Python in Heliophysics Community Meeting

November 13-15, 2018

DOI: 10.5281/zenodo.2537188

Meeting materials are located at http://heliopython.org/meetings/

Table of Contents

Participants

Will Barnes, Rice University
Julie Barnum, LASP
Robert Candey, GSFC
Baptiste Cecconi, Observatoire de Paris
Steven Christe, GSFC
Alexandria Ware DeWolfe, LASP
Jeremy Faden, Cottage Systems
Eric Grimes, UC Berkeley
Bernie Harris, GSFC
Bryan Harter, LASP
Liam Kilcommons, University of Colorado
Alan Loh, Observatoire de Paris
Bob McGuire, GSFC
Stuart Mumford
Ayris Narock, GSFC
Quynh Nhu Nguyen, Observatoire de Paris
Jonathan Niehof, University of New Hampshire
Armando Maldonado
Nicholas Murphy, CfA
Russ Panneton, LASP
Asher Pembroke, GSFC
Chris Piker, University of Iowa
Kevin Reardon, NSO
Aaron Roberts, GSFC
Anthony Rogers, University of New Hampshire
Daniel Ryan, GSFC
Sabrina Savage, MSFC
Russell Stoneback, UT Dallas
Fraser Watson, NSO
Don Woodraska, LASP

Introduction

In 2018, the Python in Heliophysics Community (PyHC) was established to bring together developers from the different Python software packages for heliophysics in order to increase collaboration and coordination of software development efforts. The goals of this group are to share knowledge and recommendations, develop standards for new and existing projects, and provide support for tools and libraries that are used by multiple projects. Numerous software projects in various branches of heliophysics are currently being developed in or converted to Python. While there are some commonalities between these projects, there has thus far been only limited unified community effort to settle on standards and ensure that related software packages are interoperable.

We began this effort with a meeting at the Triennial Earth-Sun Summit (TESS) in spring 2018, and held bi-weekly telecons starting in July. We agreed that an in-person meeting would help us clarify the path forward and establish priorities. Our first community meeting was held at LASP in Boulder, Colorado, on November 13-15, 2018. The meeting consisted of a mix of presentations, discussions, and an "unconference," which allowed us to organize a set of discussions based around consensus on topics of interest. This report summarizes the discussions and outcomes of this meeting, and is intended to act both as a report to NASA, a historical document capturing the status of this project, and a resource for other communities which may choose to carry out a similar effort.

State of the Field

At the time of this writing there are a total of 51 Python packages that are self-reported as being part of PyHC (listed on the heliopython.org website). This represents a total of approximately 200,000 lines of Python code as well as 100,000 comment lines as measured using the open source Ohcount library for counting lines of source code (openhub.net). The vast majority (>90%) of this code is currently available on GitHub and is therefore open source and follows an open development model. The top five largest projects (see Figure 1) in terms of lines of code+comments are in order of decreasing size PlasmaPy, SunPy, SpacePy, PyDARN, and SpiceyPy. In total, these 5 projects represent 62% of the total codebase while the other 46 projects represent the other 38%.

