CUTTLEFISH V1.0
Aggressive Data Reduction

NPOI Data-Reduction Software Design and User Manual

Nick Elias
OAM Solutions, LLC
2014 August 03

Table of Contents

1.0 Introduction

In 2013 February, Gerard van Belle hired me through Lowell Observatory to write a simplified NPOI data-reduction software package. The top-level requirements were:

- Run from the linux command line
- Extract squared-visibility and closure phase data from *.con files
- Average the data
- Remove bias from the data
- Calibrate the data
- Write the data to an ASCII in fixed-column format, suitable for viewing in an editor or spreadsheet

Based on a preliminary design, I estimated that I could come up with a simple, ergonomic, easily expandable, and well documented C++ package for 0.25 FTE in ~ 15 months.

The NRL contingent of NPOI immediately became interested in this work. After some discussion, additional features were requested:

- Writing averaged, bias removed, and calibrated data to *.cha files so that they can be read by oyster.
- Handling complex visibilities
- Loading data into IDL (including oyster) and/or other scripting language for specialized analysis, plotting, etc.
- Creating a modern version of constrictor based on code written by Dave Mozurkewich and Anders Jorgensen, which includes saving reorganized NPOI data packets to disk and the most up-to-date algorithms for calculating complex visibilities

Given the amount of time allocated in the Lowell contract, only the first additional feature could be done for this contract. To avoid not fulfilling the original contract because of "mission creep," I modified the original design to include "hooks" to these new features, under the assumption that additional funds will be available in the future to implement them.

The working name for this package is "cuttlefish." It was begun approximately 15 years ago before it was suspended.

2.0 Design Considerations

In this section, I discuss and explain my design choices for this software.

2.1 Language(s)

The original data reduction software for the NPOI laser metrology, called inchworm, was written by me in an object-oriented manner from the low-level algorithmic code to the top-level CLI/GUI code. The low-level code was written in "object-oriented C." The low-level code was linked to the CLI through a layer [incr tcl], a true object-oriented version of the tcl scripting language. The GUI was written in the tk scripting language, which has object-oriented features, that is bundled with tcl. The GUI mirrored the object-oriented "classes" from the low-level code.

Experience has shown me that the object-oriented low-level code is easier to organize, understand, and update. I have also found that an object-oriented CLI/GUI makes software easier to link to object-oriented low-level code and is much easier for users to understand. Therefore, cuttlefish will employ the object-oriented paradigm at all levels.

Except for applications where extreme speed is a requirement, C++ is far superior to "object-oriented C". tcl, tk, and [incr tcl] are still available, but they are slow scripting languages now used mostly for embedded systems and do not have the required data-manipulation capabilities (e.g., array slicing, built-in statistical functions, etc.). Given the increased speed of computers and the relative small size of *.con and *.cha datasets, I had considered writing cuttlefish completely in object-oriented python. Python, however, is less stable than C++ (there are major differences between versions 2.x and 3.x), so I ultimately decided to employ the latter since NPOI will be in operation for many more years.

There are three viable candidates for the cuttlefish CLI/GUI code:

- IDL – It has impressive data-manipulation facilities, although its object-oriented API is primitive and leaves much to be desired. Licenses are expensive, but a free version called GDL is available. Fabien Baron (GSU/CHARA) is part of the GDL development team and has been promoting it. Christian Hummel has gotten most parts of oyster to work using GDL.
- MatLab – Its capabilities are very similar to IDL and it has a better object-oriented API. Licenses are also expensive. No other NPOI code uses MatLab, which makes licenses less cost effective.
- python – It is a free package used by a lot of astronomical software that has a decent (although imperfect) object-oriented API. It has acceptable data-manipulation facilities, and linking low-level code to the API is relatively simple compared to IDL and is almost automatic.

I immediately rule out MatLab because its features, which are similar to IDL/GDL and python, are not worth the extra cost. IDL/GDL and python, on the other hand, are very viable options and are available on all platforms. At this point, it is not required to commit to any of them. I will clarify this reasoning in Section 2.3.

2.2 File I/O Libraries

The constrictor program creates *.con files that contain configuration information and siderostat-/baseline- based point-averaged data. A point-averaged datum, usually 0.5-2.0 seconds long, is an average of raw data. Oyster (chameleon) and cuttlefish processes data in *.con files to create *.cha files that contain the same configuration information and siderostat-/baseline- based scan-averaged data. A scan-averaged datum is an average of point data.

The *.con and *.cha files are presently saved in HDS (Hierarchical Database Software) format. It was originally written by Starlink (now defunct), and it is now supported by the Joint Astronomy Centre at the University of Hawaii. The code and build system of the HDS version used by NPOI are somewhat clunky. Thus, I have created a simplified HDS build system for cuttlefish which builds only the static libraries for linux OSes (no builds for obscure/old OSes, no documentation extraction, etc.).

I had considered migrating to the more modern file format, HDF (Hierarchical Database Format). It has dedicated C/C++ APIs and can be directly accessed by both IDL and python. But, recall that all *.cha files must be readable by oyster (Section 1.0). To accommodate the oyster requirement and a possible future file format migration, I wrote the file I/O interface, called HData<FORMAT>{}, as a C++ template, which makes switching between HDS and HDF seamless. I have already created the HData<HDS4>{} specialization, which is the interface to the HDS library. Creating the HData<HDF5>{} specialization, with the same member functions, will be relatively simple but it must be postponed until a future contract.

The cuttlefish package has linux programs which write data to the terminal and ASCII files (terminal output can also be piped into ASCII files). There are two types of ASCII file formats, free-form and column-based. Programs that gather configuration data write to free-form ASCII files (exception: the program that reads/writes scan times, star positions, etc.). Programs that process point-averaged data to scan-averaged data write to column-based ASCII files. Both fixed-length space-delimited and comma-delimited files can easily be imported into spreadsheet programs. Ultimately, I decided to use fixed-length space-delimited columns because they are much easier to read in editors.

2.3 Code Organization

Cuttlefish is organized into a number of layers (Figure 1). The user has access to the package's functionality through simple linux programs, a CLI (TBD), and a GUI (TBD). The linux programs perform specific tasks such as: 1) getting configuration data from *.con and *.cha files and printing them to the terminal (they can be piped to ASCII files); 2) processing *.con files to *.cha files; and 3) getting all data from *.cha files and writing them to ASCII files.

The first coding task was to create a C++ interface to the HDS library (the HData<HDS4>{} class template specialization). This interface is used by all high-level code. I investigated linking HData<HDS4>{} directly to the HDS FORTRAN-77 library, but I determined that this design would lead to a very large and unwieldy class template.

In 1998, I wrote a C API to the HDS FORTRAN-77 library for the inchworm laser-metrology package. I now use this C API as an intermediary layer between HData<HDS4>{} and the HDS FORTRAN-77 library, i.e., HData<HDS4>{} calls the C API which in turn calls the HDS FORTRAN-77 library. Sixteen years ago, the reduction of processing speed due to an extra layer would have been an issue. There are no speed issues now.

C and C++ employ zero-based arrays. FORTRAN-77, on the other hand, employs one-based arrays, which complicates linking FORTRAN-77 libraries to C and C++ code. I isolated the "zero-to-one" interface to helper static member functions that create HData<HDS4>{}. This scheme works perfectly well, but in the future I may modify the code such that the zero-to-one interface is located within HData<HDS4>{} so that all of that code is conveniently hidden within a single class. In other words, the only code exposed to one-based arrays is HData<HDS4>{} and the C API to the HDS FORTRAN-77 library.

The next higher layer consists of NPOI-based C++ class templates (and helper functions) that load, manipulate, write specific types of data within *.con and *.cha files. The template parameter is the file format (HDS4=done, HDF5=future, etc.) that is sent to HData<FORMAT>{}. I chose this scheme in order to keep the class interfaces relatively simple. For example, there is a class that handles geodetic parameters, a class that handles input beam data, a class that handles output beam configuration data, etc. I will provide a list of these NPOI-based classes in Section 4.0.

