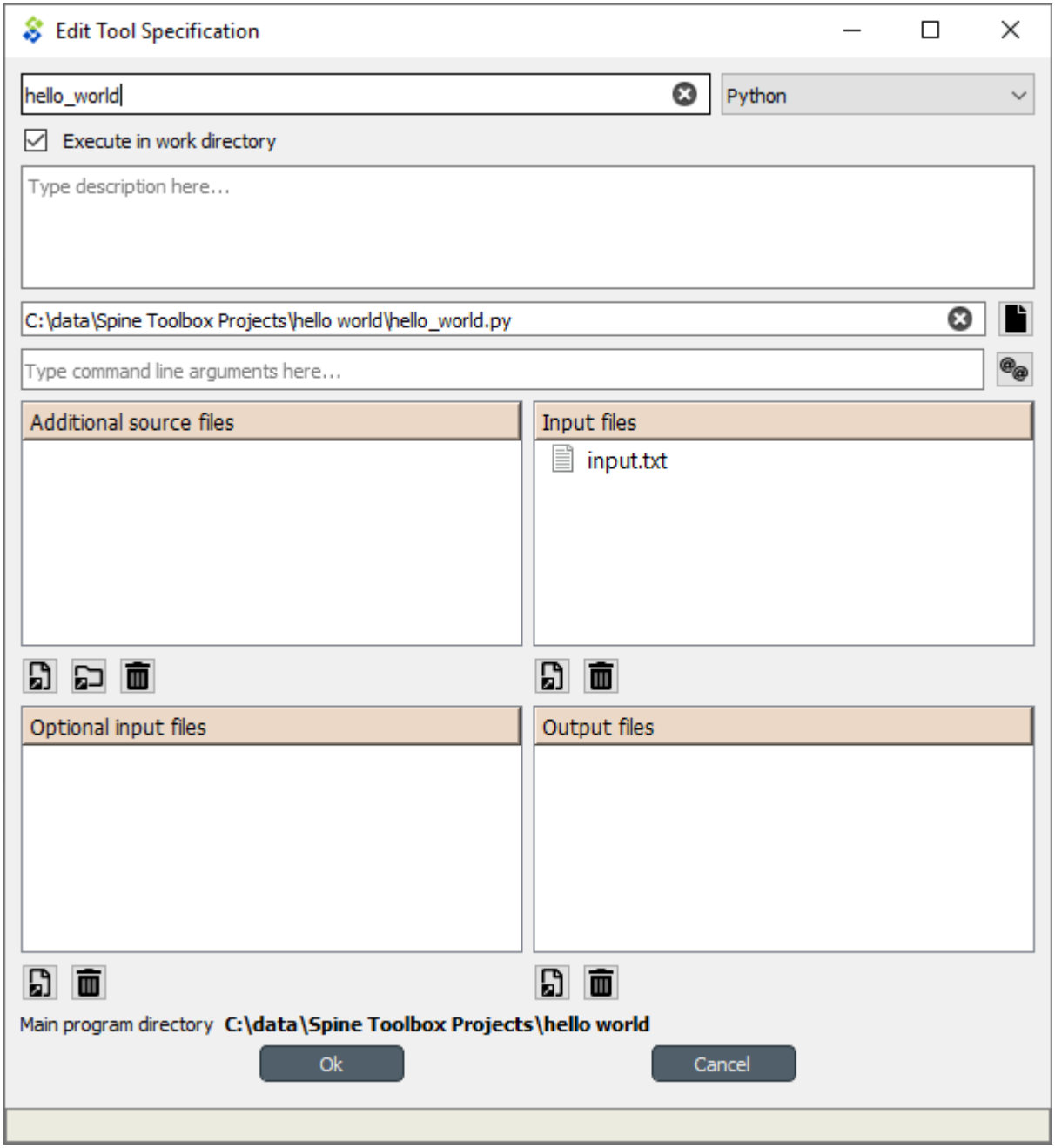
1.6 Editing a Tool specification

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.

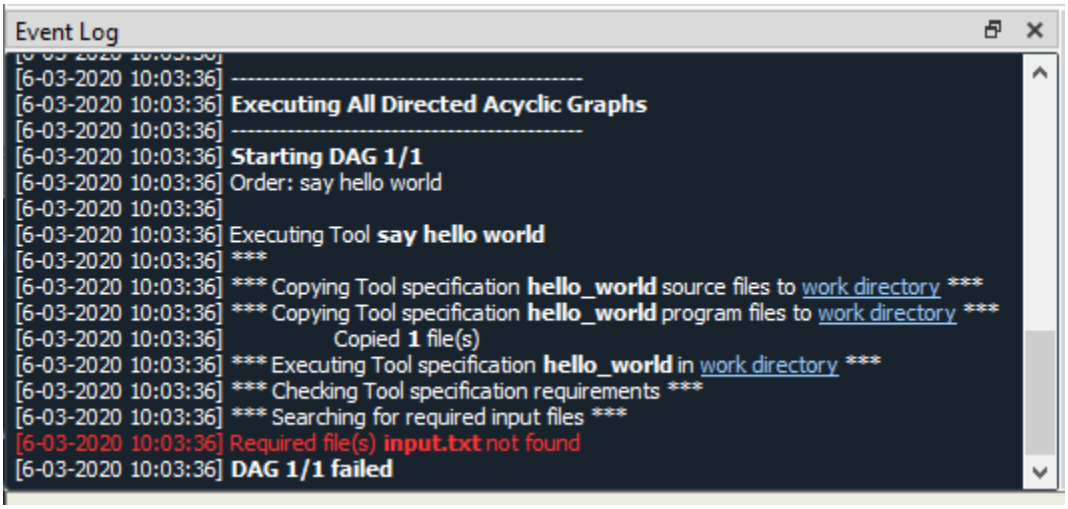
Delete whatever it's in the file and enter the following instead:

```
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:

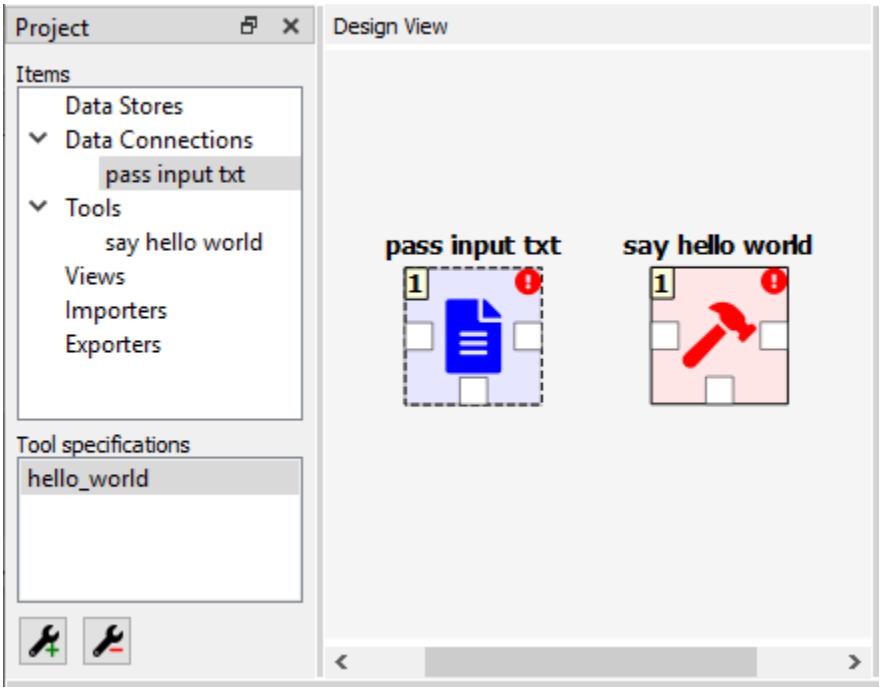


1.7 Adding a Data Connection item to the project

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.

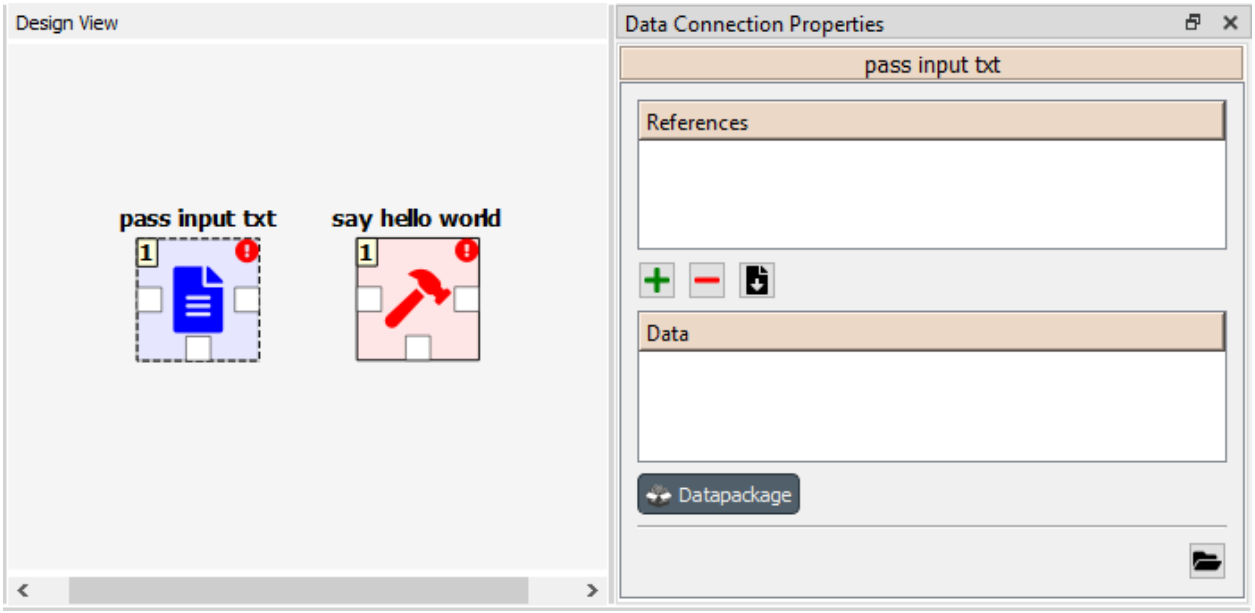
Let's add a Data Connection item to our project, so that we're able to pass the file 'input.txt' to 'say hello world'. To add a Data Connection item drag-and-drop the Data Connection icon () from the main window toolbar onto the *Design View*.

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:



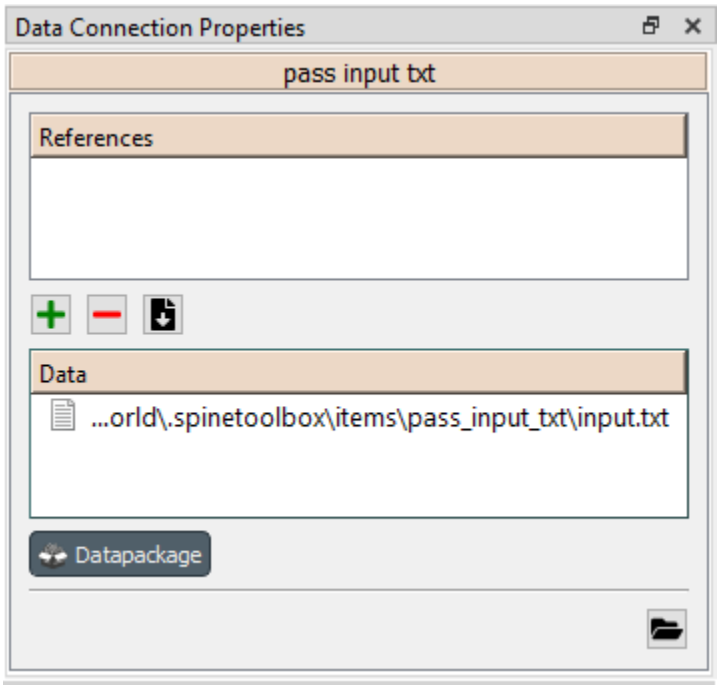
1.8 Adding data files to a Data Connection

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:

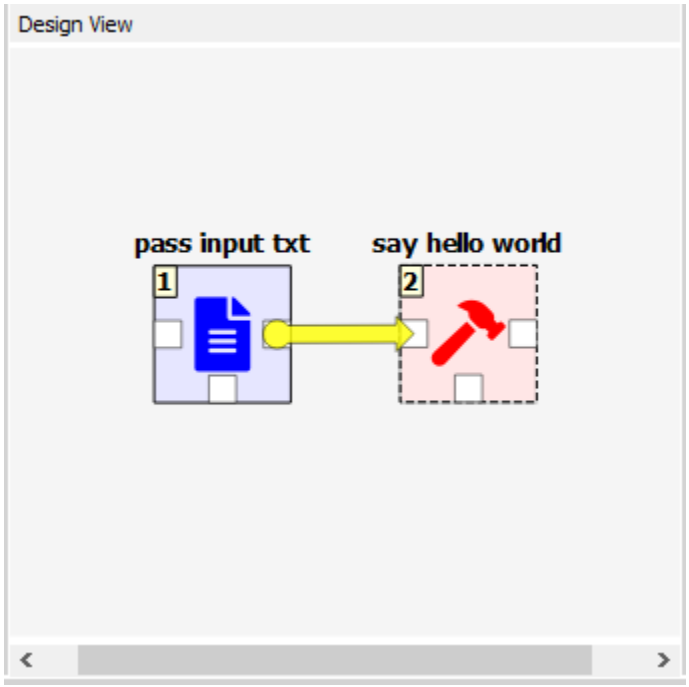
```
Hello again, World!
```

Save the file.

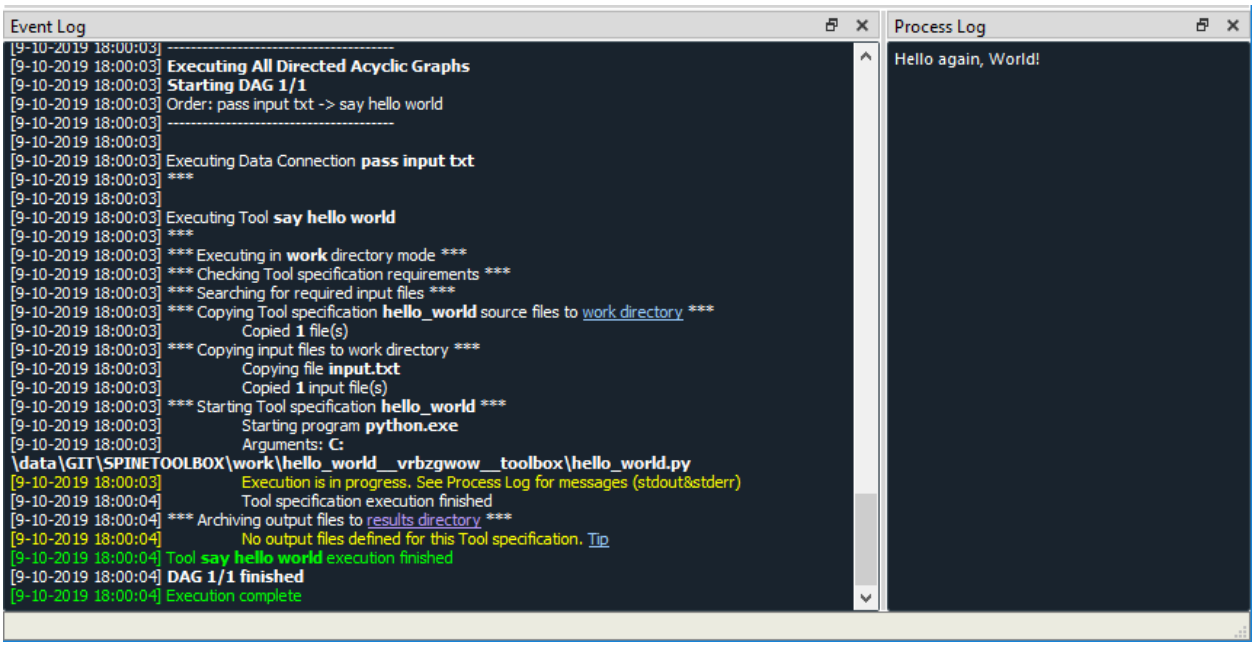
1.9 Connecting project items

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 SpineModel.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.:

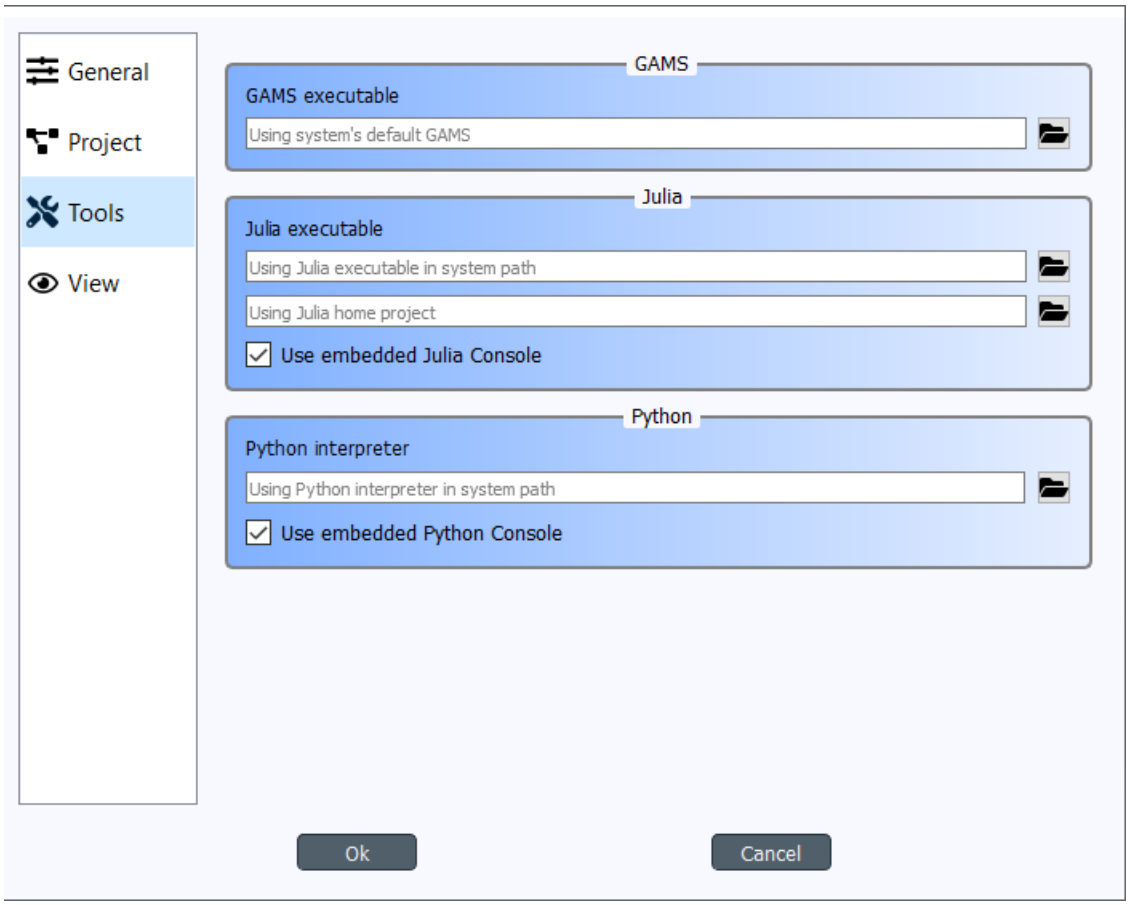
```
Couldn't determine Python version. Please check the Python interpreter option in␣
↪Settings.
```

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.

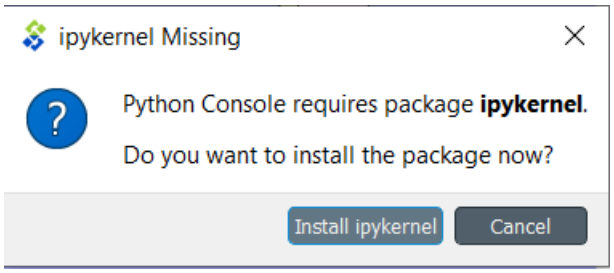
Shell execution

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.**

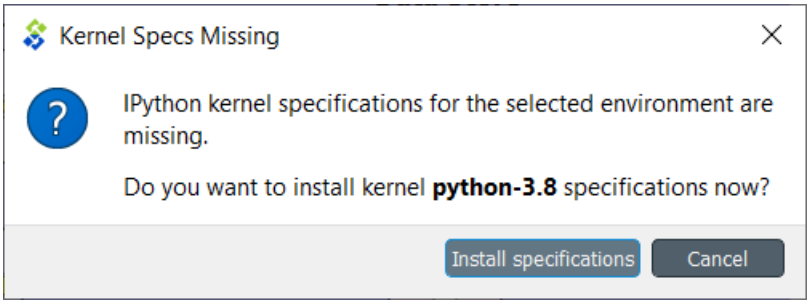
1. Go to https://www.python.org/downloads/ and download the Python you want
2. Run the Python installer and follow instructions
3. Either let the installer put Python in your PATH or memorize the path where you installed it (e.g. *C:\Python38*)
4. Start Spine Toolbox
5. Go to File -> Settings (or press 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.

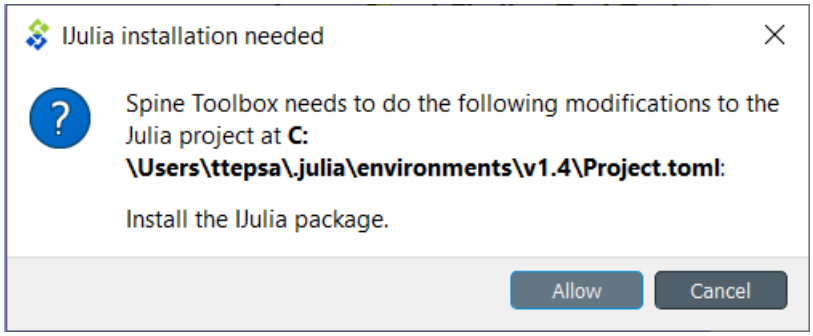
Julia Console execution

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 SpineModel.jl

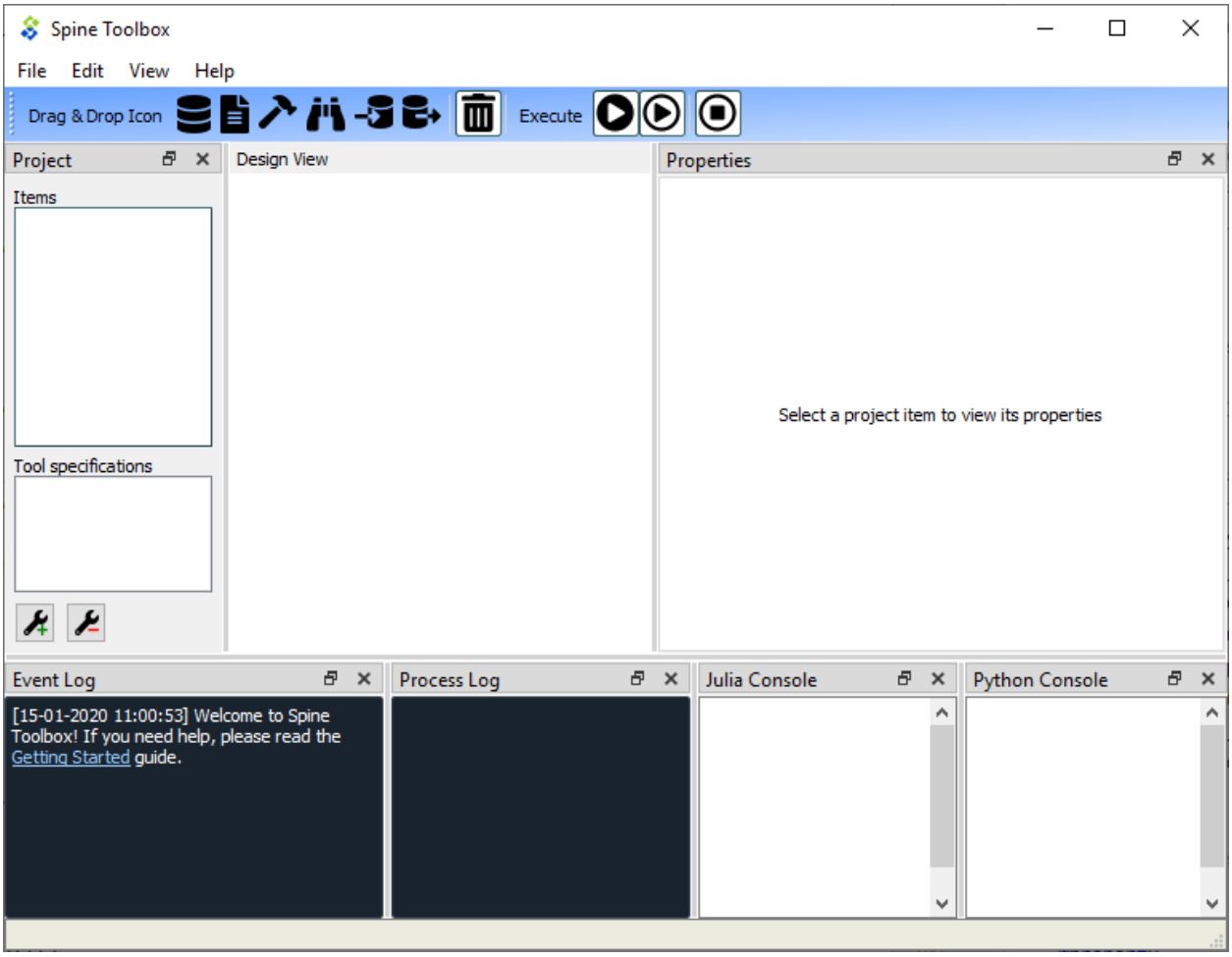
There's a built-in configuration assistant in Spine Toolbox that downloads and configures SpineModel.jl for you automatically. You can find the configuration assistant in the main window menu **File->Tool configuration assistants...->SpineModel.jl** Before you run this, you need to set up Julia for Spine 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`.

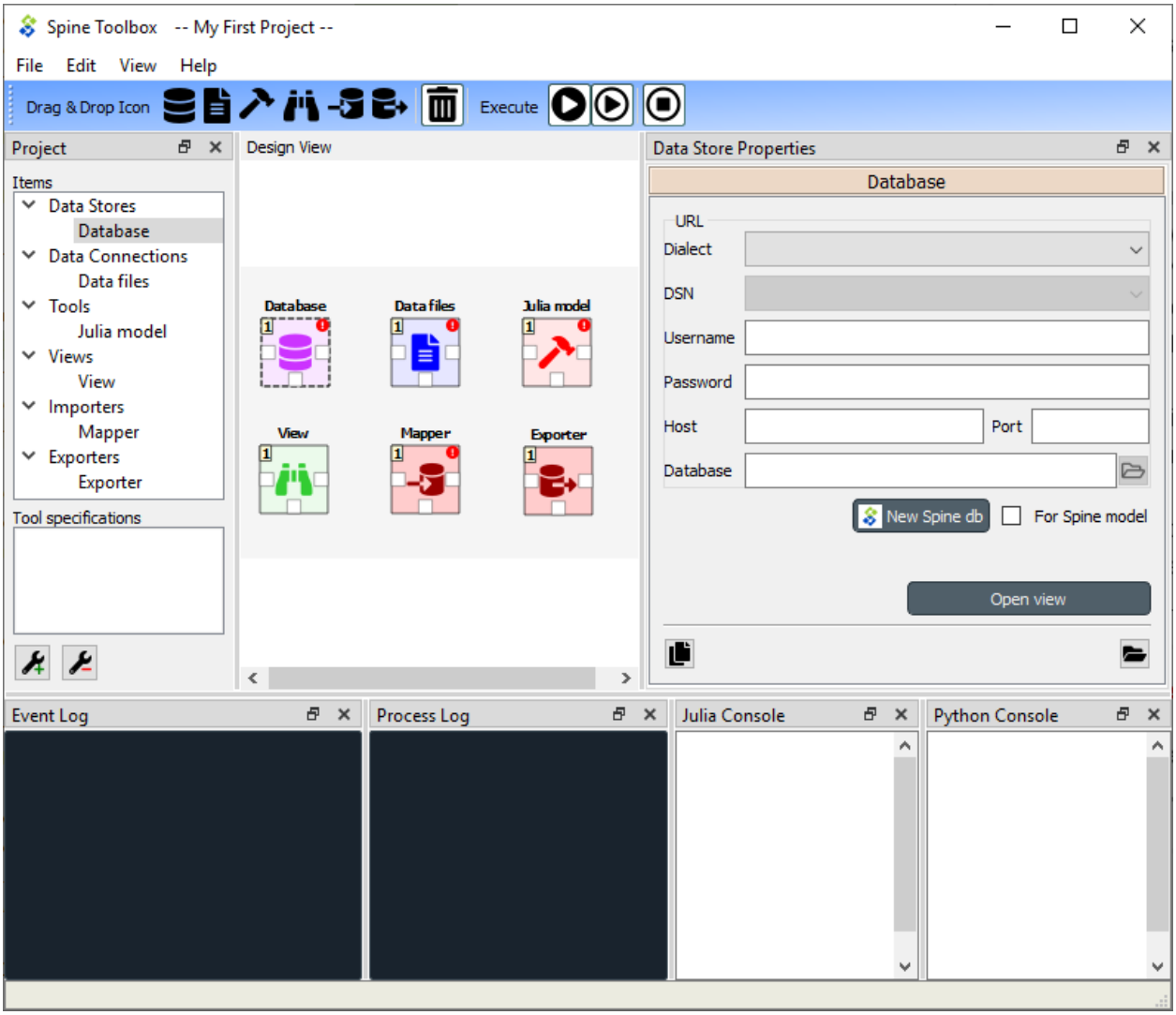
The menu bar in the top of the application contains *File*, *Edit*, *View*, and *Help* menus. In the *File* menu you can create a new project, open an existing project, save the project, upgrade an old project to modern directory-based project, and open the application Settings among other things. Spine 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 six categories. A Data Store called *Database*, a Data Connection called *Data files*, A Tool called *Julia model*, a View called *View*, an Importer called *Mapper*, and an Exporter called *Exporter*. The project items are also listed in the *Project* dock widget.



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. You can also find information on how resources are passed between project items at execution time there.

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 model database. Currently, the item supports sqlite and mysql dialects. The database can be accessed and modified in *Data store view* 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 View

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

4.2.5 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.

4.2.6 Exporter

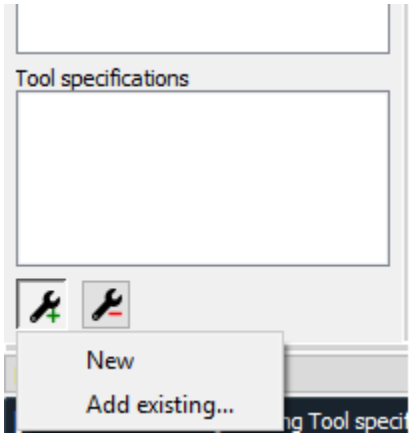
This item exports databases contained in a *Data Store* into `.gdx` format for GAMS Tools. See *Importing and exporting data* for more information.

CHAPTER 5

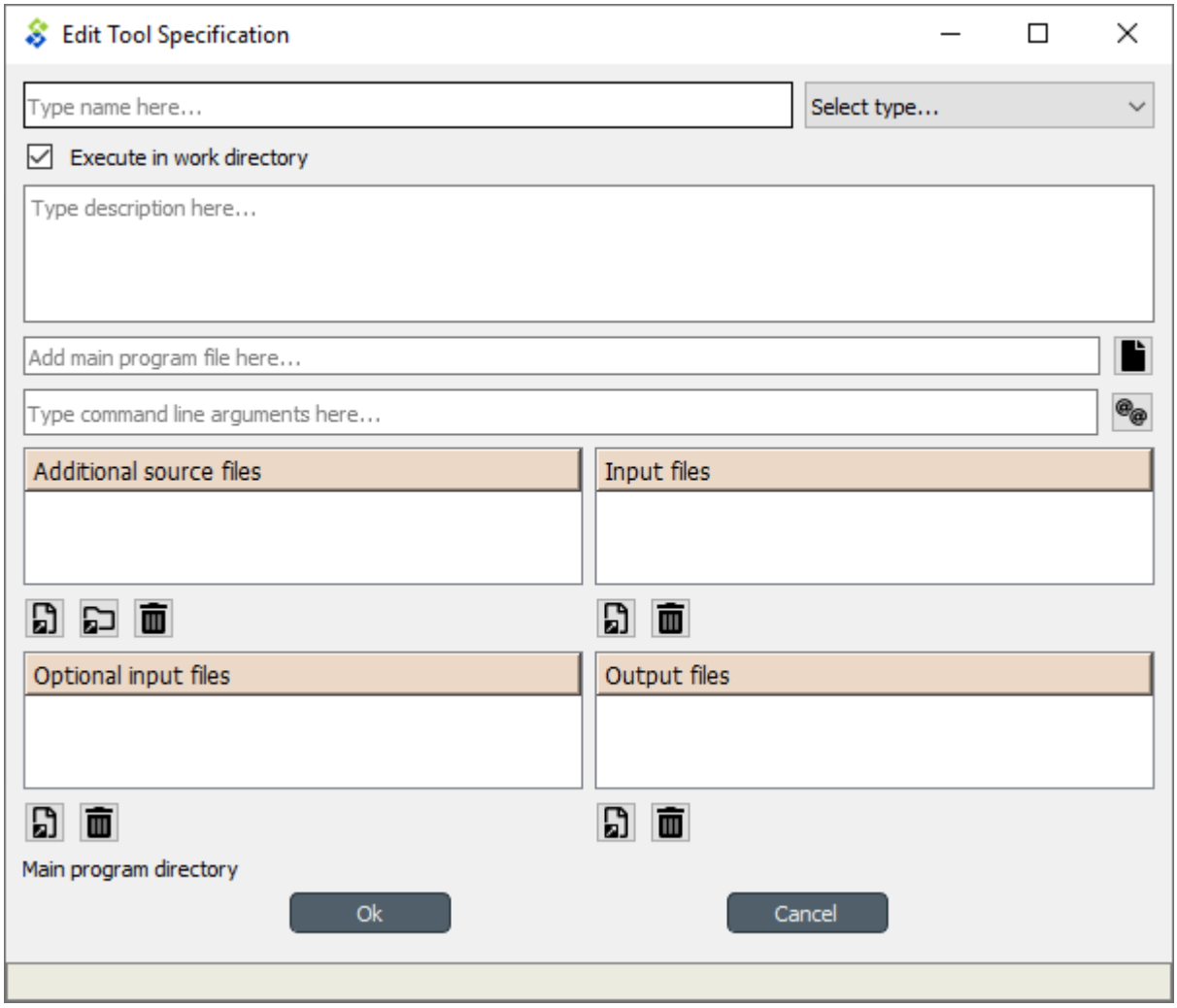
Tool specification editor

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.

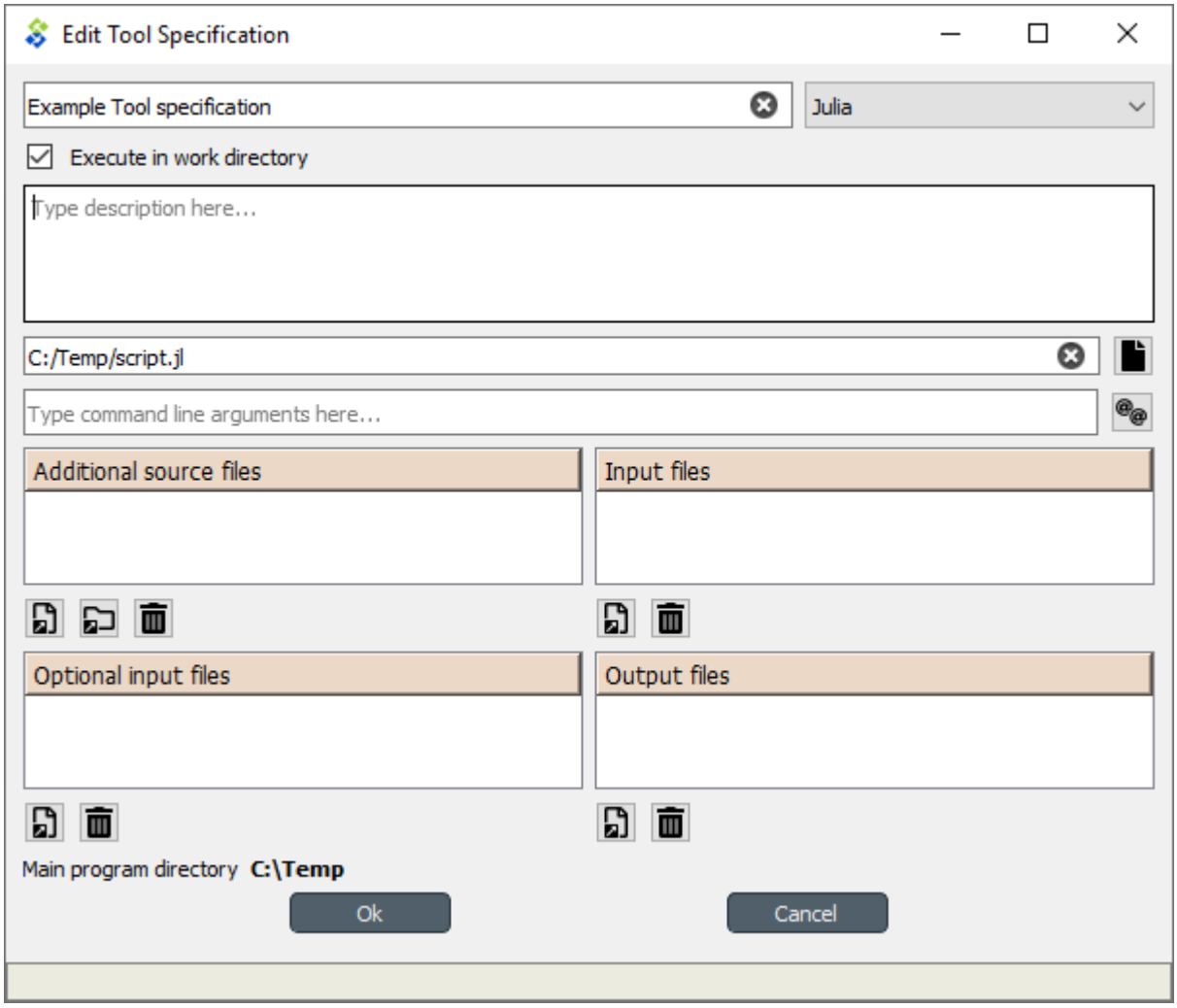
Examples:

- **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.

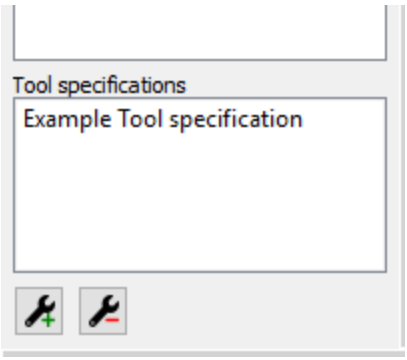
Here is a minimal Tool specification for a Julia script *script.jl*



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.

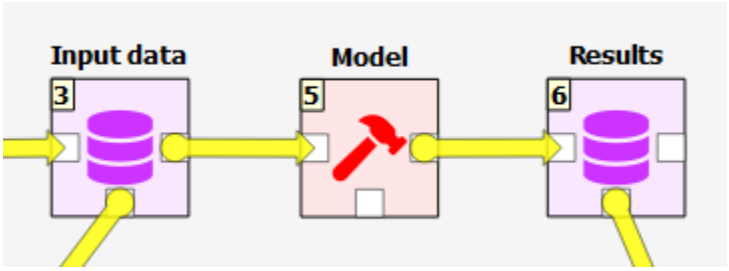


To edit this Tool specification, just right-click on the Tool specification name and select *Edit Tool specification* from the context-menu.

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

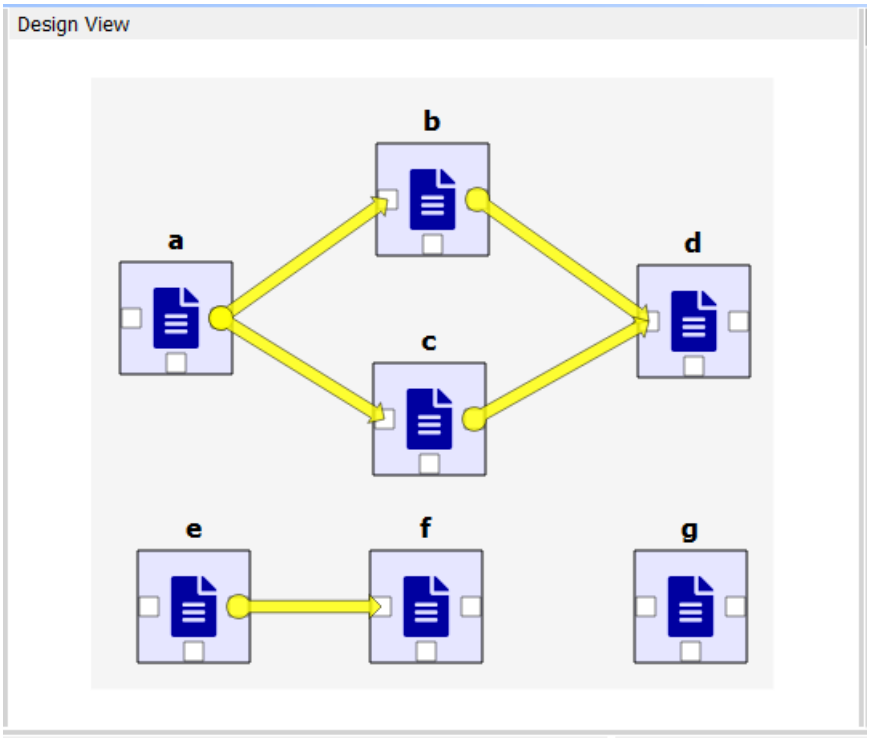
Executing Projects

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 Passing Resources between Project Items

All project items are visited when a DAG is executed but the actual processing only happens when a Tool, an Importer, or an Exporter project item is visited. The processing is done in a subprocess, in order to not clog the GUI until the project item has been executed.

When project items are connected to each other, the resources that are passed between project items at execution

depends on the project item type. The following table describes the resources that project items use from their predecessors and what resources are passed to their successors.

Note: Resources are only transmitted to **direct successors** and **direct predecessors**.

Type	Notes	Accepts from predecessor	Accepts from successor	Provides to predecessor	Provides to successor	Properties
Data Connection	1	n/a	n/a	n/a	File URLs	File paths
Data Store	2	n/a	n/a	Database URL	Database URL	Database URL
Exporter		Database URL	n/a	n/a	File URLs	Export settings
Importer	3	File URLs	Database URL	n/a	n/a	Import mappings
Tool	4	File URLs, database URLs	Database URLs	n/a	File URLs	Tool specification,
						cmd line arguments,
						execute in work dir
View		Database URLs	n/a	n/a	n/a	n/a

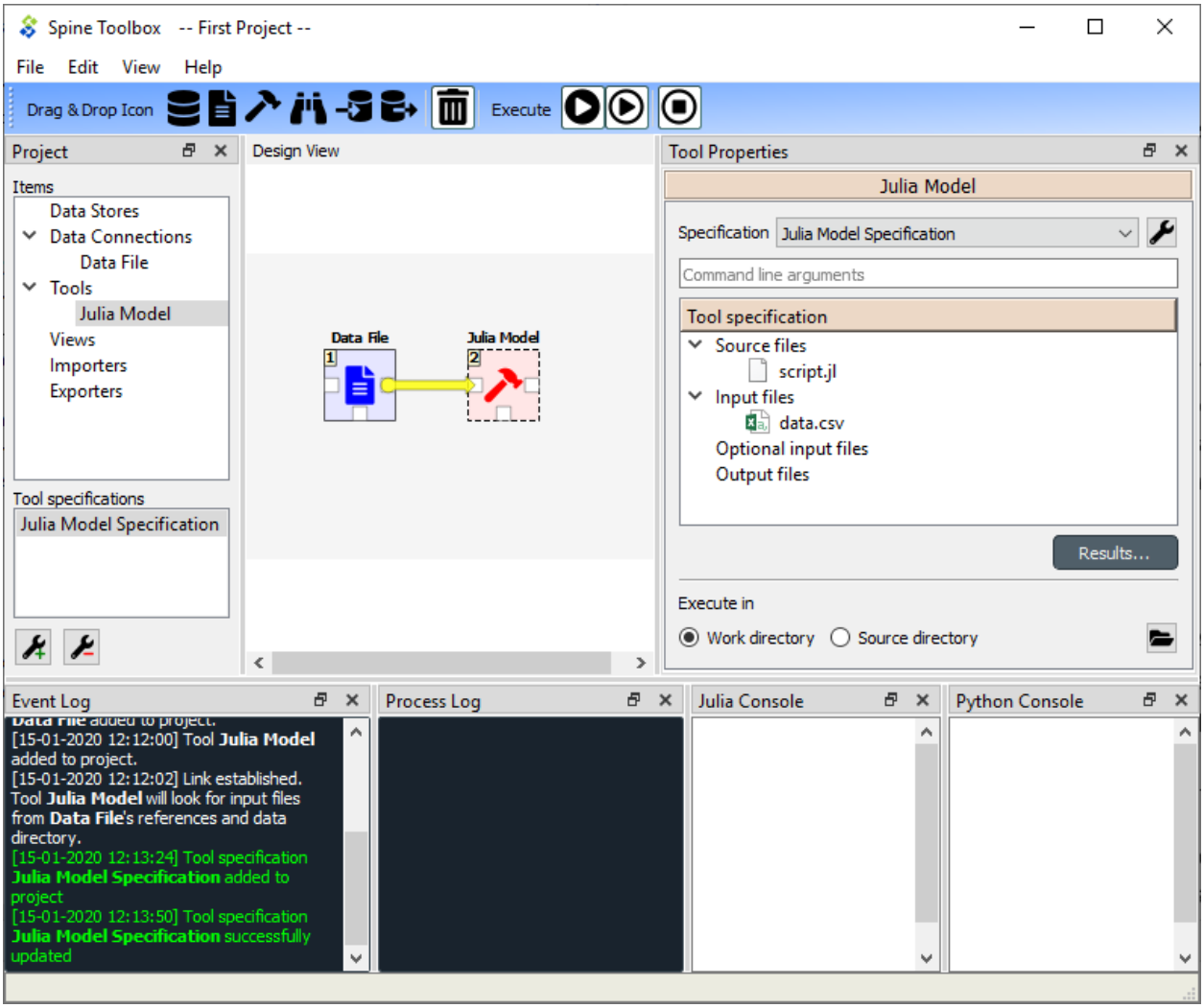
Notes:

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 predecessor.
