Automating Metadata Compliance Checking

Improving Earth Science Metadata Consistency

Oliver Chang, University of Miami, Miami, Florida
o.chang@umiami.edu
George Chang, Jet Propulsion Laboratory, Pasadena, California
george.w.chang@jpl.nasa.gov
Ed Armstrong, Jet Propulsion Laboratory, Pasadena, California
edward.m.armstrong@jpl.nasa.gov

August 18, 2014

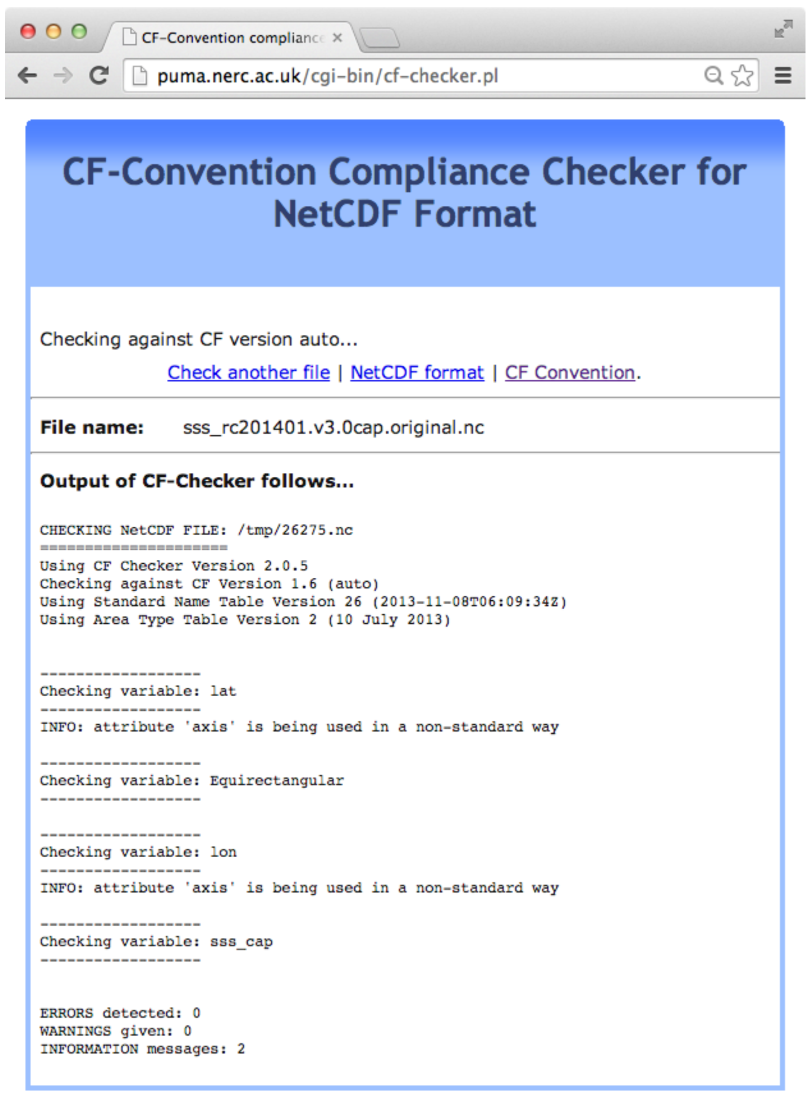
Abstract

Metadata are extra pieces of information that accompany data to help make the data more self-describing and eliminate ambiguity in how the data might be processed by both human operators and machines. However, because they are extra bits of information, metadata often vary dramatically in quantity and quality between different providers of data. Metadata standards such as the Attribute Conventions for Dataset Discovery[1] (ACDD) and Climate and Forecast[2] (CF) Metadata Conventions exist to describe uniform methods and vocabularies for metadata to adhere to. To check metadata against these standards by hand is tedious; the CF standard without appendices is over 100 pages. We have developed an automated tool that automatically checks metadata attributes for presence and correctness, and generates human-readable reports that reference the metadata alongside the standard. We have targeted the ACDD, CF, and GDS2 metadata standards for both ease of implementation and practical use by engineers at the Physical Oceanography Distributed Active Archive Center[3] (PO.DAAC). In keeping with the goal of ease-of-use, the tool is implemented as a web application with a public API that returns JavaScript Object Notation (JSON) so that new tools can be built that easily leverage the metadata checker.