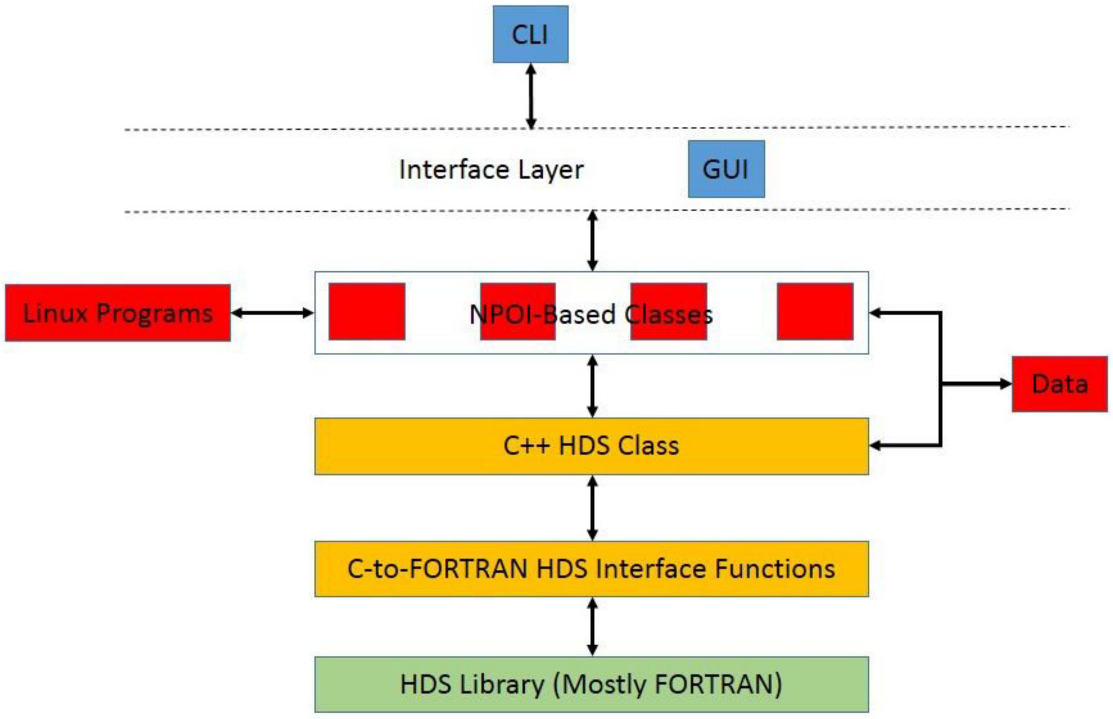

Figure 1. The cuttlefish organizational layers. Green: Imported code. Yellow: Finished code. Red: Working code that may or may not be augmented in the future. Blue: TBD code.

After I finished the C++ API HData<HDS4>{} to the HDS FORTRAN-77 library, I was very tempted to link it directly to an interface layer so that I would have a simple and powerful CLI to HDS files which I could employ to write the NPOI-based classes in the CLI scripting language instead of C++.

.con/.cha files are small enough, computer processor speeds are fast enough, and disk I/O is fast enough such that they could not be used to determine which language to

employ. Writing the code in the scripting language would be faster than writing it in C++, but the savings is not so great given that the amount of design time is ~ independent of language. In the end, I decided to fall back upon the old maxim, "Keep the interface layer between the low-level code and the CLI as thin as possible." NB: If additional funding appears, I would actually create a direct CLI to HData<HDS4>{} so that users and I can tests ideas via prototyping scripts. If the scripts are sufficiently useful, they could eventually be converted C++ code.

"Thin" means "minimal," just enough code to link the C++ classes to the CLI. It is possible to create a thin interface to python almost automatically. A thin interface to IDL takes a little more work, but it is straightforward.

One might be tempted to save time by writing helper functions in the scripting language, making the interface layer "thick," but this scheme represents a false economy. If a change in scripting language is required, these helper functions become obsolete and must be rewritten. C++ code, apart from small changes to the input/output variables, never needs to be rewritten.

An exception to the thin-interface rule is GUI code. GUIs are notoriously tedious to write in compiled languages, so writing them in a scripting language is an attractive alternative. Granted, if the scripting language is changed (or if another is added) new GUI code will have to be developed. Fortunately, GUI elements (buttons, list boxes, menus, etc.) are very similar or even identical between scripting languages, so creating a new GUI script is simply a matter of changing syntax.

There were three types of GUIs in inchworm: HDS browser, main GUI consisting of a simple button bar with drop-down menus, and sub-GUIs (called by the drop-down menus) corresponding to the high-level classes. Buttons and drop-down menus were deactivated until the required processing steps had been performed. This scheme is very ergonomic and requires minimal documentation as long as the user understands the underlying algorithms. If additional funding appears, this scheme can be adapted easily for cuttlefish.

In many software packages, there is a "math" or "processing" layer between the file I/O libraries and the top-level NPOI-based classes. The mathematics required for cuttlefish are not too extensive, so I have incorporated them as private member functions of (or friend functions to) the top-level NPOI classes.

The cuttlefish code requires two additional features. First, it must be able to convert one-dimensional untyped blocks of FORTRAN-77 memory (used by HDS) into multidimensional C/C++ arrays (used by the rest of the cuttlefish code). Second, it must be able to express multidimensional C/C++ arrays as/ pointers, STL objects, etc.

To those ends, I created the Data<TYPE,NUM>{} class template, where TYPE is the data type (char, short int, int, float, double, char*) and NUM is the dimension integer (0, 1, 2, 3; higher dimensions may be added later). Most of the inputs and outputs of the functions associated with the HData<HDS4>{} class template and the NPOI-based class templates employ the Data<TYPE,NUM>{} class template.

3.0 File Formats

In this section, I present the detailed formats for the outputs of constrictor and chameleon (part of oyster) that are used by cuttlefish. Eventually, these formats must be completely standardized and regularized.

Multidimensional arrays are now considered abstract, since what the high-level code and the user sees is independent of how the data are actually stored. For example, HDS data are stored in 1-based FORTRAN order, while the user sees 0-based C order.

The elements in the file format lists are color coded according to C++ class.

- Purple – Header<FORMAT>{}
- Orange – GeoParms<FORMAT>{}
- Green – GenConfig<FORMAT>{}
- Light Blue – InputBeamConfig<FORMAT>{}
- Dark Blue – OutputBeamConfig<FORMAT>{}
- Brown – TripleConfig<FORMAT>{}
- Black – BGScanData<FORMAT>{}
- Light Green – ScanData<FORMAT>{}
- Magenta – InputBeam<FORMAT>{}
- Gray – OutputBeam<FORMAT>{}
- Bright Green – Triple<FORMAT>{}

*3.1 *.con*

I have only included the elements used by cuttlefish. If oyster requires others, I can add the required code easily to the appropriate classes.