3. Importer requires a database URL from its successor for writing the mapped data. This can be provided by a Data Store.
4. Tool *program* is defined by its *Tool specification*, which also contains the required files, optional files, and output files of the *program*. The output files are provided to successors as file URLs. 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 an URL is actually required by a tool *program*. For more information, see *Tool specification editor*.
5. The **Properties** column describes the resources that the user is expected to set for each project item in Spine Toolbox.

To learn more about Project items and their responsibilities, please see *Project Items*.

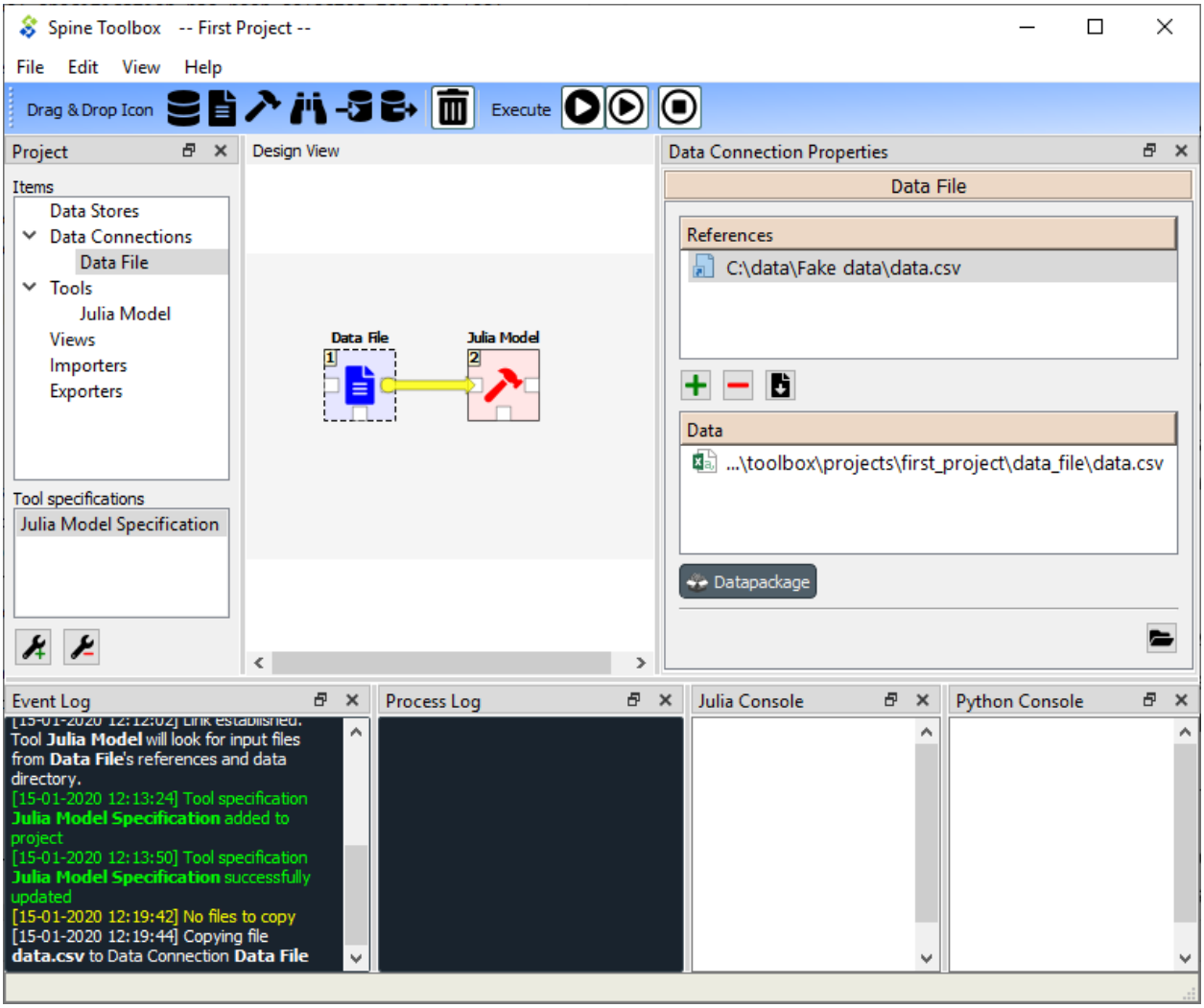
6.2 Example DAG

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.

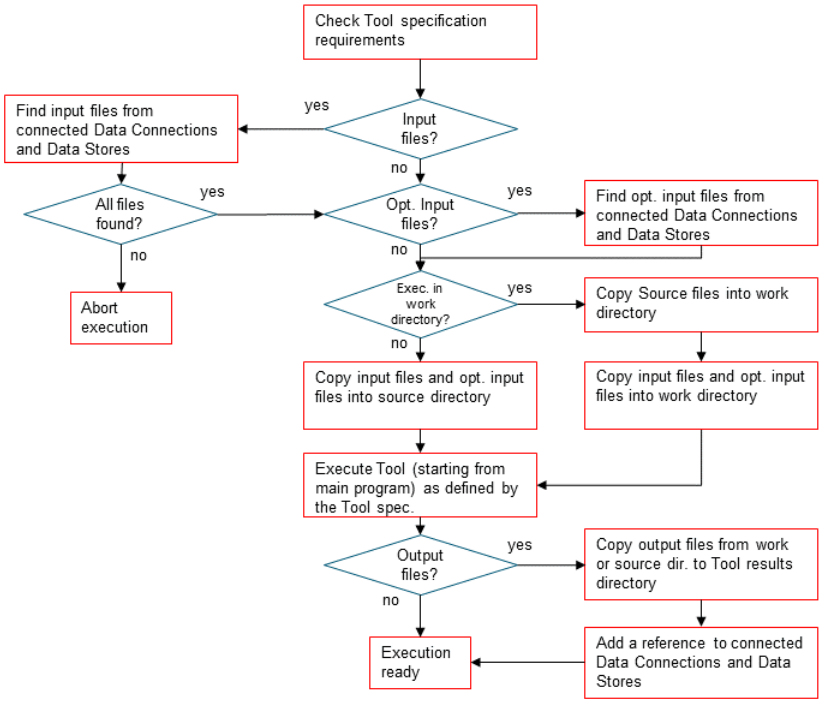


When it's the *Julia Model* tools turn to be executed, it checks if it finds the file *data.csv* from project items, that have already been executed. When the DAG is set up like this, the Tool finds the input file that it requires and then starts processing the Tool specification starting with the main 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).

6.3 Tool execution algorithm

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

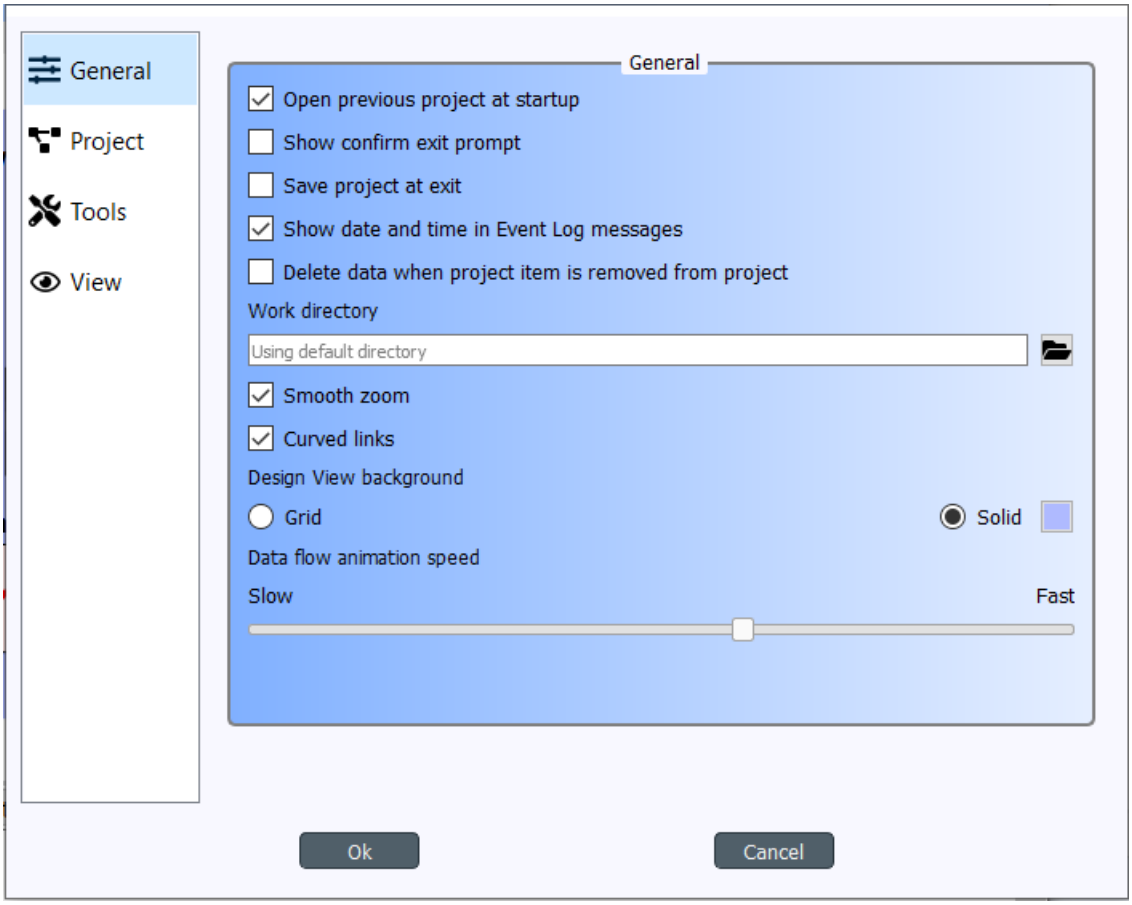

Check Tool specification requirements
Input files?
yes
Find input files from connected Data Connections and Data Stores
All files found?
yes
no
Abort execution
Opt. Input files?
yes
Find opt. input files from connected Data Connections and Data Stores
no
Exec. in work directory?
yes
Copy Source files 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

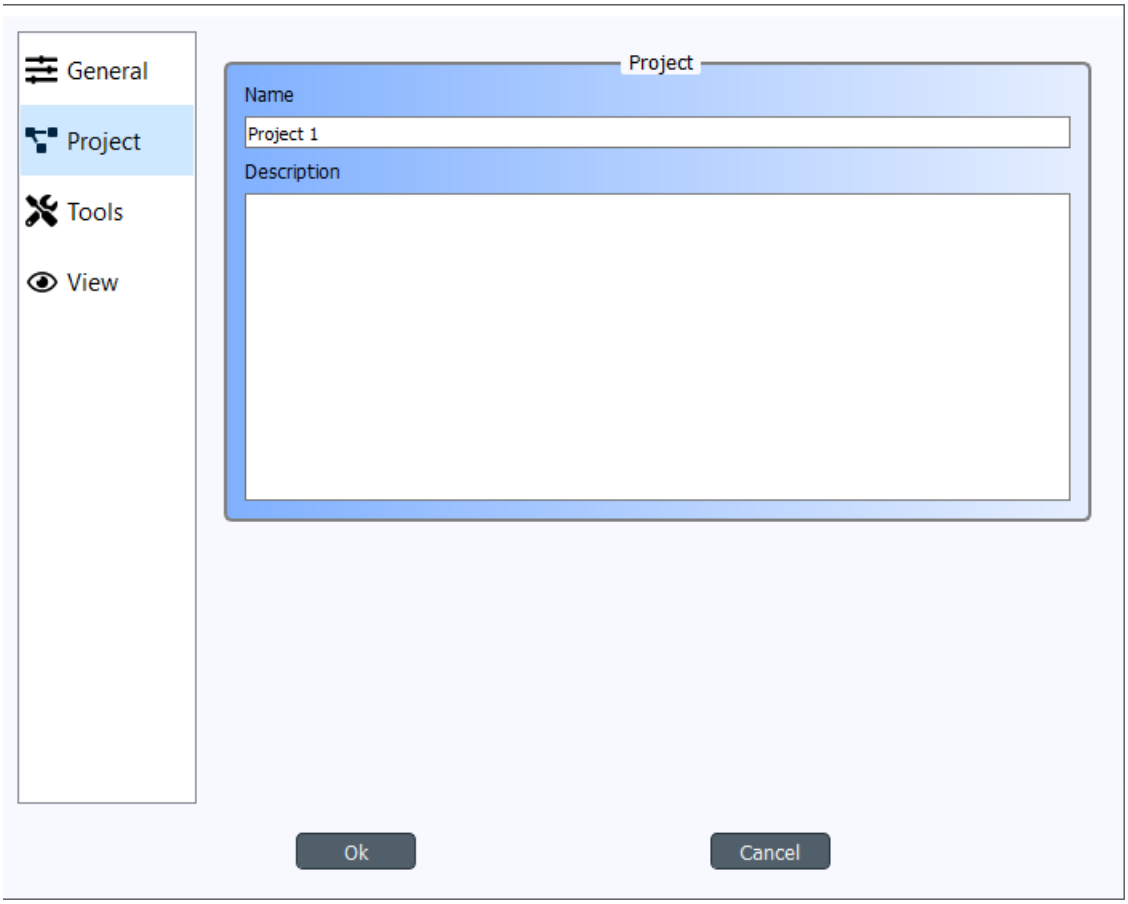


The General tab contains the general application settings.

- **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 Data Store View. 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.

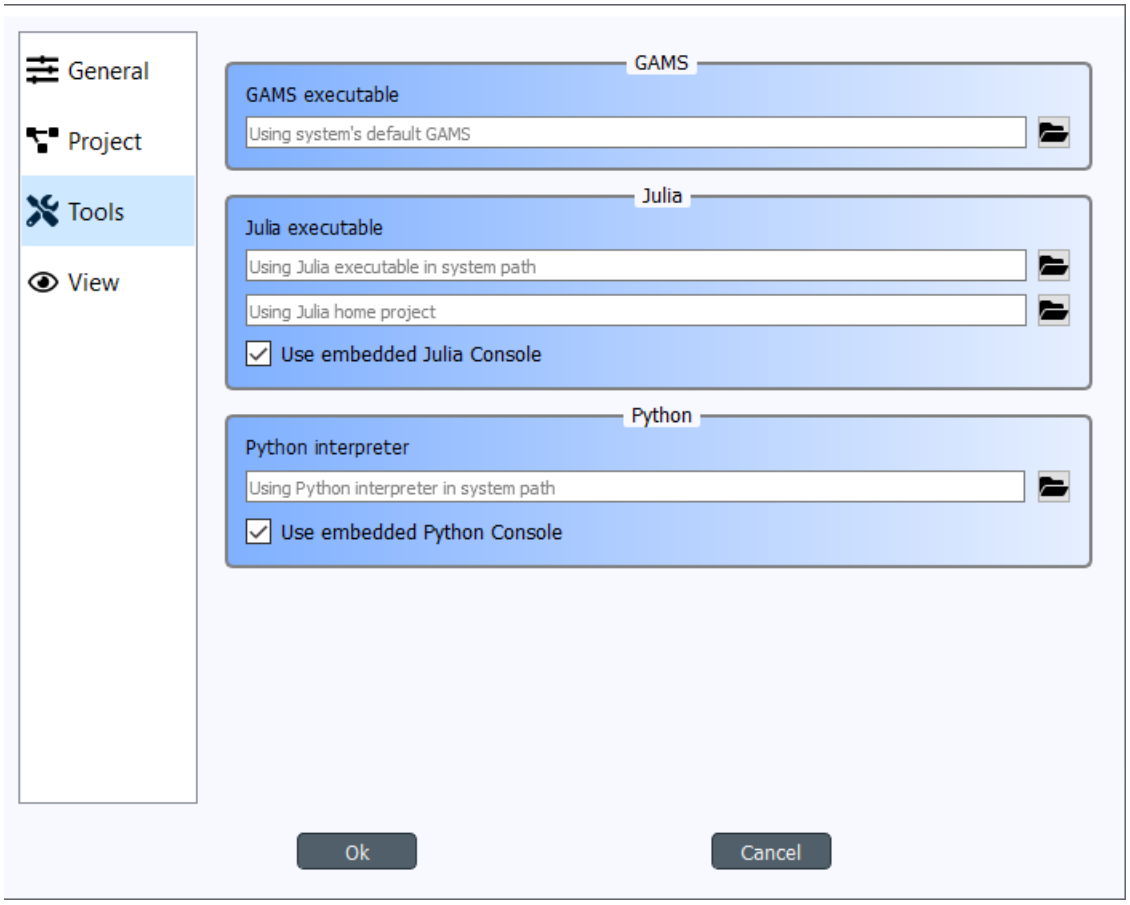
7.2 Project settings



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.

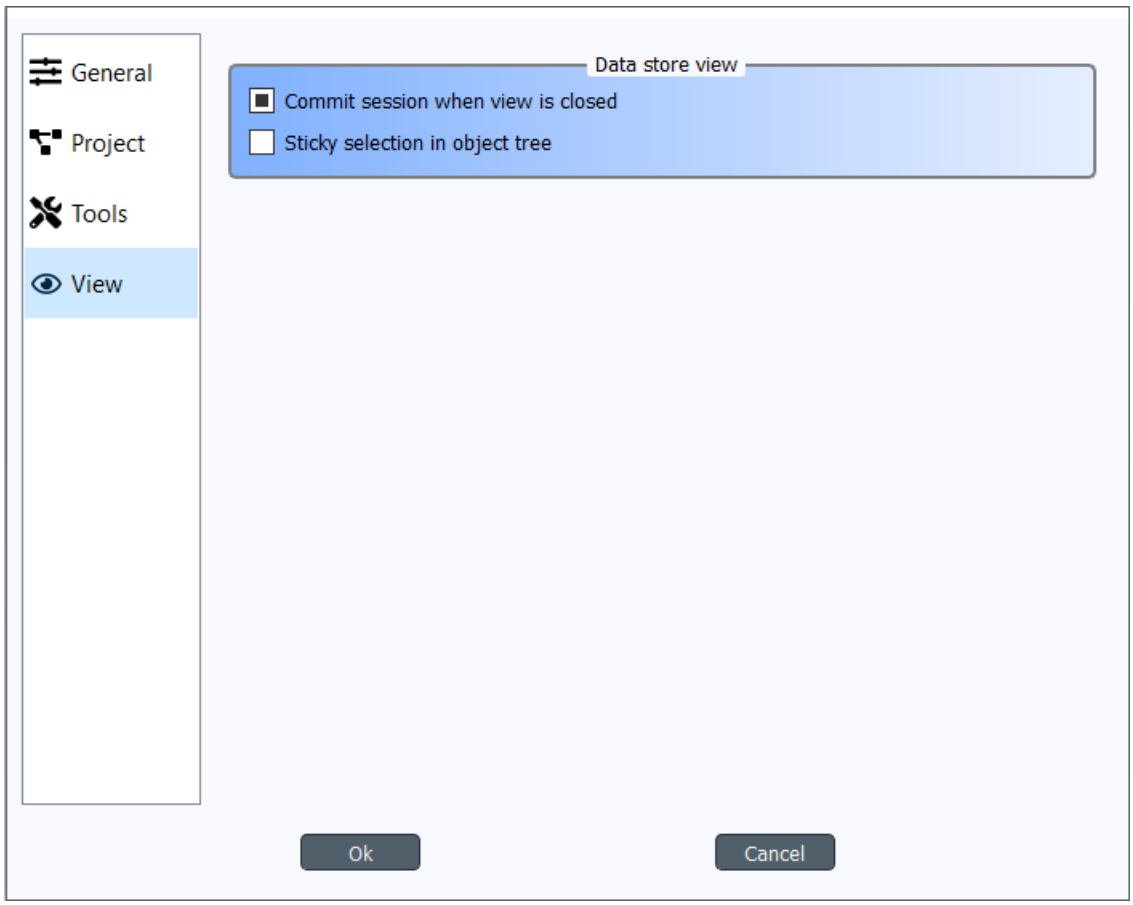
7.3 Tools settings



- **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 Data Store View 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 Data Store View'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.

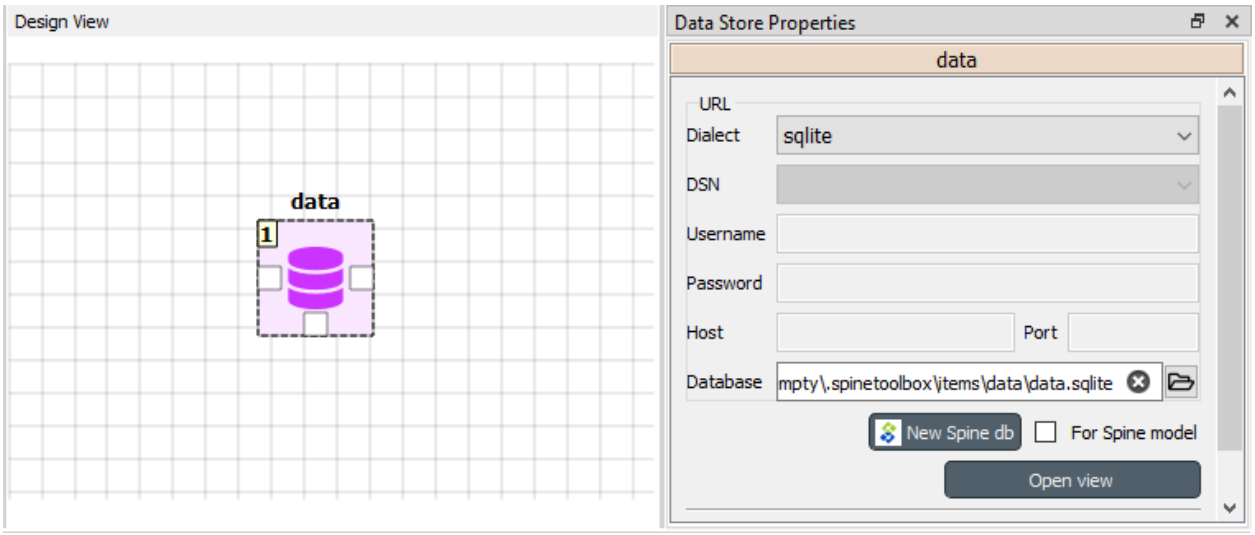
Note: If you are looking for information on project item properties, see *Project Items*.

CHAPTER 8

Data store view

This section describes the different interfaces available in *Data Store view* for viewing and editing data in a Spine database.

To open Data store view, select a **Data Store** and click the *Open view* button in its *Properties*:



Data store view consists of a number of dockable views that display the data in the database in different ways. Three pre-defined layouts or *styles* are available from the *View* menu:

- In Tree style layout () you can edit, add and remove all database entities and parameters
- In Tabular style layout () you can display the the data in a pivot table
- In Graph style layout () you can view the structure of classes, objects and relationships in a graph representation

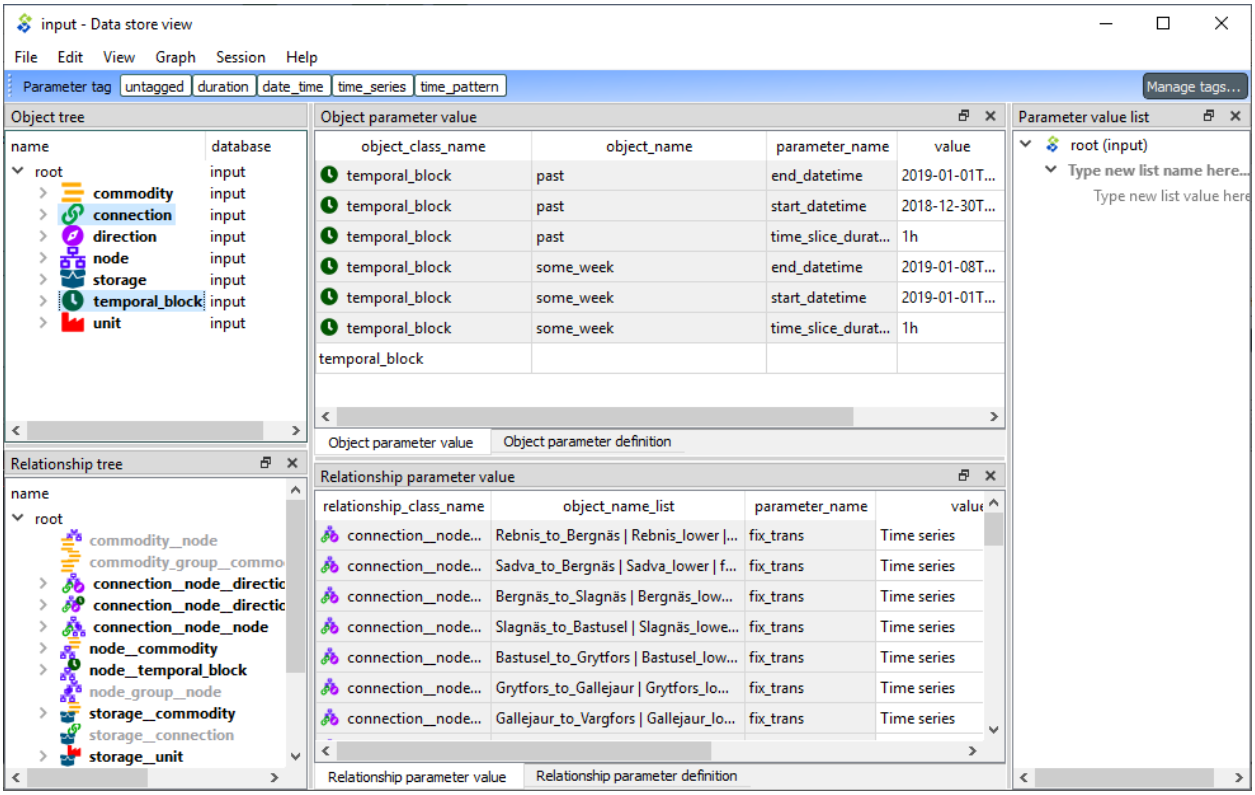
8.1 Commit/Rollback changes

Changes are not immediately saved to the database. They need to be committed separately. To do that select **Session -> Commit** from the menu bar, enter a commit message and press **Commit**. Any changes made in the Data store view will be saved into the database.

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

8.2 Tree style

The **Tree style** layout is useful to get an overview of the data and the relationships within a Spine database:



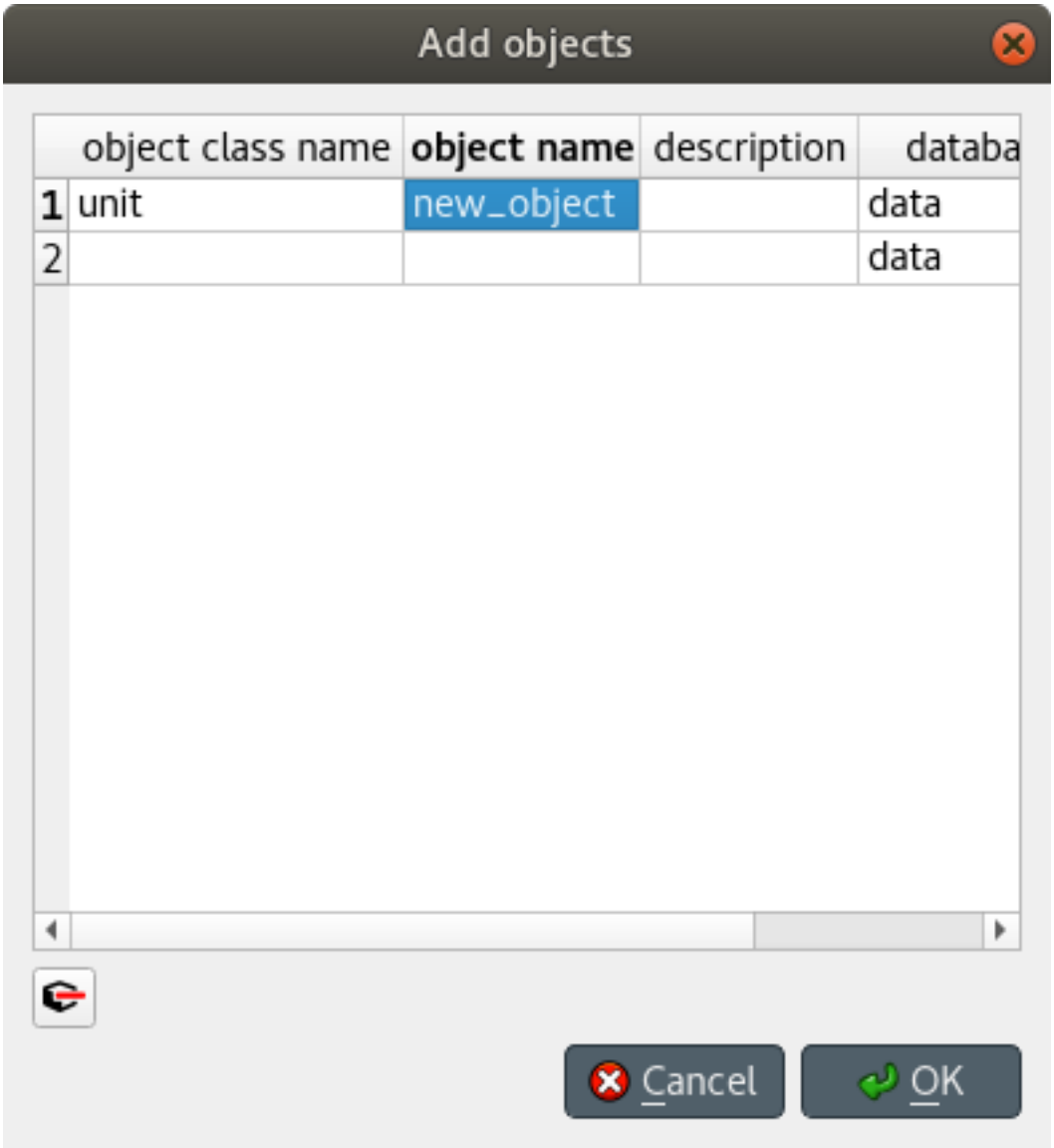
In the default **Tree style** layout the left side of the Data store view is occupied by two tree views which display the different object and relationship classes, with their objects and relationships in a hierarchical tree. These and the other views allow you to add, edit, and delete object classes, relationship classes, objects, relationships, parameters and parameter values.

The interface has five main components:

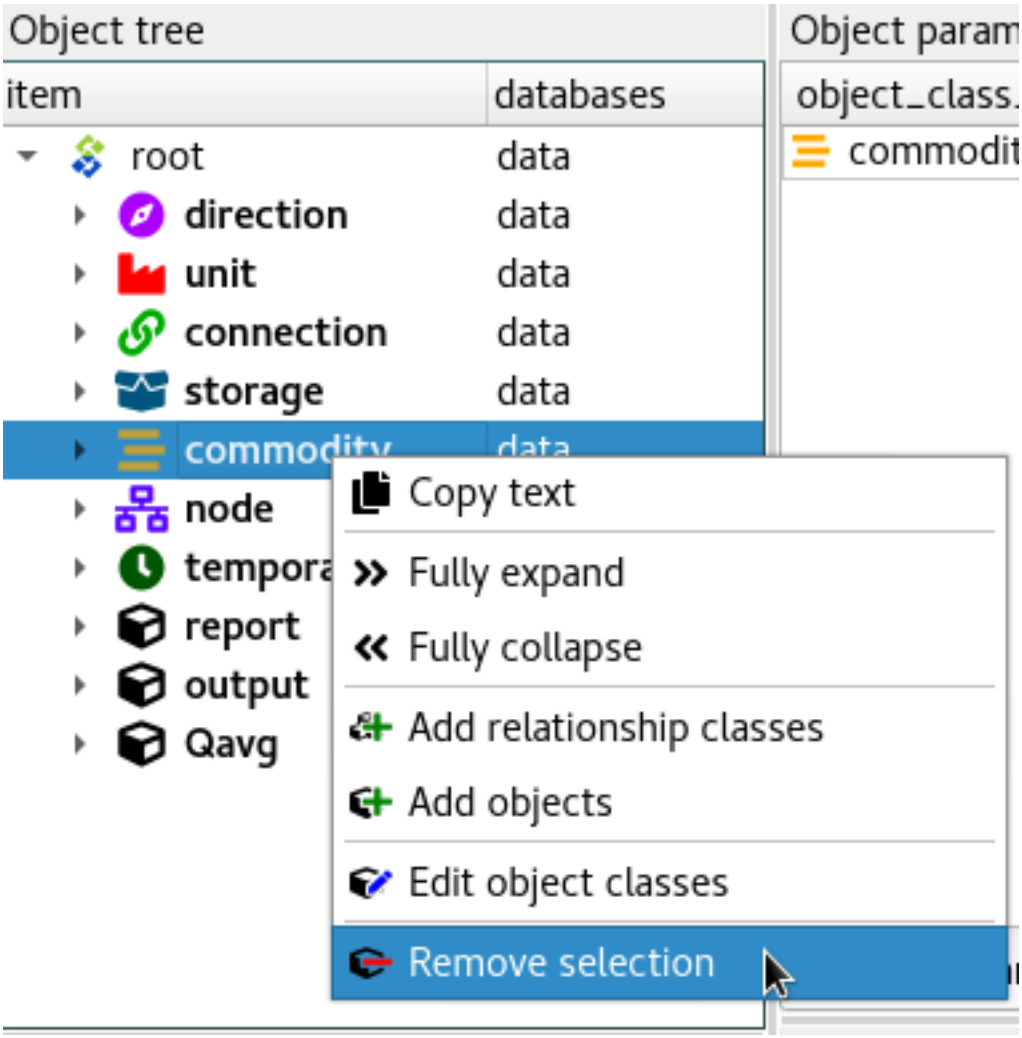
1. *Object tree*, where you can expand and collapse the different levels of the hierarchy. It also acts as a filtering tool for the two table components, so that only items selected in the *Object tree* are shown in the *Parameter tables*.
2. *Relationship tree*, similar to *Object tree* but for relationships.
3. *Object parameter value* table, where you can view, add, edit, and delete object parameter definitions and values.
4. *Relationship parameter value* table, where you can view, add, edit, and delete relationship parameter definitions and values.
5. *Parameter value list* allows you to create value lists that can be associated to parameter definitions.

8.2.1 Editing items

To add object classes, relationship classes, objects or relationships you can use the **Edit** menu from the main menu bar, as well as the context menu from the *Object tree*. In the dialog that pops up you can enter new items by typing their names or pasting data from the clipboard.



To delete an item, you can again use the **Edit** menu from the main menu bar or the item's context menu from the *Object tree*.



Editing items is done following a similar procedure.

8.2.2 Viewing parameter definitions and values

In the *Parameter tables*, you can switch between viewing parameter definitions or values by using the tabs in the lower left corner.

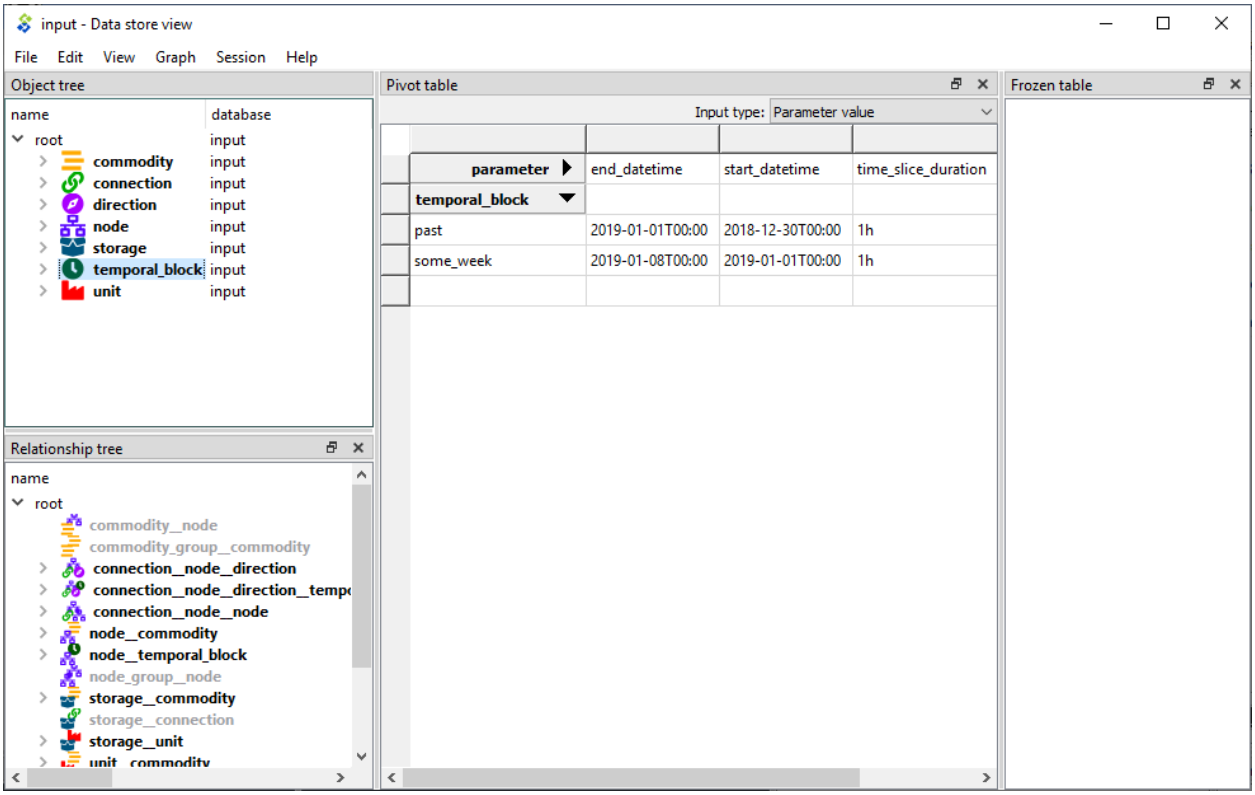
You can also (further) filter the tables by clicking on the column headers.

8.2.3 Editing parameters definitions and values

To add new parameter definitions or values you can directly do it in the last row of each table. The tables also support pasting values from the clipboard.

8.3 Tabular style

The **Tabular style** layout is used to display and edit data in a Spine database via a table-like interface. The interface lets you filter and pivot the data for exploration and editing.



The interface has four main components:

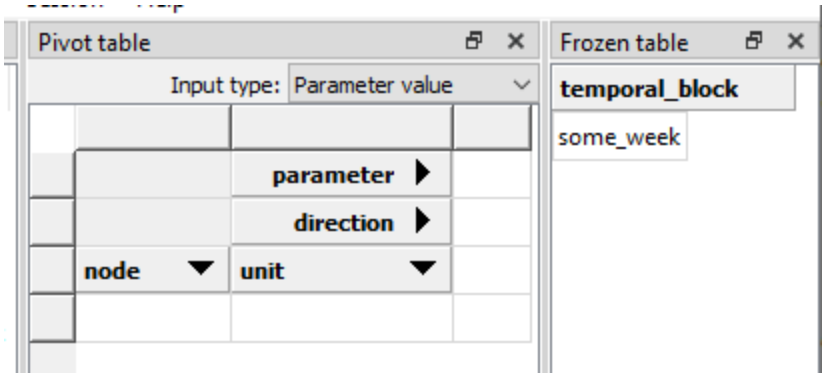
1. *Object tree*, where you can expand and collapse the different levels of the hierarchy. The item selected in the *Object tree* is shown in the *Pivot table*.
2. *Relationship tree*, similar to *Object tree* but for relationships.
3. *Pivot table*, where you can transform the data view by dragging and dropping the header blocks. You can choose, e.g., which items go into rows and which into columns.
4. *Frozen table*: dragging header blocks here freezes the selected items in the *Pivot table*.

From the drop-down *Input type* list at the top of the Pivot table, you can select two different input types:

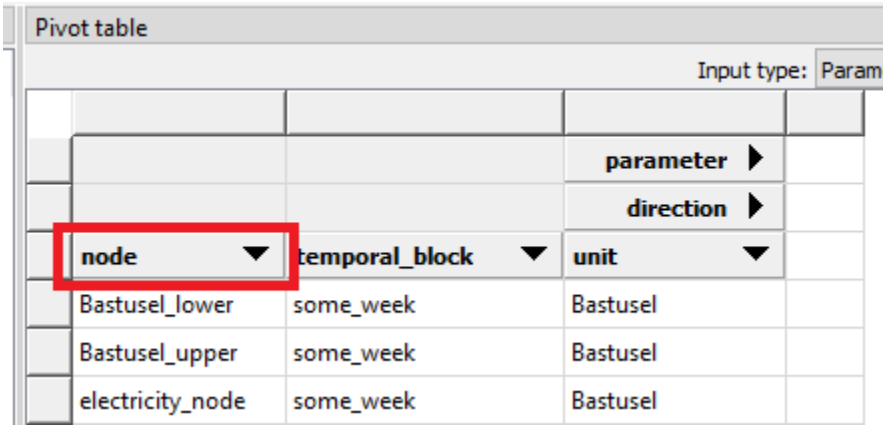
- *Parameter value*: display all objects (or relationships), as well as all parameters and parameter values for the selected object (or relationship) class.
- *Relationship*: display only the objects (or relationships) for the selected object (or relationship) class.

8.3.1 Pivoting and filtering data

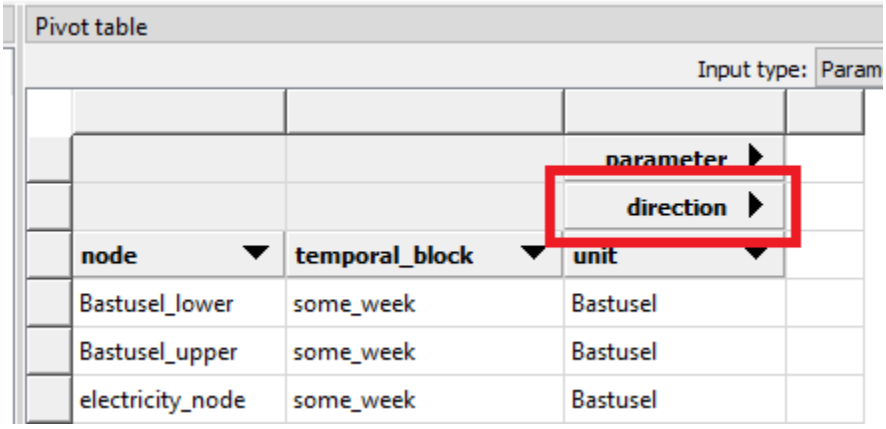
You can transform (pivot) the data view by dragging header blocks across the *Pivot table* and to *Frozen table*:



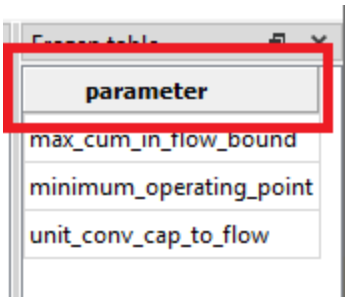
When you drag a header block in Pivot table in a column header position like the 'node' header below, items under that header are displayed in that column of the Pivot table so there is a unique row for each item.



When you drag a header block in Pivot table in a row header position like the 'direction' header below, items under that header are displayed in that row of the Pivot table.



