McStasScript

Mads Bertelsen

March 7, 2019

1 Introduction

This document serves as the documentation for the McStasScript scripting language for python. Its purpose is to generate McStas instrument files from python which is simply another way of writing an instrument file. The main advantages is the possibility of using for-loops and that it can be used directly from a python terminal. The simulation described by the instrument can be executed from the scripting language and the data can be manipulated before plotting.

2 Documentation

This section describes the classes and their methods.

class McStas_instr

Holds methods for creating a McStas instrument file

Initiating an instance of the class requires a name to be given as the first argument and two optional keyword arguments currently supported, allowing information on the author an origin of the code. In the table below the positional arguments are above the dotted line and the keyword arguments are below.

input	type	explanation
first argument	string	name of the instrument
author	string	name of the author
origin	string	origin of the work
mcrun_path	string	path to local mcrun

Below an instance called detector will contain an instrument called "LOKI_detector" while the instance named example has a instrument named test with a specified author.

```
detector = McStasScript.McStas_instr("LOKI_detector")
example = McStasScript.McStas_instr("test",author="Mads Bertelsen")
```

McStas_instr method add_parameter

Adds input parameter to instrument, uses class parameter_variable

input	type	explanation
first argument (optional)	string	variable type
second argument	string	name of the parameter
value	any	default value for the parameter
comment	string	comment that will be displayed with the variable

Here four different parameters are added to the instrument file using the different allowed keywords.

```
detector.add_parameter("wavelength")
detector.add_parameter("height",value=1.0,comment="Height in [m]")
detector.add_parameter("string","reflection_filename")
detector.add_parameter("string","data_filename",value="\"data.dat\"")
```

The two first variables called *wavelength* and *height* are of the default type because no type was given. In McStas the default type is a double. The *height* variable was given a default value and a comment. The *refelction_filename* and *data_filename* are both specified to be strings and the latter was given a default value, note the \" needed to insert the quotation marks into strings.

McStas_instr method add_declare_var

Adds declared variable to the instrument file

input	type	explanation
first argument	string	variable type
second argument	string	name of the parameter
value	any	value for the parameter (can be array)
array	int	length of array
comment	string	comment that will be displayed with the variable

Here four different variables are added to the instrument file using some of the different allowed keywords.

```
detector.add_declare_var("double","energy")
detector.add_declare_var("int","flag")
detector.add_declare_var("double","tube_radius",value=0.013)
detector.add_declare_var("double","displacements",array=7)
detector.add_declare_var("double","t_array",array=4,value=[0.65E-6,0.65E-6,1E-6,1E-6])
```

When declaring an array the array keyword must be used even when setting the values. The values are given as a python array as shown in the last example. The declared variables will appear in the declare section of the instrument file.

McStas_instr method append_initialize

Adds line of code to initialize section

This methods adds a line of text to the initialize section of the McStas file and has no keyword arguments. A similar method called append_initialize_no_new_line exists for adding to the same line with multiple calls.

```
detector.append_initialize("energy=pow(2*PI/wavelength*K2V,2)*VS2E;")
```

McStas_instr method add_component

Method for adding a new component to the instrument file

Components are added using the component class. Only the two first arguments are required, the rest can be modified later. The default values for the location and rotation of a component is (0,0,0) ABSOLUTE. If the keyword arguments *before* or *after* is not used, the added component will be the last in the component sequence.

input	type	explanation
first argument	string	name of the component instance
second argument	string	name of the component to use
AT	float list[3]	position in (x,y,z)
AT_RELATIVE	string	name of earlier component used as reference for position
ROTATED	float list[3]	rotation around (x,y,z)
ROTATED_RELATIVE	string	name of earlier component used as reference for rotation
RELATIVE	string	name of earlier component used as reference
before	string	name of component this component should be before
after	string	name of component this component should be after
comment	string	comment that will be displayed with the variable

A component in McStas needs a name, which is the first argument. The second argument select what component should be used from the component library. Below are some examples of simple use.

```
detector.add_component("Origin","Arm")
detector.add_component("source","Source_simple",RELATIVE="Origin")
detector.add_component("beam_extraction","Guide_gravity",AT=[0,0,2],RELATIVE="source")
```

If one wishes to insert another component between the source and beam_extraction it can be done with the *before* or *after* keyword.

```
detector.add_component("pre_guide_slit","Slit",before="beam_extraction",
    AT=[0,0,1],RELATIVE="source",comment="Slit before the guide")
```

McStas_instr method print_components

Method for printing current list of components to the terminal

To check that the components defined in the documentation so far are in the expected order, the print_components method is demonstrated. Data on the rotation of components is normally included but is omited here. This method has no arguments.

```
detector.print_components()
Origin              Arm             AT  [0, 0, 0]     ABSOLUTE
source              Source_simple   AT  [0, 0, 0]     RELATIVE Origin
pre_guide_slit      Slit            AT  [0, 0, 1]     RELATIVE Origin
beam_extraction     Guide_gravity   AT  [0, 0, 2]     RELATIVE source
```