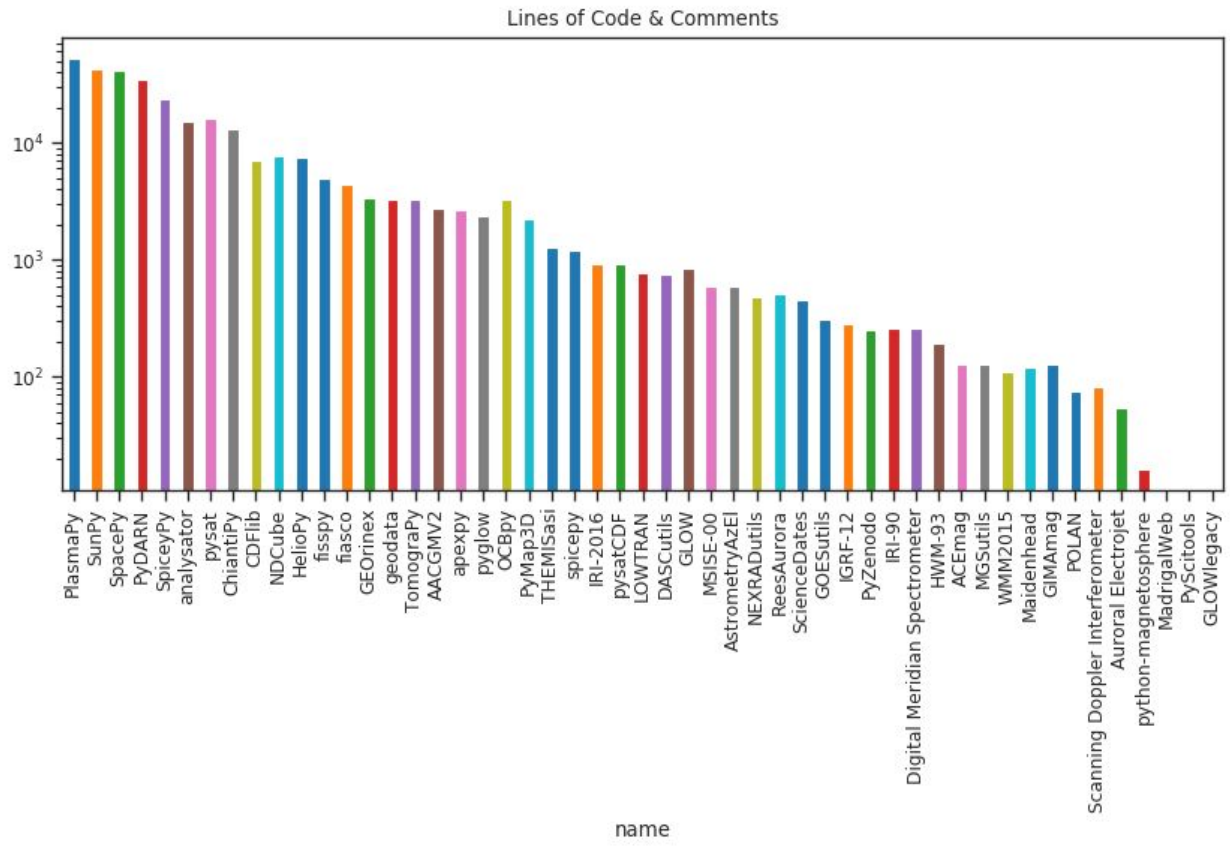

Figure 1: A size comparison of Heliophysics Python projects as measured by total lines of code+comments. The five largest package represent about 62% of the total code base with the remaining 46 projects accounting for the other 38%.

This code base has been developed by hundreds of individual contributors. In general, there is a roughly linear relationship between the number of contributors and the total size of the project therefore the largest projects have the largest number of contributors. PlasmaPy, SunPy, and PyDARN each have over 35 contributors with SunPy is the outlier with over 100 individual contributors. On the other hand the majority of projects are smaller and depend on fewer contributors. The majority of projects (75%) have less than 10 contributors while 50% have only 1 to 5 contributors (see Figure 2). Ten projects have only

one contributor.

