

2020-07-02: ioda-engines

Yannick opened the meeting by announcing the focused topic, namely the new refactoring of ioda IO interfaces we're calling ioda-engines. This is work that has been led by Steve H and Ryan H. Yannick also remarked on the number of meeting participants, which later reached as high as 60. This is a topic of much interest.

Steve then presented the following slides. Please note that the presentation is focused on what the ioda-converter will need to use, and that the ioda-engines interface is more extensive than what is shown in these slides.



What follows is a brief description; see slides for further details. Steve started by explaining that this is work by Ryan and himself that builds on hdf5 tools previously developed by Ryan. Steve proceeded to review the motivation and data model. Much of the new functionality centers on the new ObsGroup class, which organizes variables and attributes (metadata) hierarchically as in hdf5, and represents metadata as attributes. Steve mentioned that attributes can be dimensional and Greg asked for an example of this. Steve responded with an example in which the attributes specify a valid range of values that can be used with QC.

Each ObsGroup object links a standard front-end API to a specific back end that determines how the data is stored, either in a file (HDF5/Netcdf, ODC /ODB) or in-memory (ObsStore). The front-end interface is powered by the various back ends. The ObsGroup methods are deliberately similar to the ObsSpace methods to help maintain compatibility and familiarity. These include, for example, getDb() and putDb() methods as well as a new appendDb() method that can be used to move data into the data store in a sequence of discrete chunks.

During the discussion of the example (slide 12), Steve showed how the setDimensionScale() method for ObsGroup can be used to set the data dimensions using the number of locations (nlocs) and the number of channels (nchans) as an example. Jianjun asked if these can be obtained from the file. Steve responded that this is generally done in the converter as described in slide 11. Jianjin also asked about obtaining more information from the files for use with QC. Steve H responded that this can indeed be done. The example shown here is highly simplified. For an actual instantiation of the converter and ObsGroup, you would in general have multiple variables and you would define more data and metadata for the ObsValues themselves but also other fields such as error estimates and pre-QC.

Wojciech asked for clarification on how the ObsGroup class relates to the ObsSpace class. Steve explained that ObsSpace does much more than ObsGroup, including the selection and parallel distribution of observations. In practice, an ObsSpace object would instantiate one or more ObsGroup objects to manage the transfer of data between different back ends. For example, you may have different ObsGroups for input and output if the file formats are different. Wojciech then asked if the ObsSpace API will remain the same. Steve responded that there may be changes but they will be incremental.

Wojciech then asked for a time estimate of when the capability to retrieve and put multi-dimensional data structures will be ready. Steve H responded that that interface will continually improve and the target is to have it ready by the first JEDI release near the end of the summer.

Breogan asked if the multi-dimensional capability can be used to organize data in terms of latitude and longitude. Steve responded that we typically handle locations as an indexed list and then specify the position (latitude, longitude, Pressure/height) as metadata for each index. But, there is a possibility of re-considering that for specific use cases using this new interface.

David S asked about how data is handled after being read in from, e.g. an ODB file where some metadata has been removed. Will it be possible to reconstruct the original data or will it remain flat? For example, reconstructing vertical profiles based on satellite id or station. Steve responded that there is a capability in ObsSpace to group observations in ways that can be specified in the yaml file, for example by station. But, this grouping is not preserved when the data is written to a file. If this is insufficient for particular use cases, then we can talk about how we might preserve more of the metadata in the future.

Steve V commented that a UML class diagram would help users understand the relationship between the ObsGroup class, the ObsSpace class, and other classes. Steve H responded that the ioda-engines code has good Doxygen documentation so these UML diagrams can be generated by running Doxygen.

Mark M asked why ioda-engines is a separate repository from ioda and ioda-converter and will it remain that way? Ryan responded that this was the easiest approach from a development perspective. Sometime before the release, in the next few months, ioda-engines will be merged into ioda.

Guillaume asked how these changes will affect their current converters and when the changes need to be made. Steve said that it will be at least 2-3 weeks before any changes are required but it's a good idea to take a look at the ioda-engines repo now in preparation. He suggested that we hold a code sprint in about a month (late July/early August) to adapt the current converters to the new interfaces.

Rahul posed a hypothetical question: If I were to write a function to read a particular file type, would it be possible to hook this into the ObsGroup directly? Guillaume rephrased this a little by asking whether the step of writing ioda files can be bypassed - will ObsGroup be able to transfer data directly to and from ObsSpace. Steve answered yes - the in-memory ObsStore is one of the back ends supported by ObsGroup so this will now be possible with the new framework.

Rahul also asked if there is a strategy for optimizing parallel IO from large files. Jianjun agreed this is important and asked how the parallel distribution of observations is handled now. Steve responded that certain file types, like netcdf4, have this parallel functionality built-in. For other file types this still has to be implemented. If there are a large number of files, one strategy may be to have different MPI tasks each grab a converter and read in a different file, that will then be re-distributed among processors after the read. Currently each MPI task has its own ObsSpace object for each Obs type that contains a subset of the observations. Steve added that further developing and optimizing the parallel data handling in ioda is something that he plans to work on extensively in the coming year.

Emily asked whether the ioda-engines restructuring will require changes in the way the radiance data and filters are implemented and when these changes need to be implemented. Steve confirmed that some changes will be needed and again confirmed that the migration to the new system will occur at the end of July or beginning of August. But, he encouraged people to take a look at ioda-engines now and start to familiarize yourself with it. It should look similar to the current ObsSpace implementation. Emily asked to be included in the code sprint. Steve added that the EMC-JCSDA JEDI Junction and JEDI 2 team meetings are good places to discuss this further and remain up to date.