McStas_instr method set_component_parameter

Method for setting parameters of a component using a dictionary

This methods sets the parameters of a defined component using a python dictionary.

input	type	explanation
first argument	string	name of the component instance to modify
second argument	dict	dictionary with parameter names and values

It is possible to add several parameters in one call, and new calls add further parameters.

```
detector.set_component_parameter("source",{"xwidth" : 0.12, "E0" : "energy"})
detector.set_component_parameter("source",{"yheight" : 0.12})
```

McStas_instr method print_component

Method for printing information contained in defined component

This method takes the name of a component and prints the current information. We can check that the paraameters and position of a component has been registred correctly.

```
detector.print_component("source")

COMPONENT source = Source_simple
  xwidth = 0.12
  E0 = energy
  yheight = 0.12
AT [0, 0, 0] RELATIVE Origin
ROTATED [0, 0, 0] RELATIVE Origin
```

This is not intended for copy-pasting into McStas instruments as the syntax is not correct. Generation of the instrument file is covered later in the documentation.

McStas_instr method set_component_AT

Method for updating position of a component

There are a range of methods for updating information on a component after it has been defined. The syntax is similar to the original call for add_component in all cases.

input	type	explanation
first argument	string	name of component to modify
first argument	float list[3]	position in (x,y,z)
RELATIVE	string	name of earlier component used as reference for position

```
detector.set_component_AT("source",[0.01,0,0])
```

McStas_instr method set_component_ROTATED

Method for updating rotation of a component

input	type	explanation
first argument	string	name of component to modify
first argument	float list[3]	rotation around (x,y,z)
RELATIVE	string	name of earlier component used as reference for rotation

```
detector.set_component_ROTATED("beam_extraction",[0,2.0,0],RELATIVE="Origin")
```

McStas_instr method set_component_RELATIVE

Method for updating RELATIVE reference for both position and rotation

This method will override both positional relative and rotational relative. It has no keyword arguments.

```
detector.set_component_RELATIVE("beam_extraction","pre_guide_slit")
```

After these updates the output from print_components is shown again to see the changes.

```
Origin           Arm            AT   [0, 0, 0]        ABSOLUTE
source           Source_simple  AT   [0.01, 0, 0]     RELATIVE Origin
pre_guide_slit   Slit           AT   [0, 0, 1]        RELATIVE Origin
beam_extraction  Guide_gravity  AT   [0, 0, 2]        RELATIVE pre_guide_slit
```

```
Origin           Arm            ROTATED  [0, 0, 0]      ABSOLUTE
source           Source_simple  ROTATED  [0, 0, 0]      RELATIVE Origin
pre_guide_slit   Slit           ROTATED  [0, 0, 0]      RELATIVE Origin
beam_extraction  Guide_gravity  ROTATED  [0, 2.0, 0]    RELATIVE pre_guide_slit
```

McStas_instr method set_component_comment

Method for updating the comment on a component

It is also possible to add a comment to a component after it was defined.

```
detector.set_component_comment("beam_extraction","Simulating severe misalignment")
detector.print_component("beam_extraction")
// Simulating severe misalignment
COMPONENT beam_extraction = Guide_gravity
AT [0, 0, 2] RELATIVE pre_guide_slit
ROTATED [0, 2.0, 0] RELATIVE pre_guide_slit
```

McStas_instr method write_c_files

Methods for writing c files to folder named generated_includes

This method will write c files describing the declare, initialize and trace sections of the generated instrument.

```
detector.write_c_files()
```

These can then be included in another McStas file. This is useful as this python tool is most often used to generate large repeating part of an instrument that can then be included in a regular instrument file. The instrument file can include them using the %include keyword from McStas as shown below.

```
DECLARE
%{
// include parameters declared from generate_LOKI_parts.py
%include "generated_includes/LOKI_detector_declare.c"
%}

INITIALIZE
%{
// include initialization code from generate_LOKI_parts.py
%include "generated_includes/LOKI_detector_initialize.c"
%}

TRACE
// include components from generate_LOKI_parts.py
%include "generated_includes/LOKI_detector_component_trace.c"
```

McStas_instr method write_full_instrument

Writes the full instrument file with name defined in original McStas_instr call

This method instead writes the entire instrument file using the provided information.

```
detector.write_full_instrument()
```

McStas_instr method run_full_instrument

Runs McStas simulation of defined instrument and returns array of mcstas_data objects

This methods runs the simulation using the mcrun commands of the system and returns the resulting data as a array of mcstas_data objects. An error will occur if the fodldername already exists.