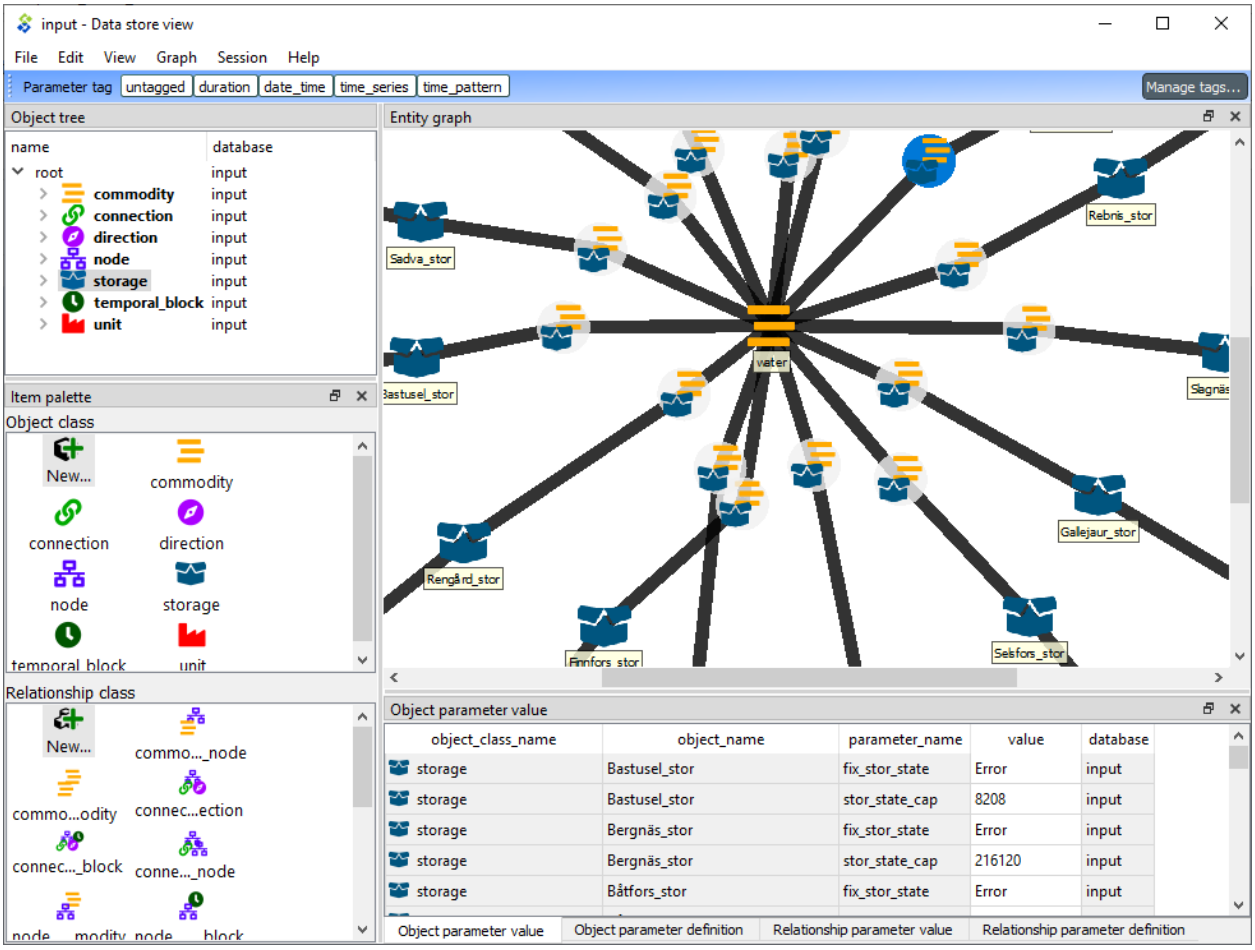
When you drag a header block to the *Frozen table* all items in under the header are excluded from the Pivot table and shown in the Frozen table instead. The Pivot table is then filtered by the selected item in the Frozen table.



To filter a specific item you can use the filter buttons in the header blocks. It is possible to apply multiple filters at the same time.

8.4 Graph style

The **Graph style** layout is used to visualize the Spine database structure into a graph. Here you can select objects to see how they are related. You can also view parameter definition and values same as in the **Tree style** layout.

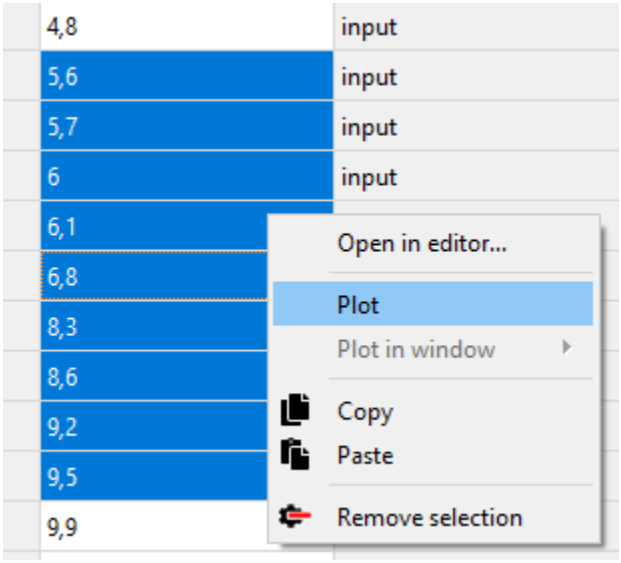


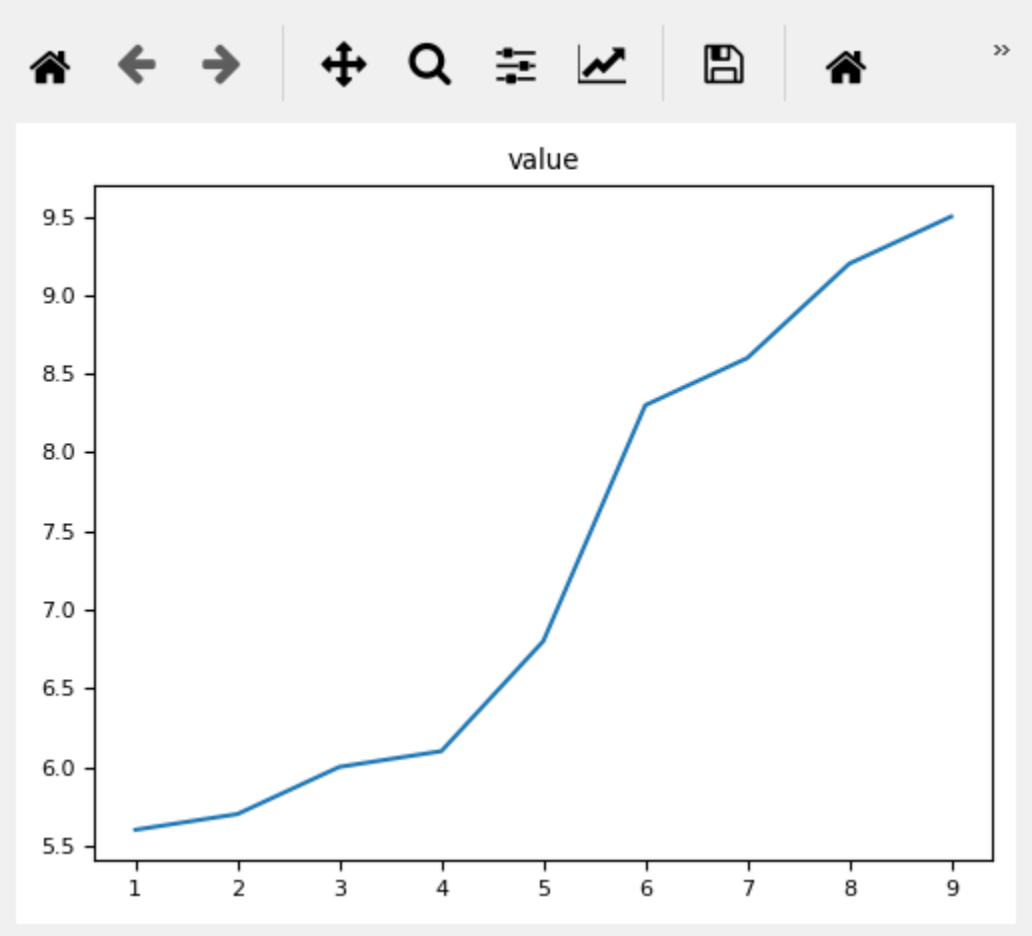
CHAPTER 9

Plotting

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

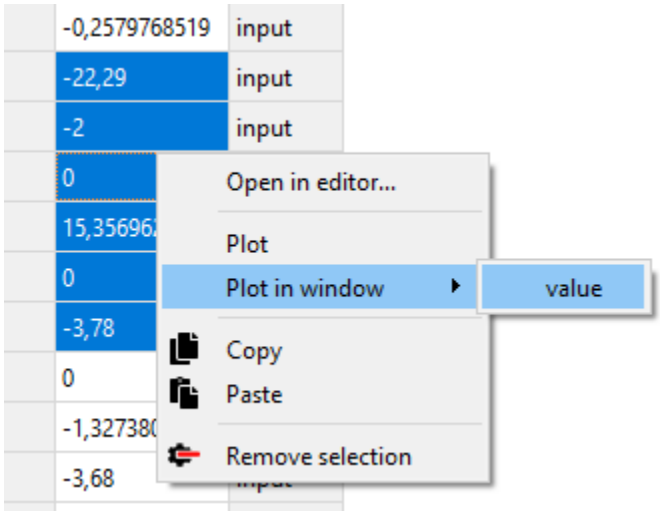
To plot a column, select the values from a table and then *Plot* from the **right click** popup menu.





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

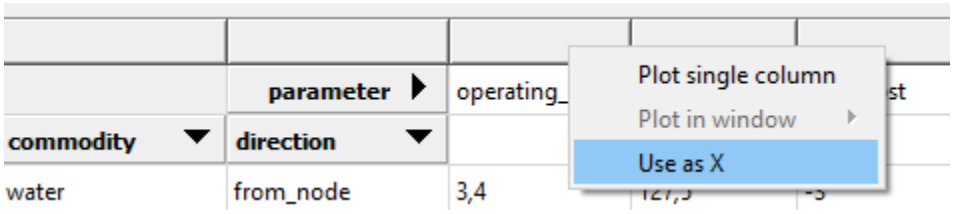
It is possible to add a plot to an existing plot window. Select the target plot window from the *Plot in window* submenu and the data will be added to the plot.



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.

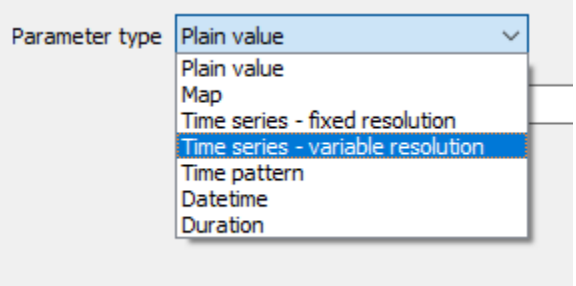
CHAPTER 10

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 data store views.

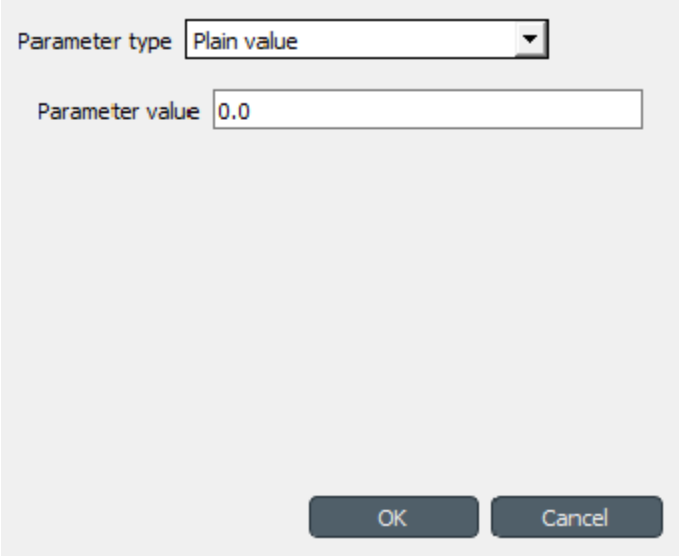
10.1 Choosing value type



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

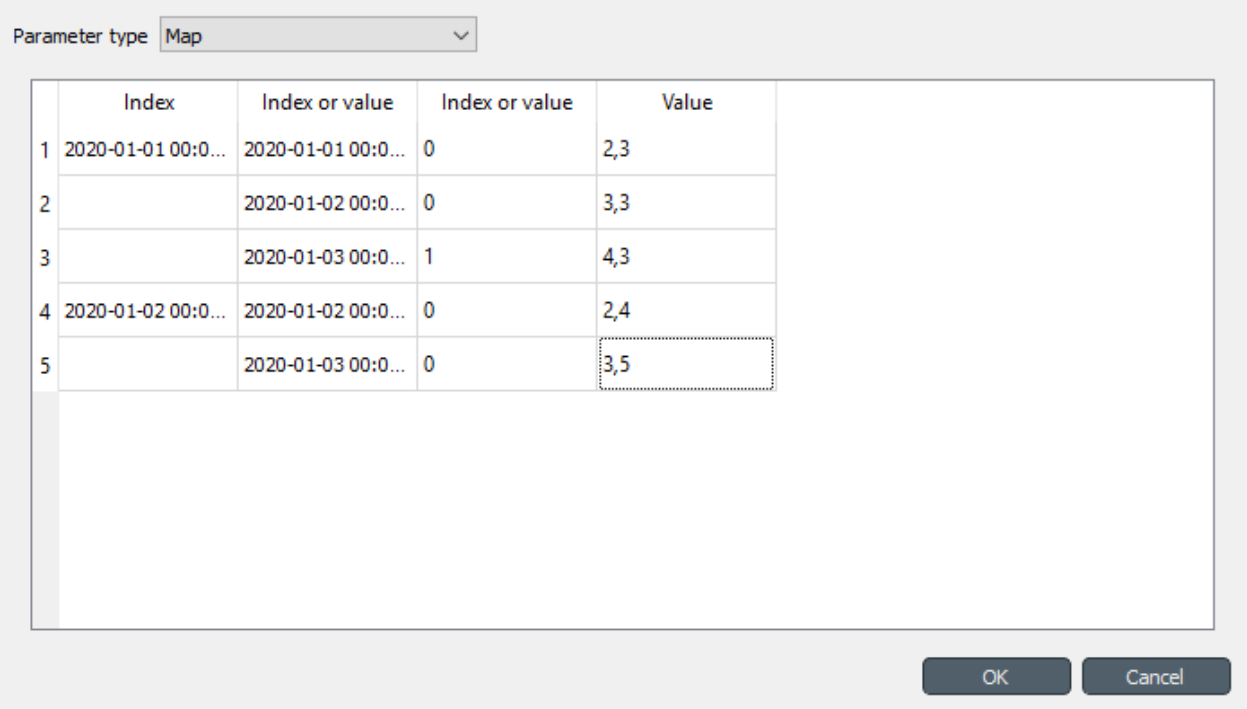
10.2 Plain values

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.

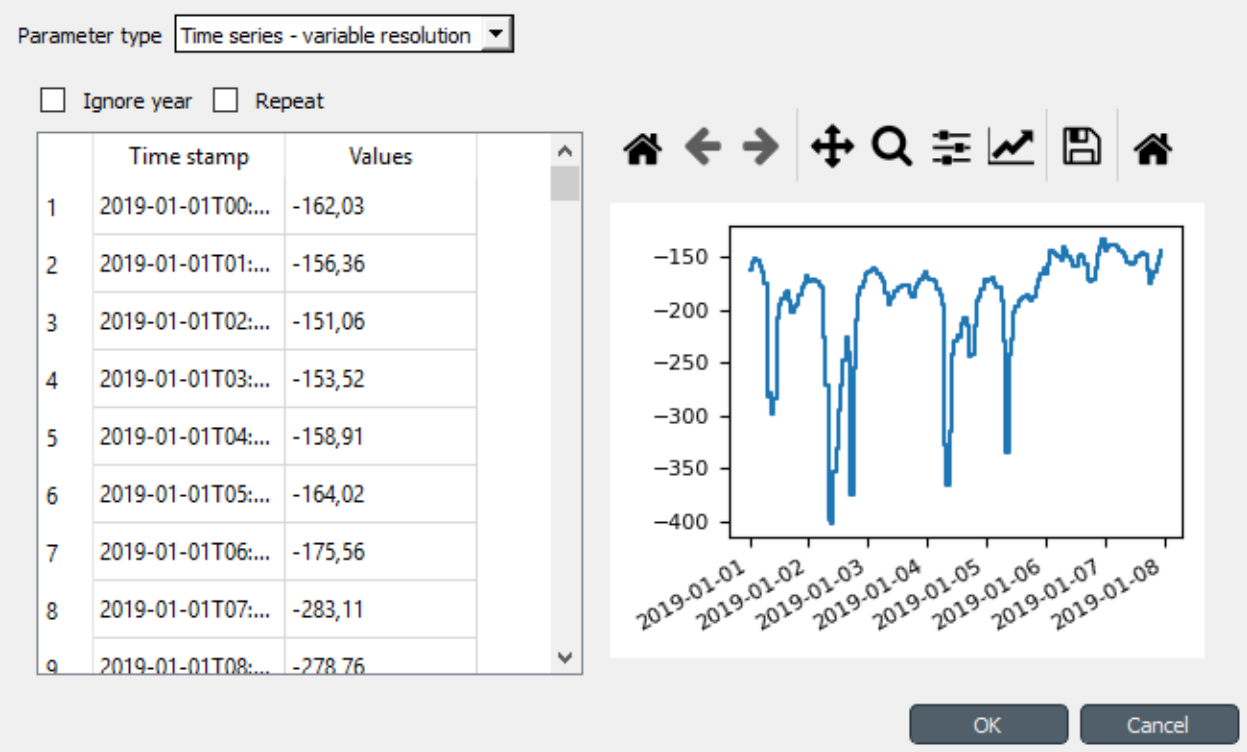


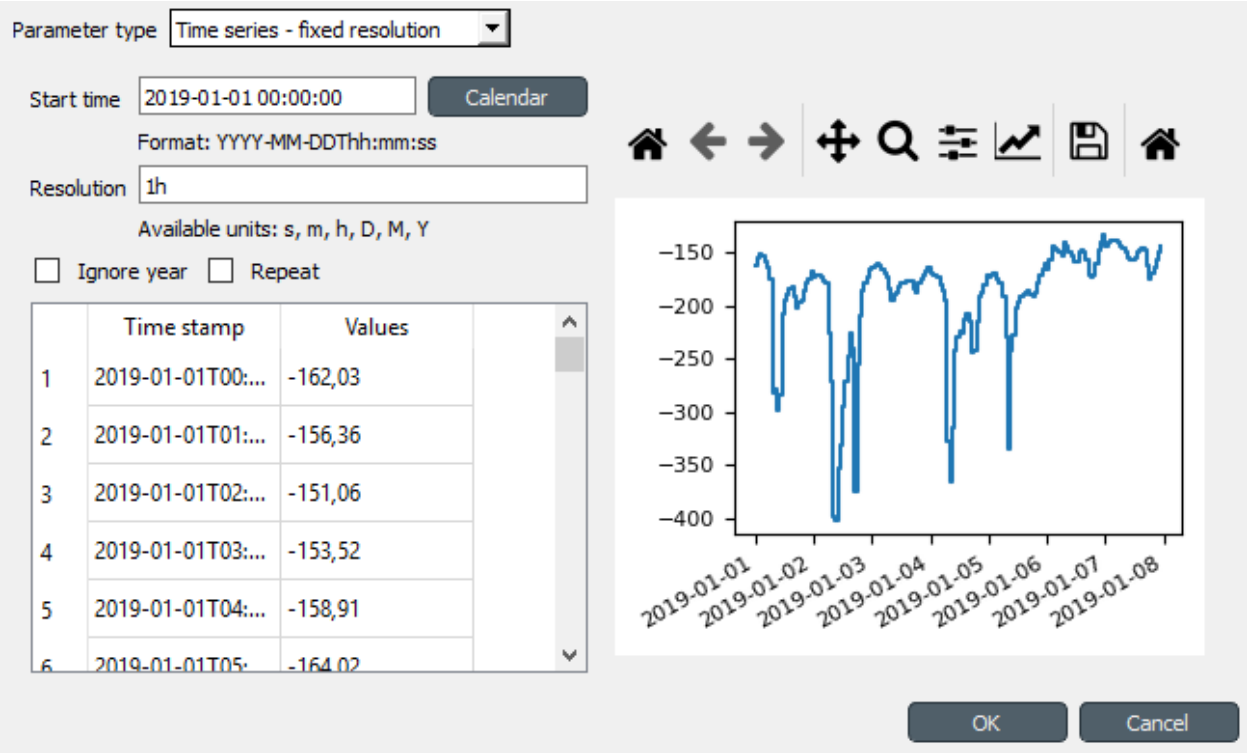
A **Right click** popup menu gives options to add rows or columns (effectively adds a new dimension to map) or trim empty columns from the right hand side.

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.

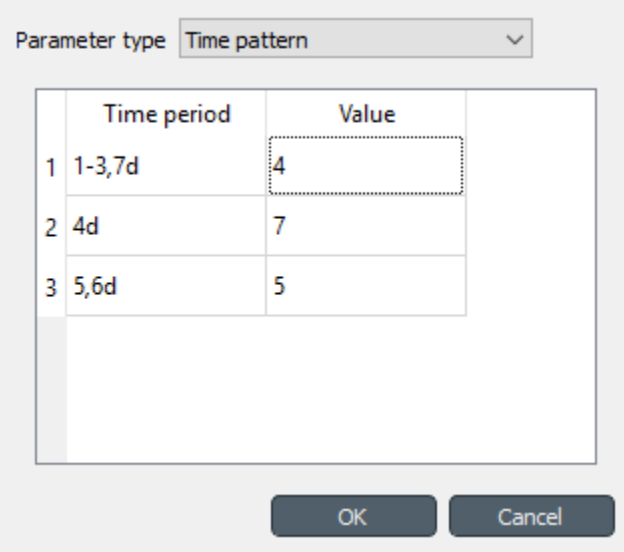
The time steps of a fixed resolution series are edited by the *Start time* and *Resolution* fields. The format for the start time is ISO8601. The *Resolution* field takes a single time step or a comma separated list of steps. If a list of resolution steps is provided then the steps are repeated so as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

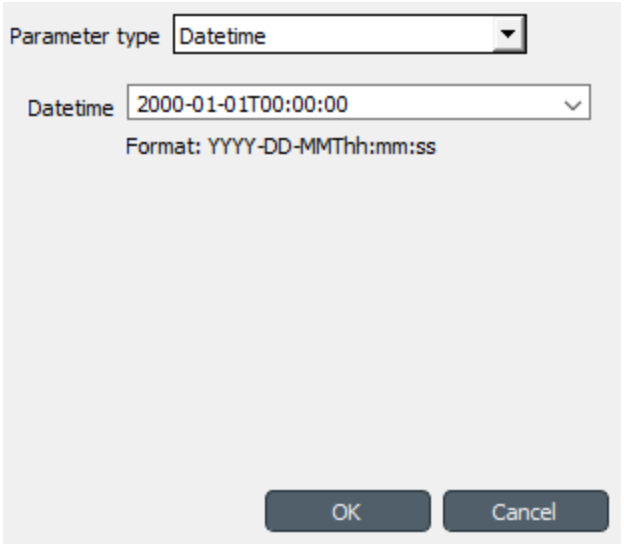
10.5 Time patterns

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.



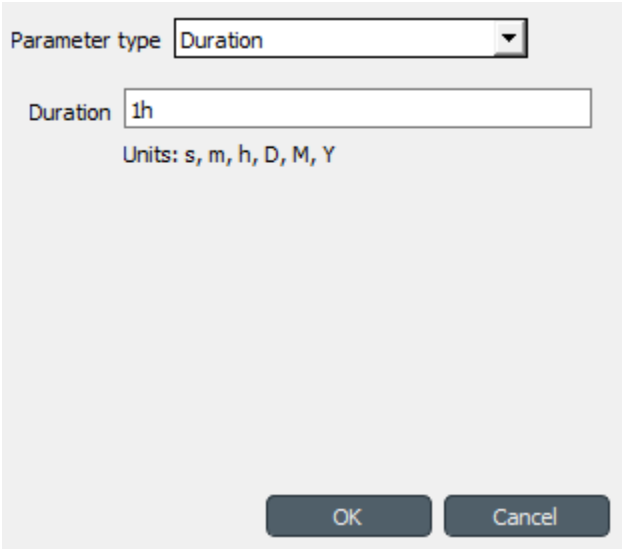
10.6 Datetimes

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

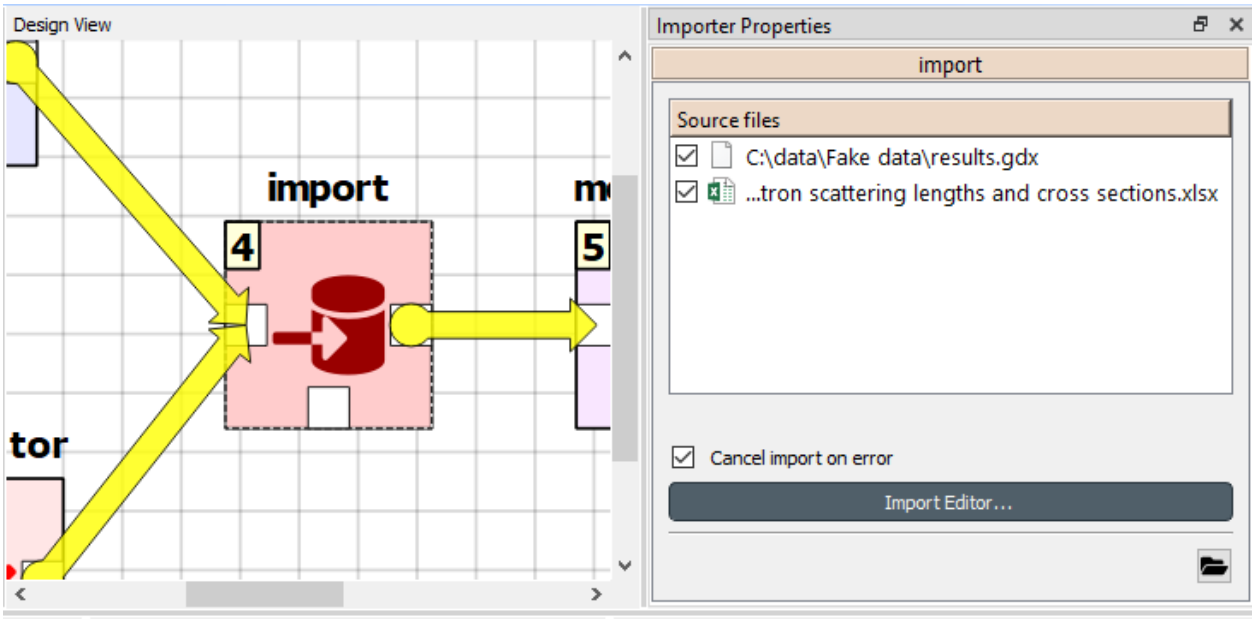
This section explains the different ways of importing and exporting data to and from a Spine database.

11.1 Importing data with Importer

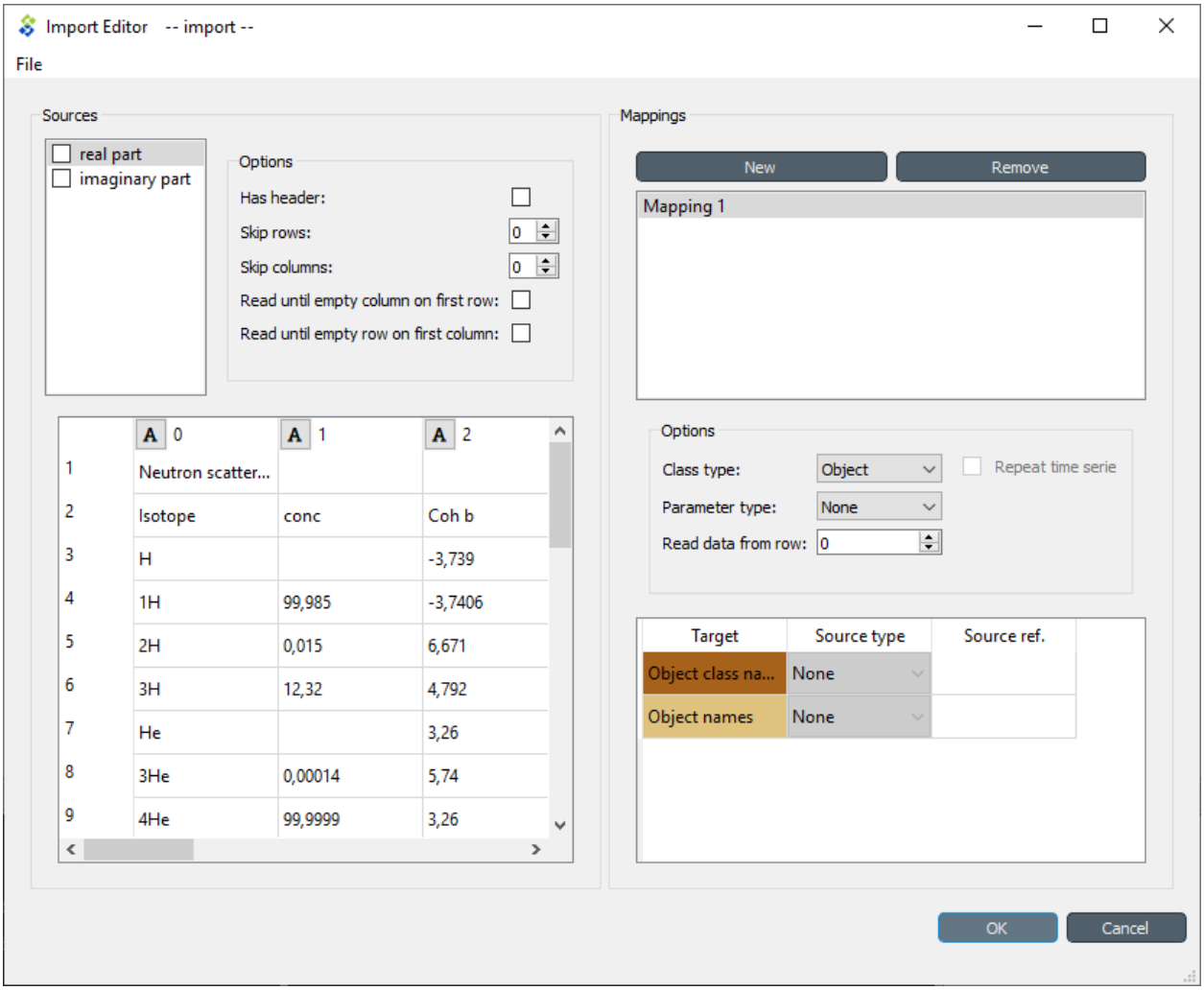
Data importing is handled by the Importer project item which can import tabulated and to some degree tree-structured data into a Spine database from various formats. The same functionality is also available in **Data store view** from **File->Import** but using an Importer item is preferred because then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

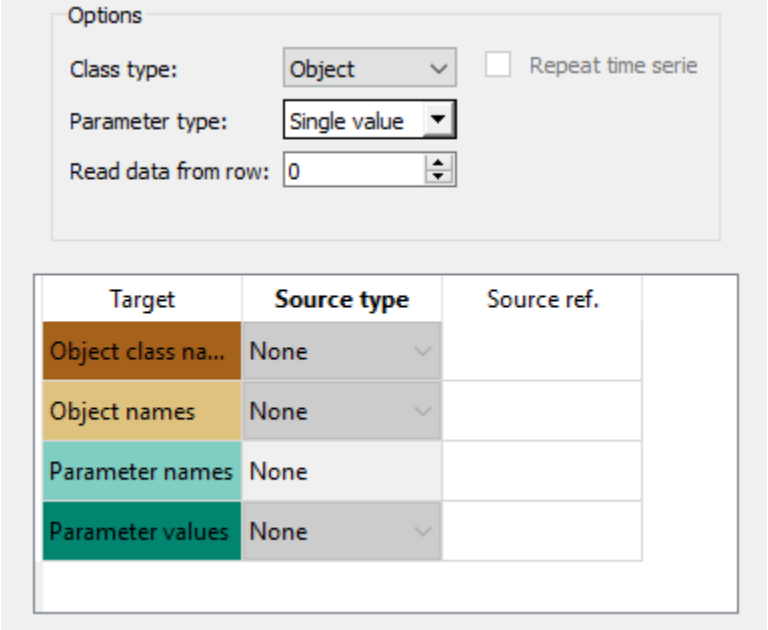
The heart of Importer is the **Import Editor** window in which the mappings from source data to Spine database entities are set up. The editor window can be accessed by the **Import Editor...** button in Importer's Properties tab. Note, that you have to select one of the files in the **Source files**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



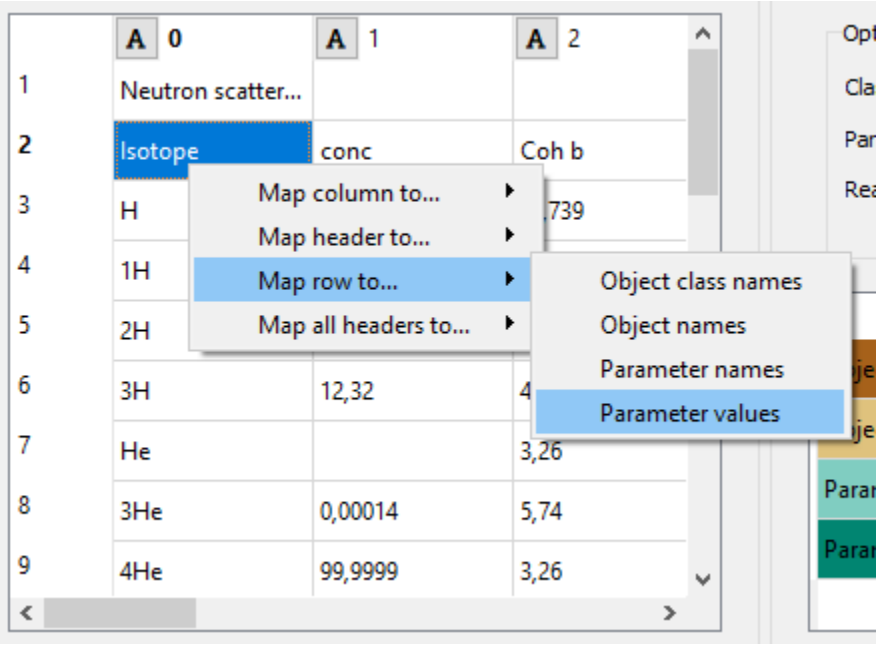
The **Import Editor** windows is divided into two parts: **Sources** shows all the 'sheets' contained in the file, some options for reading the file correctly, and a preview table to visualize and configure how the data on the selected sheet would be mapped. **Mappings**, on the other hand, shows the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



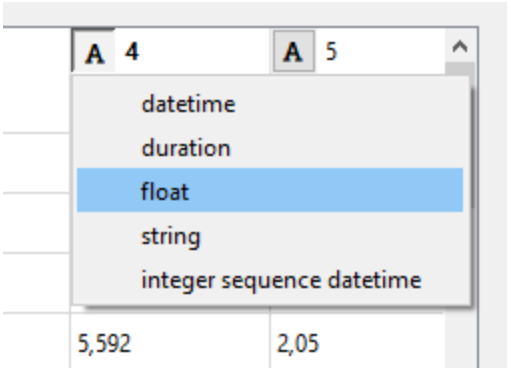
The options in the Mappings part declare if the currently selected sheet will be imported as an object or relationship and what type of parameters, if any, the sheet contains. The table can be used to configure how the input data is interpreted: which row or column contains the entity class names, parameter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

To export a Spine database to an Excel file, select a **Data store** and open the **Data store view**. Then select **File -> Export** from the main menu.

Tip: An easy way to get an Excel template is to export an existing Spine database to Excel.

11.2.1 Format

The exported Excel files are formatted in the following way:

Object classes:

	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

Object timeseries:

	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	

Relationship classes:

	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	

Relationship timeseries:

	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				

When exporting all object classes and relationship classes are exported. Only parameter values with time series data are exported in the time series format.

11.3 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)**

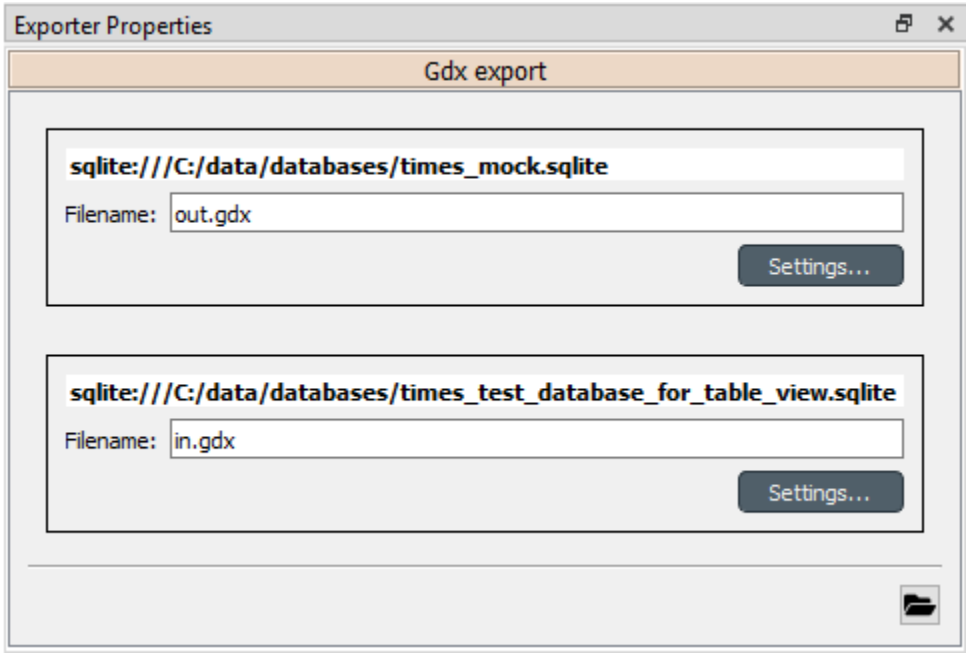
instead of **bar(foo)**.

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.

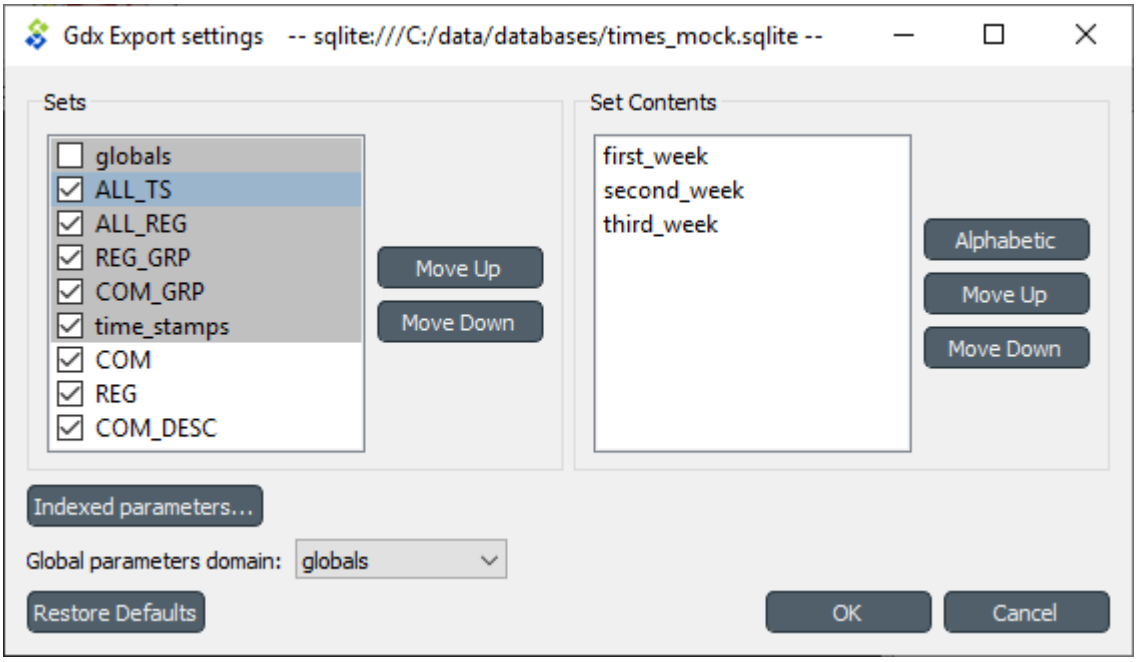
Some GAMS models need their data to be in a specific order in the `.gdx`. This is not directly supported by the database. Rather, user has to specify the desired exporting order using the *Exporter* item's settings.

11.3.1 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.

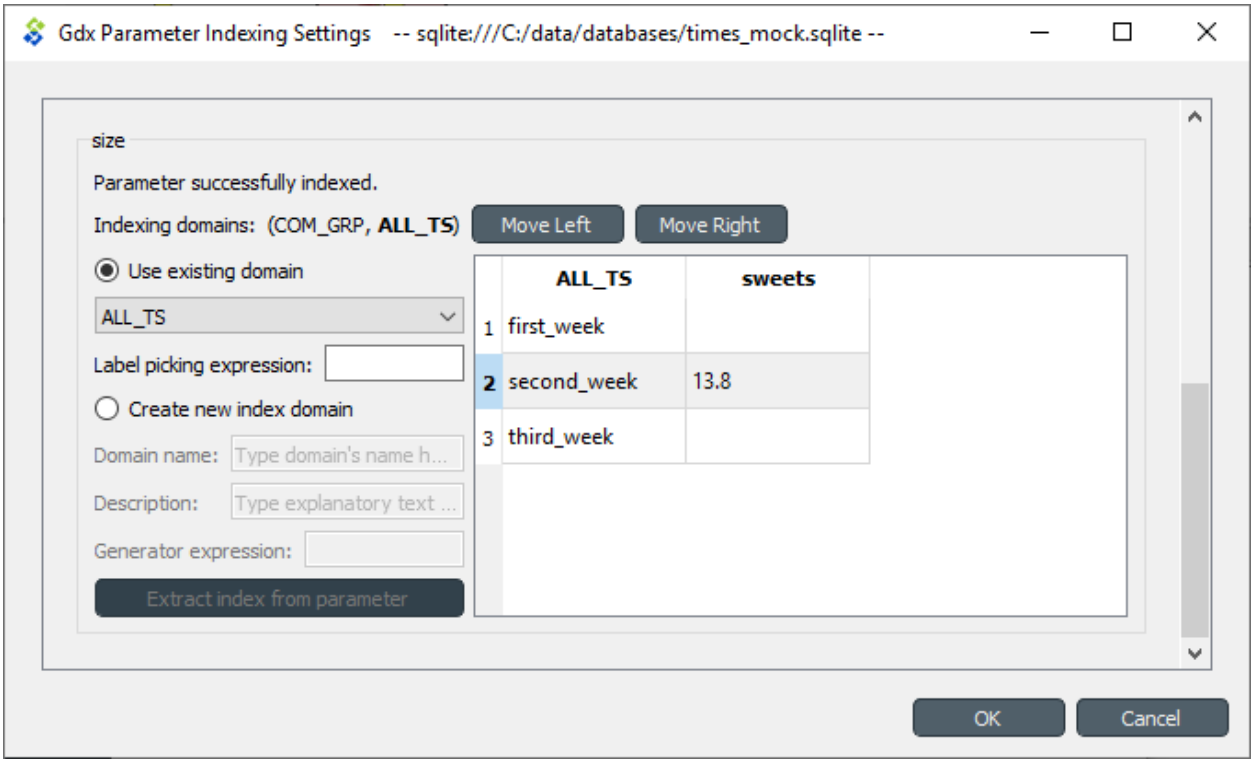
The *Set Contents* box lists the members of the selected set or subset. Their order of export can be changed the same way as with sets by *Move Up* and *Move Down*. The *Alphabetic* button sorts the members alphabetically.

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.

11.3.2 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.