Background

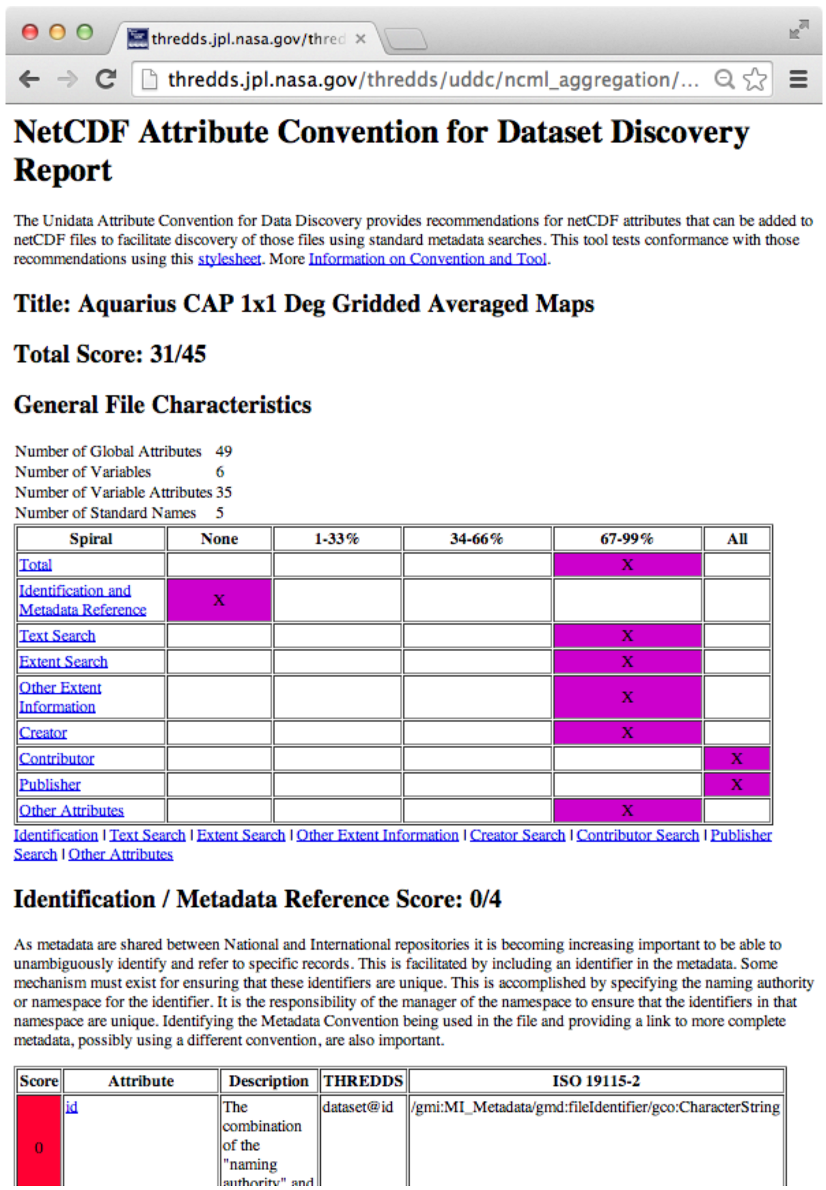
At the Physical Oceanography DAAC, data generally comes in as netCDF and HDF container files that handle Sea Surface Topography, Salinity, Gravity, Temperature, and other measurements at various levels of processing. These files can vary from dozens of megabytes to gigabytes in size each while the quantity of the files as well as their size increase over time, especially with the upcoming launch of missions such as GRACE Follow-On (GRACE-FO) and the proposed launch of the Surface Water and Ocean Topography (SWOT).

To keep the data for effective archival and useful research, there needs to be a system for cataloging the attributes of the data in a consistent manner. NetCDF and HDF contain hierarchical structures solely for metadata that can be extracted and parsed in an XML format. This XML document can then be verified against a series of checker tests that confirm conformance to certain sets of overall guidelines like ACDD and CF, while also conforming to more precise guidelines like the Group for High Resolution (GHRSST) Data Specification[4] (GDS2) for GHRSST data.



Automated metadata validation is not a new idea. If we investigate the checkers that came before, we eventually develop a checklist of what we do and do not want in a checker. The first one we will look at is a script that is available as a web interface[5] that checks for compliance to the CF Standards. What this tool offers is ease of use: simply point your browser to a URL, upload a file, and wait. However, it does not give a lot of transparency into what checks are being performed. In other words, it is very opaque since it does not show what checks pass, only those that fail. In addition, the checker is overly permissive; it does not catch some legitimate errors or even bother to show a warning. Finally, the entire presentation is more or less flat, with little of the hierarchy inherent in either the data or the CF metadata specification.

The second tool is the UDDC plugin to the THREDDS data server[6]. This tool is another single-checker tool, in this case checking dataset conformance to the ACDD standard. Also a web tool, it offers several important features: summary scoring, near instant test time, attribute descriptions in a tabular format, and grouping of attributes into "spirals," overarching categories that say what the attribute describes. This facet is important because it acknowledges that not all the metadata is important in all instances. Instead, it gives the option to easily prioritize the quality of certain attributes. But this checker lacks customizability. Most obviously, you can only perform a check against ACDD, which is mostly focused on global attributes, not variable attributes, which diminishes its coverage dramatically. In addition, the tool infers some data from the actual data, not the headers, which can lead to misleading results for some global attributes that might not exist, e.g. geospatial_lon_units and geospatial_lon_resolution could be non-existent in the metadata, but since



NetCDF Attribute Convention for Dataset Discovery Report

The Unidata Attribute Convention for Data Discovery provides recommendations for netCDF attributes that can be added to netCDF files to facilitate discovery of those files using standard metadata searches. This tool tests conformance with those recommendations using this stylesheet. More Information on Convention and Tool.