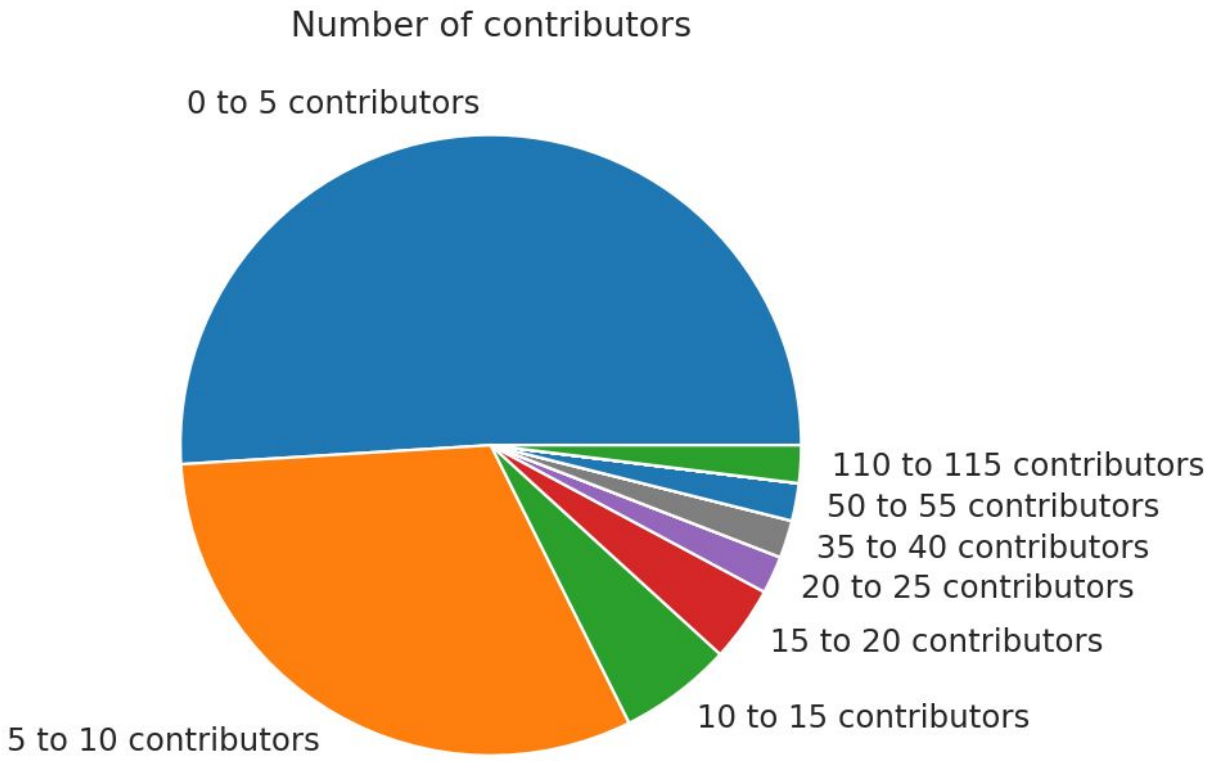

Figure 2: Most projects (~75%) have relatively few contributors (fewer than 10) while the largest projects have a large number of contributors.

The age of a project may be an indicator of its maturity. The oldest project is 8.5 years old while the youngest is only about 2 months old. The age of the project does not seem to correlate with the total lines of code which suggests that most package are built to fulfill a certain scope and do not expand beyond that scope.

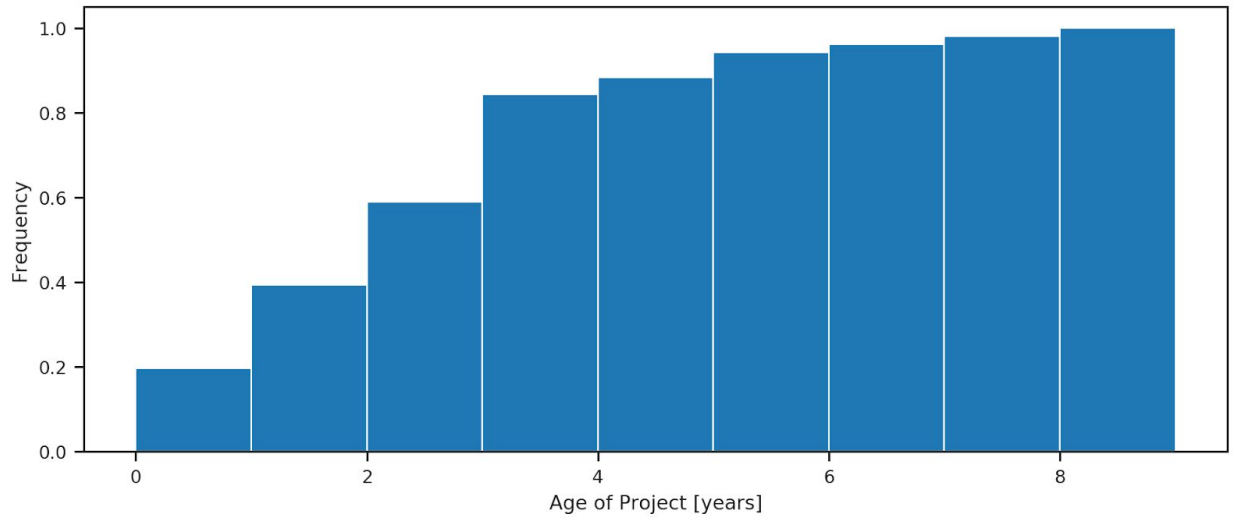

Figure 3: The cumulative age distribution of all projects. No project is older than 8.5 years old. About 50% of projects are between 2 and 3 years old while the remaining projects are from 3 to 8 years old.

Few projects make use of Sphinx for documentation (only 30%) while 43% of projects use some form of automated testing.