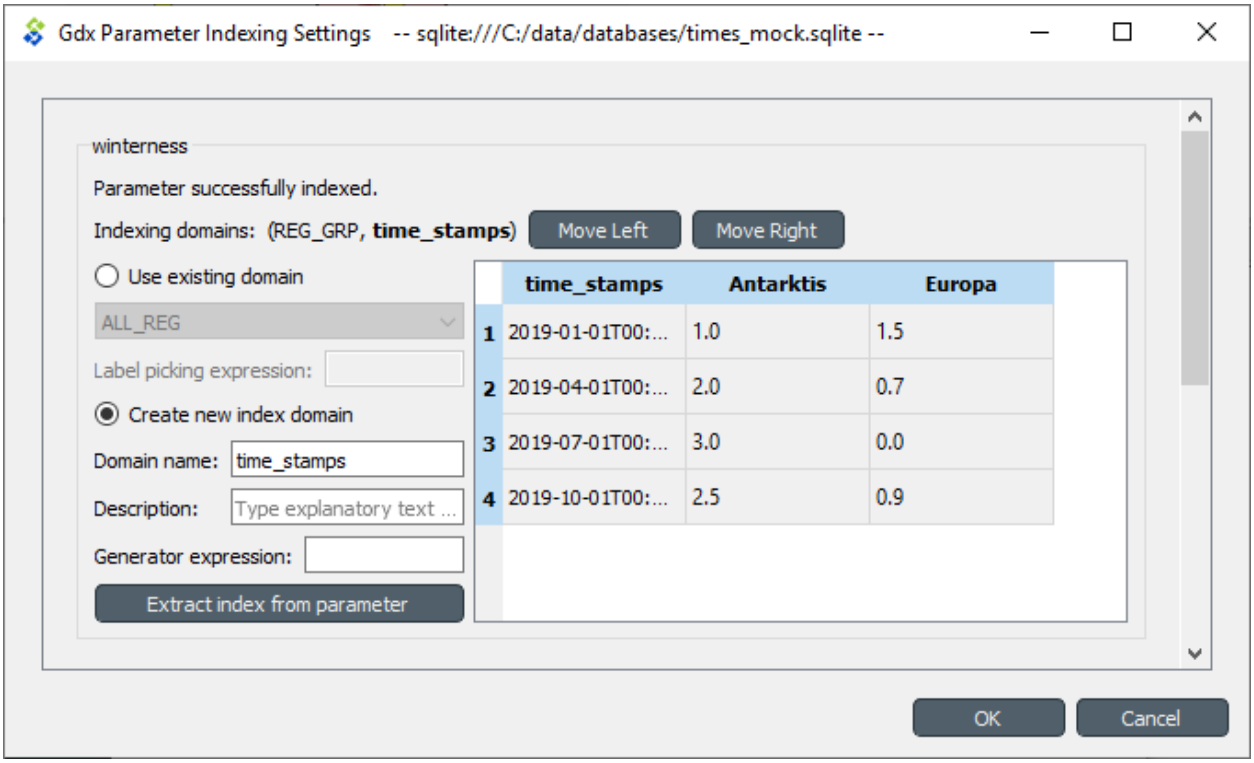


The above figure shows the indexing settings when an existing GAMS set is used to replace the original time stamps of a time series in a parameter called 'size'. The choice between using an existing set or creating a new one can be changed by the *Use existing domain* and *Create new index domain* radio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

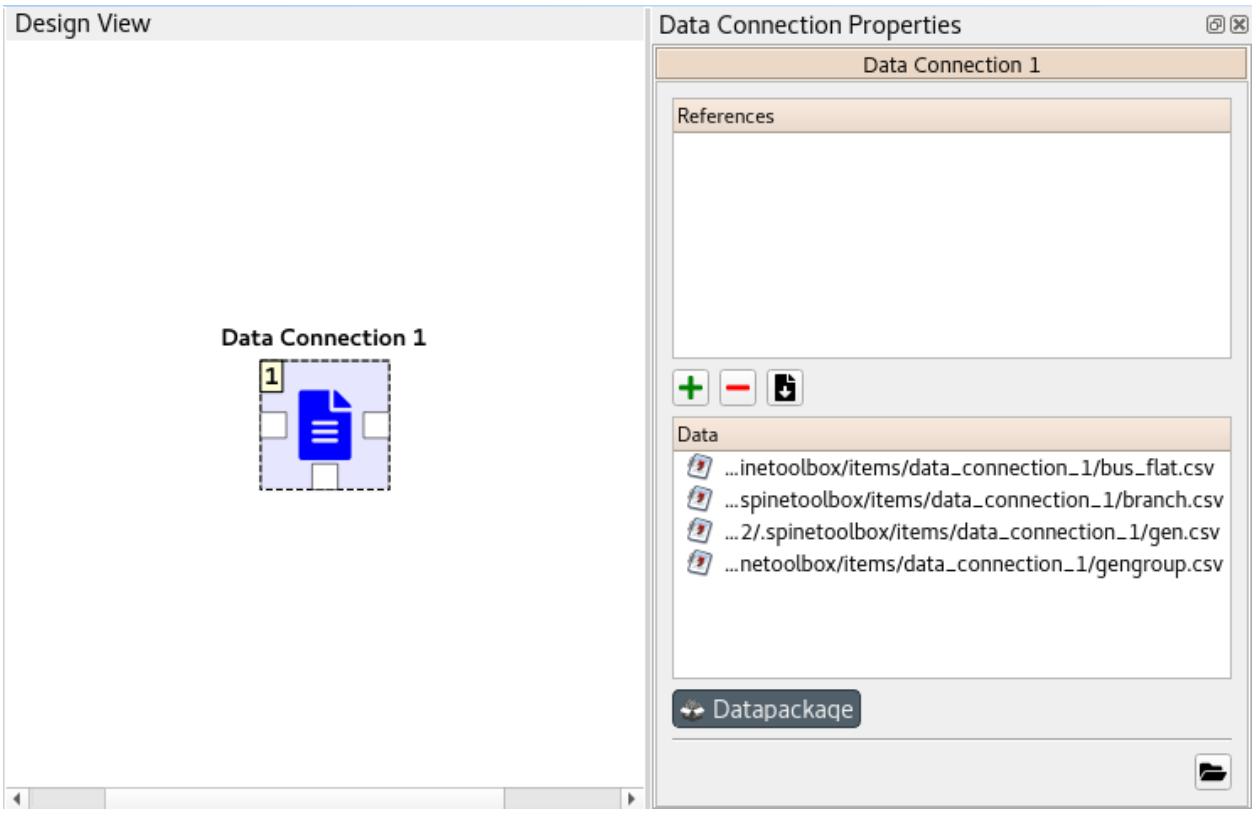
CHAPTER 12

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, Spine Model, 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 Spine Model 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 Spine Model will be able to directly utilize as well as output them.
- **Spine Model** An interpreter, which formulates a solver-ready mixed-integer optimization problem based on the input data and the equations defined in the Spine Model. 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. Spine Model 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 Spine Model 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 less than 5.12, which 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	<5.12	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

API Reference

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

16.1 spinetoolbox

spinetoolbox package.

16.1.1 Subpackages

spinetoolbox.configuration_assistants

Standard assistants.

author

M. Marin (KTH)

date 17.2.2020

Subpackages

spinetoolbox.configuration_assistants.spine_model

SpineModel.jl config assistant.

author

M. Marin (KTH)

date 17.2.2020

[1] Created with sphinx-autoapi

Submodules

spinetoolbox.configuration_assistants.spine_model.configuration_assistant

Widget for assisting the user in configuring SpineModel.jl.

author

M. Marin (KTH)

date 9.1.2019

Module Contents

class spinetoolbox.configuration_assistants.spine_model.configuration_assistant.**SpineModelC**

Bases: *spinetoolbox.widgets.state_machine_widget.StateMachineWidget*

_required_julia_version = 1.1.0

py_call_program_check_needed

spine_model_process_failed

py_call_installation_needed

py_call_reconfiguration_needed

py_call_process_failed

spine_model_ready

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_model(*self*)

_make_prompt_to_install_latest_spine_model(*self*)

_make_installing_latest_spine_model(*self*)

_make_report_spine_model_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_model_ready(*self*)

_make_reconfiguring_py_call(*self*)

_make_installing_py_call(*self*)

_make_report_py_call_process_failed(*self*)

update_spine_model(*self*)

install_spine_model(*self*)

_handle_spine_model_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 given python program.

_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

class spinetoolbox.configuration_assistants.spine_model.**make_assistant**(*toolbox*)
Bases: *spinetoolbox.widgets.state_machine_widget.StateMachineWidget*

_required_julia_version = 1.1.0

py_call_program_check_needed

spine_model_process_failed

py_call_installation_needed

py_call_reconfiguration_needed

py_call_process_failed

spine_model_ready

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_model(*self*)

_make_prompt_to_install_latest_spine_model(*self*)

_make_installing_latest_spine_model(*self*)

_make_report_spine_model_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_model_ready(*self*)

_make_reconfiguring_py_call(*self*)

_make_installing_py_call(*self*)

_make_report_py_call_process_failed(*self*)

update_spine_model(*self*)

install_spine_model(*self*)

_handle_spine_model_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 given python program.

_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_model.**assistant_name = SpineModel.jl**

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.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

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundParameterModel**(*parent*, *db_mngr*, **db_maps*)

Bases: *spinetoolbox.mvcmodels.compound_table_model.CompoundWithEmptyTableModel*

A model that concatenates several single parameter models and one empty parameter model.

Initializes model.

Parameters

- **parent** (DataStoreForm) – the parent object
- **db_mngr** (SpineDBManager) – the database manager
- ***db_maps** (*DiffDatabaseMapping*) – the database maps included in the model

remove_selection_requested

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 of the entity class id in the model items (either "object_class_id" or "relationship_class_id")

Returns str

_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

fetchMore(*self*, *parent=QModelIndex()*)
Populates filter menus as submodels are fetched.

_add_data_to_filter_menus(*self*, *sub_model*)
Adds data of given sub-model to filter menus.

Parameters **sub_model** (SingleParameterModel) –

_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.

_get_entity_classes(*self*, *db_map*)
Returns a list of entity classes from the given db_map.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns list

_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 should be included in this compound model.

Parameters **model** (SingleParameterModel, EmptyParameterModel) –

Returns bool

_main_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

update_main_filter(*self*)
Updates and applies the main filter.

update_compound_main_filter(*self*)
Updates the main filter in the compound model by setting the _accepted_entity_class_ids attribute.

Returns True if the filter was updated, None otherwise

Return type bool

update_single_main_filter(*self*, *model*)
Updates the filter in the given single model by setting its _selected_param_def_ids attribute.

Parameters **model** (SingleParameterModel) –

Returns True if the filter was updated, None otherwise

Return type bool

update_auto_filter(*self*, *field*, *valid_values*, *has_filter*)
Updates and applies the auto filter.

Parameters

- **field** (*str*) – the field name
- **valid_values** (*list(str)*) – accepted values for the field
- **has_filter** (*bool*) –

static _build_auto_filter(*field_menu_data*, *valid_values*, *has_filter*)

update_compound_auto_filter(*self*, *field*, *auto_filter*)
Updates 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

update_single_auto_filter(*self*, *model*, *field*)
Updates 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

_item_ids_per_class_id(*self*, *items*)
Returns a dict mapping entity class ids to a set of item ids.

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

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*)

value_name(*self*, *index*)

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundObjectParameterMixin**
Implements the interface for populating and filtering a compound object parameter model.

entity_class_type

_get_entity_classes(*self*, *db_map*)

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundRelationshipParameterMixin**
Implements the interface for populating and filtering a compound relationship parameter model.

entity_class_type

_get_entity_classes(*self*, *db_map*)

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundParameterDefinitionMixin**
Handles signals from db mngr for parameter definition models.

item_type

receive_parameter_definition_tags_set(*self*, *db_map_data*)

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundParameterValueMixin**
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

update_single_main_filter(*self*, *model*)
Update the filter for the given model.

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundObjectParameterDefinitionMo**
Bases: *spinetoolbox.mvcmodels.compound_parameter_models.CompoundObjectParameterMixin*, *spinetoolbox.mvcmodels.compound_parameter_models.CompoundParameterDefinitionMixin*, *spinetoolbox.mvcmodels.compound_parameter_models.CompoundParameterModel*

A model that concatenates several single object parameter definition models and one empty object parameter definition model.

_make_header(*self*)

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundRelationshipParameterDefinit**
Bases: *spinetoolbox.mvcmodels.compound_parameter_models.CompoundRelationshipParameterMixin*, *spinetoolbox.mvcmodels.compound_parameter_models.CompoundParameterDefinitionMixin*, *spinetoolbox.mvcmodels.compound_parameter_models.CompoundParameterModel*

A model that concatenates several single relationship parameter definition models and one empty relationship parameter definition model.

_make_header(*self*)

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundObjectParameterValueModel**
Bases: *spinetoolbox.mvcmodels.compound_parameter_models.CompoundObjectParameterMixin*, *spinetoolbox.mvcmodels.compound_parameter_models.CompoundParameterValueMixin*, *spinetoolbox.mvcmodels.compound_parameter_models.CompoundParameterModel*

A model that concatenates several single object parameter value models and one empty object parameter value model.

entity_type

_make_header(*self*)

class spinetoolbox.mvcmodels.compound_parameter_models.**CompoundRelationshipParameterValueM**
Bases: *spinetoolbox.mvcmodels.compound_parameter_models.CompoundRelationshipParameterMixin*, *spinetoolbox.mvcmodels.compound_parameter_models.CompoundParameterValueMixin*, *spinetoolbox.mvcmodels.compound_parameter_models.CompoundParameterModel*

A model that concatenates several single relationship parameter value models and one empty relationship parameter value model.

entity_type

_make_header(*self*)

spinetoolbox.mvcmodels.compound_table_model

Models that vertically concatenate two or more table models.

authors

M. Marin (KTH)

date 9.10.2019

Module Contents

class spinetoolbox.mvcmodels.compound_table_model.**CompoundTableModel**(*parent*, *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

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*)

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

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()*)
Insert count rows after the given row under the given parent. Localizes the appropriate submodel and calls insertRows on it.

class spinetoolbox.mvcmodels.compound_table_model.**CompoundWithEmptyTableModel**
Bases: *spinetoolbox.mvcmodels.compound_table_model.CompoundTableModel*

A compound parameter table model where the last model is an empty row model.

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

class spinetoolbox.mvcmodels.data_package_models.**DatapackageResourcesModel**(*parent*)
Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A model of datapackage resource data, used by SpineDatapackageWidget.

Parameters **parent** (SpineDatapackageWidget) –

reset_model(*self*, *resources*)

flags(*self*, *index*)

class spinetoolbox.mvcmodels.data_package_models.**DatapackageFieldsModel**(*parent*)
Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A model of datapackage field data, used by SpineDatapackageWidget.

Parameters **parent** (SpineDatapackageWidget) –

reset_model(*self*, *schema*)

class spinetoolbox.mvcmodels.data_package_models.**DatapackageForeignKeysModel**(*parent*)
Bases: *spinetoolbox.mvcmodels.empty_row_model.EmptyRowModel*

A model of datapackage foreign key data, used by SpineDatapackageWidget.

Parameters **parent** (SpineDatapackageWidget) –

reset_model(*self*, *foreign_keys*)

spinetoolbox.mvcmodels.empty_parameter_models

Empty models for parameter definitions and values.

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

class spinetoolbox.mvcmodels.empty_parameter_models.**EmptyParameterModel**(*parent*, *header*, *db_mngr*)

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) –

entity_class_type
Either 'object class' or 'relationship class'.

entity_class_id_key

entity_class_name_key

can_be_filtered

accepted_rows(*self*)

db_item(*self*, *_index*)

flags(*self*, *index*)

_make_unique_id(*self*, *item*)
Returns a unique id for the given model item (name-based). Used by receive_parameter_data_added.

get_entity_parameter_data(*self*, *db_map*, *ids=None*)
Returns object or relationship parameter definitions or values. Must be reimplemented in subclasses according to the entity type and to whether it's a definition or value model. 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.mvcmodels.empty_parameter_models.**EmptyParameterDefinitionModel**
Bases: *spinetoolbox.mvcmodels.parameter_mixins.FillInValueListIdMixin*, *spinetoolbox.mvcmodels.parameter_mixins.FillInEntityClassIdMixin*, *spinetoolbox.mvcmodels.parameter_mixins.FillInParameterNameMixin*, *spinetoolbox.mvcmodels.empty_parameter_models.EmptyParameterModel*

An empty parameter definition model.

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.mvcmodels.empty_parameter_models.**EmptyObjectParameterDefinitionModel**
Bases: *spinetoolbox.mvcmodels.empty_parameter_models.EmptyParameterDefinitionModel*

An empty object parameter definition model.

entity_class_type

get_entity_parameter_data(*self*, *db_map*, *ids=None*)
Returns object parameter definitions. Used by receive_parameter_data_added.

class spinetoolbox.mvcmodels.empty_parameter_models.**EmptyRelationshipParameterDefinitionMo**
Bases: *spinetoolbox.mvcmodels.empty_parameter_models.EmptyParameterDefinitionModel*

An empty relationship parameter definition model.

entity_class_type

get_entity_parameter_data(*self*, *db_map*, *ids=None*)
Returns relationship parameter definitions. Used by receive_parameter_data_added.

flags(*self*, *index*)
Additional hack to make the object_class_name_list column non-editable.

class spinetoolbox.mvcmodels.empty_parameter_models.**EmptyParameterValueModel**
Bases: *spinetoolbox.mvcmodels.parameter_mixins.InferEntityClassIdMixin*, *spinetoolbox.mvcmodels.parameter_mixins.FillInParameterDefinitionIdsMixin*, *spinetoolbox.mvcmodels.parameter_mixins.FillInEntityIdsMixin*, *spinetoolbox.mvcmodels.parameter_mixins.FillInEntityClassIdMixin*, *spinetoolbox.mvcmodels.empty_parameter_models.EmptyParameterModel*

An empty parameter value model.

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.mvcmodels.empty_parameter_models.**EmptyObjectParameterValueModel**
Bases: *spinetoolbox.mvcmodels.empty_parameter_models.EmptyParameterValueModel*

An empty object parameter value model.

entity_class_type

entity_type

get_entity_parameter_data(*self*, *db_map*, *ids=None*)
Returns object parameter values. Used by receive_parameter_data_added.

class spinetoolbox.mvcmodels.empty_parameter_models.**EmptyRelationshipParameterValueModel**
Bases: *spinetoolbox.mvcmodels.parameter_mixins.MakeRelationshipOnTheFlyMixin*, *spinetoolbox.mvcmodels.empty_parameter_models.EmptyParameterValueModel*

An empty relationship parameter value model.

_add_entities_on_the_fly = True

entity_class_type

entity_type

get_entity_parameter_data(*self*, *db_map*, *ids=None*)
Returns relationship parameter values. Used by receive_parameter_data_added.

add_items_to_db(*self*, *rows*)
Add items to db.

Parameters **rows** (*set*) – add data from these rows

spinetoolbox.mvcmodels.empty_row_model

Contains a table model with an empty last row.

authors

M. Marin (KTH)

date 20.5.2018

Module Contents

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()*)

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

flags (*self*, *index*)
Return default flags except if forcing defaults.

set_default_row (*self*, ***kwargs*)
Set default row data.

clear (*self*)

reset_model (*self*, *main_data=None*)

_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.entity_list_models

List models for object and relationship classes.

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

class spinetoolbox.mvcmodels.entity_list_models.**EntityListModel** (*graph_view_form*, *db_mngr*, *db_map*)

Bases: PySide2.QtGui.QStandardItemModel

A model for listing entity classes in the GraphViewForm.

Initialize class

add_more_icon

entity_type

_get_entity_class_ids (*self*)

populate_list (*self*)
Populate model.

add_entity_class (*self*, *entity_class_id*)
Add entity class item to model.

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

_data (*self*, *index*)

receive_entity_classes_added (*self*, *db_map_data*)
Runs when entity classes are added.

receive_entity_classes_updated(*self*, *db_map_data*)
Runs when entity classes are update.

receive_entity_classes_removed(*self*, *db_map_data*)
Runs when entity classes are removed.

flags(*self*, *index*)

class spinetoolbox.mvcmodels.entity_list_models.**ObjectClassListModel**
Bases: *spinetoolbox.mvcmodels.entity_list_models.EntityListModel*

A model for listing object classes in the GraphViewForm.

add_more_icon

entity_type

_get_entity_class_ids(*self*)

class spinetoolbox.mvcmodels.entity_list_models.**RelationshipClassListModel**
Bases: *spinetoolbox.mvcmodels.entity_list_models.EntityListModel*

A model for listing relationship classes in the GraphViewForm.

add_more_icon

entity_type

_get_entity_class_ids(*self*)

spinetoolbox.mvcmodels.entity_tree_item

Classes to represent entities in a tree.

authors

P. Vennström (VTT), M. Marin (KTH)

date 11.3.2019

Module Contents

class spinetoolbox.mvcmodels.entity_tree_item.**MultiDBTreeItem**(*model=None*, *db_map_id=None*)
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_name
"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.

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_remove_db_map(*self*, *db_map*)
Removes given db_map from this item and all its descendants.

deep_take_db_map(*self*, *db_map*)
Takes given db_map from this item and all its descendants. Returns a new item from taken data or None if db_map is not present in the first place.

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

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 first id is True, 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.

class spinetoolbox.mvcmodels.entity_tree_item.**TreeRootItem**
Bases: *spinetoolbox.mvcmodels.entity_tree_item.MultiDBTreeItem*

item_type = root

display_id
"See super class.

display_name
"See super class.

class spinetoolbox.mvcmodels.entity_tree_item.**ObjectTreeRootItem**(**args*, ***kwargs*)
Bases: *spinetoolbox.mvcmodels.entity_tree_item.TreeRootItem*

An object tree root item.

child_item_type
Returns an ObjectClassItem.

_get_children_ids(*self*, *db_map*)
Returns a list of object class ids.

class spinetoolbox.mvcmodels.entity_tree_item.**RelationshipTreeRootItem**(**args*, ***kwargs*)
Bases: *spinetoolbox.mvcmodels.entity_tree_item.TreeRootItem*

A relationship tree root item.

child_item_type
Returns a RelationshipClassItem.

_get_children_ids(*self*, *db_map*)
Returns a list of relationship class ids.

class spinetoolbox.mvcmodels.entity_tree_item.**EntityClassItem**
Bases: *spinetoolbox.mvcmodels.entity_tree_item.MultiDBTreeItem*

An entity class item.

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

class spinetoolbox.mvcmodels.entity_tree_item.**ObjectClassItem**(**args*, ***kwargs*)
Bases: *spinetoolbox.mvcmodels.entity_tree_item.EntityClassItem*

An object class item.

item_type = object class

display_icon
Returns the object class icon.

child_item_type
Returns an ObjectItem.

_get_children_ids(*self*, *db_map*)
Returns a list of object ids in this class.

default_parameter_data(*self*)
Return data to put as default in a parameter table when this item is selected.

class spinetoolbox.mvcmodels.entity_tree_item.**RelationshipClassItem**(**args*, ***kwargs*)
Bases: *spinetoolbox.mvcmodels.entity_tree_item.EntityClassItem*

A relationship class item.

visual_key = ['name', 'object_class_name_list']

item_type = relationship class

display_icon
Returns relationship class icon.

child_item_type
Returns a RelationshipItem.

_get_children_ids(*self*, *db_map*)
Returns a list of relationship ids in this class. If the parent is an ObjectItem, then only returns ids of relationships involving that object.

default_parameter_data(*self*)
Return data to put as default in a parameter table when this item is selected.

class spinetoolbox.mvcmodels.entity_tree_item.**EntityItem**
Bases: *spinetoolbox.mvcmodels.entity_tree_item.MultiDBTreeItem*

An entity item.

data(*self*, *column*, *role=Qt.DisplayRole*)
Returns data for given column and role.

class spinetoolbox.mvcmodels.entity_tree_item.**ObjectItem**(**args*, ***kwargs*)
Bases: *spinetoolbox.mvcmodels.entity_tree_item.EntityItem*

An object item.

item_type = object

child_item_type
Returns a RelationshipClassItem.

display_icon
Returns the object class icon.

_get_children_ids(*self*, *db_map*)
Returns a list of relationship class ids involving this item's class.

default_parameter_data(*self*)
Return data to put as default in a parameter table when this item is selected.

class spinetoolbox.mvcmodels.entity_tree_item.**RelationshipItem**(**args*, ***kwargs*)
Bases: *spinetoolbox.mvcmodels.entity_tree_item.EntityItem*

An object item.

Overridden method to parse some data for convenience later. Also make sure we never try to fetch this item.

visual_key = ['name', 'object_name_list']

item_type = relationship

object_name_list

display_name
"Returns the name for display.

display_icon
Returns relationship class icon.

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.

_get_children_ids (*self*, *db_map*)
See base class.

spinetoolbox.mvcmodels.entity_tree_models

Models to represent entities in a tree.

authors

P. Vennström (VTT), M. Marin (KTH)

date 11.3.2019

Module Contents

class spinetoolbox.mvcmodels.entity_tree_models.**EntityTreeModel**(*parent*, *db_mngr*, **db_maps*)
Bases: *spinetoolbox.mvcmodels.minimal_tree_model.MinimalTreeModel*

Base class for all entity tree models.

Init class.

Parameters

- **parent** (DataStoreForm) –
- **db_mngr** (SpineDBManager) – A manager for the given db_maps
- **db_maps** (*iter*) – DiffDatabaseMapping instances

remove_selection_requested

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*)

headerData (*self*, *section*, *orientation*, *role*)

_select_index (*self*, *index*)
Marks the index as selected.

select_indexes (*self*, *indexes*)
Marks given indexes as selected.

find_leaves(*self*, *db_map*, **ids_path*, *parent_items=()*, *fetch=False*)
Returns leaf-nodes following the given path of ids, where each element in ids_path is a set of ids to jump from one level in the tree to the next. Optionally fetches nodes as it goes.

class spinetoolbox.mvcmodels.entity_tree_models.**ObjectTreeModel**(**args*, ***kwargs*)
Bases: *spinetoolbox.mvcmodels.entity_tree_models.EntityTreeModel*

An 'object-oriented' tree model.

root_item_type

selected_object_class_indexes

selected_object_indexes

selected_relationship_class_indexes

selected_relationship_indexes

_group_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 result (dict)

_group_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 result (dict)

_group_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 result (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*)

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.mvcmodels.entity_tree_models.**RelationshipTreeModel**(**args*, ***kwargs*)

Bases: *spinetoolbox.mvcmodels.entity_tree_models.EntityTreeModel*

A relationship-oriented tree model.

root_item_type

selected_relationship_class_indexes

selected_relationship_indexes

_group_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 result (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.mvcmodels.filter_checkbox_list_model

Provides FilterCheckboxListModel for FilterWidget.

author

P. Vennström (VTT)

date 1.11.2018

Module Contents

class spinetoolbox.mvcmodels.filter_checkbox_list_model.**SimpleFilterCheckboxListModel**(*parent*, *show_*

Bases: PySide2.QtCore.QAbstractListModel

Init class.

Parameters **parent** (*QWidget*) –

reset_selection(*self*)

_select_all_clicked(*self*)

_check_all_selected(*self*)

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

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

click_index(*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*, *search_for*)

apply_filter(*self*)

_remove_and_add_filtered(*self*)

_remove_and_replace_filtered(*self*)

remove_filter(*self*)

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** (DataStoreForm) –
- **source_model** (CompoundParameterModel) – a model to lazily get data from

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

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

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** (DataStoreForm) –
- **data_to_value** (*method*) – a method to translate item data to a value for display role

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

spinetoolbox.mvcmodels.frozen_table_model

Contains FrozenTableModel class.

author

P. Vennström (VTT)

date 24.9.2019

Module Contents

class spinetoolbox.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.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

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

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.

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.**_reconstruct_map**(*rows*, *first_row*, *last_row*, *column_index*)

spinetoolbox.mvcmodels.minimal_table_model

Contains a minimal table model.

authors

M. Marin (KTH)

date 20.5.2018

Module Contents

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

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 meaningfull.

children

parent_item

display_name

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 0 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.

class spinetoolbox.mvcmodels.minimal_tree_model.**MinimalTreeModel**(*parent=None*)
Bases: PySide2.QtCore.QAbstractItemModel

Base class for all tree models.

Init class.

Parameters **parent** (DataStoreForm) –

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.parameter_mixins

Miscelaneous mixins for parameter models

authors

M. Marin (KTH)

date 4.10.2019

Module Contents

spinetoolbox.mvcmodels.parameter_mixins.**_parse_csv_list** (*csv_list*)

class spinetoolbox.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.mvcmodels.parameter_mixins.**FillInParameterNameMixin**
Bases: *spinetoolbox.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.mvcmodels.parameter_mixins.**FillInValueListIdMixin**(**args*, ***kwargs*)

Bases: *spinetoolbox.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.mvcmodels.parameter_mixins.**MakeParameterTagMixin**(**args*, ***kwargs*)

Bases: *spinetoolbox.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.mvcmodels.parameter_mixins.**FillInEntityClassIdMixin**(**args*, ***kwargs*)

Bases: *spinetoolbox.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.mvcmodels.parameter_mixins.**FillInEntityIdsMixin**(**args*, ***kwargs*)

Bases: *spinetoolbox.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.mvcmodels.parameter_mixins.**FillInParameterDefinitionIdsMixin**(**args*, ***kwargs*)
Bases: *spinetoolbox.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.mvcmodels.parameter_mixins.**InferEntityClassIdMixin**
Bases: *spinetoolbox.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.mvcmodels.parameter_mixins.**MakeRelationshipOnTheFlyMixin**(**args*, ***kwargs*)
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.mvcmodels.parameter_value_list_model

A tree model for parameter value lists.

authors

M. Marin (KTH)

date 28.6.2019

Module Contents

class spinetoolbox.mvcmodels.parameter_value_list_model.**EditableMixin**

flags(*self*, *column*)
Makes items editable.

class spinetoolbox.mvcmodels.parameter_value_list_model.**GrayFontMixin**
Paints the text gray.

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

class spinetoolbox.mvcmodels.parameter_value_list_model.**BoldFontMixin**
Bolds text.

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

class spinetoolbox.mvcmodels.parameter_value_list_model.**AppendEmptyChildMixin**
Provides a method to append an empty child if needed.

append_empty_child(*self*, *row*)
Append empty child if the row is the last one.

class spinetoolbox.mvcmodels.parameter_value_list_model.**DBItem**(*db_map*)
Bases: *spinetoolbox.mvcmodels.parameter_value_list_model.AppendEmptyChildMixin*, *spinetoolbox.mvcmodels.minimal_tree_model.TreeItem*

An item representing a db.

Init class.

Args db_mngr (SpineDBManager) db_map (DiffDatabaseMapping)

db_mngr

fetch_more(*self*)

empty_child(*self*)

data(*self*, *column*, *role=Qt.DisplayRole*)
Shows Spine icon for fun.

class spinetoolbox.mvcmodels.parameter_value_list_model.**ListItem**(*db_map*, *identifier=None*, *name=None*, *value_list=()*)
Bases: *spinetoolbox.mvcmodels.parameter_value_list_model.GrayFontMixin*, *spinetoolbox.mvcmodels.parameter_value_list_model.BoldFontMixin*, *spinetoolbox.mvcmodels.parameter_value_list_model.AppendEmptyChildMixin*, *spinetoolbox.mvcmodels.parameter_value_list_model.EditableMixin*, *spinetoolbox.mvcmodels.minimal_tree_model.TreeItem*

A list item.

db_mngr

fetch_more(*self*)

compile_value_list(*self*)

empty_child(*self*)

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

set_data(*self*, *column*, *name*)

set_child_data(*self*, *child*, *value*)

update_name_in_db(*self*, *name*)

update_value_list_in_db(*self*, *child*, *value*)

add_to_db(*self*)
Add item to db.

handle_updated_in_db(*self*, *name*, *value_list*)
Runs when an item with this id has been updated in the db.

handle_added_to_db(*self*, *identifier*, *value_list*)
Runs when the item with this name has been added to the db.

reset_value_list(*self*, *value_list*)

class spinetoolbox.mvcmodels.parameter_value_list_model.**ValueItem**(*value=None*)
Bases: *spinetoolbox.mvcmodels.parameter_value_list_model.GrayFontMixin*, *spinetoolbox.mvcmodels.parameter_value_list_model.EditableMixin*, *spinetoolbox.mvcmodels.minimal_tree_model.TreeItem*

A value item.

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

set_data(*self*, *column*, *value*)

class spinetoolbox.mvcmodels.parameter_value_list_model.**ParameterValueListModel**(*parent*, *db_mngr*, **db_maps*)
Bases: *spinetoolbox.mvcmodels.minimal_tree_model.MinimalTreeModel*

A model to display parameter value list data in a tree view.

Parameters

- **parent** (DataStoreForm) –
- **db_mngr** (SpineDBManager) –
- **db_maps** (*iter*) – DiffDatabaseMapping instances

Initialize class

remove_selection_requested

remove_icon

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.

spinetoolbox.mvcmodels.pivot_model

Provides PivotModel.

author

P. Vennström (VTT)

date 1.11.2018

Module Contents

class spinetoolbox.mvcmodels.pivot_model.**PivotModel**

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.mvcmodels.pivot_table_models

Provides pivot table models for the Tabular View.

author

P. Vennström (VTT)

date 1.11.2018

Module Contents

class spinetoolbox.mvcmodels.pivot_table_models.**PivotTableModel**(*parent*)

Bases: PySide2.QtCore.QAbstractTableModel

Parameters **parent** (*TabularViewForm*) –

_V_HEADER_WIDTH = 5

_ITEMS_TO_FETCH = 1024

plot_x_column
Returns the index of the column designated as Y values for plotting or None.

reset_data_count(*self*)

canFetchMore(*self*, *parent*)

fetchMore(*self*, *parent*)

fetch_more_rows(*self*, *parent*)

fetch_more_columns(*self*, *parent*)

reset_model(*self*, *data*, *index_ids*, *rows=()*, *columns=()*, *frozen=()*, *frozen_value=()*)

clear_model(*self*)

update_model(*self*, *data*)

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 Y flag on a column

first_data_row(*self*)
Returns the row index to the first data row.

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

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*, *top_left_id*, *header_id*)
Returns the name of the header given by top_left_id and header_id.

Parameters

- **top_left_id** (*int*) – The id of the top left header
- **header_id** (*int*) – The header id

Returns str

header_name(*self*, *index*)
Returns the name corresponding to the given header index.

Parameters **index** (*QModelIndex*) –

Returns str

header_data(*self*, *index*, *role=Qt.DisplayRole*)
Returns the data corresponding to the given header index based on role enum.

Parameters

- **index** (*QModelIndex*) –
- **role** (*enum Qt.ItemDataRole*) –

Returns str

header_names(*self*, *index*)
Returns the header names corresponding to the given data index.

Parameters **index** (*QModelIndex*) –

Returns object names str: parameter name

Return type list(str)

value_name(*self*, *index*)
Returns a string that concatenates the header names corresponding to the given data index.

Parameters **index** (*QModelIndex*) –

Returns str

column_name(*self*, *column*)
Returns a string that concatenates the header names corresponding to the given column.

Parameters **column** (*int*) –

Returns str

_color_data(*self*, *index*)

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*)

_batch_set_parameter_value_data(*self*, *row_map*, *column_map*, *data*, *values*)

_checked_parameter_values(*self*, *items*)

_add_parameter_values(*self*, *items*)

_update_parameter_values(*self*, *items*)

_batch_set_relationship_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*)

class spinetoolbox.mvcmodels.pivot_table_models.**PivotTableSortFilterProxy**(*parent=None*)

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.mvcmodels.project_item_model

Contains a class for storing project items.

authors

P. Savolainen (VTT)

date 23.1.2018

Module Contents

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.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

class spinetoolbox.mvcmodels.single_parameter_models.**SingleParameterModel**(*parent*, *header*, *db_mngr*, *db_map*, *entity_class_id*, *lazy=True*)

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

- **parent** (*CompoundParameterModel*) – the parent object
- **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.

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.

fetchMore(*self*, *parent=None*)
Fetch data and use it to reset the model.

_fetch_data(*self*)
Returns data to reset the model with and call it fetched. Reimplement in subclasses if you want to populate your model automatically.

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*)

_main_filter_accepts_row(*self*, *row*)
Applies the main filter, defined by the selections in the grand parent.

_auto_filter_accepts_row(*self*, *row*)
Applies the autofilter, defined by the autofilter drop down menu.

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.mvcmodels.single_parameter_models.**SingleObjectParameterMixin**
Associates a parameter model with a single object class.

entity_class_type

class spinetoolbox.mvcmodels.single_parameter_models.**SingleRelationshipParameterMixin**
Associates a parameter model with a single relationship class.

entity_class_type

class spinetoolbox.mvcmodels.single_parameter_models.**SingleParameterDefinitionMixin**
Bases: *spinetoolbox.mvcmodels.parameter_mixins.FillInParameterNameMixin*, *spinetoolbox.mvcmodels.parameter_mixins.FillInValueListIdMixin*, *spinetoolbox.mvcmodels.parameter_mixins.MakeParameterTagMixin*

A parameter definition model for a single entity class.

item_type

update_items_in_db(*self*, *items*)
Update items in db.

Parameters **item** (*list*) – dictionary-items

class spinetoolbox.mvcmodels.single_parameter_models.**SingleParameterValueMixin**(**args*, ***kwargs*)
Bases: *spinetoolbox.mvcmodels.parameter_mixins.ConvertToDBMixin*

A parameter value model for a single entity class.

item_type

_main_filter_accepts_row(*self*, *row*)
Reimplemented to filter objects.

update_items_in_db(*self*, *items*)
Update items in db.

Parameters **item** (*list*) – dictionary-items

class spinetoolbox.mvcmodels.single_parameter_models.**SingleObjectParameterDefinitionModel**
Bases: *spinetoolbox.mvcmodels.single_parameter_models.SingleObjectParameterMixin*, *spinetoolbox.mvcmodels.single_parameter_models.SingleParameterDefinitionMixin*, *spinetoolbox.mvcmodels.single_parameter_models.SingleParameterModel*

An object parameter definition model for a single object class.

_fetch_data(*self*)
Returns object parameter definition ids.

class spinetoolbox.mvcmodels.single_parameter_models.**SingleRelationshipParameterDefinitionM**
Bases: *spinetoolbox.mvcmodels.single_parameter_models.SingleRelationshipParameterMixin*, *spinetoolbox.mvcmodels.single_parameter_models.SingleParameterDefinitionMixin*, *spinetoolbox.mvcmodels.single_parameter_models.SingleParameterModel*

A relationship parameter definition model for a single relationship class.

_fetch_data(*self*)
Returns relationship parameter definition ids.

class spinetoolbox.mvcmodels.single_parameter_models.**SingleObjectParameterValueModel**
Bases: *spinetoolbox.mvcmodels.single_parameter_models.SingleObjectParameterMixin*, *spinetoolbox.mvcmodels.single_parameter_models.SingleParameterValueMixin*, *spinetoolbox.mvcmodels.single_parameter_models.SingleParameterModel*

An object parameter value model for a single object class.

_fetch_data(*self*)
Returns object parameter value ids.

class spinetoolbox.mvcmodels.single_parameter_models.**SingleRelationshipParameterValueModel**
Bases: *spinetoolbox.mvcmodels.single_parameter_models.SingleRelationshipParameterMixin*, *spinetoolbox.mvcmodels.single_parameter_models.SingleParameterValueMixin*, *spinetoolbox.mvcmodels.single_parameter_models.SingleParameterModel*

A relationship parameter value model for a single relationship class.

_fetch_data(*self*)
Returns relationship parameter value ids.

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

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

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

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

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

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.mvcmodels.tool_specification_model

Contains a class for storing Tool specifications.

authors

P. Savolainen (VTT)

date 23.1.2018

Module Contents

class spinetoolbox.mvcmodels.tool_specification_model.**ToolSpecificationModel**
Bases: PySide2.QtCore.QAbstractListModel

Class to store tools that are available in a project e.g. GAMS or Julia models.

rowCount (*self*, *parent=None*)
Must be reimplemented when subclassing. Returns the number of Tools in the model.

Parameters **parent** (*QModelIndex*) – Not used (because this is a list)

Returns Number of rows (available tools) 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 Tool

insertRow (*self*, *tool*, *row=None*, *parent=QModelIndex()*)
Insert row (tool specification) into model.

Parameters

- **tool** (Tool) – Tool added to the model
- **row** (*str*) – Row to insert tool to
- **parent** (*QModelIndex*) – Parent of child (not used)

Returns Void

removeRow (*self*, *row*, *parent=QModelIndex()*)
Remove row (tool specification) from model.

Parameters

- **row** (*int*) – Row to remove the tool from
- **parent** (*QModelIndex*) – Parent of tool on row (not used)

Returns Boolean variable

update_tool_specification (*self*, *row*, *tool*)
Update tool specification.

Parameters

- **row** (*int*) – Position of the tool to be updated
- **tool** (ToolSpecification) – new tool, to replace the old one

Returns Boolean value depending on the result of the operation

tool_specification (*self*, *row*)
Returns tool specification on given row.

Parameters **row** (*int*) – Row of tool specification

Returns ToolSpecification from tool specification list or None if given row is zero

find_tool_specification (*self*, *name*)
Returns tool specification with the given name.

Parameters **name** (*str*) – Name of tool specification to find

tool_specification_row (*self*, *name*)
Returns the row on which the given specification is located or -1 if it is not found.

tool_specification_index (*self*, *name*)
Returns the QModelIndex on which a tool specification with the given name is located or invalid index if it is not found.

spinetoolbox.project_items

Standard project item plugins.

author A.Soininen (VTT)

date 27.9.2019

Subpackages

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

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

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

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

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.data_connection

Module for data connection class.

author

P. Savolainen (VTT)

date 19.12.2017

Module Contents

class spinetoolbox.project_items.data_connection.data_connection.**DataConnection**(*name*, *description*, *x*, *y*, *toolbox*, *project*, *logger*, *references=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Data Connection class.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **references** (*list*) – a list of file paths

set_up(*self*)

static item_type()
See base class.

static 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.

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*)
Let user select references to files for this data connection.

remove_references(*self*, *checked=False*)
Remove selected references from reference list. Do not remove anything if there are no references selected.

do_remove_references(*self*, *references*)

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*, *path=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.

output_resources_forward(*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_icon

Module for data connection icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

class spinetoolbox.project_items.data_connection.data_connection_icon.**DataConnectionIcon**(*to*
x,
y,
w,
h,
na

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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

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.

Package Contents

class spinetoolbox.project_items.data_connection.**DataConnectionIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)

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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

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.

class spinetoolbox.project_items.data_connection.**item_maker**(*name*, *description*, *x*, *y*, *toolbox*, *project*, *logger*, *references=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Data Connection class.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **references** (*list*) – a list of file paths

set_up(*self*)

static item_type()
See base class.

static 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.

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*)
Let user select references to files for this data connection.

remove_references(*self*, *checked=False*)
Remove selected references from reference list. Do not remove anything if there are no references selected.

do_remove_references(*self*, *references*)

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*, *path=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.

output_resources_forward(*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.

class spinetoolbox.project_items.data_connection.**DataConnectionPropertiesWidget**(*toolbox*)
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

class spinetoolbox.project_items.data_connection.**AddDataConnectionWidget**(*toolbox*, *x*, *y*)
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_connection.**item_rank = 1**

spinetoolbox.project_items.data_connection.**item_category**

spinetoolbox.project_items.data_connection.**item_type**

spinetoolbox.project_items.data_connection.**item_icon = :/icons/project_item_icons/file-alt**

spinetoolbox.project_items.data_connection.**icon_maker**

spinetoolbox.project_items.data_connection.**properties_widget_maker**

spinetoolbox.project_items.data_connection.**add_form_maker**

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

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

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

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.data_store

Module for data store class.

authors

P. Savolainen (VTT), M. Marin (KTH)

date 18.12.2017

Module Contents

class spinetoolbox.project_items.data_store.data_store.**DataStore**(*name*, *description*, *x*, *y*, *toolbox*, *project*, *logger*, *url=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Data Store class.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **url** (*str or dict*) – SQLAlchemy url

static item_type()
See base class.

static category()
See base class.

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.

save_selections(*self*)
Save checkbox state.

url(*self*)
Return the url attribute, for saving the project.

_update_sa_url(*self*, *log_errors=True*)

_make_url(*self*, *log_errors=True*)
Returns a sqlalchemy url from the current url attribute or None if not valid.

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_view (*self*, *checked=False*)
Opens current url in the data store view.

do_open_ds_view (*self*)
Opens current url in the data store view.

_handle_ds_view_destroyed (*self*)

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

tear_down(*self*)
Tears down this item. Called by toolbox just before closing. Closes the DataStoreForm instance opened by this item.

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

output_resources_backward(*self*)
See base class.

output_resources_forward(*self*)
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

class spinetoolbox.project_items.data_store.data_store_icon.**DataStoreIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)

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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

Package Contents

class spinetoolbox.project_items.data_store.**item_maker**(*name*, *description*, *x*, *y*, *toolbox*, *project*, *logger*, *url=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Data Store class.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **url** (*str or dict*) – SQLAlchemy url

static item_type()
See base class.

static category()
See base class.

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.

save_selections (*self*)
Save checkbox state.

url (*self*)
Return the url attribute, for saving the project.

_update_sa_url (*self*, *log_errors=True*)

_make_url (*self*, *log_errors=True*)
Returns a sqlalchemy url from the current url attribute or None if not valid.

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_view(*self*, *checked=False*)
Opens current url in the data store view.

do_open_ds_view(*self*)
Opens current url in the data store view.

_handle_ds_view_destroyed(*self*)

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

tear_down(*self*)
Tears down this item. Called by toolbox just before closing. Closes the DataStoreForm instance opened by this item.

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

output_resources_backward(*self*)
See base class.

output_resources_forward(*self*)
See base class.

class spinetoolbox.project_items.data_store.**DataStoreIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)
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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

class spinetoolbox.project_items.data_store.**DataStorePropertiesWidget**(*toolbox*)
Bases: PySide2.QtWidgets.QWidget

Widget for the Data Store Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embedded

class spinetoolbox.project_items.data_store.**AddDataStoreWidget**(*toolbox*, *x*, *y*)
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.**item_rank = 0**

spinetoolbox.project_items.data_store.**item_category**

spinetoolbox.project_items.data_store.**item_type**

spinetoolbox.project_items.data_store.**item_icon = :/icons/project_item_icons/database.svg**

spinetoolbox.project_items.data_store.**icon_maker**

spinetoolbox.project_items.data_store.**properties_widget_maker**

spinetoolbox.project_items.data_store.**add_form_maker**

spinetoolbox.project_items.exporter

Exporter project item plugin.

author

A. Soininen (VTT)

date 25.9.2019

Subpackages

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

class spinetoolbox.project_items.exporter.widgets.add_exporter_widget.**AddExporterWidget**(*toolbox*, *x*, *y*)

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

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

open_settings_clicked
signal that is triggered when settings window should be opened

file_name_changed
signal that is fired when the file name field is changed

out_file_name_edit
export file name QLineEdit

url_field
Text in the database URL field.

handle_settings_state_changed(*self*, *state*)

_emit_file_name_changed(*self*)
Emits file_name_changed signal.