Title: Aquarius CAP 1x1 Deg Gridded Averaged Maps

Total Score: 31/45

General File Characteristics

Number of Global Attributes	49
Number of Variables	6
Number of Variable Attributes	35
Number of Standard Names	5

Spiral	None	1-33%	34-66%	67-99%	All
Total				X	
Identification and Metadata Reference	X				
Text Search				X	
Extent Search				X	
Other Extent Information				X	
Creator				X	
Contributor					X
Publisher					X
Other Attributes				X	

Identification | Text Search | Extent Search | Other Extent Information | Creator Search | Contributor Search | Publisher Search | Other Attributes

Identification / Metadata Reference Score: 0/4

As metadata are shared between National and International repositories it is becoming increasing important to be able to unambiguously identify and refer to specific records. This is facilitated by including an identifier in the metadata. Some mechanism must exist for ensuring that these identifiers are unique. This is accomplished by specifying the naming authority or namespace for the identifier. It is the responsibility of the manager of the namespace to ensure that the identifiers in that namespace are unique. Identifying the Metadata Convention being used in the file and providing a link to more complete metadata, possibly using a different convention, are also important.

Score	Attribute	Description	THREDDS	ISO 19115-2
0	id	The combination of the "naming authority" and	dataset@id	/gmi:MI_Metadata/gmd:fileIdentifier/gco:CharacterString

UDDC can infer them from the data the check would pass; it would be ideal to customize this behavior.

The final single check tool we will consider a GDS2 validator[7] that checks the global attributes, variables, and variable attributes for conformance to the GDS2 standard with regards to name, attribute existence, and data type. It is implemented as a set of Python scripts with simple, human-readable configuration files that make the checker a good base to build similar checks on top of. It also implements different levels of priority (Required, Possibly Required, and Optional). However, its simplicity means that it cannot check some of the more complicated aspects of the GDS2 standard like possible values from a vocabulary. Plus its pure terminal output makes it hard to get a summary score while the deluge of possibly required values makes it hard to decipher what is truly necessary and what is not.

```
Terminal
$ ./ghrsst_format_check.py -f ~/podaac/datasets/netcdf/20140508-MODIS_A-JPL-L2P-A
2014128024500.L2_LAC_GHRSST_N-v01.nc
 ---- Validate metadata and structure of a GHRSST GDS v2 file ----
                          ver 1.1


 Checking global attributes . . .
        Notice: Global attribute name GDS_version_id not recognized
        Notice: Global attribute name DSD_entry_id not recognized
        Notice: Global attribute name stop_date not recognized
        Notice: Global attribute name creation_date not recognized
        Notice: Global attribute name file_quality_index not recognized
        Notice: Global attribute name contact not recognized
        Notice: Global attribute name start_date not recognized

        Fatal: Required attribute  geospatial_lat_units was not found
        Fatal: Required attribute  geospatial_lon_units was not found
        Fatal: Required attribute  Metadata_Conventions was not found
        Fatal: Required attribute  keywords was not found
        Fatal: Required attribute  publisher_name was not found
        Fatal: Required attribute  id was not found
        Fatal: Required attribute  naming_authority was not found
        Fatal: Required attribute  uuid was not found
        Fatal: Required attribute  source was not found
        Fatal: Required attribute  standard_name_vocabulary was not found
        Fatal: Required attribute  creator_email was not found
        Fatal: Required attribute  publisher_url was not found
        Fatal: Required attribute  processing_level was not found
        Fatal: Required attribute  gds_version_id was not found
        Fatal: Required attribute  publisher_email was not found
        Fatal: Required attribute  keywords_vocabulary was not found
        Fatal: Required attribute  geospatial_lat_resolution was not found
        Fatal: Required attribute  time_coverage_start was not found
        Fatal: Required attribute  metadata_link was not found
        Fatal: Required attribute  date_created was not found
        Fatal: Required attribute  acknowledgment was not found
        Fatal: Required attribute  geospatial_lon_resolution was not found
        Fatal: Required attribute  license was not found
        Fatal: Required attribute  creator_name was not found
        Fatal: Required attribute  time_coverage_end was not found
        Fatal: Required attribute  summary was not found
        Fatal: Required attribute  project was not found
        Fatal: Required attribute  cdm_data_type was not found
        Fatal: Required attribute  file_quality_level was not found
        Fatal: Required attribute  creator_url was not found
         Review errors above!
```

