2020-04-09: Parameters class and YAML File Handling

Yannick opened the meeting declaring that this is a focused topic meeting and the topic of the day is the new oops:Parameters class developed by Wojciech and related issues regarding how YAML/configuration files are handed in JEDI.

Yannick also announced that there will be **no JEDI meeting next week** because the JEDI team will be participating in the JCSDA Quarterly Review meeting.

Then Wojceich presented the slides below. What follows is a brief description; consult the slides for further details.

parameters-and-yaml.pdf

Wojceich started by outlining the two main topics of his presentation. In addition to describing the oops:Parameters class, he also wanted to bring up the possibility of using JSON schema in JEDI as a tool for the verification of configuration files.

Wojceich then proceed to describe the main motivation for introducing the Parameters classes. In particular, it provides a way to encapsulate collections of parameters for each class so users have a single place to look to find out what options they need to specify for each corresponding section in the yaml file. Furthermore, it encapsulates the documentation together with the code, adding clarity and helping to ensure that the documentation remains up to date.

Slide 5 lists the four different Parameter classes that are currently available and the subsequent slides give examples on how to use them. As shown on slide 6, oops:Parameters can be used as a base class to define subclasses for specific sets of parameters. Each of these subclasses can define parameters using the RequiredParameter, Parameter, and OptionalParameter classes as appropriate.

Slide 6 illustrates how to use such a subclass. Note in particular the priorityVariable member variable of the ThinningParameters subclass. This is an optional parameter and returns an object of type boost::optional<Variable>. The boost::optional<> class allows for optional parameters to be undefined. If the variable is defined, then this object holds its value but if it has not been defined then it holds a value of boost::none.

Wojeich then addressed the use case of a parameter hierarchy and how this can be handled by defining a vector of oops:Parameters objects. He then showed how the get() method of the ParameterTraits class template is used to extract the parameter from an eckit configuration object and how this can be customized for different types of objects.

Wojciech mentioned that documentation for the new Parameters classes has been added to the JCSDA ReadTheDocs website.

After finishing this first part of his presentation, Wojeich paused for questions.

Chris H asked how the Parameters classes could be used to extract configuration objects for use in fortran. Wojeich responded that there is currently no Fortran interface for the Paramters classes. So, a possible approach might be to read the variables into C++ and do error checking there before passing them to Fortran (possibly as individual variables or as components of an eckit::Configuration object).

Mark M asked how external documentation might be generated from the Parameter classes and whether they are Printable (as suggested, e.g. by bottom of slide 7). Wojeich answered that the base classes are not Printable (though some components may be) but the parameter descriptions in the files should be expressed as doxygen-formatted comments (for example, the triple slashes /// on slide 6) so they will be included in doxygen output (e.g. html or pdf).

Yannick said it would be nice to have the parameters class write out a yaml file as a template showing all available options. Travis mentioned that MOM6 does something similar to this, though not with yaml files. Here are some further comments from Travis on that point:

> *In MOM6 all the parameters are defined in the code (like our Parameters class), and additionally each parameter has in the code a string with the description. At run time the model dumps out the complete configuration file including parameters that weren't given as input but instead left at their default values. This has been very useful in the past for me with the MOM6 model because it provides 1) a good auto generated documentation of the parameters 2) a good template to start from when getting familiar with the system 3) a record of the exact parameters that were used for an experiment, so even if default values change in future versions of the code I have a record of the exact parameters used to more easily recreate an experiment*

Dave S asked how json schema might be incorporated into this and Wojceich proceeded with part 2 of his presentation, which addressed this exact topic.

JSON schemas are not implemented in JEDI currently. Wojceich presented the topic for consideration as a possible code enhancement. They can be used to define a structure to validate configuration files. To use them in JEDI, we would have to define schema and then make use of one of several yaml to json converters.

Wojeich then listed several tools that can be used to define json schema and edit yaml (or json files) based on those schema (slide 19). And, he gave a demonstration of one one these tools, namely the yaml plugin for Visual Studio Code. He showed how the editor can verify your yaml format as you type, making sure it follows the schema and the basic yaml syntax. It also include auto-completion and auto-correction features.

When the presentation was finished, Yannick commented that these schema could be a useful tool. However, he added that in the future, many JEDI users will not be editing yaml files manually as they do now. Instead, the yaml files will be generated automatically when they run the jedi-rapids workflow. So the question becomes which level in the workflow should we apply such schema?

Mark O agreed that json schema could be very useful for the workflow. Currently yaml files are common sources of error and take a substantial amount of time to debug (both person-time and computer time). Having an automated validation tool like this could be very beneficial.

David S mentioned that they have similar validation tools in place at MetoFrance. But, a problem is that the metadata can get separated from the code so it can be a challenge to keep them consistent. He suggested that the code should be taken as the "truth" and metadata such as schema should be generated automatically from the code. Yannick asked if the Parameters class can be used to generate schema. Wojeich answered that this might be possible.

Wojeich said that a challenge is what to do when you have multiple possibilities. For example, different obs or model types might have different yaml configurations that would require different schema. He thought that it might be possible to handle this somehow through polymorphism. Yannick added that we do not want to be too rigid in our schema definitions - we want some run-time flexibility in, for example, the specification of QC filters through the config files.

The meeting ended with a reminder from Yannick to please suggest other focused topics that you would like to see addressed in the future, either by contacting the JEDI core team or by entering your request directly onto the ZenHub board.