_emit_open_settings_clicked(*self*, *_*)
Emits open_settings_clicked signal.

spinetoolbox.project_items.exporter.widgets.exporter_properties

Exporter properties widget.

author

A. Soininen (VTT)

date 25.9.2019

Module Contents

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

class spinetoolbox.project_items.exporter.widgets.gdx_export_settings.**State**
Bases: enum.Enum

Gdx Export Settings window state

OK
Settings are ok.

BAD_INDEXING
Not all indexed parameters are set up correctly.

class spinetoolbox.project_items.exporter.widgets.gdx_export_settings.**GdxExportSettings**(*sett*
in-
dex
*ing_*
new
mer
*ing_*
new
dat
par
ent

Bases: PySide2.QtWidgets.QWidget

A setting window for exporting .gdx files.

Parameters

- **settings** (Settings) – export settings
- **indexing_settings** (*dict*) – indexing domain information for indexed parameter values

- **new_indexing_domains** (*list*) – list of additional domains needed for indexed parameter
- **merging_settings** (*dict*) – parameter merging settings
- **new_merging_domains** (*list*) – list of additional domains needed for parameter merging
- **database_path** (*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.

settings
the settings object

indexing_settings
indexing settings dict

indexing_domains
list of additional domains needed for indexing

merging_settings
dictionary of merging settings

merging_domains
list of additional domains needed for parameter merging

reset_settings(*self*, *settings*, *indexing_settings*, *new_indexing_domains*, *merging_settings*, *new_merging_domains*)
Resets all settings.

_check_state(*self*)
Checks if there are parameters in need for indexing.

_populate_global_parameters_combo_box(*self*, *settings*)
(Re)populates the global parameters combo box.

_update_new_domains_list(*self*, *domains*, *old_list*)
Merges entries from new and old domain lists.

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*)
Hides 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.

_approve_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.

class spinetoolbox.project_items.exporter.widgets.gdx_export_settings.**GAMSSetListModel** (*settin*
Bases: PySide2.QtCore.QAbstractListModel

A model to configure the domain and set name lists in gdx export settings.

This model combines the domain and set name lists 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 **settings** (spine_io.exporters.gdx.Settings) – settings whose domain and set name lists should be modelled

add_domain (*self*, *domain*)
Adds a new domain.

drop_domain (*self*, *domain*)
Removes a domain.

update_domain (*self*, *domain*)
Updates an existing domain.

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.

index_for_domain(*self*, *domain_name*)
Returns the model index for a domain.

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.

class spinetoolbox.project_items.exporter.widgets.gdx_export_settings.**GAMSRecordListModel**
Bases: PySide2.QtCore.QAbstractListModel

A model to manage record ordering within domains and sets.

domain_records_reordered

data(*self*, *index*, *role=Qt.DisplayRole*)
With *role == Qt.DisplayRole* returns the record's keys as comma separated string.

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()*)
Return the number of records in the model.

sort_alphabetically (*self*)

spinetoolbox.project_items.exporter.widgets.merging_error_flag

Error condition flags for Parameter merging.

author

A. Soininen (VTT)

date 2.3.2020

Module Contents

class spinetoolbox.project_items.exporter.widgets.merging_error_flag.**MergingErrorFlag**
Bases: enum.Flag

Error flags for parameter merging.

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

class spinetoolbox.project_items.exporter.widgets.parameter_index_settings.**IndexSettingsSta**
Bases: enum.Enum

An enumeration indicating the state of the settings window.

OK

DOMAIN_MISSING_INDEXES

DOMAIN_NAME_MISSING

DOMAIN_NAME_CLASH

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_existing_domains** (*dict*) – a dict from existing domain name to a list of its record keys
- **new_domains** (*dict*) – a dict from new domain name to a list of its record keys
- **parent** (*QWidget*) – a parent widget

new_domain_name
name of the new domain

state
widget's state

is_using_domain(*self*, *domain_name*)

indexing_domain(*self*)
Provides information needed to expand the parameter's indexed values.

Returns a tuple of IndexingDomain and a Set if a new domain is needed for indexing, otherwise None

Return type tuple

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.

reorder_indexes(*self*, *first*, *last*, *target*)

_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*, *domain_name=None*)
Updates the model's header and the label showing the indexing domains.

Parameters **domain_name** (*str*) – indexing domain's name or None to read it from the other widgets.

_domain_name_changed(*self*, *text*)
Reacts to changes in indexing domain name.

_set_enabled_use_existing_domain_widgets(*self*, *enabled*)
Enables and disables controls used to set up indexing based on an existing domain.

_set_enabled_create_domain_widgets(*self*, *enabled*)
Enables and disables controls used to set up indexing based on a new domain.

_existing_domain_changed(*self*, *index*)
Reacts to changes in existing domains combo box.

_update_index_list_selection(*self*, *expression*, *clear_selection_if_expression_empty=True*)
Updates selection according to changed selection expression.

_update_model_to_selection(*self*, *selected*, *deselected*)
Updates the model after table selection has changed.

_generate_index(*self*, *expression*)
Builds indexes according to given expression.

_extract_index_from_parameter(*self*, *_=True*)
Assigns indexes from the parameter to the model.

_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.

class spinetoolbox.project_items.exporter.widgets.parameter_index_settings.**_IndexingTableM**
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

indexes
a string list of all new indexing keys

index_selection
a boolean list of selected index keys, so called pick list

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.

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

reorder_indexes(*self*, *first*, *last*, *target*)
Moves indexes around.

Parameters

- **first** (*int*) – first index to move
- **last** (*int*) – last index to move (inclusive)
- **target** (*int*) – where to move the first index

rowCount(*self*, *parent=QModelIndex()*)
Return the number of rows.

selection_changed(*self*, *selected*, *deselected*)
Updates selected and deselected rows on the table.

set_index_name(*self*, *name*)
Sets the indexing domain name.

set_indexes(*self*, *indexes*)
Overwrites all new indexes.

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

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 IndexingSettings

- **available_existing_domains** (*dict*) – a map from existing domain names to lists of record keys
- **new_domains** (*dict*) – a map from new domain names to lists of record keys
- **database_path** (*str*) – a 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.

indexing_settings
indexing settings dictionary

new_domains
list of additional domains needed for indexing

reorder_indexes(*self*, *domain_name*, *first*, *last*, *target*)

_collect_and_hide(*self*)
Collects settings from individual ParameterIndexSettings widgets and hides the window.

_reject_and_close(*self*)

closeEvent(*self*, *event*)

spinetoolbox.project_items.exporter.widgets.parameter_merging_settings

Parameter merging settings widget.

author

A. Soininen (VTT)

date 19.2.2020

Module Contents

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

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.exporter

Exporter project item.

author

A. Soininen (VTT)

date 5.9.2019

Module Contents

class spinetoolbox.project_items.exporter.exporter.**Exporter**(*name*, *description*, *settings_packs*, *x*, *y*, *toolbox*, *project*, *logger*)

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

- **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
- **toolbox** (ToolboxUI) – a ToolboxUI instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance

set_up(*self*)
See base class.

static item_type()
See base class.

static category()
See base class.

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*)

_update_properties_tab(*self*)
Updates the database list in the properties tab.

execute_forward(*self*, *resources*)
See base class.

_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.

_update_export_settings(*self*, *database_url*, *settings*)
Sets new settings for given database.

_update_indexing_settings(*self*, *database_url*, *indexing_settings*)
Sets new indexing settings for given database.

_update_indexing_domains(*self*, *database_url*, *domains*)
Sets new indexing domains for given database.

_update_merging_settings(*self*, *database_url*, *settings*)
Sets new merging settings for given database.

_update_merging_domains(*self*, *database_url*, *domains*)
Sets new merging domains for given database.

_worker_finished(*self*, *database_url*)
Cleans up after a worker has finished fetching export settings.

_worker_failed(*self*, *database_url*, *exception*)
Clean up after a worker has failed 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_settings_from_settings_window(*self*, *database_path*)
Pushes a new UpdateExporterSettingsCommand to the toolbox undo stack.

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*, *indexing_domains*, *merging_settings*, *merging_domains*, *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.

_resolve_gams_system_directory(*self*)
Returns GAMS system path from Toolbox settings or None if GAMS default is to be used.

notify_destination(*self*, *source_item*)
See base class.

_update_settings_after_db_commit(*self*, *committed_db_maps*)
Refreshes export settings for databases after data has been committed to them.

static default_name_prefix()
See base class.

output_resources_forward(*self*)
See base class.

tear_down(*self*)
See base class.

class spinetoolbox.project_items.exporter.exporter.**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 *Settings*

indexing_settings
parameter indexing settings

Type dict

indexing_domains
extra domains needed for parameter indexing

Type list

merging_settings
parameter merging settings

Type dict

merging_domains
extra domains needed for parameter merging

Type list

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.exporter.**_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.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_icon

Icon class for the Exporter project item.

authors

A. Soininen (VTT)

date 25.9.2019

Module Contents

class spinetoolbox.project_items.exporter.exporter_icon.**ExporterIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)
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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

spinetoolbox.project_items.exporter.list_utils

Contains list helper functions for list manipulation.

author

A. Soininen (VTT)

date 12.12.2019

Module Contents

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.settings_state

Provides the SettingsState enum.

author

A. Soininen (VTT)

date 20.12.2019

Module Contents

class spinetoolbox.project_items.exporter.settings_state.**SettingsState**
Bases: enum.Enum

State of export settings.

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

class spinetoolbox.project_items.exporter.worker.**Worker**(*database_url*)
Bases: PySide2.QtCore.QThread

A worker thread to construct export settings for a database.

Parameters **database_url** (*str*) – database's URL

errored
Emitted when an error occurs.

finished
Emitted when the worker has finished.

indexing_domains_read
Sends new additional domains away.

indexing_settings_read
Sends the indexing settings away.

merging_settings_read
Sends updated merging settings away.

merging_domains_read
Sends updated merging domains away.

settings_read
Sends the settings away.

reset_previous_settings(*self*)
Makes worker send new settings instead of updating old ones.

run(*self*)
Constructs settings and parameter index settings and sends them to interested parties using signals.

set_previous_settings(*self*, *previous_settings*, *previous_indexing_settings*, *previous_indexing_domains*, *previous_merging_settings*)
Makes worker update existing settings instead of just making new ones.

Parameters

- **previous_settings** (Settings) – existing settings
- **previous_indexing_settings** (*dict*) – existing indexing settings
- **previous_indexing_domains** (*list*) –
- **previous_merging_settings** (*dict*) – existing merging settings

_read_settings(*self*)

_update_indexing_settings(*self*, *updated_settings*, *new_indexing_settings*)

_update_merging_settings(*self*, *updated_settings*)

Package Contents

class spinetoolbox.project_items.exporter.**item_maker**(*name*, *description*, *settings_packs*, *x*, *y*, *toolbox*, *project*, *logger*)
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

- **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
- **toolbox** (ToolboxUI) – a ToolboxUI instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance

set_up(*self*)
See base class.

static item_type()
See base class.

static category()
See base class.

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*)

_update_properties_tab(*self*)
Updates the database list in the properties tab.

execute_forward(*self*, *resources*)
See base class.

_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.

_update_export_settings(*self*, *database_url*, *settings*)
Sets new settings for given database.

_update_indexing_settings(*self*, *database_url*, *indexing_settings*)
Sets new indexing settings for given database.

_update_indexing_domains(*self*, *database_url*, *domains*)
Sets new indexing domains for given database.

_update_merging_settings(*self*, *database_url*, *settings*)
Sets new merging settings for given database.

_update_merging_domains(*self*, *database_url*, *domains*)
Sets new merging domains for given database.

_worker_finished(*self*, *database_url*)
Cleans up after a worker has finished fetching export settings.

_worker_failed(*self*, *database_url*, *exception*)
Clean up after a worker has failed 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_settings_from_settings_window(*self*, *database_path*)
Pushes a new UpdateExporterSettingsCommand to the toolbox undo stack.

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*, *indexing_domains*, *merging_settings*, *merging_domains*, *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.

_resolve_gams_system_directory(*self*)
Returns GAMS system path from Toolbox settings or None if GAMS default is to be used.

notify_destination(*self*, *source_item*)
See base class.

_update_settings_after_db_commit(*self*, *committed_db_maps*)
Refreshes export settings for databases after data has been committed to them.

static default_name_prefix()
See base class.

output_resources_forward(*self*)
See base class.

tear_down(*self*)
See base class.

class spinetoolbox.project_items.exporter.**icon_maker**(*toolbox*, *x*, *y*, *w*, *h*, *name*)
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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

class spinetoolbox.project_items.exporter.**add_form_maker**(*toolbox*, *x*, *y*)
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.

class spinetoolbox.project_items.exporter.**properties_widget_maker**(*toolbox*)
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.**item_rank = 5**

spinetoolbox.project_items.exporter.**item_category**

spinetoolbox.project_items.exporter.**item_type**

spinetoolbox.project_items.exporter.**item_icon = :/icons/project_item_icons/database-export**

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

class spinetoolbox.project_items.importer.widgets.add_importer_widget.**AddImporterWidget**(*toolbox*, *x*, *y*)

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

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

class spinetoolbox.project_items.importer.widgets.importer_properties_widget.**ImporterPrope**
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.importer

Contains Importer project item class.

authors

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

date 10.6.2019

Module Contents

spinetoolbox.project_items.importer.importer.**_CONNECTOR_NAME_TO_CLASS**

class spinetoolbox.project_items.importer.importer.**Importer**(*name*, *description*, *mappings*, *x*, *y*, *toolbox*, *project*, *logger*, *cancel_on_error=True*)
Bases: *spinetoolbox.project_item.ProjectItem*

Importer class.

Parameters

- **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
- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **cancel_on_error** (*bool*) – if True the item's execution will stop on import error

importing_finished

static item_type()
See base class.

static category()
See base class.

_handle_file_model_item_changed(*self*, *item*)

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*)
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*) – Path to 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*) – Absolute path to a 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

update_file_model(*self*, *items*)

Adds given list of items to the file model. If None or an empty list is given, the model is cleared.

Parameters **items** (*set*) – Set of absolute file paths

_prepare_importer_program(*self*, *importer_args*)

Prepares an execution manager instance for running importer_process.py in a QProcess.

If app is not frozen, the Python to run it is the python that was used in starting the app.

If app is frozen, we are running the importer_program application found in application install directory.

Parameters **importer_args** (*list*) – Arguments for the importer_program. Source file paths, their mapping specs, URLs downstream, logs directory, cancel_on_error

Returns True if preparing the program succeeded, False otherwise.

Return type bool

_handle_importer_program_process_finished(*self*, *exit_code*)

Handles the return value from importer program when it has finished. Emits a signal to indicate that this Importer has been executed.

Parameters **exit_code** (*int*) – Process return value. 0: success, !0: failure

python_exists(*self*, *program*)

Checks that Python is set up correctly in Settings. This executes 'python -V' in a QProcess and if the process finishes successfully, the python is ready to be used.

Parameters **program** (*str*) – Python executable that is currently set in Settings

Returns True if Python is found, False otherwise

Return type bool

execute_backward(*self*, *resources*)

See base class.

execute_forward(*self*, *resources*)

See base class.

stop_execution(*self*)

Stops executing this Importer.

_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*, *file_list*)
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 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

static serialize_mappings(*mappings*, *project_path*)
Returns a list of mappings, where each element contains two dictionaries, the 'serialized' path in a dictionary and the mapping dictionary.

Parameters

- **mappings** (*dict*) – Dictionary with mapping specifications
- **project_path** (*str*) – Path to project directory

Returns List where each element contains two dictionaries.

Return type list

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_icon

Module for Importer icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

class spinetoolbox.project_items.importer.importer_icon.**ImporterIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)

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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

spinetoolbox.project_items.importer.importer_program

Contains importer_program script.

authors

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

date 10.6.2019

Module Contents

spinetoolbox.project_items.importer.importer_program.**_create_log_file_timestamp**()

Creates a new timestamp string that is used as Importer and Data Store error log file.

Returns Timestamp string or empty string if failed.

spinetoolbox.project_items.importer.importer_program.**run**(*checked_files*, *all_settings*, *urls_downstream*, *logs_dir*, *cancel_on_error*)

spinetoolbox.project_items.importer.importer_program.**_import**(*all_data*, *url*, *logs_dir*, *cancel_on_error*)

spinetoolbox.project_items.importer.importer_program.**stdout**

Package Contents

class spinetoolbox.project_items.importer.**Importer**(*name*, *description*, *mappings*, *x*, *y*, *toolbox*, *project*, *logger*, *cancel_on_error=True*)

Bases: *spinetoolbox.project_item.ProjectItem*

Importer class.

Parameters

- **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
- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **cancel_on_error** (*bool*) – if True the item's execution will stop on import error

importing_finished

static item_type()
See base class.

static category()
See base class.

_handle_file_model_item_changed(*self*, *item*)

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*)
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*) – Path to 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*) – Absolute path to a 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

update_file_model(*self*, *items*)
Adds given list of items to the file model. If None or an empty list is given, the model is cleared.

Parameters **items** (*set*) – Set of absolute file paths

_prepare_importer_program(*self*, *importer_args*)
Prepares an execution manager instance for running importer_process.py in a QProcess.

If app is not frozen, the Python to run it is the python that was used in starting the app.

If app is frozen, we are running the importer_program application found in application install directory.

Parameters **importer_args** (*list*) – Arguments for the importer_program. Source file paths, their mapping specs, URLs downstream, logs directory, cancel_on_error

Returns True if preparing the program succeeded, False otherwise.

Return type bool

_handle_importer_program_process_finished(*self*, *exit_code*)
Handles the return value from importer program when it has finished. Emits a signal to indicate that this Importer has been executed.

Parameters **exit_code** (*int*) – Process return value. 0: success, !0: failure

python_exists(*self*, *program*)
Checks that Python is set up correctly in Settings. This executes 'python -V' in a QProcess and if the process finishes successfully, the python is ready to be used.

Parameters **program** (*str*) – Python executable that is currently set in Settings

Returns True if Python is found, False otherwise

Return type bool

execute_backward(*self*, *resources*)
See base class.

execute_forward(*self*, *resources*)
See base class.

stop_execution(*self*)
Stops executing this Importer.

_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*, *file_list*)
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 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

static serialize_mappings(*mappings*, *project_path*)
Returns a list of mappings, where each element contains two dictionaries, the 'serialized' path in a dictionary and the mapping dictionary.

Parameters

- **mappings** (*dict*) – Dictionary with mapping specifications
- **project_path** (*str*) – Path to project directory

Returns List where each element contains two dictionaries.

Return type list

class spinetoolbox.project_items.importer.**ImporterIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)
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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

class spinetoolbox.project_items.importer.**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

class spinetoolbox.project_items.importer.**AddImporterWidget**(*toolbox*, *x*, *y*)
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.**item_rank = 4**

spinetoolbox.project_items.importer.**item_category**

spinetoolbox.project_items.importer.**item_type**

spinetoolbox.project_items.importer.**item_icon = :/icons/project_item_icons/database-import**

spinetoolbox.project_items.importer.**item_maker**

spinetoolbox.project_items.importer.**icon_maker**

spinetoolbox.project_items.importer.**properties_widget_maker**

spinetoolbox.project_items.importer.**add_form_maker**

spinetoolbox.project_items.shared

Submodules

spinetoolbox.project_items.shared.import_export_animation

Animation class for the Exporter and Importer items.

authors

M. Marin (KTH)

date 12.11.2019

Module Contents

class spinetoolbox.project_items.shared.import_export_animation.**ImportExportAnimation**(*parent*, *src_ite*, *dst_ite*, *duration*, *ration=2*)

Initializes animation stuff.

Parameters

- **parent_item** (*QGraphicsItem*) – The item on top of which the animation should play.
- **src_item** (*QGraphicsItem*) – The source item.
- **dst_item** (*QGraphicsItem*) – The destination item.
- **duration** (*int. optional*) – The desired duration of each loop in milliseconds, defaults to 1000.

_handle_timer_value_changed(*self*, *value*)

start(*self*)
Starts the animation.

stop(*self*)
Stops the animation

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

class spinetoolbox.project_items.tool.widgets.add_tool_widget.**AddToolWidget**(*toolbox*, *x*, *y*)

Bases: PySide2.QtWidgets.QWidget

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

Initialize class.

connect_signals(*self*)
Connect signals to slots.

update_args(*self*, *row*)
Show Tool specification command line arguments in text input.

Parameters **row** (*int*) – Selected row number

name_changed(*self*)
Update label to show upcoming folder name.

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.

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.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

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.

spinetoolbox.project_items.tool.widgets.tool_properties_widget

Tool properties widget.

author

M. Marin (KTH)

date 12.9.2019

Module Contents

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 embeded

Init class.

connect_signals(*self*)
Connect signals to slots.

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

Submodules

spinetoolbox.project_items.tool.tool

Tool class.

author

P. Savolainen (VTT)

date 19.12.2017

Module Contents

class spinetoolbox.project_items.tool.tool.**Tool**(*name*, *description*, *x*, *y*, *toolbox*, *project*, *logger*, *tool=''*, *execute_in_work=True*, *cmd_line_args=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Tool class.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **tool** (*str*) – Name of this Tool's Tool specification
- **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 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*)
Pushed 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*)

update_tool_specification(*self*, *row*)
Update Tool specification according to selection in the specification comboBox.

Parameters **row** (*int*) – Selected row in the comboBox

update_tool_cmd_line_args(*self*)
Updates tool cmd line args list as line edit text is changed.

do_update_tool_cmd_line_args(*self*, *cmd_line_args*)

set_tool_specification(*self*, *tool_specification*)
Pushes a new SetToolSpecificationCommand to the toolbox' undo stack.

do_set_tool_specification(*self*, *tool_specification*)
Sets Tool specification for this Tool. Removes Tool specification if None given as argument.

Parameters **tool_specification** (ToolSpecification) – Tool specification of this Tool. None removes the specification.

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_tool_specification(*self*)
Open Tool specification editor for the Tool specification attached to this Tool.

open_tool_specification_file(*self*)
Open Tool specification file.

open_tool_main_program_file(*self*)
Open Tool specification main program file in an external text edit application.

open_tool_main_directory(*self*)
Open directory where the Tool specification main program is located in file explorer.

tool_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.

_update_base_directory(*self*)
Updates the path to the base directory, depending on *execute_in_work*.

output_resources_forward(*self*)
See base class.

_find_last_output_files(*self*)
Returns a list of most recent output files from the results directory.

Returns list

execute_backward(*self*, *resources*)
See base class.

execute_forward(*self*, *resources*)
See base class.

count_files_and_dirs(*self*)
Count the number of files and directories in required input files model.

Returns Tuple containing the number of required files and directories.

_optional_output_destination_paths(*self*, *paths*)
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

Returns a map from source path to destination path

Return type dict

create_input_dirs(*self*)
Iterate items in required input files and check if there are any directories to create. Create found directories directly to work or source directory.

Returns Boolean variable depending on success

copy_input_files(*self*, *paths*)
Copy 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.

Returns Boolean variable depending on operation success

_copy_optional_input_files(*self*, *paths*)
Copy 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_program_files(*self*)
Copies Tool specification source files to base directory.

_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

_find_optional_input_files(*self*, *resources*)
Tries to find optional input files from previous project items in the DAG. Returns found paths.

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.

static _filepaths_from_resources(*resources*)
Returns file paths from given resources.

Parameters **resources** (*list*) – resources available

Returns a list of file paths, possibly including patterns

_find_file(*self*, *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

static _find_optional_files(*pattern*, *available_file_paths*)
Returns a list of found paths to files that match the given pattern in files available from the execution instance.

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

handle_execution_finished(*self*, *return_code*)
Handles Tool specification execution finished.

Parameters **return_code** (*int*) – Process exit code

handle_output_files(*self*, *ret*)
Copies Tool specification output files from work directory to result directory.

Parameters **ret** (*int*) – Tool specification process return value

create_output_dirs(*self*)
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.

Returns True for success, False otherwise.

Return type bool

Raises OSError – If creating an output directory to work fails.

copy_output_files(*self*, *target_dir*)
Copies Tool specification output files from work directory to given target directory.

Parameters **target_dir** (*str*) – Destination directory for Tool specification output files

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.

stop_execution(*self*)
Stops executing this Tool.

_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.

_flatten_file_path_duplicates(*self*, *file_paths*, *log_duplicates=False*)
Flattens the extra duplicate dimension in file_paths.

static _database_urls(*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.tool_icon

Module for tool icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

class spinetoolbox.project_items.tool.tool_icon.**ToolIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)

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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

static _value_for_time(*msecs*)

start_animation(*self*)

Start the animation that plays when the Tool associated to this GraphicsItem is running.

stop_animation(*self*)

Stop animation

Package Contents

class spinetoolbox.project_items.tool.**item_maker**(*name*, *description*, *x*, *y*, *toolbox*, *project*, *logger*, *tool=''*, *execute_in_work=True*, *cmd_line_args=None*)

Bases: *spinetoolbox.project_item.ProjectItem*

Tool class.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **toolbox** (ToolboxUI) – QMainWindow instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance
- **tool** (*str*) – Name of this Tool's Tool specification
- **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 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*)
Pushed 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*)

update_tool_specification(*self*, *row*)
Update Tool specification according to selection in the specification comboBox.

Parameters **row** (*int*) – Selected row in the comboBox

update_tool_cmd_line_args(*self*)
Updates tool cmd line args list as line edit text is changed.

do_update_tool_cmd_line_args(*self*, *cmd_line_args*)

set_tool_specification(*self*, *tool_specification*)
Pushes a new SetToolSpecificationCommand to the toolbox' undo stack.

do_set_tool_specification(*self*, *tool_specification*)
Sets Tool specification for this Tool. Removes Tool specification if None given as argument.

Parameters **tool_specification** (ToolSpecification) – Tool specification of this Tool. None removes the specification.

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_tool_specification(*self*)
Open Tool specification editor for the Tool specification attached to this Tool.

open_tool_specification_file(*self*)
Open Tool specification file.

open_tool_main_program_file(*self*)
Open Tool specification main program file in an external text edit application.

open_tool_main_directory(*self*)
Open directory where the Tool specification main program is located in file explorer.

tool_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.

_update_base_directory (*self*)
Updates the path to the base directory, depending on *execute_in_work*.

output_resources_forward (*self*)
See base class.

_find_last_output_files (*self*)
Returns a list of most recent output files from the results directory.

Returns list

execute_backward (*self*, *resources*)
See base class.

execute_forward (*self*, *resources*)
See base class.

count_files_and_dirs (*self*)
Count the number of files and directories in required input files model.

Returns Tuple containing the number of required files and directories.

_optional_output_destination_paths (*self*, *paths*)
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

Returns a map from source path to destination path

Return type dict

create_input_dirs (*self*)
Iterate items in required input files and check if there are any directories to create. Create found directories directly to work or source directory.

Returns Boolean variable depending on success

copy_input_files (*self*, *paths*)
Copy 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.

Returns Boolean variable depending on operation success

_copy_optional_input_files(*self*, *paths*)
Copy 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_program_files(*self*)
Copies Tool specification source files to base directory.

_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

_find_optional_input_files(*self*, *resources*)
Tries to find optional input files from previous project items in the DAG. Returns found paths.

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.

static _filepaths_from_resources(*resources*)
Returns file paths from given resources.

Parameters **resources** (*list*) – resources available

Returns a list of file paths, possibly including patterns

_find_file(*self*, *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

static _find_optional_files(*pattern*, *available_file_paths*)
Returns a list of found paths to files that match the given pattern in files available from the execution instance.

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

handle_execution_finished(*self*, *return_code*)
Handles Tool specification execution finished.

Parameters **return_code** (*int*) – Process exit code

handle_output_files(*self*, *ret*)
Copies Tool specification output files from work directory to result directory.

Parameters **ret** (*int*) – Tool specification process return value

create_output_dirs(*self*)
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.

Returns True for success, False otherwise.

Return type bool

Raises `OSError` – If creating an output directory to work fails.

copy_output_files(*self*, *target_dir*)
Copies Tool specification output files from work directory to given target directory.

Parameters **target_dir** (*str*) – Destination directory for Tool specification output files

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.

stop_execution(*self*)
Stops executing this Tool.

_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.

_flatten_file_path_duplicates(*self*, *file_paths*, *log_duplicates=False*)
Flattens the extra duplicate dimension in file_paths.

static _database_urls(*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

class spinetoolbox.project_items.tool.**ToolIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)
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
- **w** (*float*) – Width of master icon
- **h** (*float*) – Height of master icon
- **name** (*str*) – Item name

static _value_for_time(*msecs*)

start_animation(*self*)
Start the animation that plays when the Tool associated to this GraphicsItem is running.

stop_animation(*self*)
Stop animation

class spinetoolbox.project_items.tool.**ToolPropertiesWidget**(*toolbox*)
Bases: PySide2.QtWidgets.QWidget

Widget for the Tool Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embeded

Init class.

connect_signals(*self*)
Connect signals to slots.

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

class spinetoolbox.project_items.tool.**AddToolWidget**(*toolbox*, *x*, *y*)
Bases: PySide2.QtWidgets.QWidget

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

Initialize class.

connect_signals(*self*)
Connect signals to slots.

update_args(*self*, *row*)
Show Tool specification command line arguments in text input.

Parameters **row** (*int*) – Selected row number

name_changed(*self*)
Update label to show upcoming folder name.

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.

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.project_items.tool.**item_rank = 2**

spinetoolbox.project_items.tool.**item_category**

spinetoolbox.project_items.tool.**item_type**

spinetoolbox.project_items.tool.**item_icon = :/icons/project_item_icons/hammer.svg**

spinetoolbox.project_items.tool.**icon_maker**

spinetoolbox.project_items.tool.**properties_widget_maker**

spinetoolbox.project_items.tool.**add_form_maker**

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

class spinetoolbox.project_items.view.widgets.add_view_widget.**AddViewWidget**(*toolbox*, *x*, *y*)

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

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

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 embeded

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.view

Module for view class.

authors

P. Savolainen (VTT), M. Marin (KHT), J. Olauson (KTH)

date 14.07.2018

Module Contents

class spinetoolbox.project_items.view.view.**View**(*name*, *description*, *x*, *y*, *toolbox*, *project*, *logger*)

Bases: *spinetoolbox.project_item.ProjectItem*

View class.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **toolbox** (ToolboxUI) – a toolbox instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance

static item_type()
See base class.

static 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.

save_selections(*self*)
Save selections in shared widgets for this project item into instance variables.

open_view(*self*, *checked=False*)
Opens references in a view window.

populate_reference_list(*self*)
Populates reference list.

update_name_label(*self*)
Update View tab name label. Used only when renaming project items.

execute_forward(*self*, *resources*)
see base class

_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.

_database_urls(*self*, *indexes*)
Returns list of tuples (url, provider) for given indexes.

_restore_existing_view_window(*self*, *view_id*)
Restores an existing view window and returns True if the operation was successful.

_make_view_window(*self*, *db_maps*)

tear_down(*self*)
Tears down this item. Called by toolbox just before closing. Closes all view windows.

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

spinetoolbox.project_items.view.view_icon

Module for view icon class.

authors

M. Marin (KTH), P. Savolainen (VTT)

date 4.4.2018

Module Contents

class spinetoolbox.project_items.view.view_icon.**ViewIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)
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
- **w** (*float*) – Width of background rectangle
- **h** (*float*) – Height of background rectangle
- **name** (*str*) – Item name

Package Contents

class spinetoolbox.project_items.view.**View**(*name*, *description*, *x*, *y*, *toolbox*, *project*, *logger*)
Bases: *spinetoolbox.project_item.ProjectItem*

View class.

Parameters

- **name** (*str*) – Object name
- **description** (*str*) – Object description
- **x** (*float*) – Initial X coordinate of item icon
- **y** (*float*) – Initial Y coordinate of item icon
- **toolbox** (ToolboxUI) – a toolbox instance
- **project** (SpineToolboxProject) – the project this item belongs to
- **logger** (LoggerInterface) – a logger instance

static item_type()
See base class.

static 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.

save_selections(*self*)
Save selections in shared widgets for this project item into instance variables.

open_view(*self*, *checked=False*)
Opens references in a view window.

populate_reference_list(*self*)
Populates reference list.

update_name_label(*self*)
Update View tab name label. Used only when renaming project items.

execute_forward(*self*, *resources*)
see base class

_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.

_database_urls(*self*, *indexes*)
Returns list of tuples (url, provider) for given indexes.

_restore_existing_view_window(*self*, *view_id*)
Restores an existing view window and returns True if the operation was successful.

_make_view_window(*self*, *db_maps*)

tear_down(*self*)
Tears down this item. Called by toolbox just before closing. Closes all view windows.

notify_destination(*self*, *source_item*)
See base class.

static default_name_prefix()
see base class

class spinetoolbox.project_items.view.**ViewIcon**(*toolbox*, *x*, *y*, *w*, *h*, *name*)
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
- **w** (*float*) – Width of background rectangle
- **h** (*float*) – Height of background rectangle
- **name** (*str*) – Item name

class spinetoolbox.project_items.view.**ViewPropertiesWidget**(*toolbox*)
Bases: PySide2.QtWidgets.QWidget

Widget for the View Item Properties.

Parameters **toolbox** (ToolboxUI) – The toolbox instance where this widget should be embeded

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

class spinetoolbox.project_items.view.**AddViewWidget**(*toolbox*, *x*, *y*)
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.**item_rank = 3**

spinetoolbox.project_items.view.**item_category**

spinetoolbox.project_items.view.**item_type**

spinetoolbox.project_items.view.**item_icon = :/icons/project_item_icons/binoculars.svg**

spinetoolbox.project_items.view.**item_maker**

spinetoolbox.project_items.view.**icon_maker**

spinetoolbox.project_items.view.**properties_widget_maker**

spinetoolbox.project_items.view.**add_form_maker**

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

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 whit 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 relationship 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

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.

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

name
set's name

Type str

records
set's elements as a list of Record objects

Type list

Parameters

- **name** (*str*) – set's name
- **description** (*str*) – set's explanatory text
- **domain_names** (*list*) – 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.

static from_object_class(*object_class*)
Constructs a Set from database's object class row.

Parameters **object_class** (*namedtuple*) – an object class row from the database

static from_relationship_class(*relationship_class*)
Constructs a Set from database's relationship class row.

Parameters **relationship_class** (*namedtuple*) – a relationship class row from the database

class spinetoolbox.spine_io.exporters.gdx.**Record**(*keys*)
Represents a GAMS set element in a Set.

Parameters

- **keys** (*tuple*) – a tuple of record's keys
- **keys** – 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.

static from_object(*object_*)
Constructs a record from database's object row.

Parameters **object** (*namedtuple*) – an object or relationship row from the database

static from_relationship(*relationship*)
Constructs a record from database's relationship row.

Parameters **relationship** (*namedtuple*) – a relationship row from the database

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

indexes
parameter's indexes

Type list

values
parameter's values

Type list

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

__eq__(*self*, *other*)

append_value(*self*, *index*, *value*)
Appends a new value.

Parameters

- **index** (*tuple*) – record keys indexing the value
- **value** – a value

append_object_parameter(*self*, *object_parameter*)
Appends a value from object parameter.

Parameters **object_parameter** (*namedtuple*) – an object parameter row from the database

append_relationship_parameter(*self*, *relationship_parameter*)
Appends a value from relationship parameter.

Parameters **relationship_parameter** (*namedtuple*) – a relationship parameter row from the database

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 contains only scalars.

is_indexed(*self*)
Returns True if this parameter contains only indexed values.

expand_indexes(*self*, *indexing_setting*)
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

static from_object_parameter(*object_parameter*)
Constructs a GAMS parameter from database's object parameter row

Parameters **object_parameter** (*namedtuple*) – a parameter row from the database

static from_relationship_parameter(*relationship_parameter*)
Constructs a GAMS parameter from database's relationship parameter row

Parameters **relationship_parameter** (*namedtuple*) – a parameter row from the database

static from_entity_class_parameter_definition(*entity_class*)
Constructs an empty GAMS parameter from database's parameter definition row

Parameters **entity_class** – a parameter definition row from the database

class spinetoolbox.spine_io.exporters.gdx.**IndexingDomain**(*name*, *description*, *indexes*, *pick_list*)
This class holds the indexes that should be used for indexed parameter value expansion.

name
indexing domain's name

Type str

description
domain's description

Type str

Picks the keys from base_domain for which the corresponding element in pick_list holds True.

Parameters

- **name** (*str*) – indexing domain's name
- **description** (*str*) – domain's description
- **indexes** (*list*) – a list of indexing key tuples
- **pick_list** (*list*) – a list of booleans

indexes
a list of picked indexing key tuples

all_indexes
a list of all indexing key tuples

pick_list
list of boolean values where True means the corresponding index should be picked

sort_indexes(*self*, *settings*)
Sorts the indexes according to settings.

Parameters **settings** (Settings) – a Settings object

to_dict(*self*)
Stores IndexingDomain to a dictionary.

static from_dict(*domain_dict*)
Restores IndexingDomain from a dictionary.

static from_base_domain(*base_domain*, *pick_list*)
Builds a new IndexingDomain from an existing Set.

Parameters

- **base_domain** (Set) – a domain set that holds the indexes
- **pick_list** (*list*) – a list of booleans

spinetoolbox.spine_io.exporters.gdx.**sort_indexing_domain_indexes**(*indexing_settings*, *settings*)
Sorts the index keys of an indexing domain in place.

Parameters

- **indexing_settings** (*dict*) – a mapping from parameter name to IndexingSetting
- **settings** (Settings) – settings

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*)

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

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*, *settings*, *db_map*)

Returns parameter merging settings updated according to new export settings.

Parameters

- **merging_settings** (*dict*) – old settings to be updated
- **settings** (Settings) – new gdx export settings
- **db_map** (*spinedb_api.DatabaseMapping*) – a database map

Returns merged old and new merging settings

Return type dict

spinetoolbox.spine_io.exporters.gdx.**merging_domain**(*merging_setting*)

Constructs the additional indexing domain which contains the merged parameters' names.

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*)

Writes parameters to .gdx file as GAMS parameters.

Parameters

- **gdx_file** (*GdxFile*) – a target file
- **parameters** (*dict*) – a list of Parameter objects

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*)
Converts object classes, objects and object parameters from a database to the intermediate format.

Object classes get converted to Set objects while objects are stored as Records in corresponding DomainSets. Lastly, object parameters are read into Parameter objects.

Parameters **db_map** (*spinedb_api.DatabaseMapping*) – a database map

Returns a tuple containing list of Set objects and a dict of Parameter objects

spinetoolbox.spine_io.exporters.gdx.**relationship_classes_to_sets**(*db_map*)
Converts relationship classes, relationships and relationship parameters from a database to the intermediate format.

Relationship classes get converted to Set objects while relationships are stored as SetRecords in corresponding Sets. Lastly, relationship parameters are read into Parameter objects.

Parameters **db_map** (*spinedb_api.DatabaseMapping*) – a database map

Returns a tuple containing a list of Set objects and a dict of Parameter objects

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** (*spinedb_api.DatabaseMapping*) – a database map

Returns a tuple containing list of domain names and a dict from domain name to its records

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*) – a database map

Returns a tuple containing list of set names and a dict from set name to its records

class spinetoolbox.spine_io.exporters.gdx.**IndexingSetting**(*indexed_parameter*)
Settings for indexed value expansion for a single Parameter.

parameter
a parameter containing indexed values

Type *Parameter*

indexing_domain
indexing info

Type *IndexingDomain*

index_position
where to insert the new index when expanding a parameter

Type int

Parameters **indexed_parameter** (Parameter) – a parameter containing indexed values

append_parameter(*self*, *parameter*)
Adds indexes and values from another parameter.

spinetoolbox.spine_io.exporters.gdx.**make_indexing_settings**(*db_map*)
Constructs skeleton indexing settings for parameter indexed value expansion.

Parameters **db_map** (*spinedb_api.DatabaseMapping*) – a database mapping

Returns a mapping from parameter name to IndexingSetting

Return type dict

spinetoolbox.spine_io.exporters.gdx.**update_indexing_settings**(*old_indexing_settings*, *new_indexing_settings*, *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
- **settings** (Settings) – new gdx export 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

spinetoolbox.spine_io.exporters.gdx.**indexing_settings_from_dict**(*settings_dict*, *db_map*)
Restores indexing settings from a json compatible dictionary.

Parameters

- **settings** (*dict*) – a JSON compatible dictionary representing parameter indexing settings.
- **db_map** (*DatabaseMapping*) – database mapping

Returns a dictionary mapping parameter name to IndexingSetting.

spinetoolbox.spine_io.exporters.gdx.**_find_parameter**(*parameter_name*, *db_map*)
Searches for parameter_name in db_map and returns Parameter.

spinetoolbox.spine_io.exporters.gdx.**filter_and_sort_sets**(*sets*, *sorted_set_names*, *metadatas*)
Returns a list of sets sorted by *sorted_set_names* and their filter flag set to True

This function removes the sets that are not supposed to be exported and sorts the rest according to the order specified by *sorted_set_names*.

Parameters

- **sets** (*list*) – a list of sets (DomainSet or Set) to be filtered and sorted
- **sorted_set_names** (*list*) – a list of set names in the order they should be in the output list, including ones to be removed

- **metadatas** (*list*) – list of SetMetadata objects in the same order as *sorted_set_names*;

Returns a list of sets

Return type list

spinetoolbox.spine_io.exporters.gdx.**sort_records_inplace**(*sets*, *settings*)
Sorts the record lists of given domains according to the order given in settings.

Parameters

- **sets** (*list*) – a list of DomainSet or Set objects whose records are to be sorted
- **settings** (Settings) – 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*, *additional_domains*, *settings*, *indexing_settings*, *merging_settings*, *gams_system_directory=None*)
Exports given database map into .gdx file.

Parameters

- **database_map** (*spinedb_api.DatabaseMapping*) – a database to export
- **file_name** (*str*) – output file name
- **additional_domains** (*list*) – a list of extra domains not in the database
- **settings** (Settings) – export settings
- **indexing_settings** (*dict*) – a dictionary containing settings for indexed parameter expansion
- **merging_settings** (*dict*) – a list of merging settings for parameter merging
- **gams_system_directory** (*str*) – path to GAMS system directory or None to let GAMS choose one for you

spinetoolbox.spine_io.exporters.gdx.**make_settings**(*database_map*)
Builds a Settings object from given database.

Parameters **database_map** (*spinedb_api.DatabaseMapping*) – a database from which domains, sets, records etc are extracted

Returns a Settings object useful for exporting the given *database_map*

class spinetoolbox.spine_io.exporters.gdx.**Settings**(*domain_names*, *set_names*, *records*, *domain_metadatas=None*, *set_metadatas=None*, *global_parameters_domain_name=''*)
This class holds some settings needed by *to_gdx_file()* for .gdx export.

Settings is mostly concerned about the order in which domains, sets and records are exported into the .gdx file. This order is paramount for some models, like TIMES.

Constructs a new Settings object.

Parameters

- **domain_names** (*list*) – a list of Set names
- **set_names** (*list*) – a list of Set names
- **records** (*dict*) – a mapping from Set names to record key tuples
- **domain_metadatas** (*list*) – a list of SetMetadata objects, one for each domain
- **set_metadatas** (*list*) – a list of SetMetadata objects, one for each set
- **global_parameters_domain_name** (*str*) – name of the Set whose parameters to export as GAMS scalars

sorted_domain_names
this list defines the order in which domains are exported into the .gdx file.

domain_metadatas
this list contains SetMetadata objects for each name in *domain_names*

sorted_set_names
this list defines the order in which sets are exported into the .gdx file.

set_metadatas
this list contains SetMetadata objects for each name in *set_names*

global_parameters_domain_name
the name of the domain, parameters of which should be exported as GAMS scalars

add_or_replace_domain(*self*, *domain*, *metadata*)
Adds a new domain or replaces an existing domain's records and metadata.

Parameters

- **domain** (Set) – a domain to add/replace
- **metadata** (SetMetadata) – domain's metadata

Returns True if a new domain was added, False if an existing domain was replaced

domain_index(*self*, *domain*)
Returns an integral index to the domain's name in sorted domain names.

del_domain_at(*self*, *index*)
Erases domain name at given integral index.

update_domain(*self*, *domain*)
Updates domain's records.

sorted_record_key_lists(*self*, *name*)
Returns a list of record keys for given domain or set name.

The list defines the order in which the records are exported into the .gdx file.