Finally, we will look at the most fully featured automatic metadata checker, an IOOS tool that checks ACDD, CF, and IOOS Asset Concept values in one package[8]. Besides being the most fully featured, it contains a very robust and detailed suite of checks, the most detailed of any tool. It implements summary scoring and prioritization like the UDDC and GDS2 tools. It also runs locally, which means that it runs much faster than a web tool would. The most onerous thing about the compliance checker is its burdensome graph of dependencies, ranging from bespoke XML tools to the udunits2 Python bindings which have poor cross-platform compatibility[9].

```
Terminal
$ ./cchecker.py -t=cf ~/podaac/datasets/netcdf/zos_AVISO_L4_199210-201012.nc
Running Compliance Checker on the dataset from: /Users/ochang/podaac/datasets/netcdf/zos_AVISO_L4_199
210-201012.nc


--------------------------------------------------------------------------------
                     The dataset scored 60 out of 69 points
                              during the cf check
--------------------------------------------------------------------------------
                               Scoring Breakdown:


                                 High Priority
--------------------------------------------------------------------------------
    Name                             :Priority: Score
Variable names                           :3:     7/7
axis                                     :3:     9/9
convention_attrs                         :3:     2/2
conventions                              :3:     0/1
data_types                               :3:     7/7
dimension_names                          :3:     7/7
latitude                                 :3:     4/4
longitude                                :3:     4/4
std_name                                 :3:     4/4
time                                     :3:     4/4
units                                    :3:     4/4


                                Medium Priority
--------------------------------------------------------------------------------
    Name                             :Priority: Score
all_features_are_same_type               :2:     0/0
contiguous_ragged_array                  :2:     0/0
coordinate_type                          :2:     3/3
coordinates_and_metadata                 :2:     0/0
feature_type                             :2:     0/0
incomplete_multidim_array                :2:     0/0
indexed_ragged_array                     :2:     0/0
missing_data                             :2:     0/0
orthogonal_multidim_array                :2:     0/0
var                                      :2:     5/13


--------------------------------------------------------------------------------
                  Reasoning for the failed tests given below:


Name                                 Priority:     Score:Reasoning
--------------------------------------------------------------------------------
conventions                             :3:     0/ 1 : Conventions field is not
                                                       "CF-1.6"
var                                     :2:     5/13 :
    lat                                 :2:     1/ 2 :
        check_independent_axis_dimensio:2:     0/ 1 : The lat dimension for the
                                                       variable lat does not have
                                                       an associated coordinate
```