- Session
    - Format <_char>
        - The file format ("CONSTRICTOR" or "CHAMELEON")
    - Date <_char>
        - The date "YYYY-MM-DD"
    - UserID <_char>
        - The user ID ("UNSPECIFIED", unless specified by user)
    - SystemID <_char>
        - The system ID "NPOI"
    - ObserverLog <_char>
        - The observer log (a long string, including carriage returns)
    - ConstrictorLog <_char>
        - The constrictor log (a long string, including carriage returns)
    - GeoParms
        - Latitude <_double>

        - The latitude of the array center (degrees)
    - Longitude <_double>
        - The longitude of the array center (degrees)
    - Altitude <_double>
        - The altitude above sea level of the array center (m)
    - EarthRadius <_double>
        - The earth radius (units?)
        - Set to zero for now
    - J2 <_double>
        - The J2 of the geoid (dimensionless)
        - Some files have zero, others have numbers that look like the flattening not J2
    - LeapSeconds <_integer>
        - The number of leap seconds for the date of observation (s)
    - EarthRotation <_double>
        - The time shift to earth orientation (s)
    - TTminusTAI <_double>
        - TT time minus TAI time (s)
- GenConfig
    - InstrCohInt <_double>
        - Instrumental coherent integration (?)
    - BeamCombinerID <_integer>
        - Beam combiner ID
    - RefStation <_int>
        - The reference station
    - InputBeam
        - NumSid <_integer>
            - The number of siderostats (input beams)
        - StationID [NumSid] <_char>
            - The station ID for each input beam
        - SiderostatID [NumSid] <_integer>
            - The siderostat ID for each input beam
        - DelayLineID [NumSid] <_integer>
            - The delay line ID for each input beam
        - BCInputID [NumSid] <_integer>
            - The beam combiner input ID for each input beam
        - StarTrackerID [NumSid] <_integer>
            - The star tracker ID for each input beam
        - StationCoord [NumSid][4] <_double>
            - The station coordinates for each input beam (x,y,z,C)
    - OutputBeam
        - NumOutBeam <_integer>
            - Number of output beams
        - SpectrometerID [NumOutBeam] <_char>
            - Spectrometer ID for each output beam
        - NumBaseline [NumOutBeam] <_integer>
            - Number of baselines per output beam

                - NumSpecChan [NumOutBeam] <_integer>
                    - Number of spectral channels per output beam
                - BaselineID [MaxNumBaseline][NumOutBeam] <_char>
                    - The baseline IDs for each output beam
                - Wavelength [MaxNumSpecChan][NumOutBeam] <_double>
                    - The wavelengths for each output beam
                - WavelengthErr [MaxNumSpecChan][NumOutBeam] <_double>
                    - The wavelength errors for each output beam
            - Triple
                - NumTriple <_integer>
                    - Not implemented yet
                - OutputBeam [NumTriple][3] <_integer>
                    - Not implemented yet
                - Baseline [NumTriple][3] <_integer>
                    - Not implemented yet
                - NumSpecChan [NumTriple] <_integer>
                    - Not implemented yet
                - SpecChan [NumTriple][3] <_integer>
                    - Not implemented yet
        - BGScanData
            - NumBGScan <_integer>
                - The number of background scans
            - ScanID [NumBGScan] <_integer>
                - The scan IDs of the background scans (corresponding to scan IDs under the ScanData structure)
            - OutputBeam [NumOutBeam] <ExtColumn>
                - Rate [NumBGScan][NumSpecChan(beam)] <_real>
                    - The number of background photons integrated over a fringe time, averaged over a point datum
        - ScanData
            - NumScan <_integer>
                - The number of valid scans for a given night
            - ScanID [NumScan] <_integer>
                - The valid scan IDs
            - StartTime [NumScan] <_double>
                - The scan start times (ms since UTC midnight)
            - StopTime [NumScan] <_double>
                - The scan stop times (ms since UTC midnight)
            - Code [NumScan] <_integer>
                - The fringe flag (1 = on fringe, 0 = off fringe)
            - StarID [NumScan] <_char>
                - The star ID for each scan
            - NumCoh [NumScan] <_integer>
                - Number of instrumental coherent integrations per constrictor coherent integration (?)
                - Not implemented
            - NumIncoh [NumScan] <_integer>

            - The number of raw data in each point datum
    - PointData [NumScan]
        - NumPoint <_integer>
            - The number of point data per scan
        - Time [NumPoint] <_double>
            - The middle time of each point datum within a scan (ms since UTC midnight)
            - Technically, these times could be different for each type of data because
                - Flagging data for each subsystem is likely different
                - The data come from independent subsystems
                - The point times are approximations, and they work OK
    - InputBeam [NumSid]
        - FDLPos [NumPoint] <_double>
            - The FDL position (m)
        - NATJitter [NumPoint] <_real>
            - The seeing measure from the narrow-angle trackers (units?)
        - NATJitterErr [NumPoint] <_real>
            - The error of NATJitter
    - OutputBeam [NumOutBeam]
        - DelayJitter [NumPoint][NumBaseline(beam)] <_real>
            - The delay standard deviation, calculated over a point datum (m)
        - VisSq [NumPoint][NumBaseline(beam)][NumSpecChan(beam)] <_real>
            - The number of correlated photons integrated over a fringe, averaged over a point datum
            - It is the non-normalized squared visibility
        - VisSqErr [NumPoint][NumBaseline(beam)][NumSpecChan(beam)] <_real>
            - The error of VisSq
            - The 1/sqrt(N) factor is included
        - PhotonRate [NumPoint][NumSpecChan(beam)] <_real>
            - The number of uncorrelated photons integrated over a fringe, averaged over a point datum
        - PhotonRateErr [NumPoint][NumSpecChan(beam)] <_real>
            - The error of PhotonRate
            - The 1/sqrt(N) factor is included
        - ComplexVis[NumPoint][NumBaseline(beam)][NumSpecChan(beam)][R/I] <_real>
            - Not implemented yet
        - ComplexVisErr [NumPoint][NumBaseline(beam)][NumSpecChan(beam)][R/I/C] <_real>
            - Not implemented yet
    - Triple [NumTriple]
        - ComplTriple [NumPoint][NumTripleChan][R/I] <_real>
            - Not implemented yet
        - ComplTripleErr [NumPoint][NumTripleChan][R/I] <_real>
            - Not implemented yet

*3.2 *.cha*

- Session

- Format <_char>
    - The file format ("CONSTRICTOR" or "CHAMELEON")
- Date <_char>
    - The date "YYYY-MM-DD"
- UserID <_char>
    - The user ID ("UNSPECIFIED", unless specified by user)
- SystemID <_char>
    - The system ID "NPOI"
- ObserverLog <_char>
    - The observer log (a long string, including carriage returns)
- ConstrictorLog <_char>
    - The constrictor log (a long string, including carriage returns)
- GeoParms (components should be the same as the *.con file but they aren't; this must be fixed)
    - Latitude <_double>
        - The latitude of the array center (degrees)
    - Longitude <_double>
        - The longitude of the array center (degrees)
    - Altitude <_double>
        - The altitude above sea level of the array center (m)
    - EarthRadius <_double>
        - The earth radius (units?)
        - Set to zero for now
    - J2 <_double>
        - The J2 of the geoid (dimensionless)
        - Some files have zero, others have numbers that look like the flattening not J2
    - TAI-UTC <_integer>
        - The number of leap seconds for the date of observation (s)
    - EarthRotation <_double>
        - The time shift to earth orientation (s)
    - TDT-TAI <_double>
        - TT time minus TAI time (s)
- GenConfig
    - InstrCohInt <_double>
        - Instrumental coherent integration (?)
    - BeamCombinerID <_integer>
        - Beam combiner ID
    - RefStation <_int>
        - The reference station
    - InputBeam
        - NumSid <_integer>
            - The number of siderostats (input beams)
        - StationID [NumSid] <_char>
            - The station ID for each input beam
        - SiderostatID [NumSid] <_integer>
            - The siderostat ID for each input beam
        - DelayLineID [NumSid] <_integer>

