The Allen Telescope Array Beamformer

Concise Operational Guide

"Forming better beams since 2008"

William C. Barott, PhD
For the SETI Institute

Document revised on 5/17/2012

Questions, comments, or other inquiries, contact author Billy Barott
barottw@erau.edu

Summary and Introduction

Calibration. Pointing. Tracking. Interference nulling.

This document contains user-level instructions on operating the ATA beamformer. At its heart, the beamformer is a rather simple instrument: Pick a direction on the sky, form a beam, and if you're ambitious, null some interferers in the process. The major functions required before observing are initialization and calibration.

The following pages give a description of the major beamformer functions and examples of how to use them, with the hope of getting a new user "on-air" as quickly as possible. Additionally, advanced information is available for those who wish to take the beamformer a bit further. Power-user information takes this a step further for those who wish to delve into the full customizability of the beamformer.

The beamformer's primary capability is to form four discrete single-polarization beams anywhere on the sky. Additionally, it can serve as a spectrometer with its built-in 16,384-channel auto-correlator (6.5 kHz resolution) on each branch node. Additional capabilities include *N* cross-correlation spectra and *N* auto-correlation spectra, each of which can be written to disk. Diagnostic capabilities include logging and writing various link-related data to the disks.

This document is maintained on log.hcro.org under the beamformer systems page.

Recent Major Changes

May 2012

The main elements of this update are to introduce the syntax for the SonATA packetizer commands and control and to add detail about more-advanced diagnostic and troubleshooting switches.

December 2008

Several major changes were introduced in the Dec 21-30 upgrades. These are summarized below (see the full documentation for details).

Circular Polarization: Single-pol beam processors now terminate in a CP concatenation stage. By default, this stage is ignored during calibration and the linearly polarized beams pass through unchanged. See new flags (ie, pol, calpol) to activate this stage.

New Beam Calculator: The engineering beam calculator group has been moved to the standard library, and a new beam calculator *beamPhasedPolBias* has been added. It functions similarly to beamDefaultPhased except that it adds the degree-value of the variable *polbias* to all inputs identified as "y" polarizations. The purpose is to provide a simplified source polarization model during calibration, and this calculator should not be used for observing.

DAC iBob Control: The bfibob script now has options for the DAC iBobs. It is necessary to run bfibob with the *dacinit* option to tune DACs to the correct beam. This must be done after each physical power-up (recommended to run at the same time as *bfibob walsh)*. The option *dacstatus* gives some status information about the DACs.

Peak Register Display: Peak registers give one output per beamformer rather than one per beam processor. This is by design to keep the single-pol beams at the same relative gain levels. The reported value is the maximum of the two single-pol beams.

Reference Antenna Selection: The method of refant selection has changed from an independent node-by-node analysis to a holistic analysis of the entire beamformer. This change facilitates scored refant selection based on noise temperature. Additionally, the beamformer prohibits combinations resulting in the correlation of two inputs from the same antenna (i.e., 1ax with 1ay).

Noise Temperature Inputs and Alarms: The file ata.tsys stores noise temperature data for each input and is now read during initialization. BFTrackepehm will now alarm out of range noise temperatures in the same manner that it alarms calibration errors. These alarms and warnings are generated from Tsys thresholds relative to the best-performing antenna in the hookup.

A Few Commands to Look At: See documentation for the following commands:
--calpol, --pol, --specsrc, --specshift, --corrsb, dacinit, dacstatus

Tutorial for Circular Polarization

The following procedure can be used for circular polarization calibration.

1. Select a calibrator and determine its phase offset. This phase offset is critical: It determines the expectations of the source and ultimately whether the beamformer will indeed produce circularly polarized beams when it is done. For example, strong circular sources would be +/- 90, and linear sources 0/180. The degree of ATA cross-pol isolation might result in source-dependent tweaks.