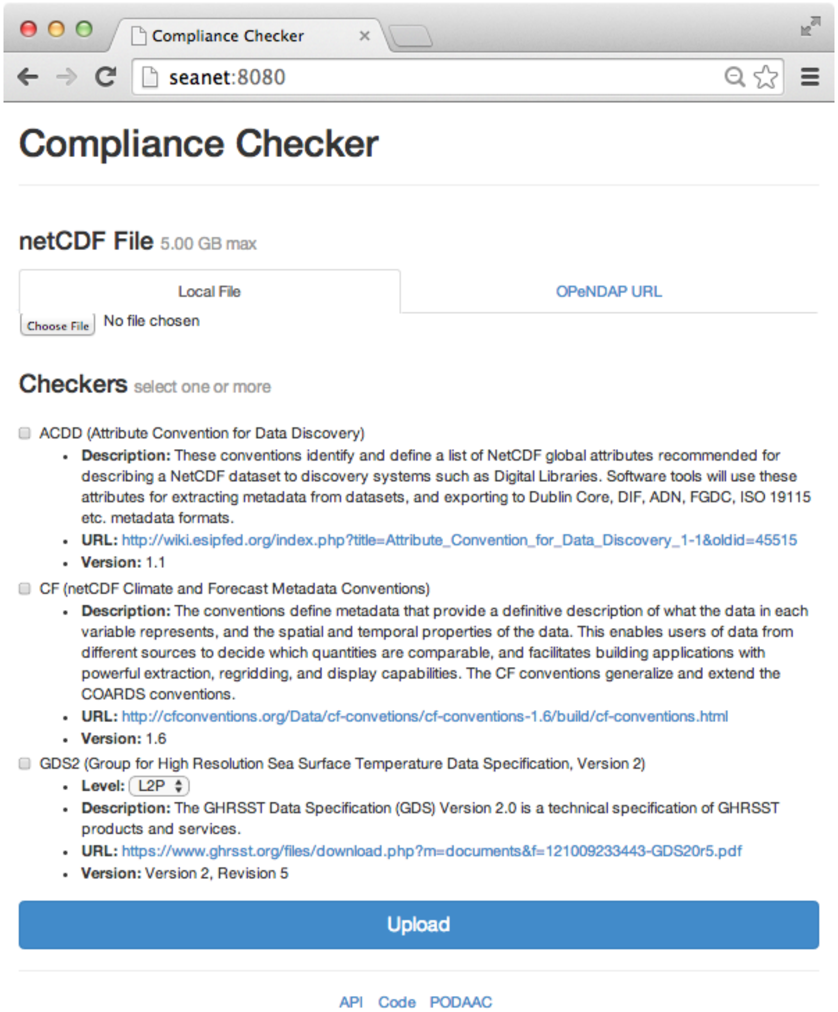
Design

Off the bat, the checker was built to integrate the functionality of the other checkers and to easily communicate results with users. The most natural end goal in that regard was to create a website that would allow users to check files without having to deal with any sort of setup. In addition, we wanted to build off of the solid foundation created by the IOOS checker and create more useful and suggestive test results than were present, further strengthening the case for a website and the possibility for textually dense output.

We built off of the IOOS compliance checker tool simply because it was the most open, most fully featured, easiest to integrate with, and most actively developed. However, some issues with the ACDD checking module dependencies, plus the need to restructure to create a web interface, led to the need to branch off and create a new system, easy in Git. (The IOOS compliance checker is implemented as a CF-only tool as a Git subtree in a separate repository from the original one.) By rewriting the ACDD portion with a new, simpler format, we can add more textual descriptions to the checker results as well as get a feel for the workflow necessary to implement a new checker suite, which was necessary to integrate the GDS2 checker. In designing the new checker, we aimed for an architecture that could deal with the variable structure of datasets and the unpredictable nature of what a check might entail in scope, e.g. filename checking, all variables, some variables, etc.

The ACDD and GDS2 checkers are each implemented as a part of a three-part architecture: one part consists of base abstract classes that create a natural hierarchy (base.py), one part checker classes that do the check computation (checkers.py), and one part the suite data necessary to perform the check (acdd.py, gds2.py). This separation of data from more involved checker code is designed for maximum reusability; a checker class can be used for multiple suites of checks.

The CF checker component is a special case. Given the scope of the project, there was not enough time to re-implement it in the more descriptive format. So instead, there exists a shim that delegates the computation to the IOOS compliance checker (which implements CF1.6)[10] and simply parses the results of that tool into a format that is compatible with the format expected by the functions that consume the checker results. In this design, the CF compliance checker is treated as a black box; we feed it a dataset to test, it runs, and we parse the results.



This image to the left is the home page for the compliance checker web interface. It takes much of its look and feel from the Bootstrap[11] library of reusable CSS and JS elements. There are two methods of loading a file in: either a local file or a remote URL for a file. From there, the next step is to select one or more checkers from the currently available checkers. Finally, once the parameters have been setup there is a submission to the server, checker execution, and the Results page (Appendix A). The results page starts with the filename, size, data model, and MD5 hash for quality checks to make sure the uploaded file is consistent. It then shows each checker's result separately with summary scoring, color-coding, and a strong hierarchical layout that is further encouraged with the availability of collapsible menus. Stylistically, the Results page is designed to be responsive to different screen sizes and to shrink down whitespace plus expand all elements if the page is printed. Of particular note is the "description" field, which contains useful text from the standard itself.

Technical Methodology

The checker is coded in Python and uses the official netCDF4 python package to interface with the netCDF C libraries. The checker was designed to be extensible and reasonably efficient but mostly to communicate results clearly. Thus, we end up using a lot of data structures, namely Python dicts.