      - The delay line ID for each input beam
    - BCInputID [NumSid] <_integer>
      - The beam combiner input ID for each input beam
    - StarTrackerID [NumSid] <_integer>
      - The star tracker ID for each input beam
    - StationCoord [NumSid][4] <_double>
      - The station coordinates for each input beam (x,y,z,C)
  - OutputBeam
    - NumOutBeam <_integer>
      - Number of output beams
    - SpectrometerID [NumOutBeam] <_char>
      - Spectrometer ID for each output beam
    - NumBaseline [NumOutBeam] <_integer>
      - Number of baselines per output beam
    - NumSpecChan [NumOutBeam] <_integer>
      - Number of spectral channels per output beam
    - BaselineID [MaxNumBaseline][NumOutBeam] <_char>
      - The baseline IDs for each output beam
    - Wavelength [MaxNumSpecChan][NumOutBeam] <_double>
      - The wavelengths for each output beam
    - WavelengthErr [MaxNumSpecChan][NumOutBeam] <_double>
      - The wavelength errors for each output beam
    - ChanWidth [MaxNumSpecChan][NumOutBeam] <_double>
      - The channel widths for each output beam
      - *Not included in cuttlefish, must determine if necessary*
    - ChanWidthErr [MaxNumSpecChan][NumOutBeam] <_double>
      - The channel widths for each output beam
      - *Not included in cuttlefish, must determine if necessary*
    - FringeMod [MaxNumSpecChan][NumOutBeam] <_int>
      - The fringe modulation multiple
      - *Not included in cuttlefish, must determine if necessary*
  - Triple
    - NumTriple <_integer>
      - Not implemented yet
    - OutputBeam [NumTriple][3] <_integer>
      - Not implemented yet
    - Baseline [NumTriple][3] <_integer>
      - Not implemented yet
    - NumSpecChan [NumTriple] <_integer>
      - Not implemented yet
    - SpecChan [NumTriple][3] <_integer>
      - Not implemented yet
- ScanData
  - NumScan <_integer>
    - The number of valid scans for a given night
  - ScanID [NumScan] <_integer>

        - The valid scan IDs
    - ScanTime [NumScan] <_double>
        - The average scan times (s since UTC midnight)
    - StartTime [NumScan] <_double>
        - The scan start times (s since UTC midnight)
    - StopTime [NumScan] <_double>
        - The scan stop times (s since UTC midnight)
    - Code [NumScan] <_integer>
        - The fringe flag (1 = on fringe, 0 = off fringe)
    - StarID [NumScan] <_char>
        - The star ID for each scan
    - NumCoh [NumScan] <_integer>
        - Number of instrumental coherent integrations per constrictor coherent integration (?)
        - Not implemented
    - NumIncoh [NumScan] <_integer>
        - The number of raw data in each point datum
    - OutputBeam [NumOutBeam]
        - VisSq [NumSpecChan(beam)][NumBaseline(beam)][NumScan] <_real>
            - The normalized squared visibilities, bias removed or not removed
        - VisSqErr [NumSpecChan(beam)][NumBaseline(beam)][NumScan] <_real>
            - The normalized calibrated squared visibility errors
        - VisSqC [NumSpecChan(beam)][NumBaseline(beam)][NumScan] <_real>
            - The calibrated normalized squared visibilities, bias removed or not removed
        - VisSqCErr [NumSpecChan(beam)][NumBaseline(beam)][NumScan] <_real>
            - The calibrated normalized calibrated squared visibility errors
        - DelayJitter [NumBaseline(beam)][NumScan] <_real>
            - The delay jitter
        - DelayJitterErr [NumBaseline(beam)][NumScan] <_real>
            - The delay jitter error
        - PhotonRate [NumSpecChan(beam)][NumScan] <_real>
            - The photon rate
        - PhotonRateErr [NumSpecChan(beam)][NumScan] <_real>
            - The photon rate error
        - BackgndRate [NumSpecChan(beam)][NumScan] <_real>
            - The background rate, copied from the *.con file and put into the correct scans
        - BackgndErr [NumSpecChan(beam)][NumScan] <_real>
            - The background rate error, copied from the *.con file and put into the correct scans
    - Triple [NumTriple]
        - ComplTriple [2][NumTripleChan(triple)][NumScan] <_real>
            - Not used in cuttlefish
        - ComplTripleErr [2][NumTripleChan(triple)][NumScan] <_real>
            - Not used in cuttlefish
        - TripleAmp [NumTripleChan(triple)][NumScan] <_real>
            - The normalized triple amplitudes, bias removed or not in complex triples
        - TripleAmpErr [NumTripleChan(triple)][NumScan] <_real>

        - The normalized triple amplitude errors
      - TriplePhase [NumTripleChan(triple)][NumScan] <_real>
        - The triple phases, bias removed or not in complex triples
      - TriplePhaseErr [NumTripleChan(triple)][NumScan] <_real>
        - The triple phase errors
      - TripleAmpC [NumTripleChan(triple)][NumScan] <_real>
        - The calibrated normalized triple amplitudes, bias removed or not in complex triples
      - TripleAmpCErr [NumTripleChan(triple)][NumScan] <_real>
        - The calibrated normalized triple amplitude errors
      - TriplePhaseC [NumTripleChan(triple)][NumScan] <_real>
        - The calibrated triple phases, bias removed or not in complex triples
      - TriplePhaseCErr [NumTripleChan(triple)][NumScan] <_real>
        - The calibrated triple phase errors
    - InputBeam [NumSid]
      - FDLPos [NumScan] <_double>
        - The FDL position (m)
      - FDLPosErr [NumScan] <_real>
        - The FDL position (m)
      - NATJitter [NumScan] <_real>
        - The seeing measure from the narrow-angle trackers (units?)
      - NATJitterErr [NumScan] <_real>
        - The error of NATJitter
      - GrpDelay [NumScan] <_double>
        - The group delay, not used by cuttlefish
      - GrpDelayErr [NumScan] <_real>
        - The group delay error, not used by cuttlefish
      - DryDelay [NumScan] <_double>
        - The dry delay, not used by cuttlefish
      - DryDelayErr [NumScan] <_real>
        - The dry delay error, not used by cuttlefish
      - WetDelay [NumScan] <_double>
        - The wet delay, not used by cuttlefish
      - WetdelayErr [NumScan] <_real>
        - The wet delay error, not used by cuttlefish

4.0 Classes

In this section, I describe the C++ classes. They can be grouped into xx categories:

- Data and I/O classes – These classes are interfaces the HDS4 (and eventually HDF5) libraries and manipulate data read/written to/from files. Eventually, they will be directly accessed from a command-line interface.
- NPOI-based classes – These classes represent the top-level interface to the data. They will be used by database developers. Eventually, they will be directly accessed from a command-line interface for general users.

- Support classes – These classes support Data, I/O, and NPOI-based classes. Database developers may need to use them, but they are not required for general users.

NPOI-based classes deal with point data in *.con and *.cha files. The member functions can be organized in the following groups:
- Construction/Destruction – The member functions used to create and destroy instances of the classes.
- Input/Output – The private member functions used to load data from *.con and *.cha files (called by the constructors), and the public member functions used to write data to *.cha files.
- Query – The member functions used to get data from a class.
- Processing – The member functions that process data (e.g., remove bias, etc.) which are called by public member functions.

Each data-processing class also has associated friend functions for averaging and calibration.

4.1 Data and I/O Classes

Data<TYPE,DIM>{}:
- This class template was originally designed to reorganize array data read/written to/from HDS4 files because the library does everything in "FORTRAN order."
- This class template is used as the output to many NPOI-based classes. I did this because I did not know at the time what the command-line interface language would be (GDL/IDL or python). TBD.
- I have added query and slicing (subarray) capabilities to the class template.
- The TYPE template parameter refers to primitive data types, such as float, double, char, char*, int, etc. I allowed char, char*, int, etc. because I decided not to perform mathematical operations within this class. Because of the design of C++, char* must have its own template specialization (the other primitive types can use the generic template).
- The DIM template parameter refers to the number of array dimensions. Up to DIM=3 is supported, except for Data<char*,3>{} which is not required for this version of cuttlefish.

HDSWrapper{}:
- This class is a collection of static member functions, each of which corresponds to a function within the HDS4 library.
- I created this class because the linking cuttlefish C++ classes to HDS4 FORTRAN libraries required a lot of bookkeeping. All of this bookkeeping is located within this class.

HData<FORMAT>{}:
- This class template represents that main C++ interface between files.
- I have written the template specialization for HDS4. See the HDS documentation for additional information about how the HDS library is used.

- In a future contract, I can create the template specialization for HDF5 which has exactly the same member functions as the HDS4 specialization. **Thus, all code that calls HData<FORMAT>{} can access HDS4 and HDF5 files with no modifications.**
- In a future contract, I will create a GDL/IDL or python command-line interface as well as a graphical-user interface.