2. Calibrate the beamformer using the beamPhasedPolBias calculator. Remember that circular calibrations generally take longer, since sources are weakly polarized or unpolarized. There's not yet a good sense for how much longer.

```
bftrackephem.rb -b 1 -f 1420 -e calibrator.ephem -i 10 --caldelay  ← Old
--calcp                          ← New: Enables calibration of CP top node.
--bctype beamPhasedPolBias       ← New: Sets the beam calculator to the CP helper.
--bcvar polbias=X                ← New: Sets the pol bias for the bc, where "X" fits your source.
```

3. Observe using beamDefaultPhased (or your favorite calculator) configured for circular polarization.

```
bftrackephem.rb -b 1 -f 1420 -e target.ephem          ← Old
--pol circular                   ← New: Selects circularly polarized output
--bctype beamDefaultPhased       ← New: Set your calculator back to normal
```

4. Done!

Table of Contents

The iBobs: Setup

Prior to any observing or tests, the beamformer iBobs must be prepared. The script bfibob accomplishes this task, and has many functions, divided into several categories. It is located in the /home/obs/ruby/bin directory and is included in the path of the obs account on most machines. The general syntax and commands are detailed below.

```
bfibob [beamformer] [command] [options]
```

status
: Verifies communications and shows the walsh and source settings.

Phase Switching (walsh functions)

walsh
: Applies phase switching to all iBobs. Required if the analog walsh function generator is enabled.

nowalsh
: Turns off the phase-switching circuitry of the iBobs

rearm
: Resynchronizes the iBob internal clocks with 1pps

Data Source Selection

sky
: Configures the iBobs to output sky data from the telescopes

fs
: Configures the iBobs to output sine wave data

noise
: Configures the iBobs to output pseudo random sequences of data

reseed
: Commands the iBobs to synchronize their random sequence generators. Required for the noise from multiple iBobs to be coherent.

Attemplifier Control

autoatten [level]
: Commands the iBobs to auto-adjust their inputs, seeking an RMS sample value of [level]. If level is omitted, a value of 11 counts RMS is sought.

getatten
: Displays the current attemplifier values for all iBobs

adcstats
: Displays the current ADC counts (rms input level) for all iBobs

DAC Control

`dacstatus`

Retrieves the current status report for the DAC iBobs on the current beamformer. Displays selected channels as well as link status.

`dacinit`

Initializes the DAC iBobs to draw data from the correct streams. DAC #1 receives X/Right and DAC #2 receives Y/Left. Resets link status registers.

Beamformer Initialization

The beamformer must be initialized prior to use or any time the hookup is changed (by rewiring or changing which antennas are used in the observation). To initialize the beamformer, ssh to a user terminal (like user1) and use the following command.

```
/home/obs/bbarott/bfu/bfinit.rb [args]
```

Basic Arguments

`-a [antenna list]` **`--ants [antenna list]`**

Specifies the list of signal paths to enable. Names should be comma separated and follow the ATA format, plus dupe codes (e.g. 5gx1). Partial names are allowed, so "1a" will enable *all* hookups beginning with 1a.

`-b [beamformer]` **`--beamselect [beamformer]`**

Selects one of two beamformers (1 or 2) to initialize.

Simple Example

To initialize beamformer #1, enabling the antennas 1a,1b,1cy. The partial naming for 1a,1b will enable both polarizations (x and y) but the specific naming on 1cy will only enable the y polarization. It will be necessary to calibrate before observing.

```
> bfinit.rb –b 1 –a 1a,1b,1cy
```

Advanced Arguments

`-r [antenna list]` **`--remove [antenna list]`**

Removes the list of antennas from the hookup. Allows partial names. Implies --preserve.

`--preserve`

Specifies that modifications are made without reinitializing the beamformer. When used with –a, the antenna list is added to (rather than replacing) the current antennas in the hookup. While this command preserves calibration data, it is possible for instrumental link delays to change as firmware is still re-loaded.

`-h [value]` **`--headroom [value]`**

Controls the linear headroom for the automatic gain controller, which itself is invoked in bftrackephem. Time domain samples have magnitudes between 0.0 and 1.0. The AGC scales data so that the peaks have a value of 1.0*value. For example, a value of 0.75 (the default, if –h is omitted) means that data will be scaled to have peak values between -0.75 and +0.75, leaving the range between 0.75 and 1.0 as white-space headroom reserved for gain variations or RFI.