Parameters **name** (*str*) – domain or set name

Returns an ordered list of record key lists

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** (Settings) – settings to merge with

static _update_names(*names*, *metadatas*, *updating_names*, *updating_metadatas*)
Updates a list of domain/set names and exportable flags based on reference names and flags.

to_dict(*self*)
Serializes the Settings object to a dict.

static from_dict(*dictionary*)
Deserializes Settings from a dict.

class spinetoolbox.spine_io.exporters.gdx.**ExportFlag**
Bases: enum.Enum

Options for exporting Set objects.

EXPORTABLE
User has declared that the set should be exported.

NON_EXPORTABLE
User has declared that the set should not be exported.

FORCED_EXPORTABLE
Set must be exported no matter what.

FORCED_NON_EXPORTABLE
Set must never be exported.

class spinetoolbox.spine_io.exporters.gdx.**SetMetadata**(*exportable=ExportFlag.EXPORTABLE*, *is_additional=False*)
This class holds some additional configuration for Sets.

exportable
set's export flag

Type *ExportFlag*

is_additional
True if the domain does not exist in the database but is supplied separately.

Type bool

Parameters

- **exportable** (ExportFlag) – set's export flag
- **is_additional** (*bool*) – True if the domain does not exist in the database but is supplied separately.

__eq__(*self*, *other*)
Returns True if other is equal to this metadata.

is_exportable(*self*)
Returns True if Set should be exported.

is_forced(*self*)
Returns True if user's export choices should be overriden.

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

spinetoolbox.spine_io.importers.csv_reader.**select_csv_file**(*parent=None*)
Launches QFileDialog with no filter

class spinetoolbox.spine_io.importers.csv_reader.**CSVConnector**
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

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

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**
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

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.**create_mapping_from_sheet**(*worksheet*)
Checks if sheet is a valid spine excel template, 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

spinetoolbox.spine_io.importers.gdx_connector.**select_gdx_file**(*parent=None*)
Launches QFileDialog with .gdx filter

class spinetoolbox.spine_io.importers.gdx_connector.**GdxConnector**
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

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

spinetoolbox.spine_io.importers.json_reader.**select_json_file**(*parent=None*)
Launches QFileDialog with .json filter

class spinetoolbox.spine_io.importers.json_reader.**JSONConnector**
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

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*)

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

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**
Bases: *spinetoolbox.spine_io.io_api.SourceConnection*

Template class to read data from another QThread.

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

class spinetoolbox.spine_io.connection_manager.**ConnectionManager**(*connection*, *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*

startTableGet

startDataGet

startMappedDataGet

connectionFailed

connectionReady

closeConnection

error

fetchingData

dataReady

tablesReady

mappedDataReady

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 **{str} -- str with table name** (*table*) –

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*)

_new_options (*self*)

set_table_options (*self*, *options*)
Sets connection manager options for current connector

Parameters **{dict} -- Dict with option settings** (*options*) –

set_table_types (*self*, *types*)
Sets connection manager types for current connector

Parameters **{dict} -- Dict with types settings, column** (*types*) –

set_table_row_types (*self*, *types*)
Sets connection manager types for current connector

Parameters **{dict} -- Dict with types settings, row** (*types*) –

option_widget (*self*)
Return a Qwidget with options for reading data from a table in source

close_connection (*self*)
Close and delete thread and worker

class spinetoolbox.spine_io.connection_manager.**ConnectionWorker** (*source*, *connection*, *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

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

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**

Template class to read data from another QThread.

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.io_models

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

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

spinetoolbox.spine_io.io_models.**Margin**

spinetoolbox.spine_io.io_models.**_MAPPING_COLORS**

spinetoolbox.spine_io.io_models.**_ERROR_COLOR**

spinetoolbox.spine_io.io_models.**_COLUMN_TYPE_ROLE**

spinetoolbox.spine_io.io_models.**_COLUMN_NUMBER_ROLE**

spinetoolbox.spine_io.io_models.**_ALLOWED_TYPES**

spinetoolbox.spine_io.io_models.**_TYPE_TO_FONT_AWESOME_ICON**

spinetoolbox.spine_io.io_models.**_MAPTYPE_DISPLAY_NAME**

spinetoolbox.spine_io.io_models.**_DISPLAY_TYPE_TO_TYPE**

spinetoolbox.spine_io.io_models.**_TYPE_TO_DISPLAY_TYPE**

class spinetoolbox.spine_io.io_models.**MappingPreviewModel**(*parent=None*)

Bases: *spinetoolbox.mvcmodels.minimal_table_model.MinimalTableModel*

A model for highlighting columns, rows, and so on, depending on Mapping specification. Used by ImportPreviewWidget.

columnTypesUpdated

rowTypesUpdated

mappingChanged

mapping(*self*)

clear(*self*)

reset_model(*self*, *main_data=None*)

set_mapping(*self*, *mapping*)

Set mapping to display colors from

Parameters **{MappingSpecModel} -- mapping model**(*mapping*) –

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*)

_mapping_data_changed(*self*)

update_colors(*self*)

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

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

data_color(*self*, *index*)

returns background color for index depending on mapping

Parameters **{PySide2.QtCore.QModelIndex} -- index**(*index*) –

Returns [QColor] – QColor of index

index_in_mapping(*self*, *mapping*, *index*)

Checks if index is in mapping

Parameters

- **{Mapping} -- mapping**(*mapping*) –
- **{QModelIndex} -- index**(*index*) –

Returns [bool] – returns True if mapping is in index

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

Returns [List[int]] – list of ints

class spinetoolbox.spine_io.io_models.**MappingSpecModel**(*model*, *parent=None*)
Bases: PySide2.QtCore.QAbstractTableModel

A model to hold a Mapping specification.

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_model_class(*self*, *new_class*)
Change model between Relationship and Object class

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

update_display_table(*self*)

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

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

get_map_append_display(*self*, *mapping*, *name*)

get_map_prepend_display(*self*, *mapping*, *name*)

data(*self*, *index*, *role*)

data_color(*self*, *display_name*)

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

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

headerData(*self*, *section*, *orientation*, *role*)

flags(*self*, *index*)

setData(*self*, *index*, *value*, *role*)

set_type(*self*, *name*, *value*)

set_value(*self*, *name*, *value*)

set_append_str(*self*, *name*, *value*)

set_prepend_str(*self*, *name*, *value*)

get_mapping_from_name(*self*, *name*)

set_mapping_from_name(*self*, *name*, *mapping*)

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

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

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

class spinetoolbox.spine_io.io_models.**MappingListModel**(*mapping_list*, *parent=None*)
Bases: PySide2.QtCore.QAbstractListModel

A model to hold a list of Mappings.

set_model(*self*, *model*)

get_mappings(*self*)

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

data_mapping(*self*, *index*)

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

add_mapping(*self*)

remove_mapping(*self*, *row*)

spinetoolbox.spine_io.io_models.**_create_allowed_types_menu**(*parent*, *trigger_slot*)
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

class spinetoolbox.spine_io.io_models.**HeaderWithButton**(*orientation*, *parent=None*)
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.

display_all

sections_with_buttons

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

set_data_types(*self*, *sections*, *type_str*, *update_viewport=True*)
Sets the data types of given sections (rows, columns).

Parameters

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

- **update_viewport** (*bool*) – True if the buttons need repaint

widget_width (*self*)
Width of widget

Returns [int] – Width of widget

widget_height (*self*)
Height of widget

Returns [int] – Height of widget

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

mousePressEvent (*self*, *mouse_event*)
Move the button to the pressed location and show or hide it if button should not be shown.

leaveEvent (*self*, *event*)
Hide button

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

Parameters

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

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

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

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 **{int} -- logical index of section** (*logical_index*) –

Returns [QSize] – Size of section

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

Parameters

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

fix_widget_positions (*self*)
Update position of interaction button

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

class spinetoolbox.spine_io.io_models.**TableViewWithButtonHeader**(*parent=None*)
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.

_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.

spinetoolbox.spine_io.type_conversion

Module Contents

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.**IntegerSequenceDateTimeConvertSpec**(*start_datetime*, *start_int*, *duration*)
Bases: *spinetoolbox.spine_io.type_conversion.ConvertSpec*

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

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_db_items_dialogs

Classes for custom QDialogs to add items to databases.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

class spinetoolbox.widgets.add_db_items_dialogs.**AddItemsDialog**(*parent*, *db_mngr*, **db_maps*)
Bases: *spinetoolbox.widgets.manage_db_items_dialog.ManageItemsDialog*

A dialog to query user's preferences for new db items.

Init class.

Args parent (DataStoreForm) db_mngr (SpineDBManager) db_maps (iter) DiffDatabaseMapping instances

connect_signals (*self*)

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.widgets.add_db_items_dialogs.**AddObjectClassesDialog**(*parent*, *db_mngr*, **db_maps*)

Bases: *spinetoolbox.widgets.manage_db_items_dialog.ShowIconColorEditorMixin*, *spinetoolbox.widgets.add_db_items_dialogs.AddItemsDialog*

A dialog to query user's preferences for new object classes.

Init class.

Args parent (DataStoreForm) db_mngr (SpineDBManager) db_maps (iter) DiffDatabaseMapping instances

connect_signals(*self*)

accept(*self*)
Collect info from dialog and try to add items.

class spinetoolbox.widgets.add_db_items_dialogs.**AddObjectsDialog**(*parent*, *db_mngr*, **db_maps*, *class_name=None*, *force_default=False*)

Bases: *spinetoolbox.widgets.manage_db_items_dialog.GetObjectClassesMixin*, *spinetoolbox.widgets.add_db_items_dialogs.AddItemsDialog*

A dialog to query user's preferences for new objects.

Init class.

Args parent (DataStoreForm) db_mngr (SpineDBManager) db_maps (iter) 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.widgets.add_db_items_dialogs.**AddRelationshipClassesDialog**(*parent*, *db_mngr*, **db_maps*, *object_class_one_name=None*, *force_default=False*)

Bases: *spinetoolbox.widgets.manage_db_items_dialog.GetObjectClassesMixin*, *spinetoolbox.widgets.add_db_items_dialogs.AddItemsDialog*

A dialog to query user's preferences for new relationship classes.

Init class.

Args parent (DataStoreForm) db_mngr (SpineDBManager) db_maps (iter) 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*)

accept(*self*)
Collect info from dialog and try to add items.

class spinetoolbox.widgets.add_db_items_dialogs.**AddRelationshipsDialog**(*parent*, *db_mngr*, **db_maps*, *relationship_class_key=None*, *object_class_name=None*, *object_name=None*, *force_default=False*)

Bases: *spinetoolbox.widgets.manage_db_items_dialog.GetObjectsMixin*, *spinetoolbox.widgets.add_db_items_dialogs.AddItemsDialog*

A dialog to query user's preferences for new relationships.

Init class.

Args parent (DataStoreForm) db_mngr (SpineDBManager) db_maps (iter) DiffDatabaseMapping instances relationship_class_key (tuple): (class_name, object_class_name_list) object_name (str): default object name object_class_name (str): default object class name force_default (bool): if True, defaults are non-editable

connect_signals(*self*)
Connect signals to slots.

call_reset_model(*self*, *index*)
Called when relationship class's combobox's index changes. Update relationship_class attribute accordingly and reset model.

reset_model(*self*)
Setup model according to current relationship class selected in combobox.

_handle_model_data_changed(*self*, *top_left*, *bottom_right*, *roles*)

accept(*self*)
Collect info from dialog and try to add items.

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

class spinetoolbox.widgets.add_project_item_widget.**AddProjectItemWidget**(*toolbox*, *x*, *y*, *initial_name=''*)

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.

name_changed(*self*)
Update label to show upcoming folder name.

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.custom_delegates

Custom item delegates.

author

M. Marin (KTH)

date 1.9.2018

Module Contents

class spinetoolbox.widgets.custom_delegates.**ComboBoxDelegate**(*parent*, *choices*)
Bases: PySide2.QtWidgets.QItemDelegate

createEditor(*self*, *parent*, *option*, *index*)

paint(*self*, *painter*, *option*, *index*)

setEditorData(*self*, *editor*, *index*)

setModelData(*self*, *editor*, *model*, *index*)

updateEditorGeometry(*self*, *editor*, *option*, *index*)

currentItemChanged(*self*)

class spinetoolbox.widgets.custom_delegates.**LineEditDelegate**
Bases: PySide2.QtWidgets.QItemDelegate

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. Set up a validator depending on datatype.

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.QItemDelegate

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.**PivotTableDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.CheckBoxDelegate*

parameter_value_editor_requested

data_committed

setModelData (*self*, *editor*, *model*, *index*)
Send signal.

_is_entity_index (*self*, *index*)

paint (*self*, *painter*, *option*, *index*)

editorEvent (*self*, *event*, *model*, *option*, *index*)

createEditor (*self*, *parent*, *option*, *index*)

class spinetoolbox.widgets.custom_delegates.**GetObjectClassIdMixin**
Allows getting the object class id from the name.

_get_object_class_id (*self*, *index*, *db_map*)

class spinetoolbox.widgets.custom_delegates.**GetRelationshipClassIdMixin**
Allows getting the relationship class id from the name.

_get_relationship_class_id (*self*, *index*, *db_map*)

class spinetoolbox.widgets.custom_delegates.**ParameterDelegate** (*parent*, *db_mngr*)
Bases: PySide2.QtWidgets.QItemDelegate

Base class for all custom parameter delegates.

parent
tree or graph view form

Type *DataStoreForm*

db_mngr

Type *SpineDBManager*

data_committed

setModelData (*self*, *editor*, *model*, *index*)
Send signal.

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.widgets.custom_delegates.**DatabaseNameDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the database name.

createEditor (*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.widgets.custom_delegates.**ParameterValueOrDefaultValueDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the either the value or the default value.

parameter_value_editor_requested

setModelData(*self*, *editor*, *model*, *index*)
Emits the data_committed signal with new data.

static _str_to_int_or_float(*string*)

_create_or_request_parameter_value_editor(*self*, *parent*, *option*, *index*, *db_map*)
Returns a CustomLineEditor or NumberParameterInlineEditor if the data from index is not of special type. Otherwise, emit the signal to request a standalone *ParameterValueEditor* from parent widget.

class spinetoolbox.widgets.custom_delegates.**ParameterDefaultValueDelegate**
Bases: *spinetoolbox.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.widgets.custom_delegates.**ParameterValueDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.ParameterValueOrDefaultValueDelegate*

A delegate for the parameter value.

_get_entity_class_id(*self*, *index*, *db_map*)

_get_value_list(*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.widgets.custom_delegates.**ObjectParameterValueDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.GetObjectClassIdMixin*, *spinetoolbox.widgets.custom_delegates.ParameterValueDelegate*

A delegate for the object parameter value.

entity_class_id_key

_get_entity_class_id(*self*, *index*, *db_map*)

class spinetoolbox.widgets.custom_delegates.**RelationshipParameterValueDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.GetRelationshipClassIdMixin*, *spinetoolbox.widgets.custom_delegates.ParameterValueDelegate*

A delegate for the relationship parameter value.

entity_class_id_key

_get_entity_class_id(*self*, *index*, *db_map*)

class spinetoolbox.widgets.custom_delegates.**TagListDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the parameter tag list.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.widgets.custom_delegates.**ValueListDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the parameter value-list.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.widgets.custom_delegates.**ObjectClassNameDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the object class name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.widgets.custom_delegates.**RelationshipClassNameDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the relationship class name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.widgets.custom_delegates.**ObjectParameterNameDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.GetObjectClassIdMixin*, *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the object parameter name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.widgets.custom_delegates.**RelationshipParameterNameDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.GetRelationshipClassIdMixin*, *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the relationship parameter name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.widgets.custom_delegates.**ObjectNameDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.GetObjectClassIdMixin*, *spinetoolbox.widgets.custom_delegates.ParameterDelegate*

A delegate for the object name.

createEditor(*self*, *parent*, *option*, *index*)
Returns editor.

class spinetoolbox.widgets.custom_delegates.**ObjectNameListDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.GetRelationshipClassIdMixin*, *spinetoolbox.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.widgets.custom_delegates.**ManageItemsDelegate**
Bases: PySide2.QtWidgets.QItemDelegate

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*)

class spinetoolbox.widgets.custom_delegates.**ManageObjectClassesDelegate**
Bases: *spinetoolbox.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.widgets.custom_delegates.**ManageObjectsDelegate**
Bases: *spinetoolbox.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.widgets.custom_delegates.**ManageRelationshipClassesDelegate**
Bases: *spinetoolbox.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.widgets.custom_delegates.**ManageRelationshipsDelegate**
Bases: *spinetoolbox.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.widgets.custom_delegates.**RemoveEntitiesDelegate**
Bases: *spinetoolbox.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.

class spinetoolbox.widgets.custom_delegates.**ManageParameterTagsDelegate**
Bases: *spinetoolbox.widgets.custom_delegates.ManageItemsDelegate*

A delegate for the model and view in ManageParameterTagsDialog.

parent
parent dialog

Type *ManageItemsDialog*

createEditor(*self*, *parent*, *option*, *index*)
Return editor.

class spinetoolbox.widgets.custom_delegates.**ForeignKeysDelegate**(*parent*)
Bases: PySide2.QtWidgets.QItemDelegate

A QComboBox delegate with checkboxes.

parent
spine datapackage widget

Type *SpineDatapackageWidget*

data_committed

close_field_name_list_editor(*self*, *editor*, *index*, *model*)

createEditor(*self*, *parent*, *option*, *index*)
Return editor.

setEditorData(*self*, *editor*, *index*)
Set editor data.

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

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.**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*)
Bases: PySide2.QtWidgets.QTableView

A check list editor.

Initialize class.

keyPressEvent(*self*, *event*)
Toggles checked state if the user presses space.

toggle_checked_state(*self*, *index*)
Toggles checked state of given index.

Parameters **index** (*QModelIndex*) –

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*)

class spinetoolbox.widgets.custom_editors.**NumberParameterInlineEditor**(*parent*)
Bases: PySide2.QtWidgets.QDoubleSpinBox

An editor widget for numeric (datatype double) parameter values.

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

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.**ToolSpecificationContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for Tool specifications.

Parameters

- **parent** (*QWidget*) – Parent for menu widget (ToolboxUI)
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

class spinetoolbox.widgets.custom_menus.**EntityTreeContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for object tree items in tree view form.

Parameters

- **parent** (*QWidget*) – Parent for menu widget
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

class spinetoolbox.widgets.custom_menus.**ObjectTreeContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.EntityTreeContextMenu*

Context menu class for object tree items in tree view form.

Parameters

- **parent** (*QWidget*) – Parent for menu widget
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

class spinetoolbox.widgets.custom_menus.**RelationshipTreeContextMenu**

Bases: *spinetoolbox.widgets.custom_menus.EntityTreeContextMenu*

Context menu class for relationship tree items in tree view form.

class spinetoolbox.widgets.custom_menus.**ParameterContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for object (relationship) parameter items in tree views.

Parameters

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

- **index** (*QModelIndex*) – Index of item that requested the context-menu

class spinetoolbox.widgets.custom_menus.**EditableParameterValueContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for object (relationship) parameter value items in tree views.

Parameters

- **parent** (*QWidget*) – Parent for menu widget
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

_plot_in_window(*self*, *action*)

Sets the option attributes ready for plotting in an existing window.

class spinetoolbox.widgets.custom_menus.**ParameterValueListContextMenu**(*parent*, *position*, *index*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for parameter enum view in tree view form.

Parameters

- **parent** (*QWidget*) – Parent for menu widget
- **position** (*QPoint*) – Position on screen
- **index** (*QModelIndex*) – Index of item that requested the context-menu

class spinetoolbox.widgets.custom_menus.**GraphViewContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for qgraphics view in graph view.

Parameters

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

class spinetoolbox.widgets.custom_menus.**EntityItemContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Context menu class for entity graphic items in graph view.

Parameters

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

class spinetoolbox.widgets.custom_menus.**ObjectItemContextMenu**(*parent*, *position*, *graphics_item*)

Bases: *spinetoolbox.widgets.custom_menus.EntityItemContextMenu*

Parameters

- **parent** (*QWidget*) – Parent for menu widget (GraphViewForm)
- **position** (*QPoint*) – Position on screen
- **graphics_item** (spinetoolbox.widgets.graph_view_graphics_items.ObjectItem) – item that requested the menu

class spinetoolbox.widgets.custom_menus.**RelationshipItemContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.EntityItemContextMenu*

Parameters

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

class spinetoolbox.widgets.custom_menus.**OpenProjectDialogComboBoxContextMenu**(*parent*, *position*)

Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

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.**AddToolSpecificationPopupMenu**(*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.**ToolSpecificationOptionsPopupmenu**(*parent*, *tool*)

Bases: *spinetoolbox.widgets.custom_menus.CustomPopupMenu*

Popup menu class for tool specification options button in Tool item.

Parameters

- **parent** (*QWidget*) – Parent widget of this menu (ToolboxUI)
- **tool** (Tool) – Tool item that is associated with the pressed button

class spinetoolbox.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.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)

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*

Parameters **parent** (DataStoreForm) –

filterChanged

emit_filter_changed(*self*, *valid_values*)

class spinetoolbox.widgets.custom_menus.**ParameterViewFilterMenu**(*parent*, *source_model*, *show_empty=True*)

Bases: *spinetoolbox.widgets.custom_menus.FilterMenuBase*

Parameters

- **parent** (DataStoreForm) –
- **source_model** (CompoundParameterModel) – a model to lazily get data from

filterChanged

emit_filter_changed(*self*, *valid_values*)

class spinetoolbox.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** (DataStoreForm) –
- **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*)

class spinetoolbox.widgets.custom_menus.**PivotTableModelMenu**(*parent*)

Bases: PySide2.QtWidgets.QMenu

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

_DELETE_OBJECT = Remove selected objects

_DELETE_RELATIONSHIP = Remove selected relationships

_DELETE_PARAMETER = Remove selected parameter definitions

delete_values(*self*)

delete_objects(*self*)

delete_relationships(*self*)

delete_parameters(*self*)

open_value_editor(*self*)
Opens the parameter value editor for the first selected cell.

plot(*self*)
Plots the selected cells in the pivot table.

request_menu(*self*, *QPos=None*)
Shows the context menu on the screen.

_find_selected_indexes(*self*)

_update_actions_enable(*self*)

_update_actions_text(*self*)

_plot_in_window(*self*, *action*)

class spinetoolbox.widgets.custom_menus.**PivotTableHorizontalHeaderMenu**(*proxy_model*, *parent=None*)

Bases: PySide2.QtWidgets.QMenu

A context menu for the horizontal header of a pivot table.

Parameters

- **proxy_model** (PivotTableSortFilterProxy) – a proxy model
- **parent** (*QWidget*) – a parent widget

request_menu(*self*, *pos*)
Shows the context menu on the screen.

_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.widgets.custom_menus.**_prepare_plot_in_window_menu**(*menu*)
Fills a given menu with available plot window names.

spinetoolbox.widgets.custom_qgraphicsscene

Custom QGraphicsScene used in the Design View.

author

P. Savolainen (VTT)

date 13.2.2019

Module Contents

class spinetoolbox.widgets.custom_qgraphicsscene.**CustomQGraphicsScene**(*parent*, *toolbox*)

Bases: *spinetoolbox.widgets.shrinking_scene.ShrinkingScene*

A scene that handles drag and drop events of DraggableWidget sources.

Parameters

- **parent** (*QObject*) – scene's parent object
- **toolbox** (ToolboxUI) – reference to the main window

connect_signals(*self*)
Connect scene signals.

resize_scene(*self*)
Resize scene to be at least the size of items bounding rectangle. Does not let the scene shrink.

scene_changed(*self*, *rects*)
Resize scene as it changes.

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_grid(*self*, *bg*)
Enable or disable background grid.

Parameters **bg** (*boolean*) – True to draw grid, False to fill background with a solid color

dragLeaveEvent(*self*, *event*)
Accept event.

dragEnterEvent(*self*, *event*)
Accept event. Then call the super class method only if drag source is not a DraggableWidget (from Add Item toolbar).

dragMoveEvent(*self*, *event*)
Accept event. Then call the super class method only if drag source is not a DraggableWidget (from Add Item toolbar).

dropEvent(*self*, *event*)
Only accept drops when the source is an instance of DraggableWidget (from Add Item toolbar). Capture text from event's mimedata and show the appropriate 'Add Item form.'

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

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

class spinetoolbox.widgets.custom_qgraphicsviews.**CustomQGraphicsView**(*parent*)
Bases: PySide2.QtWidgets.QGraphicsView

Super class for Design and Graph QGraphicsViews.

parent
Parent widget

Type QWidget

Init CustomQGraphicsView.

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

enterEvent (*self*, *event*)
Overridden method. Do not show the stupid open hand mouse cursor.

Parameters **event** (*QEvent*) – event

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.

wheelEvent (*self*, *event*)
Zoom 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

_update_zoom_limits (*self*, *rect*)
Updates the minimum zoom limit and the zoom level with which the entire scene fits the view.

Parameters **rect** (*QRectF*) – the scene's rect

scaling_time (*self*, *pos*)
Called when animation value for smooth zoom changes. Perform zoom.

anim_finished (*self*)
Called when animation for smooth zoom finishes. Clean up.

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*)
Reset zoom to the default factor.

gentle_zoom (*self*, *factor*, *zoom_focus*)
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

_ensure_item_visible (*self*, *item*)
Resets zoom if item is not visible.

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)

mousePressEvent(*self*, *event*)

Manage drawing of links. Handle the case where a link is being drawn and the user doesn't hit a connector button.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Mouse event

mouseMoveEvent(*self*, *event*)

Update line end position.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Mouse event

set_ui(*self*, *toolbox*)

Set a new scene into the Design View when app is started.

init_scene(*self*, *empty=False*)

Resize scene and add a link drawer on scene. The scene must be cleared before calling this.

Parameters **empty** (*boolean*) – True when creating a new project

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.

draw_links(*self*, *connector*)
Draw links when slot button is clicked.

Parameters **connector** (ConnectorButton) – Connector button that triggered the drawing

notify_destination_items(*self*)
Notify destination items that they have been connected to a source item.

connect_engine_signals(*self*, *engine*)
Connects signals needed for icon animations from given engine.

_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.

class spinetoolbox.widgets.custom_qgraphicsviews.**GraphQGraphicsView**
Bases: *spinetoolbox.widgets.custom_qgraphicsviews.CustomQGraphicsView*

QGraphicsView for the Graph View.

item_dropped

context_menu_requested

dragLeaveEvent(*self*, *event*)
Accept event. Then call the super class method only if drag source is not DragListView.

dragEnterEvent(*self*, *event*)
Accept event. Then call the super class method only if drag source is not DragListView.

dragMoveEvent(*self*, *event*)
Accept event. Then call the super class method only if drag source is not DragListView.

dropEvent(*self*, *event*)
Only accept drops when the source is an instance of DragListView. Capture text from event's mimedata and emit signal.

contextMenuEvent(*self*, *e*)
Show context menu.

Parameters **e** (*QContextMenuEvent*) – Context menu event

gentle_zoom(*self*, *factor*, *zoom_focus*)
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

reset_zoom(*self*)
Reset zoom to the default factor.

init_zoom(*self*)
Init zoom.

adjust_items_to_zoom(*self*)
Update items geometry after performing a zoom.

Some items (e.g. ArcItem) need this to stay the same size after a zoom.

spinetoolbox.widgets.custom_qlineedit

Classes for custom line edits.

author

M. Marin (KTH)

date 11.10.2018

Module Contents

class spinetoolbox.widgets.custom_qlineedit.**CustomQLineEdit**
Bases: PySide2.QtWidgets.QLineEdit

A custom QLineEdit that accepts file drops and displays the path.

parent
Parent for line edit widget (DataStoreWidget)

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.

spinetoolbox.widgets.custom_qlistview

Classes for custom QListView.

author

M. Marin (KTH)

date 14.11.2018

Module Contents

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

spinetoolbox.widgets.custom_qtableview

Custom QTableView classes that support copy-paste and the like.

author

M. Marin (KTH)

date 18.5.2018

Module Contents

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.**PivotTableView**(*parent=None*)
Bases: *spinetoolbox.widgets.custom_qtableview.CopyPasteTableView*

Custom QTableView class with pivot capabilities.

parent
The parent of this view

Type QWidget

Initialize the class.

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)

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

class spinetoolbox.widgets.custom_qtextbrowser.**CustomQTextBrowser**(*parent*)
Bases: PySide2.QtWidgets.QTextBrowser

Custom QTextBrowser class.

parent
Parent widget

Type QWidget

max_blocks
Returns the upper limit of text blocks that can be appended to the widget.

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

spinetoolbox.widgets.custom_qtreeview

Classes for custom QTreeView.

author

M. Marin (KTH)

date 25.4.2018

Module Contents

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.**EntityTreeView**(*parent*)
Bases: *spinetoolbox.widgets.custom_qtreeview.CopyTreeView*

Custom QTreeView class for object tree in DataStoreForm.

parent
The parent of this view

Type QWidget

Initialize the view.

edit_key_pressed

edit(*self*, *index*, *trigger*, *event*)
Send signal instead of editing item, so DataStoreForm can catch this signal and open a custom QDialog for edition.

class spinetoolbox.widgets.custom_qtreeview.**StickySelectionEntityTreeView**
Bases: *spinetoolbox.widgets.custom_qtreeview.EntityTreeView*

Custom QTreeView class for object tree in DataStoreForm.

parent
The parent of this view

Type QWidget

mousePressEvent(*self*, *event*)
Overrides selection behaviour if the user has selected sticky selection in Settings. If sticky selection is enabled, multi-selection is enabled when selecting items in the Object tree. Pressing the Ctrl-button down, enables single selection. If sticky selection is disabled, single selection is enabled and pressing the Ctrl-button down enables multi-selection.

Parameters **event** (*QMouseEvent*) –

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

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*

Init class.

Parameters **parent** (*QWidget*) –

class spinetoolbox.widgets.custom_qwidgets.**DataToValueFilterWidget**(*parent*, *data_to_value*, *show_empty=True*)
Bases: *spinetoolbox.widgets.custom_qwidgets.FilterWidgetBase*

Init class.

Parameters

- **parent** (*QWidget*) –
- **data_to_value** (*method*) – a method to translate item data to a value for display role

class spinetoolbox.widgets.custom_qwidgets.**LazyFilterWidget**(*parent*, *source_model*, *show_empty=True*)
Bases: *spinetoolbox.widgets.custom_qwidgets.FilterWidgetBase*

Init class.

Parameters

- **parent** (DataStoreForm) –
- **source_model** (CompoundParameterModel, *optional*) – a model to lazily get data from

set_model(*self*)

class spinetoolbox.widgets.custom_qwidgets.**ZoomWidgetAction**(*parent=None*)
Bases: PySide2.QtWidgets.QWidgetAction

A zoom widget action.

Class constructor.

Parameters **parent** (*QWidget*) – the widget's parent

minus_pressed

plus_pressed

reset_pressed

_handle_hovered(*self*)
Runs when the zoom widget action is hovered. Hides other menus under the parent widget which are being shown. This is the default behavior for hovering QAction, but for some reason it's not the case for hovering QWidgetAction.

class spinetoolbox.widgets.custom_qwidgets.**ZoomWidget**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

Class constructor.

Parameters **parent** (*QWidget*) – the widget's parent

minus_pressed

plus_pressed

reset_pressed

paintEvent(*self*, *event*)
Overridden method.

spinetoolbox.widgets.data_store_widget

Contains the DataStoreForm class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

class spinetoolbox.widgets.data_store_widget.**DataStoreFormBase**(*db_mngr*, **db_urls*)
Bases: PySide2.QtWidgets.QMainWindow

Base class for DataStoreForm

Initializes form.

Parameters

- **db_mngr** (SpineDBManager) – The manager to use
- ***db_urls** (*tuple*) – Database url, codename.

msg

msg_error

error_box

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*)
Appends regular message to status bar.

Parameters **msg** (*str*) – String to show in QStatusBar

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.

_find_focus_child(*self*)

selected_entity_class_ids(*self*, *entity_class_type*)
Returns object class ids selected in object tree *and* parameter tag toolbar.

_accept_selection(*self*, *widget*)
Clears selection from all widgets except the given one, so there's only one selection in the form at a time. In addition, registers the given widget as the official source for all operations involving selections (copy, remove, edit), but only in case it *has* a selection.

remove_selection(*self*, *checked=False*)
Removes selection of items.

copy(*self*, *checked=False*)
Copies data to clipboard.

paste(*self*, *checked=False*)
Pastes data from clipboard.

show_import_file_dialog(*self*, *checked=False*)
Shows dialog to allow user to select a file to import.

export_database(*self*, *checked=False*)
Exports data from database into a file.

_select_database(*self*)
Lets user select a database from available databases.

Shows a dialog from which user can select a single database. If there is only a single database it is selected automatically and no dialog is shown.

Returns the database map of the database or None if no database was selected

export_to_excel(*self*, *db_map*, *file_path*)
Exports data from database into Excel file.

export_to_sqlite(*self*, *db_map*, *file_path*)
Exports data from database into SQlite file.

refresh_session(*self*, *checked=False*)

commit_session(*self*, *checked=False*)
Commits session.

rollback_session(*self*, *checked=False*)

receive_session_committed(*self*, *db_maps*)

receive_session_rolled_back(*self*, *db_maps*)

receive_session_refreshed(*self*, *db_maps*)

_handle_tag_button_toggled(*self*, *db_map_ids*, *checked*)
Updates filter according to selected tags.

show_manage_parameter_tags_form(*self*, *checked=False*)

_handle_parameter_value_list_selection_changed(*self*, *selected*, *deselected*)
Accepts selection.

show_parameter_value_list_context_menu(*self*, *pos*)
Shows the context menu for parameter value list tree view.

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

remove_parameter_value_lists(*self*)
Removes selection of parameter value-lists.

notify_items_changed(*self*, *action*, *item_type*, *db_map_data*)
Enables or disables actions and informs the user about what just happened.

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_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_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_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_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

class spinetoolbox.widgets.data_store_widget.**DataStoreForm**(*db_mngr*, **db_urls*)
Bases: *spinetoolbox.widgets.tabular_view_mixin.TabularViewMixin*, *spinetoolbox.widgets.graph_view_mixin.GraphViewMixin*, *spinetoolbox.widgets.parameter_view_mixin.ParameterViewMixin*, *spinetoolbox.widgets.tree_view_mixin.TreeViewMixin*, *spinetoolbox.widgets.data_store_widget.DataStoreFormBase*

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*)

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_tree_style(*self*, *checked=False*)
Applies the tree style, inspired in the former tree view.

apply_tabular_style(*self*, *checked=False*)
Applies the tree style, inspired in the former tabular view.

apply_graph_style(*self*, *checked=False*)
Applies the tree style, inspired in the former graph view.

spinetoolbox.widgets.datetime_editor

An editor widget for editing datetime database (relationship) parameter values.

author

A. Soininen (VTT)

date 28.6.2019

Module Contents

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.db_session_history_dialog

Classes to show db session history

author

M. Marin (KTH)

date 5.2.2020

Module Contents

class spinetoolbox.widgets.db_session_history_dialog.**DBSessionHistoryModel**(*parent*, *db_mngr*, **db_maps*)
Bases: PySide2.QtGui.QStandardItemModel

build(*self*)

class spinetoolbox.widgets.db_session_history_dialog.**DBSessionHistoryView**(*parent*, *db_mngr*, **db_maps*)
Bases: PySide2.QtWidgets.QColumnView

class spinetoolbox.widgets.db_session_history_dialog.**DBSessionHistoryDialog**(*parent*, *db_mngr*, **db_maps*)
Bases: PySide2.QtWidgets.QDialog

Initialize class

spinetoolbox.widgets.duration_editor

An editor widget for editing duration database (relationship) parameter values.

author

A. Soininen (VTT)

date 28.6.2019

Module Contents

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.edit_db_items_dialogs

Classes for custom QDialogs to edit items in databases.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

class spinetoolbox.widgets.edit_db_items_dialogs.**EditOrRemoveItemsDialog**(*parent*, *db_mngr*)
Bases: *spinetoolbox.widgets.manage_db_items_dialog.ManageItemsDialog*

all_databases(*self*, *row*)
Returns a list of db names available for a given row. Used by delegates.

class spinetoolbox.widgets.edit_db_items_dialogs.**EditObjectClassesDialog**(*parent*, *db_mngr*, *selected*)
Bases: *spinetoolbox.widgets.manage_db_items_dialog.ShowIconColorEditorMixin*, *spinetoolbox.widgets.edit_db_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for updating object classes.

Init class.

Parameters

- **parent** (DataStoreForm) – data store widget
- **db_mngr** (SpineDBManager) – the manager to do the update
- **selected** (*set*) – set of ObjectClassItem instances to edit

connect_signals(*self*)

accept(*self*)
Collect info from dialog and try to update items.

class spinetoolbox.widgets.edit_db_items_dialogs.**EditObjectsDialog**(*parent*, *db_mngr*, *selected*)

Bases: *spinetoolbox.widgets.edit_db_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for updating objects.

Init class.

Parameters

- **parent** (DataStoreForm) – 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.widgets.edit_db_items_dialogs.**EditRelationshipClassesDialog**(*parent*, *db_mngr*, *selected*)

Bases: *spinetoolbox.widgets.edit_db_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for updating relationship classes.

Init class.

Parameters

- **parent** (DataStoreForm) – 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.widgets.edit_db_items_dialogs.**EditRelationshipsDialog**(*parent*, *db_mngr*, *selected*, *class_key*)

Bases: *spinetoolbox.widgets.manage_db_items_dialog.GetObjectsMixin*, *spinetoolbox.widgets.edit_db_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for updating relationships.

Init class.

Parameters

- **parent** (DataStoreForm) – 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.widgets.edit_db_items_dialogs.**RemoveEntitiesDialog**(*parent*, *db_mngr*, *selected*)
Bases: *spinetoolbox.widgets.edit_db_items_dialogs.EditOrRemoveItemsDialog*

A dialog to query user's preferences for removing tree items.

Init class.

Parameters

- **parent** (DataStoreForm) – 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.

class spinetoolbox.widgets.edit_db_items_dialogs.**ManageParameterTagsDialog**(*parent*, *db_mngr*, **db_maps*)
Bases: *spinetoolbox.widgets.manage_db_items_dialog.ManageItemsDialog*

A dialog to query user's preferences for managing parameter tags.

Init class.

Parameters

- **parent** (DataStoreForm) – data store widget
- **db_mngr** (SpineDBManager) – the manager to do the removal
- **db_maps** (*iter*) – DiffDatabaseMapping instances

all_databases(*self*, *row*)
Returns a list of db names available for a given row. Used by delegates.

accept(*self*)
Collect info from dialog and try to update, remove, add items.

spinetoolbox.widgets.frozen_table_view

Custom QTableView classes that support copy-paste and the like.

author

M. Marin (KTH)

date 18.5.2018

Module Contents

class spinetoolbox.widgets.frozen_table_view.**FrozenTableView**(*parent=None*)
Bases: PySide2.QtWidgets.QTableView

header_dropped

area

dragEnterEvent(*self*, *event*)

dragMoveEvent(*self*, *event*)

dropEvent(*self*, *event*)

spinetoolbox.widgets.graph_view_demo

Contains the GraphViewForm class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

class spinetoolbox.widgets.graph_view_demo.**GraphViewDemo**(*parent*)
Bases: *spinetoolbox.widgets.state_machine_widget.StateMachineWidget*

A widget that shows a demo for the graph view.

Initializes class.

Parameters **parent** (*GraphViewForm*) –

_make_select_one(*self*)

_make_select_more(*self*)

_make_good_bye(*self*)

set_up_machine(*self*)

class spinetoolbox.widgets.graph_view_demo.**SelectionAnimation**(*view*, *command*, *duration=2000*, *max_steps=4*)
Bases: PySide2.QtCore.QVariantAnimation

Parameters

- **view** (*QAbstractItemView*) –
- **command** (*QItemSelectionModel.SelectionFlags*) –
- **duration** (*int*) – milliseconds
- **max_steps** (*int*) –

static _random_point(*rect*)

updateState(*self*, *new*, *old*)

_handle_value_changed(*self*, *value*)

_handle_current_loop_changed(*self*, *loop*)

_handle_finished(*self*)

spinetoolbox.widgets.graph_view_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

class spinetoolbox.widgets.graph_view_graphics_items.**EntityItem**(*graph_view_form*, *x*, *y*, *extent*, *entity_id=None*, *entity_class_id=None*)

Bases: PySide2.QtWidgets.QGraphicsPixmapItem

Initializes item

Parameters

- **graph_view_form** (*GraphViewForm*) – 'owner'
- **x** (*float*) – x-coordinate of central point
- **y** (*float*) – y-coordinate of central point
- **extent** (*int*) – Preferred extent
- **entity_id** (*int, optional*) – The entity id in case of a non-wip item
- **entity_class_id** (*int, optional*) – The entity class id in case of a wip item

entity_type

entity_name

entity_class_type

entity_class_id

entity_class_name

boundingRect(*self*)

_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) –

become_wip(*self*)
Turns this item into a work-in-progress.

_make_wip_tool_tip(*self*)

become_whole(*self*)
Removes the wip status from this item.

adjust_to_zoom(*self*, *transform*)
Saves the view's transform to determine collisions later on.

Parameters **transform** (*QTransform*) – The view's transformation matrix after the zoom.

device_rect(*self*)
Returns the item's rect in devices's coordinates. Used to accurately determine collisions.

_find_merge_target(*self*)
Returns a suitable merge target if any.

Returns spinetoolbox.widgets.graph_view_graphics_items.EntityItem, NoneType

_is_target_valid(*self*)
Whether or not the registered merge target is valid.

Returns bool

merge_into_target(*self*, *force=False*)
Merges this item into the registered target if valid.

Returns True if merged, False if not.

Return type bool

mousePressEvent(*self*, *event*)
Saves original position for bouncing purposes.

Parameters **event** (*QGraphicsSceneMouseEvent*) –

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*) –

move_arc_items(*self*, *pos_diff*)
Moves arc items.

Parameters **pos_diff** (*QPoint*) –

mouseReleaseEvent(*self*, *event*)
Merges the item into the registered target if any. Bounces it if not possible. Shrinks the scene if needed.

Parameters **event** (*QGraphicsSceneMouseEvent*) –

_bounce_back(*self*, *current_pos*)
Bounces the item back from given position to its original position.

Parameters **current_pos** (*QPoint*) –

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*) –

wipe_out(*self*)
Removes this item and all its arc items from the scene.

contextMenuEvent(*self*, *e*)
Shows context menu.

Parameters **e** (*QGraphicsSceneMouseEvent*) – Mouse event

_show_item_context_menu_in_parent(*self*, *pos*)

class spinetoolbox.widgets.graph_view_graphics_items.**RelationshipItem**
Bases: *spinetoolbox.widgets.graph_view_graphics_items.EntityItem*

Relationship item to use with GraphViewForm.

entity_type

object_class_id_list

object_name_list

object_id_list

db_representation

_init_bg(*self*)

validate_member_objects(*self*)
Goes through connected object items and tries to complete the relationship.

move_arc_items(*self*, *pos_diff*)
Moves arc items.

Parameters **pos_diff** (*QPoint*) –

_make_wip_tool_tip(*self*)

become_whole(*self*)

_show_item_context_menu_in_parent(*self*, *pos*)

class spinetoolbox.widgets.graph_view_graphics_items.**ObjectItem**(*graph_view_form*, *x*, *y*, *extent*, *entity_id=None*, *entity_class_id=None*)
Bases: *spinetoolbox.widgets.graph_view_graphics_items.EntityItem*

Initializes the item.

Parameters

- **graph_view_form** (*GraphViewForm*) – 'owner'
- **x** (*float*) – x-coordinate of central point
- **y** (*float*) – y-coordinate of central point
- **extent** (*int*) – preferred extent
- **entity_id** (*int, optional*) – object id, if not given the item becomes a template
- **entity_class_id** (*int, optional*) – object class id, for template items

Raises ValueError – in case object_id and object_class_id are both not provided

entity_type

db_representation

refresh_name(*self*)
Refreshes the name.

_paint_as_selected(*self*)

_make_wip_tool_tip(*self*)

become_whole(*self*)

refresh_description(*self*)

edit_name(*self*)
Starts editing the object name.