Status<FORMAT>{}:
- Each HDS4 library function returns a status number which depends on whether the function executed successfully and the error type.
- This class template provides a uniform interface to the HDS status.
- When I create an HData<HDF5>{} template specialization, I will create the Status<HDF5>{} template specialization.

4.2 NPOI-Based Classes

BGScanData<FORMAT>{}:
- This class template is used to get background data from *.con files.
- It does not write them to *.cha files because they written to PhotonRate file components by the OutputBeam<FORMAT>{} class template.

Catalog{}:
- This class represents the interface to star catalogs that contain astrometric information. It allows for various queries, including astrometric information, catalog aliasing, and membership flags.
- At present, it handles the Bright Star catalog, the Henry Draper catalog, and FK5 catalog through the Catalog::Star{} nested class and the Catalog::Star::parse<NAME>() template specialization function.
- It is very easy to add another catalog by adding a corresponding Catalog::Star::parse<NAME>() template specialization function.
- All catalogs are presently ASCII, but it should be no problem adding non-ASCII catalogs as long as its library can be linked.

Diameter{}:
- This class represents the interface to the star diameter list, which is an easily editable fixed-format ASCII file. NB: Cuttlefish cannot read minds. For the calibration to work, the star and its diameter must be included in the ASCII file.
- The star names must come from the Henry Draper catalog. I decided on this scheme to reduce processing time. It is not too much of a bother because the Catalog{} class has an aliasing feature.

EO{}:
- This class represents the interface to earth orientation parameter ASCII file.
- The time and earth orientation changes are used in the calculation of the baseline vectors.

- The information is copied from the http://toshi.nofs.navy.mil/ser7/finals2000A.all ASCII file to the same ASCII file in the cuttlefish package.
- If the cuttlefish version is out of date for a given NPOI dataset, warning messages will be printed.

File<FORMAT>{}:

- This class template represents the main interface to the data read/written to/from files.
- In each member function it calls the corresponding member functions of all of the class templates that interact with files so that users and developers don't have call them by hand.
    - GenConfig<FORMAT>{}
    - GeoParms<FORMAT>{}
    - Header<FORMAT>{}
    - InputBeamConfig<FORMAT>{}
    - InputBeam<FORMAT>{}
    - Log<FORMAT,LOG>{}
    - OutputBeamConfig<FORMAT>{}
    - OutputBeam<FORMAT>{}
    - ScanData<FORMAT>{}
    - TripleConfig<FORMAT>{}
    - Triple<FORMAT>{}

GenConfig<FORMAT>{}:

- This class template represents the interface to generic configuration data
    - beam combiner ID, reference station, etc.

GeoParms<FORMAT>{}:

- This class template represents the interface to geodetic parameters and the location of the array
    - Latitude, longitude, altitude of array
    - Earth radius, number of leap seconds, TT-TAI, etc.
- NB: The parameters in the *.con and *.cha files are not well defined (e.g., J2 is wrong). Their uses and values must be clarified.

Header<FORMAT>{}:

- This class template represents the interface to the header information
    - File format (CONSTRICTOR or CHAMELEON), date, system ID, user ID, observer log, and constrictor log

InputBeamConfig<FORMAT>{}:

- This class template represents the interface to the input beam (i.e., siderostat-based) configuration information
    - number of input beams (siderostats), siderostat IDs, siderostat controls IDs, beam combiner input IDs, star tracker IDs, station IDs, delay line IDs, station coordinates, and the stroke multipliers

InputBeam<FORMAT>{}:

- This class template represents the interface to the input beam data
  - NAT jitters, NAT counts, FDL positions
- This class also identifies which scans have all shutters opened (for fringe and background observations) and which scans have one or more shutters closed

LeapSecond{}:

- This class template represents the interface to the tai-utc.dat ASCII file that contains the dates and signs of leap seconds
  - Leap seconds can be instituted every January 1 or June 1. Consult the IERS web site for information.

Log<FORMAT>{}:

- This class template represents the interface to ASCII logs
  - The currently supported logs are CONSTRICTORLOG and OBSERVERLOG, which are written to *.con files and copied to *.cha files
  - It is possible to create new types of logs (perhaps a CHAMELEONLOG?) that can appended as necessary and written to *.cha files.

OutputBeamConfig<FORMAT>{}:

- This class template represents the interface to the output beam (i.e., baseline-based) configuration information
  - Number of output beams (spectrographs), number of baselines, number of spectral channels, spectrometer IDs, baseline IDs, wavelengths and their errors

OutputBeam<FORMAT>{}:

- This class template represents the interface to the output beam data
  - squared visibilities, delay jitters, photon rates (including backgrounds)
- This class averages, removes bias, and calibrates squared visibilities
- This class calculates the uvw

ScanData<FORMAT>{}:

- This class template represents the interface to the scan data and astrometric information
  - scan IDs, star IDs, on-/off- source codes, scan times, star positions and related quantities

TripleConfig<FORMAT>{}:

- This class template represents the interface to the triple configuration information
  - The triple IDs, output beams and baselines, spectral channel maps, and wavelengths and their errors

Triple<FORMAT>{}:

- This class template represents the interface to the triple data

    - complex triples, triple amplitudes, closure (triple) phases
- This class averages, removes bias, and calibrates triples

4.3 Support Classes

CalCommon{}:
- This class contains calibration-related static member functions that are used by both OutputBeam<FORMAT>{} and Triple<FORMAT>{}
    - Averaging background rates, determining the number of siderostats consistent with a number of baselines, finding star diameters, calculating uvw, rotating horizontal coordinates to earth coordinates, rotation earth coordinates to star coordinates, time weighting, angle weighting, time+angle weighting

HDataAux<FORMAT>{}:
- This class template contains static member functions that call specific groups of HData<FORMAT>{} member functions to getting data from files.
- This class template was created in order to simplify and shorten the load() member functions of the NPOI-based classes.

MathLocal{}:
- This class contains static member functions that perform common mathematical calculations
    - min and max, normalization, mean, standard deviation, Bessel function $J_1(x)$ and related functions, great circle distances

Utility:
- Utility.cc and Utility.h do not actually contain a class called Utility{}, they contain C-like utility functions
- file type ID, character string splitting
- If more utility functions are added, it would be beneficial to create a Utility{} class with static member functions

5.0 Use

In this section, I briefly outline the installation and use of the cuttlefish package.

5.1 Installation

The README file in the npoi directory contains the install information. After untarring and gzipping the code distribution, which includes all catalogs as well, go to the npoi directory and type:

export NPOI=`pwd`

The code needs this environmental parameter so that it knows how to locate various data files. Load the Makefile in this directory into an editor and change the NPOI directory variable (use the complete path). This Makefile compiles not only the cuttlefish code but the HDS4 library as well. HDS4 requires a FORTRAN compiler. You must choose either gfortran or g77 (and the associated libraries) in the Makefile. Make sure that the desired FORTRAN package is installed! After exiting from the editor, type:

```
make
```

Everything should compile. If not, contact Nick Elias. NB: This code has been compiled on Fedora and Ubuntu virtual machines. I have not been successful compiling the code in cygwin because the make program does not work as expected.

After compilation, the linux path must be appended. Go to the bin directory under the npoi directory and type:

```
export PATH=$PATH:`pwd`
```

Now you can run the executables on any file in any directory that contains *.con and *.cha files.

5.2 Helper Programs

The cuttlefish package contains many "helper programs" so that users can quickly and easily learn about the contents of *.con and *.cha files. If you type just the program name, you will receive a message on how to use the program. The outputs go to the terminal, but they can easily be piped into ASCII files. Unless otherwise specified, all helper programs can be used with *.con and *.cha files. Here is the list:

- bgSum: This program returns the backgrounds from a *.con file.

