McStasScript developer reference

Mads Bertelsen

January 21, 2020

1 Introduction

This document serves as developer documentation for the McStasScript API for Python. There is separate user documentation (**link**) that covers installation and use. The purpose of McStasScript 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 control structures like `for`-loops and that it can be used directly from a Python terminal or a Jupyter notebook. The instrument simulation can be executed from the scripting language and the data can be manipulated before plotting. It is possible to convert existing McStas instruments to a Python version using an included converter. The code is open source and on GitHub: `https://github.com/PaNOSC-ViNYL/McStasScript`.

2 Dependencies

McStasScript requires Python 3.0 or newer and the following packages: numpy, matplotlib, PyYAML.

A local installation of McStas is required (2.0 or later), and McStasScript needs to be configured to find the components and mcrun executable. The configuration only has to be done once, and is explained further in the user documentation (**link**).

3 Platforms

McStasScript uses the Python module **subprocess** to perform McStas simulations through a system call. The syntax is currently Unix-like, so McStasScript will work on OS X and Unix, but probably not on Windows.

4 Distribution

The package is on PyPi and is manually updated. The package is owned by the user mbertelsen. The setup.py file is included in the GitHub repository. New versions are uploaded using these commands:

```
python3 setup.py sdist bdist_wheel
python3 -m twine upload dist/*
```

The package is installed through pip with the following command:

```
python3 -m pip install McStasScript --upgrade
```

5 Class diagram

The class diagram of the software is shown in figure 1. The package relies on a large interface class called McStas_instr that attempts to facilitate everything a user would normally do with a McStas instrument file. This section provides a small overview of the classes and how they fit together.

5.1 mcstas_objects

The *parameter_variable* and *declare_variable* classes describe a simple instrument parameter and declare variable, respectively. They both keep the type of the variable, the name and similar information. The *component* class describes a component without any component parameters, which is used as a parent for dynamically created component classes.

5.2 instr

The main interface class is called *McStas_instr* and describes an instrument file, including the methods to perform the corresponding simulation. It includes lists of instances for *parameter_variable*, *declare_variable* and the dynamically generated components. This class also writes the overall instrument file to disk using the methods of the mcstas_object classes.

5.3 data

Data from McStas simulations is loaded into a container class called *McStasData*, which consists of the actual data arrays and instances of *McStasMetaData / McStasPlotOptions*. The *McStasMetaData* class contains metadata such as information on axis, units and similar, while *McStasPlotOptions* contain preferences on how this data should be plotted. One instance of *McStasData* holds one 1D or 2D dataset.

5.4 plotter

The plotter classes takes *McStasData* instances and plots the contents using the settings from *McStasPlotOptions* included therein. The *make_sub_plot* is most commonly used, as it shows an array of *McstasData* in one figure. *make_plot* can also handle arrays, but will make a figure for each dataset. The *make_animation* class can create an animation from an array of *McStasData* and save as a gif.

5.5 component_reader

The *ComponentReader* class handles reading McStas component files from the local McStas installation, gathering information about their input parameters, units and similar. This information is stored in a *ComponentInfo* instance. *McStas_instr* creates the dynamic component classes from *component* and an instance of *ComponentInfo*. Each dynamic class is only created once, and kept in a dictionary to avoid duplication.

5.6 managed_mcrun

The *ManagedMcrun* class handles executing McStas simulations and loading the resulting data into *McStasData* objects.

5.7 instr_reader

These classes are responsible for reading existing McStas instrument files, and either translating these into *McStas_instr* instances or writing Python files that when executed produces these *McStas_instr* objects. This helps migrate projects to McStasScript from traditional instrument files, but the feature is still not in a finished state. The interface is through the *reader* class.