finish_name_editing(*self*, *text*)
Runs when the user finishes editing the name. Adds or updates the object in the database.

Parameters **text** (*str*) – The new name.

move_arc_items(*self*, *pos_diff*)
Moves arc items.

Parameters **pos_diff** (*QPoint*) –

keyPressEvent(*self*, *event*)
Starts editing the name if F2 is pressed.

Parameters **event** (*QKeyEvent*) –

mouseDoubleClickEvent(*self*, *event*)
Starts editing the name.

Parameters **event** (*QGraphicsSceneMouseEvent*) –

_is_in_wip_relationship(*self*)

_is_target_valid(*self*)
Whether or not the registered merge target is valid.

Returns bool

merge_into_target(*self*, *force=False*)
Merges this item into the registered target if valid.

Parameters **force** (*bool*) –

Returns True if merged, False if not.

Return type bool

_show_item_context_menu_in_parent(*self*, *pos*)

class spinetoolbox.widgets.graph_view_graphics_items.**EntityLabelItem**(*entity_item*)
Bases: PySide2.QtWidgets.QGraphicsTextItem

Label item for items in GraphViewForm.

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.

set_bg_color(*self*, *bg_color*)
Sets background color.

Parameters **bg_color**(*QColor*) –

start_editing(*self*)
Starts editing.

keyPressEvent(*self*, *event*)
Keeps text centered as the user types. Gives up focus when the user presses Enter or Return.

Parameters **event**(*QKeyEvent*) –

focusOutEvent(*self*, *event*)
Ends editing and sends entity_name_edited signal.

mouseDoubleClickEvent(*self*, *event*)
Starts editing the name.

Parameters **event**(*QGraphicsSceneMouseEvent*) –

class spinetoolbox.widgets.graph_view_graphics_items.**ArcItem**(*rel_item*, *obj_item*, *width*, *is_wip=False*)
Bases: PySide2.QtWidgets.QGraphicsLineItem

Arc item to use with GraphViewForm. 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

mousePressEvent(*self*, *event*)
Accepts the event so it's not propagated.

other_item(*self*, *item*)

become_wip(*self*)
Turns this arc into a work-in-progress.

become_whole(*self*)
Removes the wip status from this arc.

move_rel_item_by(*self*, *pos_diff*)
Moves source point.

Parameters **pos_diff** (*QPoint*) –

move_obj_item_by(*self*, *pos_diff*)
Moves destination point.

Parameters **pos_diff** (*QPoint*) –

adjust_to_zoom(*self*, *transform*)
Adjusts the item's geometry so it stays the same size after performing a zoom.

Parameters **transform** (*QTransform*) – The view's transformation matrix after the zoom.

wipe_out(*self*)

class spinetoolbox.widgets.graph_view_graphics_items.**OutlinedTextItem**(*text*, *parent*, *font=QFont()*, *brush=QBrush(Qt.white)*, *outline_pen=QPen(Qt.black, 3, Qt.SolidLine)*)

Bases: PySide2.QtWidgets.QGraphicsSimpleTextItem

Outlined text item.

Initializes item.

Parameters

- **text** (*str*) – text to show
- **font** (*QFont, optional*) – font to display the text
- **brush** (*QBrush, optional*) –
- **outline_pen** (*QPen, optional*) –

spinetoolbox.widgets.graph_view_mixin

Contains the GraphViewMixin class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

class spinetoolbox.widgets.graph_view_mixin.**GraphViewMixin**(**args*, ***kwargs*)

Provides the graph view for the DS form.

graph_created

_node_extent = 64

_arc_width

_arc_length_hint

add_menu_actions(*self*)

Adds toggle view actions to View menu.

connect_signals(*self*)

Connects signals.

setup_zoom_widget_action(*self*)

Setups zoom widget action in view menu.

init_models(*self*)

Initializes models.

receive_object_classes_added(*self*, *db_map_data*)

receive_object_classes_updated(*self*, *db_map_data*)

receive_object_classes_removed(*self*, *db_map_data*)

receive_relationship_classes_added(*self*, *db_map_data*)

receive_relationship_classes_updated(*self*, *db_map_data*)

receive_relationship_classes_removed(*self*, *db_map_data*)

receive_objects_added(*self*, *db_map_data*)

Runs when objects are added to the db. Builds a lookup dictionary consumed by `add_object`. Also, adds the new objects to the graph if needed.

Parameters **db_map_data** (*dict*) – list of dictionary-items keyed by DiffDatabaseMapping instance.

_ensure_objects_in_graph(*self*)

Makes sure all objects in `self._added_objects` are materialized in the graph if corresponds. It is assumed that `self._added_objects` doesn't contain information regarding objects added from the graph itself (through Item Palette etc.). These are materialized in `add_object()`.

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_added(*self*, *db_map_data*)

Runs when relationships are added to the db. Builds a lookup dictionary consumed by `add_relationship`. Also, adds the new relationships to the graph if needed.

Parameters **db_map_data** (*dict*) – list of dictionary-items keyed by DiffDatabaseMapping instance.

_ensure_relationships_in_graph(*self*)
Makes sure all relationships in `self._added_relationships` are materialized in the graph if corresponds. It is assumed that `self._added_relationships` doesn't contain information regarding relationships added from the graph itself (through Item Palette etc.). These are materialized in `add_relationship()`.

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.

_add_more_object_classes(*self*, *index*)
Runs when the user clicks on the Item palette Object class view. Opens the form to add more object classes if the index is the one that sayes 'New...'.

Parameters **index** (*QModelIndex*) – The clicked index.

_add_more_relationship_classes(*self*, *index*)
Runs when the user clicks on the Item palette Relationship class view. Opens the form to add more relationship classes if the index is the one that sayes 'New...'.

Parameters **index** (*QModelIndex*) – The clicked index.

_handle_zoom_minus_pressed(*self*)
Performs a zoom out on the view.

_handle_zoom_plus_pressed(*self*)
Performs a zoom in on the view.

_handle_zoom_reset_pressed(*self*)
Resets the zoom on the view.

_handle_menu_graph_about_to_show(*self*)
Enables or disables actions according to current selection in the graph.

_handle_item_palette_dock_location_changed(*self*, *area*)
Runs when the item palette dock widget location changes. Adjusts splitter orientation accordingly.

_handle_entity_graph_visibility_changed(*self*, *visible*)

_handle_item_palette_visibility_changed(*self*, *visible*)

_handle_object_tree_selection_changed(*self*, *selected*, *deselected*)
Builds graph.

build_graph(*self*, *timeit=False*)
Builds the graph.

_get_selected_object_ids(*self*)
Returns a set of ids corresponding to selected objects in the object tree.

Returns set

_get_graph_data(*self*)
Returns data for making graph according to selection in Object tree.

Returns integer object ids list: integer relationship ids list: arc source indices list: arc destination indices

Return type list

_get_new_items(*self*)
Returns new items for the graph.

Returns ObjectItem instances list: RelationshipItem instances list: ArcItem instances

Return type list

_get_wip_items(*self*)
Removes wip items from the current scene and returns them.

Returns ObjectItem instances list: RelationshipItem instances list: ArcItem instances

Return type list

static _add_new_items(*scene*, *object_items*, *relationship_items*, *arc_items*)

static _add_wip_items(*scene*, *new_object_items*, *wip_object_items*, *wip_relationship_items*, *wip_arc_items*)
Adds wip items to the given scene, merging wip object items with existing ones by entity id.

Parameters

- **scene** (*QGraphicsScene*) –
- **new_object_items** (*list*) –
- **wip_object_items** (*list*) –
- **wip_relationship_items** (*list*) –
- **wip_arc_items** (*list*) –

static shortest_path_matrix(*N*, *src_inds*, *dst_inds*, *spread*)
Returns the shortest-path matrix.

Parameters

- **N** (*int*) – The number of nodes in the graph.
- **src_inds** (*list*) – Source indices
- **dst_inds** (*list*) – Destination indices
- **spread** (*int*) – The desired 'distance' between neighbours

static sets(*N*)
Returns sets of vertex pairs indices.

Parameters **N** (*int*) –

static vertex_coordinates(*matrix*, *heavy_positions=None*, *iterations=10*, *weight_exp=-2*, *initial_diameter=1000*)
Returns x and y coordinates for each vertex in the graph, computed using VSGD-MS.

new_scene(*self*)
Replaces the current scene with a new one.

tear_down_scene(*self*)
Removes all references to this form in graphics items and schedules the scene for deletion.

extend_scene(*self*)
Extends the scene to show all items.

_handle_scene_selection_changed(*self*)
Filters parameters by selected objects in the graph.

_handle_scene_changed(*self*, *region*)
Enlarges the scene rect if needed.

_handle_item_dropped(*self*, *pos*, *text*)
Runs when an item is dropped from Item palette onto the view. Creates the object or relationship template.

Parameters

- **pos** (*QPoint*) –
- **text** (*str*) –

add_wip_relationship(*self*, *scene*, *pos*, *relationship_class_id*, *center_item=None*, *center_dimension=None*)
Makes items for a wip relationship and adds them to the scene at the given coordinates.

Parameters

- **scene** (*QGraphicsScene*) –
- **pos** (*QPointF*) –
- **relationship_class_id** (*int*) –
- **center_item_dimension** (*tuple, optional*) – A tuple of (ObjectItem, dimension) to put at the center of the wip item.

add_object(*self*, *object_class_id*, *name*)
Adds object to the database.

Parameters

- **object_class_id** (*int*) –
- **name** (*str*) –

Returns The id of the added object if successful, None otherwise.

Return type int, NoneType

update_object(*self*, *object_id*, *name*)
Updates object in the db.

Parameters

- **object_id** (*int*) –
- **name** (*str*) –

add_relationship(*self*, *class_id*, *object_id_list*, *object_name_list*)
Adds relationship to the db.

Parameters

- **class_id** (*int*) –
- **object_id_list** (*list*) –

show_graph_view_context_menu(*self*, *global_pos*)
Shows context menu for graphics view.

Parameters **global_pos** (*QPoint*) –

hide_selected_items(*self*, *checked=False*)
Hides selected items.

show_hidden_items(*self*, *checked=False*)
Shows hidden items.

prune_selected_items(*self*, *checked=False*)
Prunes selected items.

restore_pruned_items(*self*, *checked=False*)
Reinstates pruned items.

show_demo(*self*, *checked=False*)

_enable_live_graph_demo_action(*self*, *obj=None*)

show_object_item_context_menu(*self*, *global_pos*, *main_item*)
Shows context menu for entity item.

Parameters

- **global_pos** (*QPoint*) –
- **main_item** (spinetoolbox.widgets.graph_view_graphics_items.ObjectItem) –

show_relationship_item_context_menu(*self*, *global_pos*)
Shows context menu for entity item.

Parameters **global_pos** (*QPoint*) –

_apply_entity_context_menu_option(*self*, *option*)

remove_graph_items(*self*, *checked=False*)
Removes all selected items in the graph.

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

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

spinetoolbox.widgets.import_errors_widget

Contains ImportErrorWidget class.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

class spinetoolbox.widgets.import_errors_widget.**ImportErrorWidget**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

Widget to display errors while importing and ask user for action.

set_import_state(*self*, *num_imported*, *errors*)
Sets state of error widget.

Parameters

- **{int} -- number of successfully imported items**(*num_imported*) –
- **{list} -- list of errors.**(*errors*) –

spinetoolbox.widgets.import_preview_widget

Contains ImportPreviewWidget, and MappingTableMenu classes.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

class spinetoolbox.widgets.import_preview_widget.**ImportPreviewWidget**(*connector*, *parent*)
Bases: PySide2.QtWidgets.QWidget

A Widget for defining one or more Mappings associated to a data Source (CSV file, Excel file, etc). Currently it's being embedded in ImportDialog and ImportPreviewWindow.

Parameters **connector**(ConnectionManager) –

tableChecked

mappedDataReady

previewDataUpdated

checked_tables

set_loading_status(*self*, *status*)
Sets widgets enable state

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

select_table(*self*, *selection*)
Set selected table and request data from connector

check_list_item(*self*, *item*)
Set the check state of item

handle_connector_error(*self*, *error_message*)

request_mapped_data(*self*)

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

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

use_settings (*self*, *settings*)

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

Returns [Dict] – dict with settings

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*)
Context menu for connection links.

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

copy_mappings (*self*, *table*)

copy_options (*self*, *table*)

paste_mappings (*self*, *table*)

paste_options (*self*, *table*)

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

A menu to let users define a Mapping from a data table. Used to generate the context menu for ImportPreviewWidget._ui_table

set_model (*self*, *model*)

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

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

class spinetoolbox.widgets.import_preview_widget.**TableMenu** (*parent*, *position*, *can_paste_option*, *can_paste_mapping*)
Bases: *spinetoolbox.widgets.custom_menus.CustomContextMenu*

Menu for tables in data source

spinetoolbox.widgets.import_preview_widget.**_sanitize_data** (*data*, *header*)
Fills empty data cells with None.

spinetoolbox.widgets.import_preview_window

Contains ImportPreviewWindow class.

authors

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

date 10.6.2019

Module Contents

class spinetoolbox.widgets.import_preview_window.**ImportPreviewWindow**(*importer*, *filepath*, *connector*, *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
- **settings** (*dict*) – Default mapping specification
- **toolbox** (*QMainWindow*) – ToolboxUI class

settings_updated

connection_failed

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*)
Handle close window.

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

spinetoolbox.widgets.import_widget

ImportDialog class.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

class spinetoolbox.widgets.import_widget.**ImportDialog**(*settings*, *parent*)
Bases: PySide2.QtWidgets.QDialog

A widget for importing data into a Spine db. Currently used by DataStoreForm. It embeds three widgets that alternate depending on user's actions: - *select_widget* is a *QWidget* for selecting the source data type (CSV, Excel, etc.) - *_import_preview* is an *ImportPreviewWidget* for defining Mappings to associate with the source data - *_error_widget* is an *ImportErrorWidget* to show errors from import operations

Parameters

- **settings** (*QSettings*) – settings for storing/restoring window state
- **parent** (*QWidget*) – parent widget

_SETTINGS_GROUP_NAME = importDialog

mapped_data

mapping_errors

connector_selected(*self*, *selection*)

set_ok_button_availability(*self*)

import_data(*self*, *data*, *errors*)

data_ready(*self*, *data*, *errors*)

ok_clicked(*self*)

cancel_clicked(*self*)

back_clicked(*self*)

launch_import_preview(*self*)

_handle_failed_connection(*self*, *msg*)
Handle failed connection, show error message and select widget

Parameters **{str} -- str with message of reason for failed connection.** (*msg*) –

set_preview_as_main_widget(*self*)

set_error_widget_as_main_widget(*self*)

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

closeEvent(*self*, *event*)
Stores window's settings and accepts the event.

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

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.julia_repl_widget

Class for a custom SpineConsoleWidget to use as julia REPL.

author

M. Marin (KTH)

date 22.5.2018

Module Contents

class spinetoolbox.widgets.julia_repl_widget.**CustomQtKernelManager**

Bases: qtconsole.manager.QtKernelManager

A QtKernelManager with a custom restarter, and a means to override the –project argument.

kernel_left_dead

project_path

kernel_spec

override_project_arg(*self*)

start_restarter(*self*)

Starts a restarter with custom time to dead and restart limit.

_handle_kernel_left_dead(*self*)

class spinetoolbox.widgets.julia_repl_widget.**JuliaREPLWidget**(*toolbox*)

Bases: *spinetoolbox.widgets.spine_console_widget.SpineConsoleWidget*

Class for a custom SpineConsoleWidget.

Parameters **toolbox** (ToolboxUI) – QMainWindow instance

name = Julia Console

julia_kernel_name(*self*)
Returns the name of the julia kernel specification according to the julia executable selected in settings. Returns None if julia version cannot be determined.

Returns str, NoneType

start_jupyter_kernel(*self*)
Starts a Julia Jupyter kernel if available.

_do_start_jupyter_kernel(*self*, *kernel_name=None*)
Starts a Jupyter kernel with the specified name.

Parameters **kernel_name** (*str, optional*) –

handle_repl_failed_to_start(*self*)
Tries using IJulia.

Returns True, False, or None if unable to determine.

Return type (bool, NoneType)

_try_installing_ijulia(*self*)
Prompts user to install IJulia.

_do_try_installing_ijulia(*self*)

_try_rebuilding_ijulia(*self*)

restart_jupyter_kernel(*self*)
Restarts the julia jupyter kernel if it's already started. Starts a new kernel if none started or if the julia version has changed in Settings.

setup_client(*self*)

_handle_kernel_restarted(*self*, *died=True*)
Called when the kernel is restarted, i.e., when time to dead has elapsed.

_handle_kernel_left_dead(*self*)
Called when the kernel is finally declared dead, i.e., the restart limit has been reached.

handle_ijulia_installation_finished(*self*, *ret*)
Runs when IJulia installation process finishes

handle_ijulia_rebuild_finished(*self*, *ret*)
Runs when IJulia rebuild process finishes

handle_ijulia_process_finished(*self*, *ret*, *process*)
Checks whether or not the IJulia process finished successfully.

_handle_execute_reply(*self*, *msg*)

_handle_status(*self*, *msg*)
Handles status message.

_handle_error(*self*, *msg*)
Handle error messages.

wake_up(*self*)
See base class.

shutdown_jupyter_kernel(*self*)
Shut down the jupyter kernel.

_context_menu_make(*self*, *pos*)
Reimplemented to add an action for (re)start REPL action.

enterEvent(*self*, *event*)
Set busy cursor during REPL (re)starts.

dragEnterEvent(*self*, *e*)
Don't accept drops from Add Item Toolbar.

copy_input(*self*)
Copy only input.

_is_complete(*self*, *source*, *interactive*)
See base class.

spinetoolbox.widgets.manage_db_items_dialog

Classes for custom QDialogs to add edit and remove database items.

author

M. Marin (KTH)

date 13.5.2018

Module Contents

class spinetoolbox.widgets.manage_db_items_dialog.**ManageItemsDialog**(*parent*, *db_mngr*)

Bases: PySide2.QtWidgets.QDialog

A dialog with a CopyPasteTableView and a QDialogButtonBox. Base class for all dialogs to query user's preferences for adding/editing/managing data items.

parent
data store widget

Type *DataStoreForm*

db_mngr

Type *SpineDBManager*

connect_signals(*self*)
Connect signals to slots.

resize_window_to_columns(*self*, *height=None*)

_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.widgets.manage_db_items_dialog.**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.widgets.manage_db_items_dialog.**GetObjectsMixin**
Provides a method to retrieve objects for AddRelationshipsDialog and EditRelationshipsDialog.

make_db_map_obj_lookup(*self*)

make_db_map_rel_cls_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.widgets.manage_db_items_dialog.**ShowIconColorEditorMixin**
Provides methods to show an *IconColorEditor* upon request.

show_icon_color_editor(*self*, *index*)

create_object_pixmap(*self*, *object_class_name*)

class spinetoolbox.widgets.manage_db_items_dialog.**CommitDialog**(*parent*, **db_names*)
Bases: PySide2.QtWidgets.QDialog

A dialog to query user's preferences for new commit.

db_names
database names

Type Iterable

Initialize class

receive_text_changed(*self*)
Called when text changes in the commit msg text edit. Enable/disable commit button accordingly.

spinetoolbox.widgets.map_editor

An editor widget for editing a map type parameter values.

author

A. Soininen (VTT)

date 11.2.2020

Module Contents

class spinetoolbox.widgets.map_editor.**MapEditor**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

A widget for editing maps.

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.mapping_widget

MappingWidget and MappingOptionsWidget class.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

spinetoolbox.widgets.mapping_widget.**MAPPING_CHOICES = ['Constant', 'Column', 'Row', 'Colum**

class spinetoolbox.widgets.mapping_widget.**MappingWidget**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

A widget for managing Mappings (add, remove, edit, visualize, and so on). Intended to be embedded in a ImportPreviewWidget.

mappingChanged

mappingDataChanged

set_data_source_column_num(*self*, *num*)

set_model(*self*, *model*)
Sets new model

data_changed(*self*)

new_mapping(*self*)
Adds new empty mapping

delete_selected_mapping(*self*)
deletes selected mapping

select_mapping(*self*, *selection*)
gets selected mapping and emits mappingChanged

class spinetoolbox.widgets.mapping_widget.**MappingOptionsWidget**(*parent=None*)
Bases: PySide2.QtWidgets.QWidget

A widget for managing Mapping options (class type, dimensions, parameter type, ignore columns, and so on). Intended to be embedded in a MappingWidget.

set_num_available_columns(*self*, *num*)

change_skip_columns(*self*, *skip_cols*)

set_model(*self*, *model*)

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

change_class(*self*, *new_class*)

change_dimension(*self*, *dim*)

change_parameter(*self*, *par*)

change_import_objects(*self*, *state*)

change_read_start_row(*self*, *row*)

_update_time_series_options(*self*)

spinetoolbox.widgets.notification

Contains a notification widget.

author

P. Savolainen (VTT)

date 12.12.2019

Module Contents

class spinetoolbox.widgets.notification.**Notification**(*parent*, *txt*, *anim_duration=500*, *life_span=2000*)

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*)

remaining_time(*self*)

class spinetoolbox.widgets.notification.**NotificationStack**(*parent*, *anim_duration=500*, *life_span=2000*)

Bases: PySide2.QtCore.QObject

push(*self*, *txt*)
Pushes a notification to the stack with the given text.

handle_notification_destroyed(*self*, *notification*, *height*)
Removes from the stack the given notification and move up subsequent ones.

spinetoolbox.widgets.object_name_list_editor

Contains the ObjectNameListEditor class.

author

M. Marin (KTH)

date 27.11.2019

Module Contents

class spinetoolbox.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.widgets.object_name_list_editor.**ObjectNameListEditor**(*parent*, *index*, *object_class_names*, *object_names_lists*, *current_object_names*)
Bases: *spinetoolbox.widgets.manage_db_items_dialog.ManageItemsDialog*

A dialog to select the object name list for a relationship using Google-like search bars.

Initializes widget.

Parameters

- **parent** (DataStoreForm) –
- **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.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

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.options_widget

Contains OptionsWidget class.

author

P. Vennström (VTT)

date 1.6.2019

Module Contents

class spinetoolbox.widgets.options_widget.**OptionsWidget**(*options*, *header='Options'*, *parent=None*)
Bases: PySide2.QtWidgets.QWidget

A widget for handling simple options. Used by ConnectionManager.

Creates OptionWidget

Parameters **options** (*Dict*) – Dict describing what options to build a widget around.

Keyword Arguments

- **header** (*str*) – Title of groupbox (default: {"Options"})
- **parent** (*QWidget, None*) – parent of widget

optionsChanged

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

set_options(*self*, *options=None*, *set_missing_default=True*)
Sets state of options

Keyword Arguments

- **{Dict} -- Dict with option name as key and value as value (default** (*options*) – {None})
- **{bool} -- Sets missing options to default if True (default** (*set_missing_default*) – {True})

get_options(*self*)
Returns current state of option widget

Returns [Dict] – Dict with option name as key and value as value

spinetoolbox.widgets.parameter_value_editor

An editor dialog for editing database (relationship) parameter values.

author

A. Soininen (VTT)

date 28.6.2019

Module Contents

class spinetoolbox.widgets.parameter_value_editor.**_Editor**
Bases: enum.IntEnum

Indexes for the specialized editors corresponding to the selector combo box and editor stack.

PLAIN_VALUE = 0

MAP = 1

TIME_SERIES_FIXED_RESOLUTION = 2

TIME_SERIES_VARIABLE_RESOLUTION = 3

TIME_PATTERN = 4

DATETIME = 5

DURATION = 6

class spinetoolbox.widgets.parameter_value_editor.**ParameterValueEditor**(*parent_index*, *value_name=''*, *value=None*, *parent_widget=None*)
Bases: PySide2.QtWidgets.QDialog

Dialog for editing (relationship) parameter values.

The dialog takes the editable value from a parent model 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 parent model.

Parameters

- **parent_index** (*QModelIndex*) – an index to a parameter value in parent_model
- **value_name** (*str*) – name of the value
- **value** – parameter value or None if it should be loaded from parent_index
- **parent_widget** (*QWidget*) – a parent widget

accept(*self*)
Saves the parameter value shown in the currently selected editor widget back to the parent model.

_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.

_select_default_view(*self*, *message=None*)
Opens the default editor widget. Optionally, displays a warning dialog indicating the problem.

Parameters **message** (*str, optional*) –

spinetoolbox.widgets.parameter_view_mixin

Contains the ParameterViewMixin class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

class spinetoolbox.widgets.parameter_view_mixin.**ParameterViewMixin**(**args*, ***kwargs*)
Provides stacked parameter tables for the data store form.

add_menu_actions(*self*)
Adds toggle view actions to View menu.

connect_signals(*self*)
Connects signals to slots.

init_models(*self*)
Initializes models.

_setup_delegate(*self*, *table_view*, *column*, *delegate_class*)
Returns a custom delegate for a given view.

setup_delegates(*self*)
Sets delegates for tables.

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*) –

show_parameter_value_editor(*self*, *index*, *value_name=''*, *value=None*)
Shows the parameter value editor for the given index of given table view.

_handle_object_parameter_tab_changed(*self*, *index*)
Updates filter.

_handle_relationship_parameter_tab_changed(*self*, *index*)
Updates filter.

_handle_object_parameter_value_visibility_changed(*self*, *visible*)

_handle_object_parameter_definition_visibility_changed(*self*, *visible*)

_handle_relationship_parameter_value_visibility_changed(*self*, *visible*)

_handle_relationship_parameter_definition_visibility_changed(*self*, *visible*)

_handle_object_parameter_definition_selection_changed(*self*, *selected*, *deselected*)
Enables/disables the option to remove rows.

_handle_object_parameter_value_selection_changed(*self*, *selected*, *deselected*)
Enables/disables the option to remove rows.

_handle_relationship_parameter_definition_selection_changed(*self*, *selected*, *deselected*)
Enables/disables the option to remove rows.

_handle_relationship_parameter_value_selection_changed(*self*, *selected*, *deselected*)
Enables/disables the option to remove rows.

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

static set_and_apply_default_rows(*model*, *default_data*)

update_filter(*self*)
Updates filters.

show_object_parameter_value_context_menu(*self*, *pos*)
Shows the context menu for object parameter value table view.

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

show_relationship_parameter_value_context_menu(*self*, *pos*)
Shows the context menu for relationship parameter value table view.

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

show_object_parameter_definition_context_menu(*self*, *pos*)
Shows the context menu for object parameter table view.

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

show_relationship_parameter_definition_context_menu(*self*, *pos*)
Shows the context menu for relationship parameter table view.

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

_show_parameter_context_menu(*self*, *position*, *table_view*, *value_column_header*)
Shows the context menu for the given parameter table.

Parameters

- **position** (*QPoint*) – local mouse position in the table view
- **table_view** (*QTableView*) – the table view where the context menu was triggered
- **value_column_header** (*str*) – column header for editable/plottable values

remove_object_parameter_values(*self*)
Removes selected rows from object parameter value table.

remove_relationship_parameter_values(*self*)
Removes selected rows from relationship parameter value table.

remove_object_parameter_definitions(*self*)
Removes selected rows from object parameter definition table.

remove_relationship_parameter_definitions(*self*)
Removes selected rows from relationship parameter definition table.

_remove_parameter_data(*self*, *table_view*, *item_type*)
Removes selected rows from parameter table.

Parameters

- **table_view** (*QTableView*) – remove selection from this view
- **item_type** (*str*) –

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

save_window_state(*self*)
Saves window state parameters (size, position, state) via QSettings.

receive_parameter_definitions_added(*self*, *db_map_data*)

receive_parameter_values_added(*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_definitions_removed(*self*, *db_map_data*)

receive_parameter_values_removed(*self*, *db_map_data*)

spinetoolbox.widgets.pivot_table_header_view

Contains custom QHeaderView for the pivot table.

author

M. Marin (KTH)

date 2.12.2019

Module Contents

class spinetoolbox.widgets.pivot_table_header_view.**PivotTableHeaderView**(*orientation*, *area*, *parent=None*)

Bases: PySide2.QtWidgets.QHeaderView

header_dropped

area

dragEnterEvent(*self*, *event*)

dragMoveEvent(*self*, *event*)

dropEvent(*self*, *event*)

spinetoolbox.widgets.pivot_table_view

Custom QTableView classes that support copy-paste and the like.

author

M. Marin (KTH)

date 18.5.2018

Module Contents

class spinetoolbox.widgets.pivot_table_view.**PivotTableView**(*parent=None*)

Bases: PySide2.QtWidgets.QTableView

Custom QTableView class with pivot capabilities.

parent
The parent of this view

Type QWidget

Initialize the class.

clipboard_data_changed(*self*)

keyPressEvent(*self*, *event*)
Copy and paste to and from clipboard in Excel-like format.

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

class spinetoolbox.widgets.plain_parameter_value_editor.**_ValueModel**(*value*)
A model to handle the parameter value in the editor. Mostly useful because of the handy conversion of strings to floats or booleans.

Parameters **value** (*float, bool*) – a parameter value

value
Returns the value held by the model.

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_value(*self*, *value*)
Sets the value to be edited in this widget.

_value_changed(*self*)
Updates the model.

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

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

class spinetoolbox.widgets.plot_widget.**PlotWidget**
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.project_form_widget

Widget shown to user when a new project is created.

authors

P. Savolainen (VTT)

date 10.1.2018

Module Contents

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.python_repl_widget

Class for a custom SpineConsoleWidget to use as Python REPL.

author

P. Savolainen (VTT)

date 14.3.2019

Module Contents

class spinetoolbox.widgets.python_repl_widget.**PythonReplWidget**(*toolbox*)
Bases: *spinetoolbox.widgets.spine_console_widget.SpineConsoleWidget*

Python Repl Widget class.

toolbox
App main window (QMainWindow) instance

Type *ToolboxUI*

Class constructor.

name = Python Console

connect_signals(*self*)
Connect signals.

disconnect_signals(*self*)
Disconnect signals. Needed before switching to another Python kernel.

python_kernel_name(*self*)
Returns the name of the Python kernel specification and its display name according to the selected Python environment in Settings. Returns None if Python version cannot be determined.

setup_python_kernel(*self*)
Context menu Start action handler.

launch_kernel(*self*, *k_name*, *k_display_name*)
Check if selected kernel exists or if it needs to be set up before launching.

check_and_install_requirements(*self*)
Prompts user to install IPython and ipykernel if they are missing. After installing the required packages, installs kernelspecs for the selected Python if they are missing.

Returns Boolean value depending on whether or not the user chooses to proceed.

is_package_installed(*self*, *package_name*)
Checks if given package is installed to selected Python environment.

Parameters **package_name** (*str*) – Package name

Returns True if installed, False if not

Return type (bool)

start_package_install_process(*self*, *package_name*)
Starts installing the given package using pip.

Parameters **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

start_kernelspec_install_process(*self*)
Install kernel specifications for the selected Python environment.

handle_kernelspec_install_process_finished(*self*, *retval*)
Handles installing package finished.

Parameters **retval** (*int*) – Process return value. 0: success, !0: failure

start_python_kernel(*self*)
Starts kernel manager and client and attaches the client to the Python Console.

wake_up(*self*)
See base class.

handle_executing(*self*, *code*)
Slot for handling the 'executing' signal. Signal is emitted when a user visible 'execute_request' has been submitted to the kernel from the FrontendWidget.

Parameters **code** (*str*) – Code to be executed (actually not 'str' but 'object')

handle_executed(*self*, *msg*)
Slot for handling the 'executed' signal. Signal is emitted when a user-visible 'execute_reply' has been received from the kernel and processed by the FrontendWidget.

Parameters **msg** (*dict*) – Response message (actually not 'dict' but 'object')

receive_iopub_msg(*self*, *msg*)
Message received from the IOPUB channel. Note: We are only monitoring when the kernel has started successfully and ready for action here. Alternatively, this could be done in the Slot for the 'executed' signal. However, this Slot could come in handy at some point. See 'Messaging in Jupyter' for details: https://jupyter-client.readthedocs.io/en/latest/messaging.html

Parameters **msg** (*dict*) – Received message from IOPUB channel

shutdown_kernel(*self*, *hush=False*)
Shut down Python kernel.

push_vars (*self*, *var_name*, *var_value*)
Push a variable to Python Console session. Simply executes command 'var_name=var_value'.

Parameters

- **var_name** (*str*) – Variable name
- **var_value** (*object*) – Variable value

Returns True if succeeded, False otherwise

Return type (bool)

test_push_vars (*self*)
QAction slot to test pushing variables to Python Console.

_context_menu_make (*self*, *pos*)
Reimplemented to add custom actions.

dragEnterEvent (*self*, *e*)
Don't accept project item drops.

_is_complete (*self*, *source*, *interactive*)
See base class.

spinetoolbox.widgets.report_plotting_failure

Functions to report failures in plotting to the user.

author

A. Soininen (VTT)

date 10.7.2019

Module Contents

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

class spinetoolbox.widgets.settings_widget.**SettingsWidget**(*toolbox*)
Bases: PySide2.QtWidgets.QWidget

A widget to change user's preferred settings.

toolbox
Parent widget.

Type *ToolboxUI*

Initialize class.

connect_signals (*self*)
Connect signals.

browse_gams_path (*self*, *checked=False*)
Open file browser where user can select a GAMS program.

browse_julia_path (*self*, *checked=False*)
Open file browser where user can select a Julia executable (i.e. julia.exe on Windows).

browse_julia_project_path (*self*, *checked=False*)
Open file browser where user can select a Julia project path.

browse_python_path (*self*, *checked=False*)
Open file browser where user can select a python interpreter (i.e. python.exe on Windows).

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*)

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.

handle_ok_clicked (*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_python_env_changed (*self*, *new_path*)
Checks if Python environment was changed. This indicates that the Python Console may need a restart.

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

file_is_valid (*self*, *file_path*, *msgbox_title*)
Checks that given path is not a directory and it's a file that actually exists. Needed because the QLineEdits are editable.

dir_is_valid(*self*, *dir_path*, *msgbox_title*)
Checks that given path is a directory. Needed because the QLineEdits are editable.

keyPressEvent(*self*, *e*)
Close settings 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.

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.shrinking_scene

A QGraphicsScene that can shrink sometimes.

author

A. Soininen (VTT)

date 18.10.2019

Module Contents

class spinetoolbox.widgets.shrinking_scene.**ShrinkingScene**(*horizontal_shrinking_threshold*, *vertical_shrinking_threshold*, *parent*)

Bases: PySide2.QtWidgets.QGraphicsScene

A QGraphicsScene class that can shrinks its scene rectangle.

Shrinking can be triggered by shrink_if_needed(). It is controlled by two threshold values which control how far the items need to be from the scene rectangle's edges to trigger the shrinking.

Parameters

- **horizontal_shrinking_threshold** (*float*) – horizontal threshold before the scene is shrank
- **vertical_shrinking_threshold** (*float*) – vertical threshold before the scene is shrank
- **parent** (*QObject*) – a parent

item_move_finished
Emitted when an item has finished moving.

shrink_if_needed(*self*)
Shrinks the scene rectangle if it is considerably larger than the area occupied by items.

spinetoolbox.widgets.spine_console_widget

Class for a custom RichJupyterWidget that can run tool instances.

authors

M. Marin (KTH)

date 22.10.2019

Module Contents

class spinetoolbox.widgets.spine_console_widget.**SpineConsoleWidget**(*toolbox*)
Bases: qtconsole.rich_jupyter_widget.RichJupyterWidget

Base class for all console widgets that can run tool instances.

Parameters **toolbox** (ToolboxUI) – QMainWindow instance

ready_to_execute

execution_failed

name = Unnamed console

wake_up(*self*)
Wakes up the console in preparation for execution.

Subclasses need to emit either ready_to_execute or execution_failed as a consequence of calling this function.

interrupt(*self*)
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

class spinetoolbox.widgets.spine_datapackage_widget.**SpineDatapackageWidget**(*data_connection*)
Bases: PySide2.QtWidgets.QMainWindow

A widget to allow user to edit a datapackage and convert it to a Spine database in SQLite.

data_connection
Data Connection associated to this widget

Type *DataConnection*

Initialize class.

msg

msg_proc

msg_error

add_toggle_view_actions(*self*)
Add toggle view actions to View menu.

show(*self*)
Called when the form shows. Init datapackage (either from existing datapackage.json or by inferring a new one from sources) and update ui.

infer_datapackage(*self*, *checked=False*)
Called when the user triggers the infer action. Infer datapackage from sources and update ui.

load_datapackage(*self*)
Load datapackage from 'datapackage.json' file in data directory, or infer one from CSV files in that directory.

infer_datapackage_(*self*)
Infer datapackage from CSV files in data directory.

update_ui(*self*)
Update ui from datapackage attribute.

connect_signals(*self*)
Connect signals to slots.

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

_handle_menu_about_to_show(*self*)
Called when a menu from the menubar is about to show. Adjust infer action depending on whether or not we have a datapackage. Adjust copy paste actions depending on which widget has the focus. TODO Enable/disable action to save datapackage depending on status.

add_message(*self*, *msg*)
Prepend regular message to status bar.

Parameters **msg** (*str*) – String to show in QStatusBar

add_process_message(*self*, *msg*)
Show process message in status bar. This messages stays until replaced.

Parameters **msg** (*str*) – String to show in QStatusBar

add_error_message(*self*, *msg*)
Show error message.

Parameters **msg** (*str*) – String to show

save_datapackage(*self*, *checked=False*)
Write datapackage to file 'datapackage.json' in data directory.

show_export_to_spine_dialog(*self*, *checked=False*)
Show dialog to allow user to select a file to export.

export_to_spine(*self*, *file_path*)
Export datapackage into Spine SQlite file.

_handle_converter_progressed(*self*, *step*, *msg*)

_handle_converter_failed(*self*, *msg*)

_handle_converter_finished(*self*)

copy(*self*, *checked=False*)
Copy data to clipboard.

paste(*self*, *checked=False*)
Paste data from clipboard.

load_resource_data(*self*)
Load resource data into a local list of tables.

reset_resource_models(*self*, *current*, *previous*)
Reset resource data and schema models whenever a new resource is selected.

reset_resource_data_model(*self*)
Reset resource data model with data from newly selected resource.

update_resource_data(*self*, *index*, *new_value*)
Update resource data with newly edited data.

_handle_resource_name_data_committed(*self*, *index*, *new_name*)
Called when line edit delegate wants to edit resource name data. Update resources model and descriptor with new resource name.

_handle_field_name_data_committed(*self*, *index*, *new_name*)
Called when line edit delegate wants to edit field name data. Update name in fields_model, resource_data_model's header and datapackage descriptor.

_handle_primary_key_data_committed(*self*, *index*)
Called when checkbox delegate wants to edit primary key data. Add or remove primary key field accordingly.

_handle_foreign_keys_data_committed(*self*, *index*, *value*)

_handle_foreign_keys_data_changed(*self*, *top_left*, *bottom_right*, *roles=None*)
Called when foreign keys data is updated in model. Update descriptor accordingly.

_handle_foreign_keys_model_rows_inserted(*self*, *parent*, *first*, *last*)

create_remove_foreign_keys_row_button(*self*, *index*)
Create button to remove foreign keys row.

remove_foreign_key_row(*self*, *button*)

closeEvent(*self*, *event=None*)
Handle close event.

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

class spinetoolbox.widgets.spine_datapackage_widget.**CustomPackage**(*descriptor=None*, *base_path=None*, *strict=False*, *storage=None*)

Bases: datapackage.Package

Custom datapackage class.

rename_resource(*self*, *old*, *new*)

rename_field(*self*, *resource*, *old*, *new*)
Rename a field.

set_primary_key(*self*, *resource*, **primary_key*)
Set primary key for a given resource in the package

append_to_primary_key(*self*, *resource*, *field*)
Append field to resources's primary key.

remove_from_primary_key(*self*, *resource*, *field*)
Remove field from resources's primary key.

insert_foreign_key(*self*, *row*, *resource_name*, *field_names*, *reference_resource_name*, *reference_field_names*)
Insert foreign key to a given resource in the package at a given row.

remove_primary_key(*self*, *resource*, **primary_key*)
Remove the primary key for a given resource in the package

remove_foreign_key(*self*, *resource*, *fields*, *reference_resource*, *reference_fields*)
Remove foreign key from the package

remove_foreign_keys_row(*self*, *row*, *resource*)
Remove foreign keys row from the package

spinetoolbox.widgets.state_machine_widget

Contains the StateMachineWidget class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

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.tabular_view_header_widget

Contains TabularViewHeaderWidget class.

authors

P. Vennström (VTT), M. Marin (KTH)

date 2.12.2019

Module Contents

class spinetoolbox.widgets.tabular_view_header_widget.**TabularViewHeaderWidget**(*identifier*, *name*, *area*, *menu=None*, *parent=None*)

Bases: PySide2.QtWidgets.QFrame

A draggable QWidget.

Parameters

- **identifier** (*int*) –
- **name** (*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.widgets.tabular_view_mixin

Contains TabularViewMixin class.

author

P. Vennström (VTT)

date 1.11.2018

Module Contents

class spinetoolbox.widgets.tabular_view_mixin.**TabularViewMixin**(**args*, ***kwargs*)

Provides the pivot table and its frozen table for the DS form.

_PARAMETER_VALUE = Parameter value

_RELATIONSHIP = Relationship

_PARAMETER = parameter

_PARAM_INDEX_ID

setup_delegates(*self*)
Sets delegates for tables.

add_menu_actions(*self*)
Adds toggle view actions to View menu.

connect_signals(*self*)
Connects signals to slots.

init_models(*self*)
Initializes models.

_handle_pivot_table_selection_changed(*self*, *selected*, *deselected*)
Accepts selection.

is_value_input_type(*self*)

_set_model_data(*self*, *index*, *value*)

current_object_class_id_list(*self*)

current_object_class_name_list(*self*)

static _is_class_index(*index*, *class_type*)
Returns whether or not the given tree index is a class index.

Parameters

- **index** (*QModelIndex*) – index from object or relationship tree
- **class_type** (*str*) –

Returns bool

_handle_pivot_table_visibility_changed(*self*, *visible*)

_handle_frozen_table_visibility_changed(*self*, *visible*)

_handle_entity_tree_selection_changed(*self*, *selected*, *deselected*)

_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 set 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 set(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*)
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*) –

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

get_pivot_preferences(*self*, *selection_key*)
Returns saved or default pivot preferences.

Parameters **selection_key** (*tuple(int,str,str)*) – Tuple of class id, class type, and input type.

Returns list: indexes in rows list: indexes in columns list: frozen indexes tuple: selection in frozen table

reload_pivot_table(*self*, *text=''*)
Updates current class (type and id) and reloads pivot table for it.

do_reload_pivot_table(*self*)
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.

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** (*int*) –
- **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*)

static _group_by_class (*items*, *get_class_id*)

receive_data_added_or_removed (*self*, *data*, *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_classes_removed (*self*, *db_map_data*)

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_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_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.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

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

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

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.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

class spinetoolbox.widgets.tool_specification_widget.**ToolSpecificationWidget**(*toolbox*, *tool_specification=*
Bases: PySide2.QtWidgets.QWidget

A widget to query user's preferences for a new tool specification.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **tool_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.

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 offer 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.

spinetoolbox.widgets.toolbars

Functions to make and handle QToolBars.

author

P. Savolainen (VTT)

date 19.1.2018

Module Contents

class spinetoolbox.widgets.toolbars.**ItemToolBar**(*parent*)
Bases: PySide2.QtWidgets.QToolBar

A toolbar to add items using drag and drop actions.

Parameters **parent** (ToolboxUI) – QMainWindow instance

add_draggable_widgets(*self*, *category_icon*)
Adds draggable widgets from the given list.

Parameters **category_icon** (*list*) – List of tuples (item_type (str), item category (str), icon path (str))

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.

class spinetoolbox.widgets.toolbars.**DraggableWidget**(*parent*, *pixmap*, *item_type*, *category*)

Bases: PySide2.QtWidgets.QLabel

A draggable QLabel.

Parameters

- **parent** (*QWidget*) – Parent widget
- **pixmap** (*QPixMap*) – Picture for the label
- **item_type** (*str*) – Item type (e.g. Data Store, Data Connection, etc...)
- **category** (*str*) – Item category (e.g. Data Stores, Data Connetions, etc...)