Meeting Overview

The meeting was attended by 21 people, with several more attending remotely. Projects represented included SunPy, SpacePy, pysat, PlasmaPy, ndcube, and Fiasco, as well as related efforts such as the International Heliophysics Data Environment Alliance (IHDEA) and the Space Physics Data Facility (SPDF). The meeting spanned two and a half days, and was held at the Laboratory for Atmospheric and Space Physics (LASP) in Boulder, Colorado. The full agenda is attached at the end of this report. The first day was primarily dedicated to presentations on the status and planned development of the major existing Python software packages, with afternoon discussions of general software development standards and the potential for developing unified software frameworks. Day two began with flash demonstrations of software packages. The afternoon of day two was an “unconference” session, in which attendees wrote a list of topics of interest, voted on which topics to discuss, and then held a series of parallel breakout sessions on those topics. Each unconference session was led by one person and had another taking notes. As the sessions were parallel, not all conference attendees were present at each session. The results of those sessions are summarized below.

Discussion Summaries

Plan for Community Development

The scientific and open source software communities both support the open development of tools and the broad dissemination of honest results. In this spirit, we propose that the first tools built as a community support areas with foundational requirements for current tools as well as ease the development of future tools.

The study of heliophysics spans a number of different sub communities, each with its own heritage of instrumentation and software practices. It is not expected that a single package could successfully meet the needs of the whole community. There are different technical challenges within each discipline that necessitate different solutions and different approaches to science software. For example, solar measurements are largely dominated by remote images however ionosphere/thermosphere measurements can include remote images, radar measurements, in situ satellite based measurements, all-sky ground based cameras, GPS networks, magnetometer networks, etc. A single package that tried to solve all problems for everyone will likely not work for anyone.

Though there are a range of requirements for each community, there does exist a common ground across heliophysics and development should begin there. All scientists need to perform some or all of the following, such as download data (heliodownload), load data (heliodata), understand when/where a measurement was performed (heliospacetime), transform measurements from one basis to another (heliocoordinates), as well as express vectors in different systems (heliovector).

The development of each of these ‘helio’ packages is based on a hub and spoke model. The ‘helio’ name is reserved for packages committed to a consistent user interface to support long term use by other packages. One method to achieve this rigidity is to generally limit the ‘helio’ package functionality, the hub, to glue binding other community packages, the spokes, that actually implement various functions towards a common goal. As an example, consider the heliodata package. This package could present a single user interface command, load, that accepts a filename and returns data in a standard format. Internally, the actual loading of the file would be handled by outside packages. netCDF4 for netCDFs, pyCDF for CDF files, pyhdf5 for HDF files, etc. As the python community evolves and data packages change, the ‘heliodata’ can simply replace the underlying packages with updated versions all while maintaining the same public interface.

Each ‘helio’ package is expected to be

- Pip installable
- Well documented with simple interfaces and appropriate unit tests
- Be available as its own package (not be wholly contained within another package)

Internal Data Representation

Led by Jon Niehof
Notes by Dan Ryan

Discussion of thinnest/lightest data object used by various projects: SpacePy dmarray; Komodo function decorators for metadata; PySat Instrument objects; various SunPy objects. It was clear there was not an obvious least-common-denominator for object types; however, we agreed that we should minimize differences in behavior.
Recommendation: basic container types should be based on numpy ndarray or xarray, or Pandas dataframe.
Recommendation: metadata for a container should be included as a key-value mapping (i.e. dict-like) accessed through an attribute named “meta” or “attrs”; ideally both at this time (further research of the existing ecosystem is planned to determine if one or the other is substantially more common)
Recommendation: The underlying data container in an object should be exposed via a “data” attribute, although not necessarily exclusively (e.g. for objects which are-a fundamental type, the data attribute may simply self-reference.) Further research will determine whether it is possible to build an object that will gracefully degrade to an ndarray by returning its data attribute in an array context; it was agreed that this is desirable if possible (so all numpy operations can be applied to a container object.)

HDEE call

Notes by Nick Murphy

We discussed plans for writing a Heliophysics Data Environment Enhancements (HDEE) call for proposals to be released in 2019, including duration and scope of funding opportunities.

Base objects (time, coordinates, units)

Led by Steve Christe
Notes by Dan Ryan
Link to raw notes
This session focused on discussing the use of base objects across packages. The goal was to reach a consensus on the use of a set of common base objects in order to promote interoperability across packages. This discussion focused on three areas; time, coordinates, and units.

For coordinates, multiple packages have implemented different approaches to coordinates and coordinate conversions. SunPy makes use of the astropy coordinates framework. SpacePy has developed its own Coords object class which currently depends on the Fortran-based IRBEM library for coordinate conversions. PySat depends on the CGC4 and PyPHEM. A demo was provided of the astropy coordinates framework which may be able to bring all packages under a single coordinate framework.

Recommendation: Packages should consider using the Astropy Coordinate framework to determine whether it has enough capabilities for all Heliophysics especially whether it can support quaternion transformations (e.g. velocity).