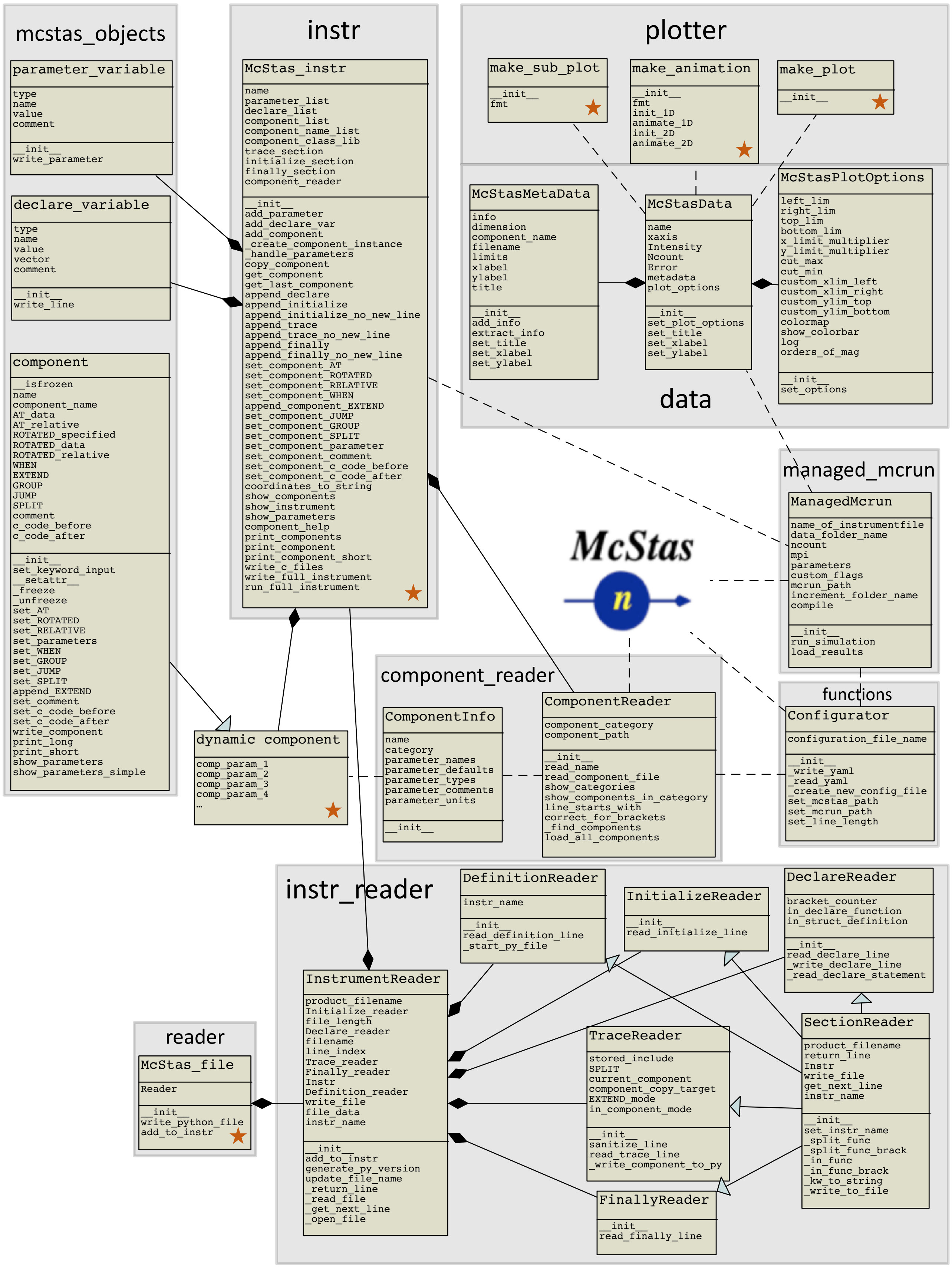

Figure 1: Map of classes in McStasScript and their relationships. The boxes indicate the file or folder that includes the class definition. Black filled arrows shows aggregation, gray filled arrows shows inheritance and dashed line shows dependency. The McStas logo shows where McStasScript depends on McStas. The dynamic component classes are generated at run time based on user demand. The user interface is through the classes marked with a star.