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.toolbars.**ParameterTagToolBar**(*parent*, *db_mngr*, **db_maps*)

Bases: PySide2.QtWidgets.QToolBar

A toolbar to add items using drag and drop actions.

Parameters

- **parent** (DataStoreForm) – tree or graph view form
- **db_mngr** (SpineDBManager) – the DB manager for interacting with the db
- **db_maps** (*iter*) – DiffDatabaseMapping instances

tag_button_toggled

manage_tags_action_triggered

init_toolbar(*self*)

receive_parameter_tags_added(*self*, *db_map_data*)

_add_db_map_tag_actions(*self*, *db_map*, *parameter_tags*)

receive_parameter_tags_removed(*self*, *db_map_data*)

_remove_db_map_tag_actions(*self*, *db_map*, *parameter_tag_ids*)

receive_parameter_tags_updated(*self*, *db_map_data*)

_update_db_map_tag_actions(*self*, *db_map*, *parameter_tags*)

spinetoolbox.widgets.tree_view_mixin

Contains the TreeViewMixin class.

author

M. Marin (KTH)

date 26.11.2018

Module Contents

class spinetoolbox.widgets.tree_view_mixin.**TreeViewMixin**(**args*, ***kwargs*)

Provides object and relationship trees for the data store form.

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 _db_map_items(*indexes*)

Groups items from given tree indexes by db map.

Returns lists of dictionary items keyed by DiffDatabaseMapping

Return type dict

static _db_map_class_id_data(*db_map_data*)

Returns a new dictionary where the class id is also part of the key.

Returns lists of dictionary items keyed by tuple (DiffDatabaseMapping, integer class id)

Return type dict

static _extend_merge(*left*, *right*)

Returns a new dictionary by uniting left and right.

Returns lists of dictionary items keyed by DiffDatabaseMapping

Return type dict

_update_object_filter(*self*)

Updates filters object filter according to object tree selection.

_update_relationship_filter(*self*)

Update filters relationship filter according to relationship tree selection.

edit_object_tree_items(*self*, *current*)
Starts editing the given index in the object tree.

edit_relationship_tree_items(*self*, *current*)
Starts editing the given index in the relationship tree.

show_object_tree_context_menu(*self*, *pos*)
Shows the context menu for object tree.

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

show_relationship_tree_context_menu(*self*, *pos*)
Shows the context for relationship tree.

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

fully_expand_selection(*self*)

fully_collapse_selection(*self*)

find_next_relationship(*self*, *index*)
Expands next occurrence of a relationship in object tree.

call_show_add_objects_form(*self*, *index*)

call_show_add_relationship_classes_form(*self*, *index*)

call_show_add_relationships_form(*self*, *index*)

show_add_object_classes_form(*self*, *checked=False*)
Shows dialog to let user select preferences for new object classes.

show_add_objects_form(*self*, *checked=False*, *class_name=''*)
Shows dialog to let user select preferences for new objects.

show_add_relationship_classes_form(*self*, *checked=False*, *object_class_one_name=None*)
Shows dialog to let user select preferences for new relationship class.

show_add_relationships_form(*self*, *checked=False*, *relationship_class_key=()*, *object_class_name=''*, *object_name=''*)
Shows dialog to let user select preferences for new relationships.

show_edit_object_classes_form(*self*, *checked=False*)

show_edit_objects_form(*self*, *checked=False*)

show_edit_relationship_classes_form(*self*, *checked=False*)

show_edit_relationships_form(*self*, *checked=False*)

show_remove_object_tree_items_form(*self*)
Shows form to remove items from object treeview.

show_remove_relationship_tree_items_form(*self*)
Shows form to remove items from relationship treeview.

notify_items_changed(*self*, *action*, *item_type*, *db_map_data*)
Enables or disables actions and informs the user about what just happened.

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_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_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*)

16.1.2 Submodules

spinetoolbox.__main__

Spine Toolbox application main file.

author

P. Savolainen (VTT)

date 14.12.2017

Module Contents

spinetoolbox.__main__.**return_code**

spinetoolbox.config

Application constants and style sheets

author

P. Savolainen (VTT)

date 2.1.2018

Module Contents

spinetoolbox.config.**REQUIRED_SPINE_ENGINE_VERSION = 0.4.0**

spinetoolbox.config.**REQUIRED_SPINEDB_API_VERSION = 0.2.2**

spinetoolbox.config.**LATEST_PROJECT_VERSION = 1**

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.**TOOL_OUTPUT_DIR = output**

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.**TOOL_TYPES = ['Julia', 'Python', 'GAMS', 'Executable']**

spinetoolbox.config.**REQUIRED_KEYS = ['name', 'tooltype', 'includes']**

spinetoolbox.config.**OPTIONAL_KEYS = ['description', 'short_name', 'inputfiles', 'inputfiles**

spinetoolbox.config.**LIST_REQUIRED_KEYS = ['includes', 'inputfiles', 'inputfiles_opt', 'outp**

spinetoolbox.config.**JL_REPL_TIME_TO_DEAD = 5.0**

spinetoolbox.config.**JL_REPL_RESTART_LIMIT = 3**

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; b**

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

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

receive_item_execution_finished(*self*, *item_finish_state*)

TODO: Method obsolete? Pop next project item to execute or finish current graph if there are no items left.

Parameters **item_finish_state** (*ExecutionState*) – an enumeration to indicate if execution should continue or not

spinetoolbox.datapackage_import_export

Functions to import/export between spine database and frictionless data's datapackage.

author

M. Marin (KTH)

date 28.8.2018

Module Contents

class spinetoolbox.datapackage_import_export.**Signaler**

Bases: PySide2.QtCore.QObject

finished

failed

progressed

class spinetoolbox.datapackage_import_export.**DatapackageToSpineConverter**(*db_url*, *datapackage_descriptor*, *datapackage_base_path*)

Bases: PySide2.QtCore.QRunnable

number_of_steps(*self*)

run(*self*)

_run(*self*)

spinetoolbox.datapackage_import_export.**datapackage_to_spine**(*db_map*, *datapackage_file_path*)

Convert datapackage from *datapackage_file_path* into Spine *db_map*.

spinetoolbox.execution_managers

Classes to manage tool instance execution in various forms.

author

P. Savolainen (VTT)

date 1.2.2018

Module Contents

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*)
Run if there is an error in the running QProcess.

Parameters **process_error** (*QProcess::ProcessError*) – Process error number

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)

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

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. Starts drawing a link.

Parameters **event** (*QGraphicsSceneMouseEvent*) – Event

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.**NotificationListItem**
Bases: PySide2.QtWidgets.QGraphicsTextItem

Notification list graphics item. Used to show notifications for a ProjectItem

setHtml(*self*, *html*)

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.**ProjectItemIcon**(*toolbox*, *x*, *y*, *w*, *h*, *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
- **w** (*float*) – Icon width
- **h** (*float*) – Icon height
- **project_item** (ProjectItem) – Item
- **icon_file** (*str*) – Path to icon resource
- **icon_color** (*QColor*) – Icon's color
- **background_color** (*QColor*) – Background color

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

update_links_geometry(*self*)
Updates geometry of connected links to reflect this item's most recent position.

mouseReleaseEvent(*self*, *event*)

shrink_scene_if_needed(*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.**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

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.

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*

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

mouseDoubleClickEvent(*self*, *e*)
Accepts event if there's a connector button underneath, to prevent unwanted creation of feedback links.

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.

Parameters **toolbox** (ToolboxUI) – main UI class instance

dst_connector

dst_rect

dst_center

start_drawing_at(*self*, *src_connector*)
Starts drawing a link from the given connector.

Parameters **src_connector** (ConnectorButton) –

spinetoolbox.helpers

General helper functions and classes.

authors

P. Savolainen (VTT)

date 10.1.2018

Module Contents

spinetoolbox.helpers.**set_taskbar_icon**()
Set application icon to Windows taskbar.

spinetoolbox.helpers.**supported_img_formats**()
Function to check if reading .ico files is supported.

spinetoolbox.helpers.**pyside2_version_check**()
Check that PySide2 version is older than 5.12, since this is not supported yet. Issue #238 in GitLab.

qt_version is the Qt version used to compile PySide2 as string. E.g. "5.11.2" qt_version_info is a tuple with each version component of Qt used to compile PySide2. E.g. (5, 11, 2)

spinetoolbox.helpers.**spinedb_api_version_check**()
Check if spinedb_api is the correct version and explain how to upgrade if it is not.

spinetoolbox.helpers.**spine_engine_version_check**()
Check if spine engine package is the correct version and explain how to upgrade if it is not.

spinetoolbox.helpers.**busy_effect**(*func*)
Decorator to change the mouse cursor to 'busy' while a function is processed.

Parameters **func** – Decorated function.

spinetoolbox.helpers.**get_datetime**(*show*, *date=True*)
Returns date and time string for appending into Event Log messages.

Parameters

- **show** (*bool*) – True returns date and time string. False returns empty string.

- **date** (*bool*) – Whether or not the date should be included in the result

spinetoolbox.helpers.**create_dir**(*base_path*, *folder=''*, *verbosity=False*)
Create (input/output) directories recursively.

Parameters

- **base_path** (*str*) – Absolute path to wanted dir
- **folder** (*str*) – (Optional) Folder name. Usually short name of item.
- **verbosity** (*bool*) – True prints a message that tells if the directory already existed or if it was created.

Returns True if directory already exists or if it was created successfully.

Raises OSError if operation failed.

spinetoolbox.helpers.**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.helpers.**copy_files**(*src_dir*, *dst_dir*, *includes=None*, *excludes=None*)
Function for copying files. Does not copy folders.

Parameters

- **src_dir** (*str*) – Source directory
- **dst_dir** (*str*) – Destination directory
- **includes** (*list*) – Included files (wildcards accepted)
- **excludes** (*list*) – Excluded files (wildcards accepted)

Returns Number of files copied

Return type count (int)

spinetoolbox.helpers.**erase_dir**(*path*, *verbosity=False*)
Deletes a directory and all its contents without prompt.

Parameters

- **path** (*str*) – Path to directory
- **verbosity** (*bool*) – Print logging messages or not

spinetoolbox.helpers.**copy_dir**(*widget*, *src_dir*, *dst_dir*)
Makes a copy of a directory. All files and folders are copied. Destination directory must not exist. Does not overwrite files.

Parameters

- **widget** (*QWidget*) – Parent widget for QMessageBoxes
- **src_dir** (*str*) – Absolute path to directory that will be copied
- **dst_dir** (*str*) – Absolute path to new directory

spinetoolbox.helpers.**recursive_overwrite**(*widget*, *src*, *dst*, *ignore=None*, *silent=True*)
Copies everything from source directory to destination directory recursively. Overwrites existing files.

Parameters

- **widget** (*QWidget*) – Enables e.g. printing to Event Log
- **src** (*str*) – Source directory

- **dst** (*str*) – Destination directory
- **ignore** – Ignore function
- **silent** (*bool*) – If False, messages are sent to Event Log, If True, copying is done in silence

spinetoolbox.helpers.**rename_dir**(*old_dir*, *new_dir*, *logger*)
Rename directory. Note: This is not used in renaming projects due to unreliability. Looks like it works fine in renaming project items though.

Parameters

- **old_dir** (*str*) – Absolute path to directory that will be renamed
- **new_dir** (*str*) – Absolute path to new directory
- **logger** (LoggerInterface) – A logger instance

spinetoolbox.helpers.**fix_name_ambiguity**(*input_list*, *offset=0*)
Modify repeated entries in name list by appending an increasing integer.

spinetoolbox.helpers.**tuple_itemgetter**(*itemgetter_func*, *num_indexes*)
Change output of itemgetter to always be a tuple even for one index

spinetoolbox.helpers.**format_string_list**(*str_list*)
Return an unordered html list with all elements in str_list. Intended to print error logs as returned by spinedb_api.

Parameters **str_list** (*list(str)*) –

spinetoolbox.helpers.**rows_to_row_count_tuples**(*rows*)
Breaks a list of rows into a list of (row, count) tuples corresponding to chunks of successive rows.

spinetoolbox.helpers.**inverted**(*input_*)
Inverts a dictionary that maps keys to a list of values. The output maps values to a list of keys that include the value in the input.

class spinetoolbox.helpers.**Singleton**
Bases: type

A singleton class from SO.

_instances

__call__(*cls*, **args*, ***kwargs*)

class spinetoolbox.helpers.**IconListManager**(*icon_size*)
A class to manage icons for icon list widgets.

init_model(*self*)
Init model that can be used to display all icons in a list.

_model_data(*self*, *index*, *role*)
Replacement method for model.data(). Create pixmaps as they're requested by the data() method, to reduce loading time.

create_object_pixmap(*self*, *display_icon*)
Create and return a pixmap corresponding to display_icon.

class spinetoolbox.helpers.**IconManager**
A class to manage object class icons for data store forms.

ICON_SIZE

create_object_pixmap(*self*, *display_icon*)
Create a pixmap corresponding to display_icon, cache it, and return it.

setup_object_pixmaps(*self*, *object_classes*)
Called after adding or updating object classes. Create the corresponding object pixmaps and clear obsolete entries from the relationship class icon cache.

object_pixmap(*self*, *object_class_name*)
A pixmap for the given object class.

object_icon(*self*, *object_class_name*)
An icon for the given object class.

relationship_pixmap(*self*, *str_object_class_name_list*)
A pixmap for the given object class name list, created by rendering several object pixmaps next to each other.

relationship_icon(*self*, *str_object_class_name_list*)
An icon for the given object class name list.

class spinetoolbox.helpers.**CharIconEngine**(*char*, *color*)
Bases: PySide2.QtGui.QIconEngine

Specialization of QIconEngine used to draw font-based icons.

paint(*self*, *painter*, *rect*, *mode=None*, *state=None*)

pixmap(*self*, *size*, *mode=None*, *state=None*)

spinetoolbox.helpers.**make_icon_id**(*icon_code*, *color_code*)
Take icon and color codes, and return equivalent integer.

spinetoolbox.helpers.**interpret_icon_id**(*display_icon*)
Take a display icon integer and return an equivalent tuple of icon and color code.

spinetoolbox.helpers.**default_icon_id**()

class spinetoolbox.helpers.**ProjectDirectoryIconProvider**
Bases: PySide2.QtWidgets.QFileIconProvider

QFileIconProvider that provides a Spine icon to the Open Project Dialog when a Spine Toolbox project directory is encountered.

icon(*self*, *info*)
Returns an icon for the file described by info.

Parameters **info** (*QFileInfo*) – File (or directory) info

Returns Icon for a file system resource with the given info

Return type QIcon

spinetoolbox.helpers.**path_in_dir**(*path*, *directory*)
Returns True if the given path is in the given directory.

spinetoolbox.helpers.**serialize_path**(*path*, *project_dir*)
Returns a dict representation of the given path.

If path is in project_dir, converts the path to relative. If path does not exist returns it as-is.

Parameters

- **path** (*str*) – path to serialize
- **project_dir** (*str*) – path to the project directory

Returns Dictionary representing the given path

Return type dict

spinetoolbox.helpers.**serialize_url**(*url*, *project_dir*)
Return a dict representation of the given URL.

If the URL is a file that is in project dir, the URL is converted to a relative path.

Parameters

- **url** (*str*) – a URL to serialize
- **project_dir** (*str*) – path to the project directory

Returns Dictionary representing the URL

Return type dict

spinetoolbox.helpers.**deserialize_path**(*serialized*, *project_dir*)
Returns a deserialized path or URL.

Parameters

- **serialized** (*dict*) – a serialized path or URL
- **project_dir** (*str*) – path to the project directory

Returns Path or URL as string

Return type str

spinetoolbox.logger_interface

A logger interface.

authors

A. Soininen (VTT)

date 16.1.2020

Module Contents

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 instead of 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).

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

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

spinetoolbox.metaobject.**shorten**(*name*)
Returns a 'shortened' version of given name.

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 Graph, Tree and Tabular views 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

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_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.

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.

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_types_inconsistent** (*values*)
Raises an exception if not all values are TimeSeries or floats.

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 floats 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 plot support object

Returns a tuple of values and label(s)

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.**_raise_if_value_types_clash**(*values*, *plot_widget*)
Raises a PlottingError if values type is incompatible with plot_widget.

spinetoolbox.plugin_loader

Contains a minimal plugin loader infrastructure.

author

P. Savolainen (VTT)

date 11.6.2019

Module Contents

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

class spinetoolbox.project.**SpineToolboxProject**(*toolbox*, *name*, *description*, *p_dir*, *project_item_model*, *settings*, *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
- **logger** (LoggerInterface) – a logger instance

dag_execution_finished

dag_execution_about_to_start
Emitted just before an engine runs. Provides a reference to the engine.

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*)

static get_connections(*links*)

save(*self*, *tool_spec_paths*)
Collects project information and objects into a dictionary and writes it to a JSON file.

Parameters **tool_spec_paths** (*list*) – List of absolute paths to tool specification files

Returns True or False depending on success

Return type bool

load(*self*, *objects_dict*)
Populates project item model with items loaded from project file.

Parameters **objects_dict** (*dict*) – Dictionary containing all project items in JSON format

Returns True if successful, False otherwise

Return type bool

load_tool_specification_from_file(*self*, *jsonfile*)
Returns a Tool specification from a definition file.

Parameters **jsonfile** (*str*) – Path of the tool specification definition file

Returns ToolSpecification or None if reading the file failed

load_tool_specification_from_dict(*self*, *definition*, *path*)
Returns a Tool specification from a definition dictionary.

Parameters

- **definition** (*dict*) – Dictionary with the tool definition
- **path** (*str*) – Directory where main program file is located

Returns ToolSpecification, NoneType

add_project_items(*self*, *category_name*, **items*, *set_selected=False*, *verbosity=True*)
Pushes an AddProjectItemsCommand to the toolbox undo stack.

make_project_tree_items(*self*, *category_name*, **items*)
Creates and returns list of LeafProjectTreeItem instances.

Parameters

- **category_name** (*str*) – The items' category
- **items** (*dict*) – one or more dict of items to add

Returns list(LeafProjectTreeItem)

_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) –

do_add_project_items(*self*, *category_name*, **items*, *set_selected=False*, *verbosity=True*)
Adds items to project at loading.

Parameters

- **category_name** (*str*) – The items' category
- **items** (*dict*) – 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.

spinetoolbox.project_commands

QUndoCommand subclasses for modifying the project.

authors

M. Marin (KTH)

date 12.2.2020

Module Contents

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*, *category_name*, **items*, *set_selected=False*, *verbosity=True*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to add items.

Parameters

- **project** (SpineToolboxProject) – the project
- **category_name** (*str*) – The items' category
- **items** (*dict*) – 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()

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.**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_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*)

class spinetoolbox.project_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*)

class spinetoolbox.project_commands.**UpdateImporterSettingsCommand**(*importer*, *settings*, *importee*)

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
- **importee** (*str*) – the filepath

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**UpdateImporterCancelOnErrorCommand**(*importer*, *cancel_on_error*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Importer cancel on error setting.

Parameters

- **importer** (spinetoolbox.project_items.importer.importer.Importer) – the Importer
- **cancel_on_error** (*bool*) – the new setting

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**SetToolSpecificationCommand**(*tool*, *specification*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to set the specification for a Tool.

Parameters

- **tool** (Tool) – the Tool
- **specification** (ToolSpecification) – the new tool spec

redo(*self*)

undo(*self*)

class spinetoolbox.project_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_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*)

class spinetoolbox.project_commands.**UpdateExporterOutFileNameCommand**(*exporter*, *file_name*, *database_path*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Exporter output file name.

Parameters

- **exporter** (Exporter) – the Exporter
- **export_list_item** (ExportListItem) – the widget that holds the name

- **file_name** (*str*) – the output filename
- **database_path** (*str*) – the associated db path

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**UpdateExporterSettingsCommand**(*exporter*, *settings*, *indexing_settings*, *indexing_domains*, *merging_settings*, *merging_domains*, *database_path*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Exporter settings.

Parameters

- **exporter** (Exporter) – the Exporter
- **database_path** (*str*) – the db path to update settings for

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**AddToolSpecificationCommand**(*toolbox*, *tool_specification*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to add Tool specs to a project.

Parameters

- **toolbox** (ToolboxUI) – the toolbox
- **tool_specification** (ToolSpecification) – the tool spec

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**RemoveToolSpecificationCommand**(*toolbox*, *row*, *ask_verification*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to remove Tool specs from a project.

Parameters

- **toolbox** (ToolboxUI) – the toolbox
- **row** (*int*) – the row in the ToolSpecificationModel
- **ask_verification** (*bool*) – if True, shows confirmation message the first time

redo(*self*)

undo(*self*)

class spinetoolbox.project_commands.**UpdateToolSpecificationCommand**(*toolbox*, *row*, *tool_specification*)

Bases: *spinetoolbox.project_commands.SpineToolboxCommand*

Command to update Tool specs in a project.

Parameters

- **toolbox** (ToolboxUI) – the toolbox
- **row** (*int*) – the row in the ToolSpecificationModel of the spec to be replaced
- **tool_specification** (ToolSpecification) – the updated tool spec

redo(*self*)

undo(*self*)

spinetoolbox.project_item

ProjectItem and ProjectItemResource classes.

authors

P. Savolainen (VTT)

date 4.10.2018

Module Contents

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

create_data_dir(*self*)

static item_type()
Item's type identifier string.

static category()
Item's category identifier string.

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*)

set_icon(*self*, *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.

stop_execution(*self*)
Stops executing this View.

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** (*str*) – either "forward" or "backward"

Returns True if execution succeeded, False otherwise

Return type bool

run_leave_animation(*self*)
Runs the animation that represents execution leaving this item. Blocks until the animation is finished.

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(*self*, *direction*)
Returns output resources for execution 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** (*str*) – Direction where output resources are passed

Returns a list of ProjectItemResources

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

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

make_execution_leave_animation(*self*)
Returns animation to play when execution leaves this item.

Returns QParallelAnimationGroup

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

available_resources_downstream(*self*, *upstream_resources*)
Returns resources available to downstream items.

Should be reimplemented by subclasses if they want to offer resources to downstream items. The default implementation returns an empty list.

Parameters **upstream_resources** (*list*) – a list of resources available from upstream items

Returns a list of ProjectItemResources

available_resources_upstream(*self*)
Returns resources available to upstream items.

Should be reimplemented by subclasses if they want to offer resources to upstream items. The default implementation returns an empty list.

Returns a list of ProjectItemResources

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** (*str*) – 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

class spinetoolbox.project_item.**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.

Init class.

Parameters

- **provider** (ProjectItem) – The item that provides the resource
- **type** (*str*) – The resource type, either "file" or "database" (for now)
- **url** (*str*) – The url of the resource
- **metadata** (*dict*) – Some metadata providing extra information about the resource. For now it has one key: - future (bool): whether the resource is from the future, e.g. Tool output files advertised beforehand

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*)

__repr__(*self*)

spinetoolbox.project_tree_item

Project Tree items.

authors

A. Soininen (VTT)

date 17.1.2020

Module Contents

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.

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*, *item_maker*, *icon_maker*, *add_form_maker*, *properties_ui*)
Bases: *spinetoolbox.project_tree_item.BaseProjectTreeItem*

Class for category project tree items.

Parameters

- **name** (*str*) – Category name
- **description** (*str*) – Category description
- **item_maker** (*function*) – A function for creating project items in this category
- **icon_maker** (*function*) – A function for creating icons (QGraphicsItems) for project items in this category
- **add_form_maker** (*function*) – A function for creating the form to add project items to this category
- **properties_ui** (*object*) – An object holding the Item Properties UI

flags(*self*)
Returns the item flags.

item_maker(*self*)
Returns the item maker method.

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
- **toolbox** (ToolboxUI) – A toolbox instance

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

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

is_valid(*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

upgrade(*self*, *project_dict*, *old_project_dir*, *new_project_dir*)

Converts the project described in given project description file to the latest version.

Parameters

- **project_dict** (*dict*) – Full path to project description file, ie. .proj or .json
- **old_project_dir** (*str*) – Path to the original project directory
- **new_project_dir** (*str*) – New project directory

Returns Latest version of the project info dictionary

Return type dict

static upgrade_to_latest(*v*, *project_dict*)

Upgrades the given project dictionary to the latest version.

NOTE: Implement this when the structure of the project file needs to be changed.

Parameters

- **v** (*int*) – project version
- **project_dict** (*dict*) – Project JSON to be converted

Returns Upgraded project information JSON

Return type dict

upgrade_from_no_version_to_version_1(*self*, *old*, *old_project_dir*, *new_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
- **new_project_dir** (*str*) – Path to new 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 Upgraded project information JSON or None if the operation failed

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

spinetoolbox.spine_db_commands

QUndoCommand subclasses for modifying the db.

authors

M. Marin (KTH)

date 31.1.2020

Module Contents

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.**CommandBase**(*db_mngr*, *db_map*)
Bases: PySide2.QtWidgets.QUndoCommand

Parameters

- **db_mngr** (SpineDBManager) – SpineDBManager instance
- **db_map** (*DiffDatabaseMapping*) – DiffDatabaseMapping instance

age

block_notifications(*self*, *func*)
Calls given function while blocking notifications on the affected Data Store forms. This is so undo() and subsequent redo() calls don't trigger the same notifications over and over.

static redomethod(*func*)
Wraps the given function with a mechanism to determine this command's completion. The command is considered completed if calling the function triggers a certain signal. Once the command is completed, we don't listen to the signal anymore and we also block notifications on the affected Data Store forms. If the signal is not received, then the command is declared obsolete.

static undomethod(*func*)
Wraps the given function with an artifact to block notifications on the affected Data Store forms.

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*)
Bases: *spinetoolbox.spine_db_commands.CommandBase*

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

_command_name

_method_name

_redo_method_name

_emit_signal_name

_receive_signal_name

redo(*self*)

undo(*self*)

receive_items_changed(*self*, *db_map_data*)

data(*self*)

class spinetoolbox.spine_db_commands.**AddCheckedParameterValuesCommand**(*db_mngr*, *db_map*, *data*)

Bases: *spinetoolbox.spine_db_commands.AddItemsCommand*

class spinetoolbox.spine_db_commands.**UpdateItemsCommand**(*db_mngr*, *db_map*, *data*, *item_type*)

Bases: *spinetoolbox.spine_db_commands.CommandBase*

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

_command_name

_method_name

_emit_signal_name

_undo_item(*self*, *db_map*, *redo_item*)

redo(*self*)

undo(*self*)

data(*self*)

class spinetoolbox.spine_db_commands.**UpdateCheckedParameterValuesCommand**(*db_mngr*, *db_map*, *data*)

Bases: *spinetoolbox.spine_db_commands.UpdateItemsCommand*

class spinetoolbox.spine_db_commands.**SetParameterDefinitionTagsCommand**(*db_mngr*, *db_map*, *data*)

Bases: *spinetoolbox.spine_db_commands.CommandBase*

_undo_item(*self*, *db_map*, *redo_item*)

redo(*self*)

undo(*self*)

class spinetoolbox.spine_db_commands.**RemoveItemsCommand**(*db_mngr*, *db_map*, *typed_data*)

Bases: *spinetoolbox.spine_db_commands.CommandBase*

Parameters

- **db_mngr** (SpineDBManager) – SpineDBManager instance
- **db_map** (*DiffDatabaseMapping*) – DiffDatabaseMapping instance
- **typed_data** (*dict*) – lists of dict-items to remove keyed by string type

_undo_method_name

_emit_signal_name

redo(*self*)

undo(*self*)

receive_items_changed(*self*, *db_map_typed_data*)

data(*self*)

spinetoolbox.spine_db_manager

The SpineDBManager class

authors

P. Vennström (VTT) and M. Marin (KTH)

date 2.10.2019

Module Contents

spinetoolbox.spine_db_manager.**do_create_new_spine_database**(*url*, *for_spine_model*)
Creates a new spine database at the given url.

class spinetoolbox.spine_db_manager.**SpineDBManager**(*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

- **logger** (*LoggingInterface*) – a general, non-database-specific logger
- **project** (SpineToolboxProject) –

session_refreshed

session_committed

session_rolled_back

object_classes_added

objects_added

relationship_classes_added

relationships_added

parameter_definitions_added

_parameter_definitions_added

parameter_values_added

_parameter_values_added

parameter_value_lists_added

parameter_tags_added

object_classes_removed

objects_removed

relationship_classes_removed

relationships_removed

parameter_definitions_removed

parameter_values_removed

parameter_value_lists_removed

parameter_tags_removed

object_classes_updated

objects_updated

relationship_classes_updated

relationships_updated

parameter_definitions_updated

_parameter_definitions_updated

parameter_values_updated

_parameter_values_updated

parameter_value_lists_updated

parameter_tags_updated

parameter_definition_tags_set

items_removed_from_cache

_GROUP_SEP =

db_maps

create_new_spine_database(*self*, *url*, *for_spine_model=False*)

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.

get_db_map(*self*, *url*, *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*) –
- **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

get_db_map_for_listener(*self*, *listener*, *url*, *upgrade=False*, *codename=None*)

remove_db_map_listener(*self*, *db_map*, *listener*)

set_logger_for_db_map(*self*, *logger*, *db_map*)

unset_logger_for_db_map(*self*, *db_map*)

refresh_session(*self*, **db_maps*)

commit_session(*self*, **db_maps*)

static _get_commit_msg(*db_map*)

rollback_session(*self*, **db_maps*)

_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

cache_parameter_definition_tags(*self*, *db_map_data*)
Caches parameter definition tags in the parameter definition dictionary.

Parameters **db_map_data** (*dict*) – lists of parameter definition items keyed by DiffDatabaseMapping

uncache_items(*self*, *item_type*, *db_map_data*)
Removes data from cache.

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*)
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

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_items_from_db(*self*, *db_map*, *item_type*)
Returns all items of the given type in the given db map. Called by the above methods whenever they don't find what they're looking for in cache.

get_field(*self*, *db_map*, *item_type*, *id_*, *field*)

get_value(*self*, *db_map*, *item_type*, *id_*, *field*, *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*) –
- **field** (*str*) – either "value" or "default_value"
- **role** (*int, optional*) –

parse_value(*self*, *value*)

static _display_data(*parsed_value*)
Returns the value's database representation formatted for Qt.DisplayRole.

static _tool_tip_data(*parsed_value*)
Returns the value's database representation formatted for Qt.ToolTipRole.

get_object_classes(*self*, *db_map*, *cache=True*)
Returns object classes from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_objects(*self*, *db_map*, *class_id=None*, *cache=True*)
Returns objects from database.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **class_id** (*int, optional*) –

Returns dictionary items

Return type list

get_relationship_classes(*self*, *db_map*, *ids=None*, *object_class_id=None*, *cache=True*)
Returns relationship classes from database.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **ids** (*set, optional*) –
- **object_class_id** (*int, optional*) –

Returns dictionary items

Return type list

get_relationships(*self*, *db_map*, *ids=None*, *class_id=None*, *object_id=None*, *cache=True*)
Returns relationships from database.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **ids** (*set, optional*) –
- **class_id** (*int, optional*) –
- **object_id** (*int, optional*) –

Returns dictionary items

Return type list

get_object_parameter_definitions(*self*, *db_map*, *ids=None*, *object_class_id=None*, *cache=True*)
Returns object parameter definitions from database.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **ids** (*set, optional*) –
- **object_class_id** (*int, optional*) –

Returns dictionary items

Return type list

get_relationship_parameter_definitions(*self*, *db_map*, *ids=None*, *relationship_class_id=None*, *cache=True*)
Returns relationship parameter definitions from database.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **ids** (*set, optional*) –
- **relationship_class_id** (*int, optional*) –

Returns dictionary items

Return type list

get_object_parameter_values(*self*, *db_map*, *ids=None*, *object_class_id=None*, *cache=True*)
Returns object parameter values from database.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **ids** (*set*) –
- **object_class_id** (*int*) –

Returns dictionary items

Return type list

get_relationship_parameter_values(*self*, *db_map*, *ids=None*, *relationship_class_id=None*, *cache=True*)
Returns relationship parameter values from database.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **ids** (*set*) –
- **relationship_class_id** (*int*) –

Returns dictionary items

Return type list

get_parameter_definitions(*self*, *db_map*, *ids=None*, *entity_class_id=None*, *cache=True*)
Returns both object and relationship parameter definitions.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **ids** (*set, optional*) –
- **entity_class_id** (*int, optional*) –

Returns dictionary items

Return type list

get_parameter_values(*self*, *db_map*, *ids=None*, *entity_class_id=None*, *cache=True*)
Returns both object and relationship parameter values.

Parameters

- **db_map** (*DiffDatabaseMapping*) –
- **ids** (*set, optional*) –
- **entity_class_id** (*int, optional*) –

Returns dictionary items

Return type list

get_parameter_value_lists(*self*, *db_map*, *cache=True*)
Returns parameter value lists from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

get_parameter_tags(*self*, *db_map*, *cache=True*)
Get parameter tags from database.

Parameters **db_map** (*DiffDatabaseMapping*) –

Returns dictionary items

Return type list

add_or_update_items(*self*, *db_map_data*, *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

- **signal_name** (*str*) – signal attribute of SpineDBManager to emit if successful

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_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_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_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_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_data*)

do_remove_items(*self*, *db_map_typed_data*)
Removes items from database.

Parameters **db_map_typed_data** (*dict*) – lists of items to remove, keyed by item type (str), keyed by DiffDatabaseMapping

static _to_ids(*db_map_data*)

cascade_remove_objects(*self*, *db_map_data*)
Removes objects in cascade when removing object classes.

Parameters **db_map_data** (*dict*) – lists of removed items keyed by DiffDatabaseMapping

cascade_remove_relationship_classes(*self*, *db_map_data*)
Removes relationship classes in cascade when removing object classes.

Parameters **db_map_data** (*dict*) – lists of removed items keyed by DiffDatabaseMapping

cascade_remove_relationships_by_class(*self*, *db_map_data*)
Removes relationships in cascade when removing objects.

Parameters **db_map_data** (*dict*) – lists of removed items keyed by DiffDatabaseMapping

cascade_remove_relationships_by_object(*self*, *db_map_data*)
Removes relationships in cascade when removing relationship classes.

Parameters **db_map_data** (*dict*) – lists of removed items keyed by DiffDatabaseMapping

cascade_remove_parameter_definitions(*self*, *db_map_data*)
Removes parameter definitions in cascade when removing entity classes.

Parameters **db_map_data** (*dict*) – lists of removed items keyed by DiffDatabaseMapping

cascade_remove_parameter_values_by_entity_class(*self*, *db_map_data*)
Removes parameter values in cascade when removing entity classes.

Parameters **db_map_data** (*dict*) – lists of removed items keyed by DiffDatabaseMapping

cascade_remove_parameter_values_by_entity(*self*, *db_map_data*)
Removes parameter values in cascade when removing entity classes when removing entities.

Parameters **db_map_data** (*dict*) – lists of removed items keyed by DiffDatabaseMapping

cascade_remove_parameter_values_by_definition(*self*, *db_map_data*)
Removes parameter values in cascade when when removing parameter definitions.

Parameters **db_map_data** (*dict*) – lists of removed 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_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_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_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

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.

do_add_parameter_definitions(*self*, *db_map_data*)
Adds parameter definitions in extended format given data in compact format.

Parameters **db_map_data** (*dict*) – lists of parameter definition items keyed by DiffDatabaseMapping

do_add_parameter_values(*self*, *db_map_data*)
Adds parameter values in extended format given data in compact format.

Parameters **db_map_data** (*dict*) – lists of parameter value items keyed by DiffDatabaseMapping

do_update_parameter_definitions(*self*, *db_map_data*)
Updates parameter definitions in extended format given data in compact format.

Parameters **db_map_data** (*dict*) – lists of parameter definition items keyed by DiffDatabaseMapping

do_update_parameter_values(*self*, *db_map_data*)
Updates parameter values in extended format given data in compact format.

Parameters **db_map_data** (*dict*) – lists of parameter value items keyed by DiffDatabaseMapping

spinetoolbox.spine_db_signaller

Spine DB Signaller class.

authors

M. Marin (KTH)

date 31.10.2019

Module Contents

class spinetoolbox.spine_db_signaller.**SpineDBSignaller**(*db_mngr*)
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_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_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_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_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_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*)

receive_session_rolled_back(*self*, *db_maps*)

spinetoolbox.tool_instance

Contains ToolInstance class.

authors

P. Savolainen (VTT), E. Rinne (VTT)

date 1.2.2018

Module Contents

class spinetoolbox.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.tool_instance.**GAMSToolInstance**

Bases: *spinetoolbox.tool_instance.ToolInstance*

Class for GAMS Tool instances.

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.tool_instance.**JuliaToolInstance**(*toolbox*, *tool_specification*, *basedir*, *settings*, *logger*)

Bases: *spinetoolbox.tool_instance.ToolInstance*

Class for Julia Tool instances.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **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_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.tool_instance.**PythonToolInstance**(*toolbox*, *tool_specification*, *basedir*, *settings*, *logger*)

Bases: *spinetoolbox.tool_instance.ToolInstance*

Class for Python Tool instances.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance

- **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_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.tool_instance.**ExecutableToolInstance**
Bases: *spinetoolbox.tool_instance.ToolInstance*

Class for Executable Tool instances.

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.tool_specifications

Contains Tool specification classes.

authors

P. Savolainen (VTT), E. Rinne (VTT), M. Marin (KTH)

date 24.1.2018

Module Contents

spinetoolbox.tool_specifications.**CMDLINE_TAG_EDGE = @@**

class spinetoolbox.tool_specifications.**CmdlineTag**

URL

URL_INPUTS

URL_OUTPUTS

OPTIONAL_INPUTS

class spinetoolbox.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.metaobject.MetaObject*

Super class for all tool specifications.

Parameters

- **name** (*str*) – Name of the tool
- **tooltype** (*str*) – Type of Tool (e.g. Python, Julia, ..)
- **path** (*str*) – Path to tool
- **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

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

set_def_path(*self*, *path*)

Sets the file path for this tool specification.

Parameters **path** (*str*) – Absolute path to the specification file.

get_def_path(*self*)

Returns tool specification file path.

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 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

static _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

class spinetoolbox.tool_specifications.**GAMSTool**(*name*, *tooltype*, *path*, *includes*, *settings*, *logger*, *description=None*, *inputfiles=None*, *inputfiles_opt=None*, *outputfiles=None*, *cmdline_args=None*, *execute_in_work=True*)

Bases: *spinetoolbox.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)

__repr__(*self*)
[OBSOLETE]. Returns instance of this class as a string.

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.tool_specifications.**JuliaTool**(*toolbox*, *name*, *tooltype*, *path*, *includes*, *settings*, *logger*, *description=None*, *inputfiles=None*, *inputfiles_opt=None*, *outputfiles=None*, *cmdline_args=None*, *execute_in_work=True*)
Bases: *spinetoolbox.tool_specifications.ToolSpecification*

Class for Julia tool specifications.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **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
- **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)

__repr__(*self*)
[OBSOLETE]. Returns instance of this class as a string.

update_julia_options(*self*, *key*, *value*)
[OBSOLETE?] Updates Julia command line options.

Parameters

- **key** (*str*) – Option name
- **value** (*int, float*) – Option value

static load(*toolbox*, *path*, *data*, *settings*, *logger*)
Creates a JuliaTool according to a tool specification file.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **path** (*str*) – Base path to tool files
- **data** (*dict*) – Dictionary of tool definitions

- **settings** (*QSetting*) – Toolbox settings
- **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.tool_specifications.**PythonTool**(*toolbox*, *name*, *tooltype*, *path*, *includes*, *settings*, *logger*, *description=None*, *inputfiles=None*, *inputfiles_opt=None*, *outputfiles=None*, *cmdline_args=None*, *execute_in_work=True*)
Bases: *spinetoolbox.tool_specifications.ToolSpecification*

Class for Python tool specifications.

Parameters

- **toolbox** (ToolboxUI) – QMainWindow instance
- **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
- **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)

__repr__(*self*)
[OBSOLETE]. Returns instance of this class as a string.

update_python_options(*self*, *key*, *value*)
[OBSOLETE?] Updates Python command line options.

Parameters

- **key** (*str*) – Option name
- **value** (*int, float*) – Option value

static load(*toolbox*, *path*, *data*, *settings*, *logger*)
Creates a PythonTool according to a tool specification file.

Parameters

- **toolbox** (ToolboxUI) – Toolbox main window
- **path** (*str*) – Base path to tool files
- **data** (*dict*) – Dictionary of tool definitions
- **settings** (*QSettings*) – Toolbox settings
- **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.tool_specifications.**ExecutableTool**(*name*, *tooltype*, *path*, *includes*, *settings*, *logger*, *description=None*, *inputfiles=None*, *inputfiles_opt=None*, *outputfiles=None*, *cmdline_args=None*, *execute_in_work=True*)
Bases: *spinetoolbox.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)

__repr__(*self*)
[OBSOLETE]. Returns instance of this class as a string.

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.ui_main

Contains ToolboxUI class.

author

P. Savolainen (VTT)

date 14.12.2017

Module Contents

class spinetoolbox.ui_main.**ToolboxUI**
Bases: PySide2.QtWidgets.QMainWindow

Class for application main GUI functions.

Initialize application and main window.

msg

msg_success

msg_error

msg_warning

msg_proc

information_box

error_box

msg_proc_error

tool_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.

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*) – If given, changes the work directory to given
- **creates the directory if it does not exist.** (*and*) –

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_tool_specification_model(*self*, *tool_specification_paths*)
Initializes Tool specification model.

Parameters **tool_specification_paths** (*list*) – List of tool definition file paths used in this project

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

open_tool_specification (*self*)
Opens a file dialog where the user can select an existing tool specification definition file (.json). If file is valid, calls add_tool_specification().

add_tool_specification (*self*, *tool_specification*)
Pushes a new AddToolSpecificationCommand to the undo stack.

do_add_tool_specification (*self*, *tool_specification*, *row=None*)
Adds a ToolSpecification instance to project, which then can be added to a Tool item. Adds the tool specification file path into project file (project.json)

Parameters **tool_specification** (ToolSpecification) – Tool specification that is added to project

update_tool_specification (*self*, *row*, *tool_specification*)
Pushes a new UpdateToolSpecificationCommand to the undo stack.

do_update_tool_specification (*self*, *row*, *tool_specification*)
Updates a Tool specification and refreshes all Tools that use it.

Parameters

- **row** (*int*) – Row of tool specification in ToolSpecificationModel
- **tool_specification** (ToolSpecification) – An updated Tool specification

update_tool_settings (*self*, *tool_settings*)
Updates tool specification and execution mode for a bunch of tool items. Called just after successfully updating a Tool Specification.

Parameters **tool_settings** (*dict*) – mapping Tool items to a tuple of (ToolSpecification instance, bool execution mode)

remove_selected_tool_specification (*self*, *checked=False*)
Removes tool specification selected in QListView.

remove_tool_specification (*self*, *row*, *ask_verification=True*)

do_remove_tool_specification (*self*, *row*, *ask_verification=True*)
Removes tool specification from ToolSpecificationModel. Removes also Tool specifications from all Tool items that use this specification.

Parameters

- **row** (*int*) – Row in ToolSpecificationModel
- **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

edit_tool_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_tool_specification_file (*self*, *index*)
Open the Tool specification definition file in the default (.json) text-editor.

Parameters **index** (*QModelIndex*) – Index of the item

open_tool_main_program_file(*self*, *index*)
Open the tool specification's main program file in the default 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_category*, *x=0*, *y=0*)
Show add project item widget.

show_tool_specification_form(*self*, *tool_specification=None*)
Show tool specification widget.

show_settings(*self*)
Show Settings widget.

show_about(*self*)
Show About Spine Toolbox form.

show_user_guide(*self*)
Open Spine Toolbox documentation index page in browser.

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

show_tool_specification_context_menu(*self*, *pos*)
Context menu for tool specifications.

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

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

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'.

major :int

minor :int

micro :int

releaselevel :str

serial :int

spinetoolbox.version.**major = 0**

spinetoolbox.version.**minor = 4**

spinetoolbox.version.**micro = 0**

spinetoolbox.version.**releaselevel = final**

spinetoolbox.version.**serial = 0**

spinetoolbox.version.**__version_info__**

spinetoolbox.version.**__version__**

16.1.3 Package Contents

spinetoolbox.**__version__**

spinetoolbox.**__version_info__**

CHAPTER 17

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

J

K

O

P

Q

R

S

T

U

V

W

X

Y

Z