Example implementation:

```
bfinit.rb –b 1 -a 1ax,1bx,1cx,1dx
```

Please see the hookups as previously distributed. All currently hooked-up antennas and iBobs are currently supported.

Power User Arguments

`--bandwidth [new fractional bandwidth]`

Changes the beamformer's operating bandwidth, as defined by the fractional delay filter.

Packetizer Arguments

The beamformer is now integrally linked with the Sonata packetizer system. While the two operate under separate daemons, control is integrated through the bfinit and bftrackephem interfaces.

`--spk`

Command the SonATA packetizer to be invoked and initialize. If omitted, the passthrough firmware is installed instead. When using the packetizer the destination addresses should be set with the ip switches.

`--ip0 [IP:PORT]` **`--ipr [IP:PORT]`** **`--ipx [IP:PORT]`**

Set a destination for the stream0 data in the beamformer output stream for the SonATA packetizer. These switches are synonymous and have no discrete meaning since the beam definition calls for both "x" (in linear mode) and "r" (in circular mode) to be stored in stream0.

`--ip1 [IP:PORT]` **`--ipl [IP:PORT]`** **`--ipy [IP:PORT]`**

Set a destination for the stream1 data in the beamformer output stream for the SonATA packetizer. These switches are synonymous and have no discrete meaning since the beam definition calls for both "y" (in linear mode) and "l" (in circular mode) to be stored in stream1.

Beamformer Tracking

The bftrackephem command directs the beamformer to either calibrate or observe (in a sense, calibration is just observing with self-adjustment). This command can be executed from the same machine as bfinit.

```
/home/obs/bbarott/bfu/bftrackephem.rb [args]
```

Basic Arguments

`-b [beamformer]` **`--beamselect [beamformer]`**
Selects one of two beamformers (1 or 2) to use for observing. Required.

`-f [frequency]` **`--freq [frequency]`**
Sets the beamformer and ATA sky frequency (enter in MHz). The sky frequency is software-limited at 15 GHz. If omitted, the beamformer will use the last sky frequency set with –f (1420 at initialization). Note that the beamformer can only learn of a sky frequency setting from this command, and does not use the current LO oscillator setting to determine sky frequency.

`-e [filename]` **`--ephem [filename]`**
Specifies the ephemeris file for the observation. Format is the same as that generated by ataephem. The path may be relative, but must be on a file system mapped by boot2 (e.g. /home/obs). The colon character [:] is not a valid file name character, and its presence will change the function of this command (see advanced functions).

`-d [duration]` **`--duration [duration]`**
Sets the duration of the observation, in seconds. Ephemeris is padded up to 100 seconds on each side. Unless overridden by –n, bftrackephem will not terminate until this duration has elapsed. Pointing data is preloaded, so long values of –d might cause a long delay before tracking starts. If not specified, this will attempt to choose a magic number based on the calibration, integration, snapshot write, and number of iteration switches. Omitting -d is not recommended for modes other than calibration. See the quickload switch for instructions on conducting long observations.

`-n [cycles]` **`--num [cycles]`**
Overrides the duration with respect to how long the beamformer operates. The beamformer will only perform software updates *cycles* times before terminating. Does not modify the loaded ephemeris. Software limited to be between 1 and 1000; settings outside this range revert to 3.

`--caldelay / --calphase / --calfreq / --calamp`
Causes a calibration cycle to occur, defaulting to 3 iterations unless overridden by –n. Here, -d applies to each iteration individually. Modes include: caldelay (delays and phases), calphase (phases only, assuming no

delay errors), calfreq (phases only, offset from calphase frequency), calamp (balance amplitudes, use with short integrations). All modes imply automatic snapshots.

-i [time] **--integrate [time]**

Specifies the integration time, in seconds, for snapshots and calibrations. See advanced *memtimer* for use with non-calibrating snapshots.