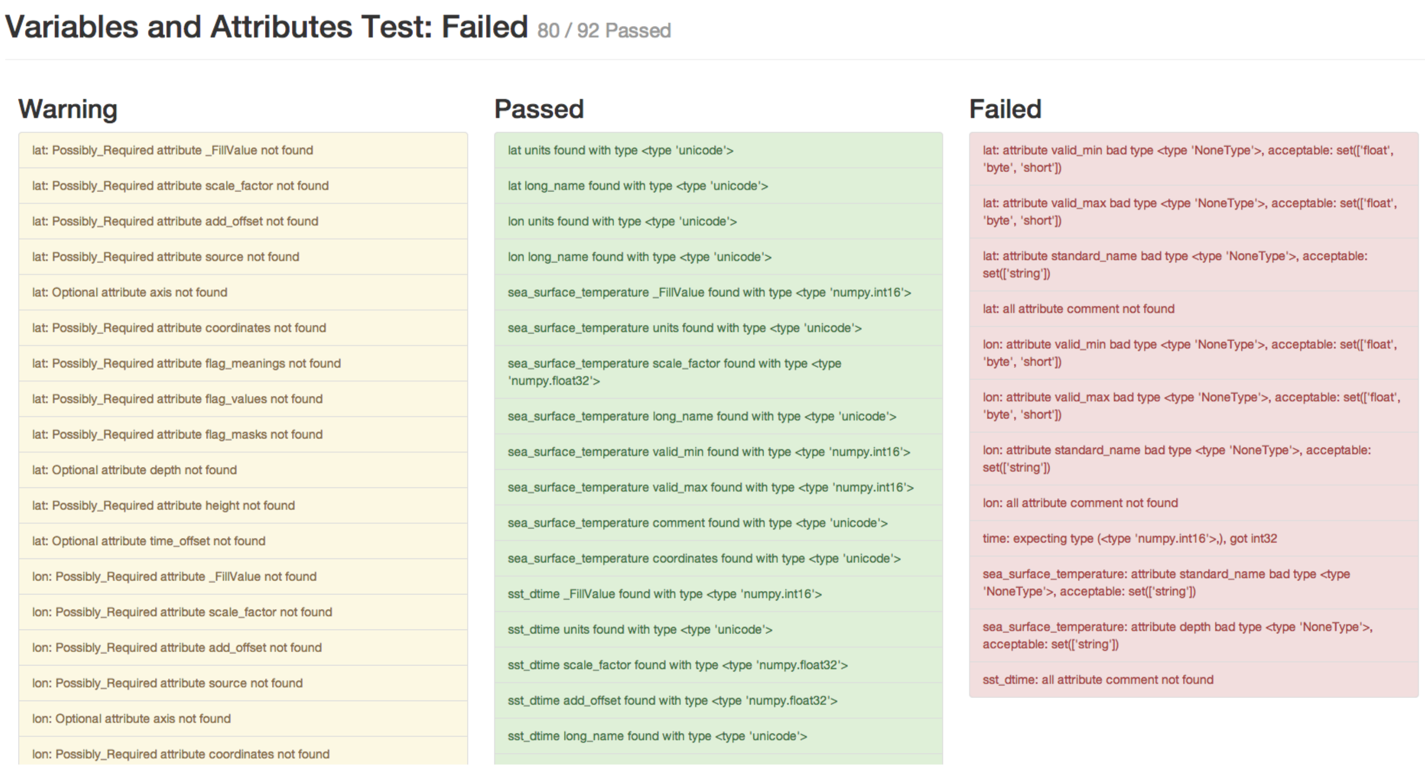
Module Overview

The checker is implemented as three major Python modules.

1. Checker (checker/)

In this module are all the major elements that were coded from the ground-up. It contains the base classes Group, CheckSuite, Blueprint, Checker, and Result. In a check suite, e.g. ACDD, each of these will be used at least once, although only a CheckSuite needs to be explicitly implemented—the rest are only used internally. As illustrated in the figure below, each suite is composed if an arbitrary number of Groups and Blueprints on those groups where each Blueprint generates a number of results equal to the number of Checkers, both inherited from groups and explicitly attached. Blueprints are created with simple python dicts; since they are data-driven, it is easy to add them programmatically (illustrated by the different styles in acdd.py and gds2.py). For a more detailed understanding of these classes, check the docstrings in base.py.

2. IOOS Compliance Checker (compliance_checker/)



The package-level directory compliance_checker is actually a symlink to the true location in the ioos-compliance-checker subdirectory so that we can keep the documentation for that Git subtree separate. In this module, all of the non-essential CF checking parts have been stripped out. All references to dogma, wicken, and petulant-bear have been removed to simplify maintenance and deployment.

3. Web Interface (web/)

This package contains all of the utilities, templates, static files, and routes for use with the server framework we use, Flask[12]. The server.py file funnels the other modules into the web interface. In a real deployment, this module would be used by an interface, i.e. mod_wsgi, which requires the configuration files in the package-level directory.

Compliance Checker Design

Visually, the design of the current checker was influenced by a proof-of-concept web interface for the GDS2 checker, seen below. By proving that the web format was really useful for viewing the results of a compliance check, we continued with the hierarchical, color-coded approach we have today.