For time, there three current standards across multiple packages. It was recognized by many that though the Python datetime object is well-supported it is lacking key aspects such as leap seconds which can be a burden. SunPy has recently switched to the Astropy Time objects which provides many advantages including properly dealing with leap seconds. It can also easily output to datetime. One downside is that the leap seconds are currently hard-coded which means that adding future leap seconds will require a new astropy release forcing packages that depend on it to support a new version. SpacePy has developed the time object referred to as TickTock. It handles leap seconds and ephemeris time therefore should be compatible with AstroPy time.
Recommendation: Packages should consider using AstroPy time to determine whether it provides the capabilities needed by this community.

There was only a short amount of time to discuss units. No unit objects were discussed but it was noted that the AstroPy units package is powerful and well-supported. The following consensus recommendations were agreed upon.

Recommendation: The use of units are a good idea and are recommended. Should be used for user-facing API. Suggest considering astropy units.

Recommendation: When defining a unit string, use a string representation that is astropy units compatible.

Another recommendation/realization is that as a community, we should agree on the values of heliospheric constants.

Atomic data

Led by Nick Murphy and Will Barnes

For topics ranging from interpreting observations of the solar corona to understanding energetic neutral atoms in the heliosphere, atomic data plays a vital role in heliophysics research. It is therefore necessary that tools that provide easy access to atomic data be included in the heliophysics Python software ecosystem. **There should be a unified framework that provides a common interface for the needed atomic data and derived quantities.** Stable, long-term funding to cover atomic database maintenance and tool development is critical. Data and software should be released under permissive open access and open source licenses to maximize potential for scientific reproducibility. In particular, data released under the Creative Commons Attribution 4.0 International license would require derived works to provide information on the provenance of the data while permitting reuse, modification, and redistribution.

Unified Downloader

Led by Steven Christe
Notes by Dan Ryan
[Link to the raw notes]

Current capabilities of existing packages to support data sources was reviewed. The following summarizes the current state of affairs.

Data Provider	**Package**	**Interface**
Virtual Solar Observatory	SunPy	HTTP API
LYRA	SunPy	files directly from constructed URL
GOES/XRS	SunPy	FITS files directly from constructed URL
NOAA Indices	SunPy	Txt file
RHESSI Summary lightcurves	SunPy	
JSOC (serves SDO/AIA)	SunPy	HTTP API
Nobeyama Radio Telescope	SunPy	files
FERMI GBM	SunPy	files
MMS	SPEDAS	Restful API
THEMIS	SPEDAS	HTTP API
CDAWeb	SPEDAS	HAPI
CDAWeb CDF	AI CDAS	
MAVEN	PyTplot	Rest
Madrigal	pysat	
CDAAC	pysat	FTP (moving to sftp)
CDAWeb	pysat	HTTPS
LASP	pysat	HTTPS

Packages whose representatives were not present are missing from this list. A discussion on the basic needs for a download client suggests that there is sufficient overlap in needed functionality that it is possible to consider a single downloader to support all of our needs.

SunPy has developed a pluggable unified downloader, referred to as Fido, which may provide most of what is needed.

Recommendation: A single unified download interface should be developed to allow access to any heliophysics data provider.

In order to accomplish this task, a new working group will be created. Steven Christe volunteered to lead the group.

Licenses

Led by Nick Murphy
Notes by Armando Maldonado

Open source licenses can be permissive or copyleft. Permissively licensed code and adapted works may be redistributed under different licenses, whereas copyleft code and adapted works must be released under the same license. This restriction means that copyleft code cannot be incorporated into a permissively licensed project, which in practice complicates code sharing and discourages use by businesses. **We recommend using a permissive license for open source scientific software** (e.g., the Open Source Initiative approved BSD 2-clause, BSD 3-clause, or BSD+Patent licenses), though developers should check with their home institution as to legal requirements.

CDF convergence

Led by Jon Niehof
Notes by Baptiste Cecconi

Discussed existing Python CDF libraries (pycdf; pysatcdf; cdflib) and ways to reduce duplication.
Recommendations are:

1) pycdf improvements, particularly to the install process (and minimizing difficulties of other dependencies with full SpacePy). Consideration of pulling pycdf out of SpacePy (but discussion of how big projects should be, not clear this is necessary)
2) pycdf devs work with pysatcdf devs to ensure pycdf can address pysatcdf use cases; pysatcdf functionality either incorporated into pycdf or thin wrapper around
3) pycdf and cdflib devs work together to expose a low-level interface (similar to CDF C internal interface) so the pycdf interface can work with the cdflib pure-Python implementation. There was agreement that a pure Python implementation may be of benefit. Eventually this may mean a convergence between the two but not obvious this is immediately desirable.
4) New code and projects should use pycdf if practical.