--agcdb [decibel output level]

Implements the automatic gain controller. Argument is dB relative to full-scale range (determined by bfinit –h). This is usually negative or zero. Note that digital clipping levels are +/-127, with an analog range of 2 volts peak to peak. The AGC is executed based on the last snapshot, and must be used in conjunction with a calibration. AGC gains are persistent.

Simple Example

To start beamformer #1 on an observation. Set the sky center frequency to 1420 MHz and load 10 minutes of ephemeris from the file indicated (relative to the current path).

```
> bftrackephem.rb –b 1 –f 1420 –d 600 –e ../myephem/mystar.ephem
```

To start beamformer #1 on a delay calibration. Set the sky center frequency to 1420 MHz and load 5 minutes of ephemeris for each of three iterations on casa. Integrate for 10 seconds, and adjust the AGC during each step to try to achieve -3dB-pk output levels.

```
> bftrackephem.rb –b 1 –f 1420 –d 300 –n 3 –e casa.ephem
                  –-caldelay –i 10 --agcdb -3
```

Advanced Arguments

Observational

--azel [az,el]

Alternative method of specifying pointing (versus –ephem). Comma separate (no space) the azimuth and elevation coordinates (degrees). *This argument is being depreciated in favor of –e [az:el] below. It is not recommended for new scripts, and will soon be removed.*

-e [az:el] **--ephem [az:el]**

There is a short-cut for making –e implement fixed-position pointing. This shortcut is required for beamcalc pointings, which have no separate azel command. Use a colon [:] to separate azimuth and elevation. For example, `--ephem 40.1:23.5 acts like --azel 40.1,23.5`

--write [mode]

Records and writes data snapshots to the disk. Modes include “out” “crosses” and “autos” and only one may be selected. Data is saved to the file atabf.bfs in a time-stamped directory for each activity. All data is stored under /opt/bfu/data/ on boot2.

--memtimer

Implements software coefficient updates during write cycle integrations. When omitted, coefficients are not updated (leading to errors during long integrations). Only optional for --write; calibration integrations enable the memtimer mode by default.

--sideband [sideband choice]

Changes the output mode. Valid choices are “upper”, “lower”, and “off”. This argument is persistent, and the power-up state is “off.” The “upper” mode centers the sky frequency at DC and increasing frequencies are higher sky frequencies. The “lower” mode centers the sky frequency at DC and increasing frequencies are lower sky frequencies. See --specshift for modifications to this. Persistent.

--specshift

An augmentation to sideband (can only be used when --sideband is present in the same call). Commands the beamformer to implement the 26.2144 MHz spectrum shifter. In sideband mode “upper,” the sky-frequency has a baseband value of 26.2144 MHz. Radio frequencies above Fsky are present above this (flipped in “lower”). --specshift has no effect when setting sideband “off,” and is persistent until the next --sideband call.

--offsetaz [offset]
--offsetel [offset]

Adjusts all pointings loaded during this cycle by the indicated offset, in degrees. Affects ephemeris tables and fixed pointings loaded with ephem and bcephem, but not with azel.

--timer [interval]

Causes the BEE processes to autonomously update coefficients with the given delay (in seconds) between updates. Releases control to the O.S. after starting the timer.

--quickload

Enables just-in-time beamforming. Normally, all table coefficients are calculated before the first beams are formed. For long observations this may create too much “startup” overhead, or may make the tables too large. JIT beamforming creates coefficients on-the-fly in updates at predetermined intervals. When the end of the current table is approached, old data is purged and new data is added. Note: JIT/BF is invalid for calibration cycles, and windows are scaled for integration times. If used in conjunction with –timer, data tables are updated in the background.

--pol [polarization choice]

Changes the polarization selection on the beamformer output. Valid choices are “linear” and “circular”. In linear, concatenated output streams will contain X/Y data. In circular, data is R/L. It is important that the user ensures proper calibration of the X/Y beams with each other by using calcp on a polarization-supporting source. Persistent.