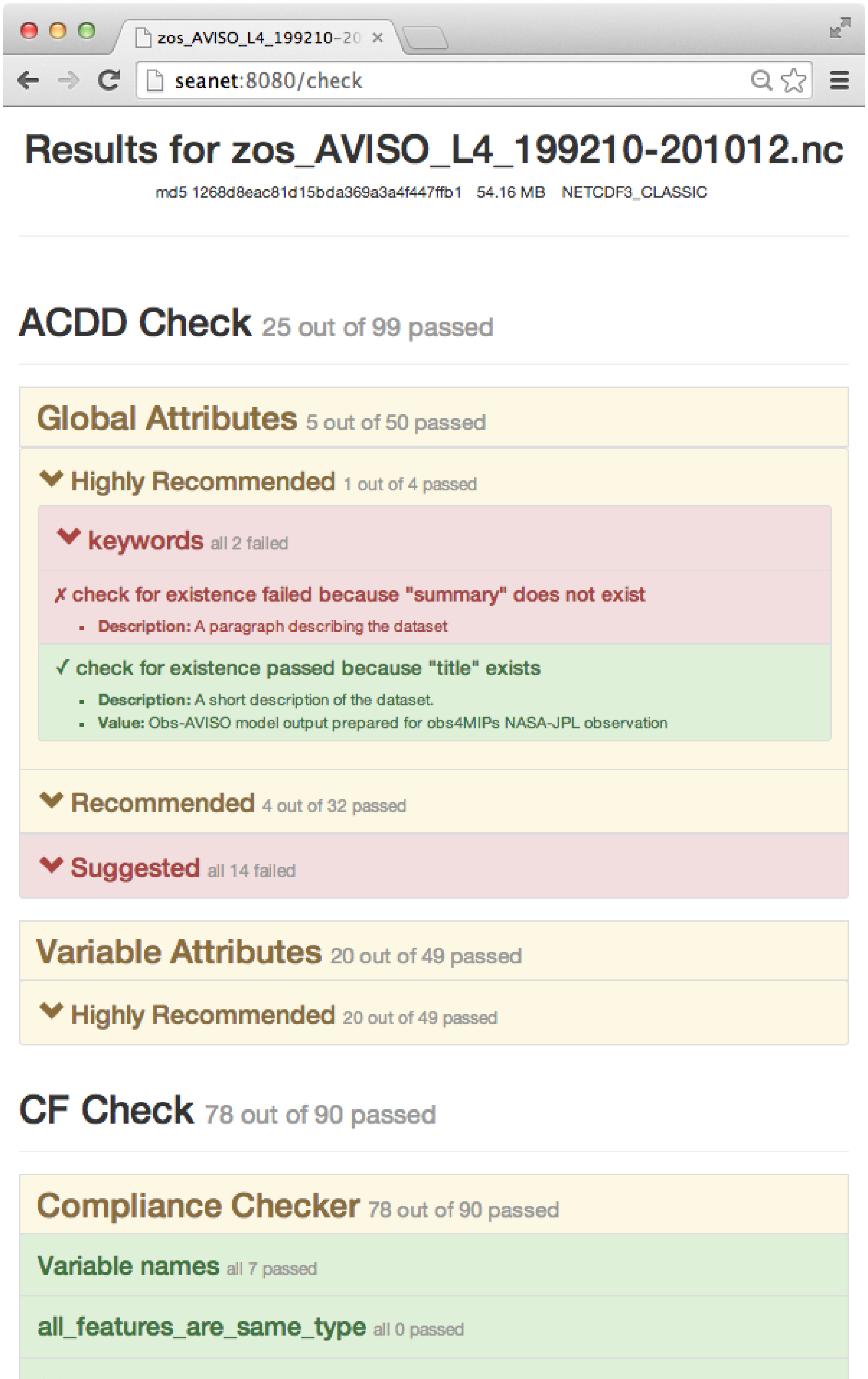
```
--- datasets/netcdf » file testMods.nc
testMods.nc: NetCDF Data Format data
--- datasets/netcdf » curl -F "ACDD=on" -F "file-upload=@testMods.nc" -F "respon
se=json" -s localhost:8080/check | head -n 15
{
  "fn": "testMods.nc",
  "md5": "cd22bae08126badfd9dedb9231b71edf",
  "model": "NETCDF3_CLASSIC",
  "results": [
    {
      "description": "These conventions identify and define a list of NetCDF glo
bal attributes recommended for describing a NetCDF dataset to discovery systems
such as Digital Libraries. Software tools will use these attributes for extracti
ng metadata from datasets, and exporting to Dublin Core, DIF, ADN, FGDC, ISO 191
15 etc. metadata formats.",
      "hash": 2524994885125708077,
      "name": "Attribute Convention for Data Discovery",
      "passed": 61,
      "results": [
        {
          "hash": 4800022785557684338,
          "name": "Global Attributes",
          "passed": 19,
```

In terms of software design, the checker uses an approach that is nearly identical to the approach taken by the IOOS checker, which itself uses an approach that is heavily influenced by unit testing frameworks. We use the data from the check suite to create a checker function which is tested against the dataset for a pass fail result. The current hybrid data + Python approach of creating check suites gives us enough flexibility for nearly every situation and has the nice side effect of making it easier to create new suites while sharing a common base and checker code. We use Object-Oriented Programming (OOP) for nouns (Checkers, Blueprints, CheckSuites, etc.), and we also use recursion to traverse the tree of Group + Checker trees top-down for execution, scoring, and template generation.