5) General recommendation to all projects to file bug reports rather than forking or reimplementing; and release often to avoid the need for forks.

Recommendations and Actions

Standards

The standards discussion held during the meeting led to the drafting of a standards document, Python in Heliophysics Community (PyHC) Standards. It is available to the community and has been signed by representatives of the PyHC community. We recommend that existing projects do their best to conform to these standards, and that new projects be held to the standards. Community education will be needed to support the adoption of the standards; how best to achieve this should be an ongoing topic of discussion at future PyHC meetings.

Software Framework

We discussed whether to adopt a unified software framework for all future heliophysics Python development. The consensus was that we are not yet at a point where this is feasible, but we identified a few opportunities for consolidation and coordination of existing code. A working group was established to develop a unified download tool. Another group is investigating how to bring together the various options for reading and writing CDF files.

Community Education

A Python ecosystem for heliophysics will be most useful if members of the heliophysics community are proficient in Python. Blog posts and newsletter items may be used to build awareness of different software projects. Software Carpentry workshops would help heliophysics students and scientists learn computing skills that are often neglected in coursework (e.g., shell scripting, version control, and introductory Python). Software Carpentry lessons for Python in heliophysics could be created, along with resources such as an IDL to Python reference sheet. Workshops and tutorials at meetings provide additional opportunities for community engagement and educations. Additional possibilities would be to have week-long annual summer schools on Python in heliophysics or a Python in Heliophysics conference. Continuing education for core package developers could include in-depth tutorials during telecons or trainings in other venues.

Future work and conclusions

The community will continue to collaborate via bi-weekly telecons, and will hold in-person standalone meetings twice yearly, as well as meeting at various conferences. The community intends to propose to the upcoming Heliophysics Data Environment Enhancements (HDEE) funding opportunity to support

improvements to the infrastructure of the existing Python packages and potentially develop new code as well. Potential proposals will be discussed in upcoming telecons.

This effort has thus far been highly beneficial to the community, resulting in increased communication, consolidation of parallel efforts, development of funding opportunities, and tangible outcomes in the form of the standards document and the improvements to the heliopython website. We encourage other communities to work toward coordinating their development efforts as well, and welcome future collaborations with other Python development communities in the sciences.

Final agenda

Tuesday, Nov 13, 2018		
9:00	Introductions and welcome	Alex DeWolfe
9:15	Project overview and meeting goals	Aaron Roberts
	Overview of Existing Packages	
9:45	SunPy	Steven Christe
10:00	SpacePy	Jon Niehof
10:15	Pysat	Russell Stoneback
10:30	Break	
11:00	SPEDAS	Eric Grimes
11:15	ChiantiPy/Fiasco	Will Barnes
11:30	PlasmaPy	Nick Murphy
11:45	ndcube	Dan Ryan
12:00	Lunch	
13:00	HAPI Python Client	Bob Weigel
13:15	SpiceyPy	Andrew Annex
13:30	MASER	Baptiste Cecconi
13:45	Development Standards Discussion	Steven Christe
14:45	Break	
15:00	Proposed development stack & software framework	Russell Stoneback
Wednesday, Nov 14, 2018		
9:00	Flash Demos Session, ~5-10 min each: Asher Pembroke, Bryan Harter, Eric Grimes, Chris Piker, Will Barnes, Abigail Azari, Dan Ryan	
10:30	Break	
10:45	Functionality Gaps and Overlap	Aaron Roberts
11:15	Data formats, readers, writers, metadata	Bobby Candey
12:00	Lunch	
13:00	Unconference I: Internal data representations, HDEE call	
14:00	Unconference II: Base objects, atomic data	
15:00	Unconference III: Unified downloader, community licenses, CDF convergence	
16:00	Unconference Outbrief	

Thursday, Nov 15, 2018		
9:00	How to encourage and maintain interoperability	Baptiste Cecconi
9:30	Student Education and Program Design	Armando Maldonado
10:00	Educating the Community	Nick Murphy
10:30	Break	
10:45	Funding Future Development	Aaron Roberts
11:15	Summary & Path Forward	Alex DeWolfe
12:00	Lunch	
13:00	(OPTIONAL) Report Planning	Alex DeWolfe