--calcp **Synonym: --calpol**

Optional flag to allow analysis of the circular polarization (top-level) beamformer node. If omitted, all correlation analysis (including calibration) is restricted to second-tier and below, i.e. within co-polar nodes. When included, the beamformer will analyze correlations of cross-polarized x/y tiers, allowing for alignment and (possibly) combining of these streams into right/left streams. Not sticky. Note that this flag should *not* be used during observing, only during calibration for cp.

--calrevert

Reverts calibrations to the "power-on" state, removing any user-observed calibration coefficients. Used to recover from catastrophic calibration failures (such as interference effects on delays, or bad antennas causing amplitudes to collapse).

--beamephem [beam,filename]

Loads unique ephemeris to each beam group. Argument is an ordered list of beam group names and ephemeris files (or fixed pointings). All beam groups and ephemeris should be entered, separated by commas. Beam group names are the polarization and duplicate identification code for antennas in the hookup. For example, antenna "2ax1" belongs to beam "x1." Note that –ephem by itself writes to all beamgroups identically.

--beamazel [beam,az,el]

Similar to azel and beamephem. Give ordered triples of beam groups and azimuth/elevation directions, separated by commas.

Diagnostic

--statusonly

Prints the last-determined array health status to the screen and then terminates. This output is lengthy and content is subject to change.

--calalarms [type]

Commands normal snapshot writes, if invoked with –write, to assert calibration alarms (but does not adjust calibration offsets). Types are "delay" or "phase" to correspond to "caldelay" and "calphase" alarms.

--clearalarms

Resets link-related alarm conditions in the beamformer, so they will no longer be reported unless a new event occurs. Recommended only after the user has addressed the alarm condition to verify valid calibration.

Power User Arguments

--bandwidth [new fractional bandwidth]

Changes the beamformer's operating bandwidth, as defined by the fractional delay filter. This does not affect sample rate. The startup bandwidth is 0.7 (70%). Analog users may specify 0.5 for improved accuracy with no loss of output signal bandwidth. This is persistent. *The calibration correlators currently require a value of at least 0.4 unless set otherwise with --calbw.*

--agclin [linear output level]

An alternative to agcdb using a linear voltage scale. Equivalent with an argument of 1=0dB, 0.5=-6dB, etc. Values greater than 1 might result in clipping the output. This switch is persistent and absolute.

--gainadjustdb [decibel adjustment]

Adjusts the power gain of the output stage. Arguments are relative dB adjustments. For example, a value of -3 will decrease the power out of the beamformer by half (and is equivalent to gainadjustlin 0.707). Persistent.

--gainadjustlin [linear adjustment]

Adjusts the voltage gain of the output stage. Arguments are multiplicative relative adjustments. For example, a value of 1.2 would increase the gain by 20 percent. Persistent. Use with care to ensure no clipping.

--bctype [string]

Changes the type of beam calculator used in processing. This will affect the entire beamformer. Persistent. The primary style of beamforming is beamDefaultPhased. It is noted that the beamformer automatically forces the bctype to beamDefaultPhased when a calibration is invoked.

--bcvar [array]

Allocates variables to beam calculators. Separate variable names and their values by the "=" character, and sets of variables by commas. For example, --bcvar nullaz=40,nullel=20. Text assignments are permissible, with the exception of reserved characters. The value "nil" (no quotes) may be used to clear a variable, as in –bcvar rfi=nil. Note that "ephemeris" is a reserved name, and should not be used. Persistent.

--bcephem [array]

Loads ephemeris files into the beam calculator. A functional fusion of bcvar and ephem, with rules from both applying. For example: --bcephem null0=xm.ephem,null1=20:41.2. Ephemeris tables are persistent between calls to bftrackephem, but are only loaded for the interval given by -d. Variable names are shared with bcvar, so there cannot be duplication.

--bclib [library file]

Loads the given library file into memory so that it can be accessed by the beamformer. Loads are persistent and repeatable, and may override (but not delete) previously loaded code. File names must be relative to the current path. Use commas to separate library names.

--bcclear

Removes all beam-calculator variables from memory (as set with bcvar or bcephem). Does this prior to bcvar/bcephem directives in the current call.