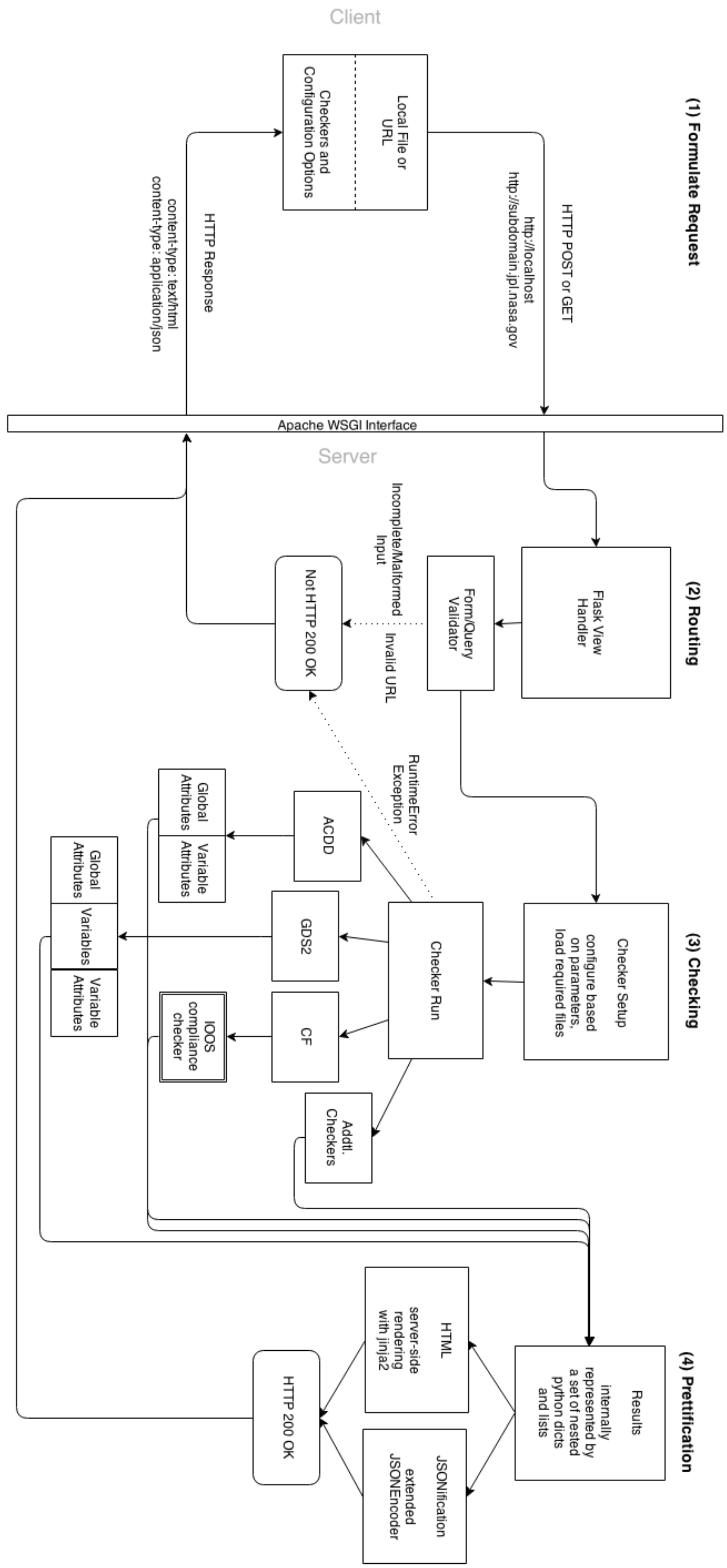
Different Interfaces

For maximum usability, there are two distinct methods of accessing the checker's rendered information. The first and most easily accessible method is through the HTML interface that offers options for local files and remote URLs and provides a GUI for easy use. In addition, there is a JSON API endpoint that generates the same data as the HTML interface in addition to useful internal values while exposing the same options as the HTML interface. With this programmatic interface, it is possible to easily integrate the tool with other tools or workflows or even create batch requests. It is our hope that the tool can be extended to be useful by nearly anybody.

Appendix A: View of the Web Interface Results



Appendix B: Lifecycle of an HTTP Request to the Checker



References

1: http://wiki.esipfed.org/index.php/Attribute_Convention_for_Data_Discovery
2: http://cfconventions.org/
3: http://podaac.jpl.nasa.gov/
4: https://www.ghrsst.org/files/download.php?m=documents&f=121009233443-GDS20r5.pdf
5: http://puma.nerc.ac.uk/cgi-bin/cf-checker.pl
6: http://thredds.jpl.nasa.gov/thredds/catalog.html
7: ftp://podaac.jpl.nasa.gov/allData/ghrsst/sw/GDS2_validation/
8: https://github.com/ioos/compliance-checker
9: https://github.com/ioos/compliance-checker/issues/65
10: http://cfconventions.org/Data/cf-convetions/cfconventions-1.6/build/cf-conventions.html
11: http://getbootstrap.com/
12: http://flask.pocoo.org/

Acknowledgements

- Ed and George for the opportunity to spend a fantastic summer at JPL
- The PO.DAAC team for their hospitality and usability improvements
- Dave Foster, Daniel Maher, and Luke Campbell for their creation and continued work on the IOOS compliance checker

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by JPL Summer Internship Program (JPLSIP) and the National Aeronautics and Space Administration.