- calTAmp: This program returns the calibrated calibrator triple amplitudes from a *.cha file.
    - **NB: Make sure that all calibrators for a particular dataset are contained in diameter.txt .**

- calTPhase: This program returns the calibrated calibrator triple (closure) phases from a *.cha file.
    - **NB: Make sure that all calibrators for a particular dataset are contained in diameter.txt .**

- calVis2: This program returns the calibrated calibrator squared visibilities from a *.cha file.
    - **NB: Make sure that all calibrators for a particular dataset are contained in diameter.txt .**

- conLog: This program returns the constrictor log.

- eoLast: This program tells the user the date of the last IERS measurement and the date of the last prediction. The latter always comes after the former. These dates change as the finals2000A.all file is updated.

- genSum: This program returns general configuration.

- geoSum: This program returns the geodetic parameters.

- headSum: This program returns the header information.

- ibcSum: This program returns the input beam configuration information.

- lSec: This program returns the number of leap seconds removed by cuttlefish.
    - If there has been a recent leap second added, it must be manually added to the tai-utc.dat file.

- obcSum: This program returns the output beam configuration information.

- obsLog: This program returns the observer log.

- sdSum: This program returns the scan data information.

- starAlias: This program returns star IDs aliased to a specified catalog.

- starCal: This program returns the star IDs in a *.con or *.cha file that are recognized as calibrators. If you don't see a calibrator in the output, you must manually add it to the diameters.txt file.

- tcSum: This program returns the triple configuration information.

5.3 Data-Processing Programs

The cuttlefish package contains two "data-processing" programs. If you type just the program name or the program name plus the --help switch, you will receive a message on how to use the program. There are terminal outputs that can be piped into ASCII files. Here is the list:

- con2cha: This is the main data-processing function which processes data in *.con files and writes them to *.cha files. There are a number of switches that control the processing. See the program help for details. **NB: Make sure that all calibrators for a particular dataset are contained in diameter.txt .**

- cha2ascii: This program reads a *.cha file and writes the data to ASCII files called *.v2 and *.triple. The data are organized as fixed format columns that can be loaded into editors, manipulated by linux ASCII file programs, and imported into spreadsheets. See the program help for details.

5.4 Typical Session

Before running the con2cha program of the cuttlefish package, you should know as much as possible about the *.con dataset. At a minimum, you should run conLog and obsLog to look at the constrictor log (a list of constrictor input parameters and how they are set) and the observer log (a diary of setup and observations) to insure that the data are of sufficiently good quantity.

You should also insure that the three internal cuttlefish files are up to date.

- Run lSec or look at the last line of the tai-utc.dat file. If the number of leap seconds is incorrect, add it manually. Leap second information can be found on the IERS or USNO web sites.
- Run eoLast or look at the finals2000A.all file. If the Earth orientation information on the night of the *.con file is incomplete or a prediction, you can add it manually. The legend for the finals2000A.all file is the readme.finals2000A file. Earth orientation information can be found on the IERS or USNO web sites.
- The most important internal file is diameter.txt, which contains the angular diameters of calibrator stars. There is no way that con2cha can determine which stars are calibrators from the *.con file (this information is not stored there; it may be in the future). Therefore, the user should know *a priori* which stars are calibrators and check to see if they are in the diameter.txt file. The star names in the diameter.txt file use only the HD catalog, so any non-HD stars must be converted to HD by querying simbad or running the starAlias helper program. If any calibrators do not appear in the diameter.txt file, add them. Also, make sure that the file doesn't have duplicates or that any of your science targets appear in diameter.txt.

With the preliminaries out of the way we can run con2cha, which is the main processing step of cuttlefish. Here is a typical command:

```
con2cha 2013-04-20.con --bias 3 --weight 1 --trim 0.03 --timehalf 0.33 –userid NICK &> gronk
```

The con2cha executable should already be in your path if you installed cuttlefish correctly. This command should be executed in the directory where the 2013-04-20.con file is located. The 2013-04-20.cha file will appear in the same directory. All information written to standard output and standard error will appear in the file gronk (the &> redirects the information).

The bias mode is 3, which corresponds to the power-law bias fit (actually, a linear fit of the log of the data). You should always set the mode to a non-zero value. Mode 0 is the default, which corresponds to not removing the bias (bad). When the optimum bias fitting algorithm is found, it will become the new default. The weight mode is set to 1, which corresponds to weighted means (the weights are the variance inverses). Weight mode 0 calculates standard (unweighted) means.

The trim parameter is set to 0.03, which means that 3% of the low point data and 3% of the high point data are trimmed before scan averaging. This value seems to work well for the datasets I've tested. The timehalf parameter is set to 0.33 hours (20 minutes). This is the characteristic smoothing time for applying the system squared visibilities, i.e., calibration. The user ID is optional.

2013-04-20.cha is a completely processed file, assuming that everything has gone as planned. Cuttlefish does not yet provide visualization facilities. *.cha files can be viewed and manipulated in oyster, which also has modeling and imaging capabilities. Alternatively, we can run the cha2ascii program in the cuttlefish package. Here is a typical command:

```
cha2ascii 2013-04-20.cha
```

This program produces two output files, 2013-04-20.v2 and 2013-04-20.triple, which contain the scan-averaged squared visibilities and triple amplitudes/phases, respectively (both uncalibrated and calibrated). They are fixed-format ASCII files with a large number of rows and columns. They also contain supplementary information, such as wavelengths, baseline IDs, baseline vectors, etc.

These ASCII files can be viewed with an editor, which tends to be a bit tedious. If they are imported into a spreadsheet, however, it is possible to employ the powerful freeze pane, hiding, sorting, and plotting capabilities for visualization. Apart from editing, modeling, and imaging, this strategy provides as much (or perhaps more) flexibility for manipulating scan-averaged data than oyster.

6.0 Future TBD

In this section, I enumerate tasks that should be undertaken in future contracts. They are ordered in approximate order of priority.

I list the benefits of the changes for all types of users and developers. The benefits are either "Excellent," "Superior," "Good," or "OK." They can be used as metrics to determine scheduling priority.

I list the impact (translate: difficulty or the amount of extra work required) of these changes according to general users (running only linux command-line code), database developers (linking cuttlefish to the NPOI database), and cuttlefish code developers (writing new code at all levels). The impacts are "None," "Minimal," "Moderate," and "Significant." They can be used as metrics to determine scheduling priority. Because of the object-oriented design of cuttlefish, the impacts are mostly minimal. The few significant impacts are reserved for cuttlefish code developers.

6.1 Reduce size of a few member functions that are longer than 100-150 lines

- **Background:**
    - As the complexity increased, so did the size of some member functions.

- **Benefits:** Good.

    - The code will be easier to understand and maintain.

- **Tasks:**
    - Identify functions that are too long.
    - Come up with meaningful non-random ways to split them into smaller functions.
    - Modify the code (a lot of cut and paste).
    - Move existing flagging capability to separate functions.

- **Impacts on general users:** None.
- **Impacts on database developers:** None.
- **Impacts on cuttlefish code developers:** Moderate.

6.2 Create an Information{} class

- **Background:**
    - Well-designed software has well-designed integrated classes to trap errors and print warning/error messages to terminal, file, and GUI.
    - The present version of cuttlefish does some error trapping and prints messages, but implementation is not consistent.

- **Benefits:** Good.
    - Improved communications with users.
    - Improved bug finding and fixing.

- **Tasks:**
    - Design the class (already done).
    - Make sure that the class can handle sending error messages to different places.
    - Write class code.
    - Locate all places in package where calls to class are required and make the required modifications.

- **Impacts on general users:** None.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Moderate.

*6.3 Improve the code that writes averaged data to *.cha files*

- **Background:** The code appears to be redundant.

- **Benefits:** OK.
    - The code will be easier to read and understand.

- **Pre-Conditions:** None.

- **Tasks:**

    - I will investigate. Only the private member functions are affected.
    - I will improve the code.

- **Impacts on general users:** None.
- **Impacts on database developers:** None.
- **Impacts on cuttlefish code developers:** Minimal.