input	type	explanation
foldername	string	name of folder that will be created for data
parameters	dict	dictionary with input parameters and their values
ncount	int	number of rays to simulate
mpi	int	number of mpi threads to use for simulation
custom_flags	string	custom mcstas flags added to mcrun launch command
mcrun_path	string	absolute path to mcrun (if not already in path)

```
data = detector.run_full_instrument(foldername="data1",
                                    parameters={"wavelength":5.1},
                                    ncount=1E7,mpi=2)
```

class mcstas_data

Holds a single McStas data set in either 1D or 2D

A class to handle data from McStas simulations in a transparent way which provides easy access to manipulation of the data. The included data is located in the following variables

variable	type	explanation
Intensity	float array	numpy array containing intensity
Error	float array	numpy array containing error on intensity
Ncount	int array	numpy array containing number of rays in each pixel
dimension	int array	size of data set, length equal to data dimensionality
xaxis	float array	numpy array of xaxis if data is one dimensional
limits	float array	limits on data axis, twice as long as dimensionality
xlabel	string	label of x-axis
ylabel	string	label of y-axis
title	string	title for data

mcstas_data method set_xlabel

Sets the xlabel of a data set

Method for setting xlabel on a data set, similar methods exists for ylabel and title with same syntax.

```
data[0].set_xlabel("custom xlabel [m]")
```

class make_plot

plots single mcstas_data object or an array of these

Class for simple plotting of mcstas_data objects. Will be expanded over time to contain more control over the resulting plots. Currently only the initialization is done so the returned object has no useful methods.

input	type	explanation
first argument	mcstas_data array	data to be plotted
log	int	enables logarithmic scale if non zero
max_orders_of_mag	float	maximum orders of magnitude to plot in log mode

Here all data in the array data is plotted on a logarithmic scale that includes the maximum and a range of at most 10 orders of magnitude.

```
plot = McStasScript.make_plot(data,log=1,max_orders_of_mag=10)
```

class make_sub_plot

plots single mcstas_data object or an array of these as subplots

Class for simple plotting of mcstas_data objects in one window. Will be expanded over time to contain more control over the resulting plots. Currently only the initialization is done so the returned object has no useful methods.

input	type	explanation
first argument	mcstas_data array	data to be plotted
log	int / array	enables logarithmic scale, array to specify for each subplot
max_orders_of_mag	float	maximum orders of magnitude to plot, use array for subplot

Here all data in the array data is plotted on a logarithmic scale that includes the maximum and a range of at most 10 orders of magnitude.

```
plot = McStasScript.make_sub_plot(data,log=[1,0,1],max_orders_of_mag=[10,2,4])
```

2.1 Advanced use

The parts of the api covered by the documentation so far is the simplest way of using the API, but some additional methods in the McStas_instr are usefull for experienced python users that want more direct access to the underlying classes.

McStas_instr method get_component

Returns the component class instance of a selected component

It is possible to get direct access to the component instances inside the McStas_instr instance for direct manipulation. This can make the syntax a bit shorter in some cases.

```
guide_piece = detector.get_component("beam_extraction")
```

McStas_instr method get_last_component

Returns the component class instance of the last component in the component sequence

Same as *get_component* but no argument is needed when returning the last component of the sequence.

```
guide_piece = detector.get_last_component()
```

class component

Holds information on a component and methods for updates and writing to file

Most of the methods contain in the component class are just passed directly to the McStas_instr and thus does not require further explanation, they are however listed here for completeness.

component method set_AT

Equivalent to set_component_AT in McStas_instr

component method set_ROTATED

Equivalent to set_component_ROTATED in McStas_instr

component method set_RELATIVE

Equivalent to set_component_RELATIVE in McStas_instr

component method set_parameters

Equivalent to set_component_parameter in McStas_instr

component method set_comment

Equivalent to set_component_comment in McStas_instr

component method write_component

Writes the component to file

input	type	explanation
first argument	file identifier	file identifier ready for writing

component method print_long

Prints information on the component to the terminal

3 Discussion

This section contains discussion on the python module.

3.1 Possible improvements / requests

Features that are still missing and should be added. Also keeps track of user requests.

3.1.1 EXTEND

It should be possible to add an extend section to a component.

3.1.2 WHEN

It should be possible to add the WHEN keyword to a component

3.1.3 JUMP

It should be possible to add the JUMP keyword to a component.

3.1.4 GROUP

It should be possible to add the GROUP keyword to a component.

3.1.5 Add code to trace

It should be possible to add code directly to trace, for example include statements. The position of this code should be relative to components.

3.1.6 FINALLY

Should be possible to add code to the FINALLY section which is ignored so far.

3.1.7 Limits on parameters

Allow user to easily set limits on parameters and generate appropriate input sanitation for instrument file with error message.

3.1.8 Printing information on parameters and variables

Methods for printing information on all parameters and variables

3.1.9 Methods for removing parameters / variables / components

When using the software from a terminal it could be useful to remove components. Might also be useful to be able to move a component to another location in the sequence.

3.2 Jupyter notebook experience

It is entirely possible to write an instrument file from a jupyter notebook using this tool, but at this point it behaves more like a script, and thus there is no inherrent benefit. The main issue is that rerunning a cell will cause errors because the same components are added again, and they recognize the names are not unique. Should instead allow to update components when the same name is used, but this adds a severe risk of users replacing an earlier component instead of creating a new.

Another issue is the lack of feedback beyond printing all added components. A simple improvement would be to have a method that prints all changes since last print was executed, which would be a natural end of each cell.