6 Documentation

The main documentation is provided as a pdf file, but the code is also heavily commented with docstrings. The focus is on docstrings for interface classes, but internal classes are documented in the same style. The following is an example of the `declare_variable` class:

```
|  declare_variable(*args, **kwargs)
|
|  Class describing a declared variable in McStas instrument
|
|  McStas parameters are declared in declare section with c syntax.
|  This class is initialized with type, name.  Using keyword
|  arguments, the variable can become an array and have its initial
|  value set.
|
|  Attributes
|  ----------
|  type : str
|      McStas type to declare: Double, Int, String
|
|  name : str
|      Name of variable
|
|  value : any
|      Initial value of variable, converted to string
|
|  comment : str
|      Comment displayed next to the declaration, could contain units
|
|  vector : int
|      0 if a single value is given, otherwise contains the length
|
|  Methods
|  -------
|  write_line(fo)
|      Writes a line to text file fo declaring the parameter in c
```

7 The two use modes of components

The software has two main ways of interacting with the dynamically generated component objects. One way hides the object oriented nature, while the other exposes it. This may be confusing for users, and a decision needs to be taken about keeping both or just one.

7.1 Through returned objects

It is possible to interact with the components through the objects returned by `McStas_instr.add_component`:

```
source = ODIN.add_component("source", "Source_simple")
source.xwidth = 0.1
source.yheight = 0.1
source.set_AT([0, 0, 0], RELATIVE="Origin")
source.set_GROUP("sources")
```

Since the dynamic component objects have attributes corresponding to the parameter names, the parameter names can be auto-completed in many editors. The name of the object can be different from the McStas component instances, this may be confusing to some.

7.2 Through McStas_instr

It is also possible to interact with these objects through the `McStas_instr.set_component_*` methods:

```
ODIN.add_component("source", "Source_simple")
ODIN.set_component_parameter("source", {"xwidth" : 0.1, "yheight" : 0.1})
ODIN.set_component_AT("source", [0, 0, 0])
ODIN.set_component_GROUP("source", "sources")
```

The interface through the McStas_instr does not require much knowledge of objects, but are a bit more prone to error. The parameters can only be set through a dictionary, this is also possible in the object version.

8 Testing

The majority of functionality contained in McStasScript is tested through unit tests or integration tests. The coverage of classes can be seen in table 1. A notable exception in test coverage is the plotting functionality. Simple integration tests are also available to test that McStas components can be loaded, a simulation can be performed and that data can be loaded. The integration tests requires the configuration to be performed and a local McStas installation.

interface			
instr.py	**McStas_instr**		
plotter.py	Make_plot	Make_sub_plot	Make_animation
functions.py	***name_search***	***name_plot_options***	**Configurator**
reader.py	**McStas_file**		
data			
data.py	**McStasData**	**McStasMetaData**	**McStasPlotOptions**
helper			
mcstas_objects.py	**parameter_variable**	**declare_variable**	**component**
component_reader.py	ComponentInfo	**ComponentReader**	
managed_mcrun.py	**ManagedMcrun**		
formatting.py	bcolors	***is_legal_parameter***	***is_legal_filename***
instr_reader			
control.py	**InstrumentReader**		
read_definition.py	**DefinitionReader**		
read_declare.py	**DeclareReader**		
read_initalize.py	**InitializeReader**		
read_trace.py	**TraceReader**		
read_finally.py	FinallyReader		
util.py	SectionReader		

Table 1: Files, classes and functions contained in McStasScript. The left side shows the folders and files included. The right side shows classes and functions. Italics signifies functions, and normal font is a class. Bold functions/classes have unit tests, while the remaining do not.