6.4 Improve the speed of averaging and calibration

- **Background:**
    - One of the top-level design considerations was to create NPOI-based classes that handle different data types, to insure that the interfaces were not too complicated. They ~ coincide with how data are stored in *.con and *.cha files.
    - When processing data, this scheme leads to a few redundancies where code must be rerun because the data from intermediate steps is not preserved or saved.

- **Benefits:** Superior.
    - I have no exact estimates, but I believe that that overall processing speed will increase by at least 50%.

- **Tasks:**
    - Determine which scheme is superior, saving temporary intermediate data in memory or files.
    - If the file scheme is superior, the format of the temporary components must be decided. Will they be deleted from the *.cha files eventually?
    - Redesign the code.
    - Write the code.
    - Test the code for accuracy and speed improvement.

- **Impacts on general users:** None.
- **Impacts on database developers:** None.
- **Impacts on cuttlefish code developers:** Moderate.

*6.5 Regularize the format of *.cha files*

- **Background:**
    - The format of *.cha files created by oyster are not consistent with the format listed on http://www.eso.org/~chummel/oyster/manual/node250.html. Also, some of the component descriptions are incomplete.
    - There are also some components in the *.cha files that are instrument dependent (e.g., VLTI).
    - Some new data components are required, e.g., flags to determine how the data were averaged, bias corrected, and calibrated; estimated star diameters and system visibilities; etc.

    - Some of the geodetic parameters do not appear consistent.

- **Benefits:** Good.
    - Less confusion about what *.cha files actually contain.
    - Confusing kludge code for handling missing components in *.cha files can be removed.

- **Tasks:**
    - The NPOI group, Christian Hummel, and I must agree upon and document the *cha file format. E.g.,
        - Add separate flag fields instead of using negative errors (computer memory is no longer an issue)? In cuttlefish, this scheme also involves changing the low-level math code in addition to obvious file format issues.
        - Add fields describing the bias fit output parameters.
        - The system visibility of calibrators.
        - Regularize the geodetic parameters.
        - Etc.
    - Eliminate confusing kludge code in load() and write() functions.
    - Add code to read and write new data components is located in load() and write() functions.
    - Add data query member functions.

- **Impacts on general users:** None.
- **Impacts on database developers:** None.
- **Impacts on cuttlefish code developers:** Minimal.

6.6 Improve bias removal

- **Background:**
    - This is an ongoing project within the NPOI group to improve processed data quality.
    - Existing V^2 bias removal can either be Poisson noise, linear fit, or log-linear (power law fit).
    - Existing triple bias removal is Poisson noise.

- **Benefits:** Excellent.
    - Improved processed data quality.

- **Tasks:**
    - Algorithms must be developed and tested for both V^2 and triples.
    - I will investigate the best way to fit and remove bias.
    - I will combine my efforts with the NPOI group.
    - Add an extra possible value to the --bias switch from the linux command line.
    - Add an extra possible value to the bias input parameter to the average() friend functions of the OutputBeam<FORMAT>{} and Triple<FORMAT>{} class templates.

    - Add member functions. It will be relatively simple because the object-oriented design dictates exactly where and how the code is created and called.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Moderate.

6.7 Read point data from other programs

- **Background:**
    - Dave Mozurkewich and Anders Jorgensen have created programs to create files containing point data.

- **Benefits:** Superior.
    - It would be nice to compare point data produced by different programs.

- **Tasks:**
    - Major discussions are required to determine the design and file formats.
    - I need a function to determine the format of these files.
    - Design the new load() functions for the NPOI-based classes.
    - Write the new load() functions.
    - Test the new load() functions.

- **Impacts on general users:** None.
- **Impacts on database developers:** None.
- **Impacts on cuttlefish code developers:** Moderate.

6.8 Create abstract base class for some or all of the NPOI-based class templates

- **Background:**
    - After the initial design, a requirement was added to be able to read raw data and (in)coherent averages from code created by Dave Mozurkewich and Anders Jorgensen.
    - After a number of class templates were created, it became clear that some member functions are redundant.

- **Benefits:** Superior.
    - It will be easier to incorporate code to read raw data and data from programs created by Dave Mozurkewich and Anders Jorgensen.
    - Redundant member functions can be eliminated, making code maintenance easier.

- **Tasks:**
    - It may be better to modify the NPOI-based classes first such that only their load() and write() member functions are templated. TBD.
    - Identify redundant member functions.

    - Create the abstract base class.
    - Eliminate redundant member functions in child classes.

- **Impacts on general users:** None.
- **Impacts on database developers:** None.
- **Impacts on cuttlefish code developers:** Minimal.

6.9 Only the load() and write() member functions are templated, not entire classes

- **Background:**
    - The original design of the class only read *.con files and wrote ASCII files. The task of reading the data from a *.con file was distributed among data-query public member functions. Thus, the NPOI-based classes were entirely templated.
    - With the additional requirement of being able to read and write *.cha files, I decided to group all reading and writing tasks into load() and write() member functions, respectively. Thus, only the load() and write() member functions need to be templated, not the entire classes.
    - I have written C++ code in CASA where only a few member functions were template specializations. It works well.

- **Benefits:** Superior.
    - Users and developers will not need *a priori* knowledge of which file format is used, which simplifies lines of C++ code that create class instances.

- **Tasks:**
    - The HData<HDF5>{}class template specialization must be written and tested.
    - Create a function that can identify whether a file is in HDS4 or HDF5 format.
    - I will rewrite the package such that only the load() and write() functions are templated. This task represents a lot of work not only in the code but in the documentation.
    - Makefiles will need to be modified to handle the new *.cc files (now only *.tcc files exist for the NPOI-based classes).
    - Modify any lines of C++ code that create NPOI-based classes (easy).

- **Impacts on general users:** None.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Significant.

6.10 Improve test suite and make sure it can all be run by a single command

- **Background:**
    - A significant number of small test programs have been create to test different parts of the cuttlefish package. Each test program corresponds to a single small class or part of a larger class.

- Although these programs were sufficient during this initial contract, they are insufficient for long-term design and development.
- A new test suite is required.

- **Benefits:** Good.
  - Makes testing code more convenient and more accurate,

- **Tasks:**
  - Determine which test programs need to be create and their purposes.
  - Define testing metric parameters.
  - Collect *.con and *.cha files, suitable for testing purposes.
  - Write the code.
  - Create scripts to run one or more of the test programs.
  - Test the test code.

- **Impacts on general users:** None.
- **Impacts on database developers:** None.
- **Impacts on cuttlefish code developers:** Significant.

6.11 Link the C++ code to the GDL/IDL or python interface

- **Background:**
  - Cuttlefish was originally designed as an automated pipeline.
  - I added hooks to cuttlefish in the first contract for an interactive mode. The Data<TYPE,DIM>{} class template was originally designed just for correctly accessing/slicing FORTRAN-ordered data in HDS4 files. I have been using it as the output for the data query member functions in the NPOI-based class templates.
  - An interactive mode is most useful for visualization and experimentation.

- **Benefits:** Superior.
  - Visualization will allow users to see how well the data are processed.
  - Experimentation will lead to new algorithms that will eventually be implemented in C++.

- **Tasks:**
  - Decide whether to use GDL/IDL or python. Creating the interface will likely be easier for python, but python visualization is not as good as GDL/IDL. NB: Given enough time, I could even create interfaces to both GDL/IDL and python.
  - Decide how to return variables within C++ code and returned to the interface (Data<TYPE,DIM>{} or something else).
  - Design the interface for the HData<FORMAT>{} class template (for direct access to files) and the NPOI-based class templates (for dealing with specific types of data).
  - Write the interfaces.
  - Test the interfaces.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Minimal to Moderate, depending on the design.
- **Impacts on cuttlefish code developers:** Significant.

6.12 Create cuttlefish graphical-user interfaces

- **Background:**
    - Cuttlefish was originally designed as an automated pipeline.
    - I added hooks to cuttlefish in the first contract for an interactive mode.
    - I created a simple linux program allowing users to browse HDS files. It is very limited but I had no time in the first contract to create anything more elaborate.
    - INCHWORM had HDS browser and data-processing GUIs. They were very ergonomic and effective. The data-processing GUIs consisted of a main bar with buttons and drop-down menus that were activated only at the correct times as well as subGUIs (run from the drop-down menus) that corresponded to the C and [incr tcl] classes.

- **Benefits:** Superior.
    - Visualization will allow users to see how well the data are processed.
    - Visualization will allow debugging.

- **Tasks:**
    - Get the INCHWORM GUI working again in a virtual machine.
    - NB: The command-line interface must be created before work on the graphical-user interface can be done.
    - Design the HData<FORMAT>{} browser GUI. It should be similar in style to the one in INCHWORM.
    - Write the HData<FORMAT>{} browser GUI.
    - Test the HData<FORMAT>{} browser GUI.
    - Design the main data-processing GUI bar. It should consist of buttons and drop-down menus that are activated at the correct time.
    - Write the main data-processing GUI bar.
    - Test the main data-processing GUI bar.
    - Design the subGUIs. Each subGUI should correspond to an NPOI-based C++ class and interface class.
    - Write the subGUIs.
    - Test the subGUIs.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Significant.

6.13 Create OI-FITS output

- **Background:**
  - OI-FITS is the standard for data transport between optical interferometry data-reduction packages.
  - Oyster has OI-FITS capability.

- **Benefits:** Good.
  - Making it easier for non-NPOI staff to view and model NPOI data.

- **Tasks:**
  - Design the code. E.g., each NPOI-based class write part of an OI-FITS file or create a separate OI-FITS class that uses instances of NPOI-based classes.
  - Write the code.
  - Consider loading OI_FITS in the future?

- **Impacts on general users:** None.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Significant.

6.14 Add additional flagging capability

- **Background:**
  - Oyster has interactive flagging capability of point- and scan- averaged data.
  - Because cuttlefish was originally written as a pipeline, it has limited flagging capabilities at this time (only too few or no data are flagged).
  - Cuttlefish needs automated flagging algorithms (looking for outliers, etc.).
  - NB: It is normally better to flag in raw data compared to average data, even if the raw data have a much lower S/N because their large number allows for better statistical analyses before flagging begins. Also, some point- and scan-averages contain some good and bad data, so flagging them reduces the science potential of a dataset.

- **Benefits:** Good.
  - Improved data quality.

- **Tasks:**
  - Find and test flagging algorithms that will work well on point data. Flagging scan-averaged data is very difficult (compare a datum to what?), so flagging them should only be done when there are no unflagged point data.
  - Design the member functions.
  - Write the member functions. Call them from the averaging friend functions.
  - Add flagging input parameters to the averaging friend functions.
  - Add flagging parameters to the linux command-line code.

- **Impacts on general users:** Minimal.

- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Significant.

6.15 Add coherent average capability

- **Background:**
    - Coherent averaging is ongoing research within the NPOI group for improved S/N and imaging (just like at radio wavelengths).
    - I am interested in it because it provides sufficient S/N and phase information for full-Stokes OIP (just like at radio wavelengths).

- **Benefits:** Superior.
    - Phases for various purposes.

- **Tasks:**
    - Coherent averaging by cuttlefish is not possible unless 1) the raw data are available; or 2) coherent point data can be averaged. At present, it would be best to employ the software of Dave Mozurkewich and Anders Jorgensen.
    - Unless additional raw data capabilities are added to cuttlefish, all it will do is read and write coherently averaged data.
    - Finalize file format for coherent averaging components.
    - Design the code.
    - Write the code.
    - Test the code.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Moderate.

6.16 Read/process raw data

- **Background:**
    - NPOI produces raw data.
    - Constrictor has been the main program for processing raw data into point data (~ 1 second averages). It is showing its age because it is difficult to add new features.
    - Dave Mozurkewich has created a program to organize raw data packets into scans. He has also created programs to process raw data into point data, both incoherently and coherently.
    - Anders Jorgensen has created a program to process raw data into point data.
    - A new standard version of the software is required.

- **Benefits:** Superior.

- **Tasks:**
    - Major discussions are required to determine the best design.

        - Modify and use constrictor.
        - Modify and use Dave's and Anders' code.
        - Incorporate the features of Dave's and Anders' code into cuttlefish.
    - Formalize the best design.
    - Develop the code.
    - Test the code.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Moderate.
- **Impacts on cuttlefish code developers:** Moderate or Significant, depending on design.

6.17 Create the HData<HDF5>{} class template specialization

- **Background:**
    - NPOI has been using the HDS4 libraries for almost 20 years.
    - The original HDS4 build system is completely broken. I had to create a new minimalist build system to get everything to compile.
    - A lot of warning messages come up for VAX-specific code. This code doesn't affect us because we don't use those particular functions, but it is a symptom of age.
    - The Hawai'i group took HDS4 over from starlink, but I've never been able to get the new version to compile. Plus, it's intertwined with other stuff that we don't need.
    - HDF5 is conceptually similar to HDS4.

- **Benefits:** Good.
    - HDF5 is more stable, has a reasonable build system, and has a large user base.
    - None of the code that calls the HData<FORMAT>{} class template needs to be modified.
    - Users won't need to know whether a file is in HDS4 or HDF5 format.

- **Tasks:**
    - Create a helper function to identify HDS4 and HDF5 files. It will be called before opening a file.
    - Create the HData<HDF5>{} class template specialization.
    - Replace HData<HDS4>{} with HData<FORMAT>{} (unspecialized) in higher level code.

- **Impacts of general users:** None.
- **Impacts on database developers:** None.
- **Impacts on cuttlefish code developers:** Moderate.

6.18 Create helper programs for leap seconds and Earth orientation

- **Background:**

    - NPOI data-reduction, whether imaging or astrometry, uses leap seconds and Earth orientation information.
    - Cuttlefish get this information from ASCII files.
    - It would be useful to have helper programs to return the latest leap second and earth orientation parameters.

- **Benefits:** Good.

- **Tasks:**
    - Create leapSecond helper function, call existing classes.
    - Create eo helper function, call existing classes.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Moderate.

*6.19 Create pipeline shell script to process all *.con files in a directory*

- **Background:**
    - Although cuttlefish has hooks to be operated in many different ways, its original intent is for pipelines.
    - There are a lot of NPOI data that need to be processed in a uniform manner.

- **Benefits:** Good.

- **Tasks:**
    - Create shell script to send all *.con files in a directory to con2cha, run all helper programs and pipe to ASCII files, and run cha2ascii.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Moderate.

6.20 Create reorganized classes

- **Background:**
    - The present high-level classes are based on the structure of *.con and *.cha files.
    - This scheme is fine, but it may be possible to improve it.
    - For example, put the photon numbers in their own class.
        - Good for users at the CLI/GUI
        - Good for developers, but more classes means more interactions between classes, so proper design is a must!
    - Many high-level classes have common member functions, so an abstract base class may be in order.

    - Because cuttlefish has been designed and written in an object-oriented manner, the original classes can be kept as long as desired (perhaps indefinitely) along with the new classes.
    - The Data<TYPE,NUMDIM>{} class templates, which are not high-level classes, may need to be redesigned as well when the CLI has been selected.

- **Benefits:** Good.

- **Tasks:**
    - Design new classes and/or modify new classes.
    - Write the code.
    - Test the code.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Significant.

6.21 Create better calibrator list

- **Background:**
    - The present calibrator list is a hodge-podge of different values that may or may not be consistent. Some have been commented out and replaced.
    - Users can (and must) add there own calibrators that do not exist in the file.
    - It would be beneficial to have a consistent list of calibrators.
    - The present high-level classes are based on the structure of *.con and *.cha

- **Benefits:** Superior.

- **Tasks:**
    - Gerard van Belle will run SED fits to spectroscopic data of a large list of stars.
    - They will be agreed upon by the NPOI group.
    - The list will be incorporated into cuttlefish.

- **Impacts on general users:** Minimal.
- **Impacts on database developers:** Minimal.
- **Impacts on cuttlefish code developers:** Minimal.