--calbw [bandwidth]

Sticky. Sets the integration bandwidth for calibrations. This should be a value less than 1. As a point of reference, the power-up value is 0.4 (40% of the band), and this must be less than the value set with –bandwidth (or less than 0.7 if –bandwidth was not used). Higher values mean lower integration times but at the expense of possibly including more RFI.

--caloffset [offset]

Sticky. Sets the offset of the calibration's integration band relative to the center of the band. This has a range of -0.5 to +0.5, but it should be noted that this must be selected with care along with calbw. IE, if caloffset is 0.4 and calbw is 0.3, the software will try to integrate from 0.25 to 0.55 – not just outside the range of the filter but also entirely outside the range of the spectrum. Proper use of caloffset and calbw can allow the user to select calibration frequencies to avoid RFI. *Note: Positive values move the calibration center to frequencies above the sky frequency.*

--calstronglock

Not Sticky. For the current phase cal, causes the beamformer to "lock on" to the dominant frequency bins within the cross correlation spectra, allowing (possible) calibration on a satellite (if the position of the satellite is known well enough…). When this argument is omitted, the astronomical calibration method for phasecal is used – astronomical methods assume a Gaussian spread within the cross correlation spectra and discard outliers (like satellites). In astronomical calibration the reported confidence interval (CI) is the CI of the mean angle estimate. In strong lock calibration the reported CI is the RMS angle fit across the band. There is generally no large penalty for using calstronglock on an astronomical calibration, other than that the astronomical mode provides improved CI estimates and slightly improved angle estimates.

--nodiag

Suppresses diagnostic sampling during observing. Provided as a courtesy function only, this option is never recommended.

--noatalo

Suppresses calls to the atasetskyfreq and atagetskyfreq commands. This is useful when running two beamformers on the same LO. The –f command must still be used (to convey the frequency to the beamformer software).

--corrsb [tiers] --corrra [tiers]

Controls the data source for cross correlations in beamformer branch nodes. Give a comma separated list of tiers, where 0 is the top tier of the beamformer (ie, CP node) and successively higher numbers represent

deeper tiers. Corrsb configures nodes on these tiers to correlate the input subbeam streams. Corrra configures nodes on these tiers to correlate the input reference antenna streams. This selection is sticky, and corrra is the default mode after startup for all nodes. As with reference antenna selection, the choice here selects the data that is passed to the next node in the tree (so corrsb will pass the reference subbeam in the refant stream).

--nocompress

Turns off data table compression. Data compression is on by default. The data compression function analyzes pointing information and stores only the most important information (for example, boundaries in discrete values). This switch should never be used unless debugging operation in which data compression is suspect. Improper use of this can cause the daemons to crash. This switch is not persistent, and is reset to use data compression on successive calls to bftrackephem.

--nofringe

Turn off the fringe rotators. If invoked, the fringe rotators will be disabled and only static delays and phases will be applied. Not recommended unless the application requires disabling fringe rotators. This switch is not persistent, and is reset to use the fringe rotators on successive calls to bftrackephem.

--conjfringe

Conjugates the fringe rotators. If invoked, the fringe rotators will be set to move opposite of the normal direction based on the fringe rate calculation. The conjugation of the starting phase is unaffected. This is a diagnostic switch only and used to debug the fringe rotators. Should not be used unless debugging new firmware. This switch is not persistent.

--specsrc [source]

Configures the source for the output spectrometers used with the –write out switch. Valid choices are “internal” and “external”. Selecting “internal” causes spectrometers in each beamformer to use their own data, which will always sample linear polarization. Selecting “external” causes spectrometers in each beamformer to use combined data from the downstream CP nodes, which is fed (post-combination) back upstream to the spectrometers. Default is “internal.” Use “external” when operating in circular polarization scenarios. This setting is persistent.

--tiergain [gain,tier]

Explicitly set the gain to the specified value across all FPGAs at the specified tier. The default gain is 2 and the maximum permitted value is 15. This is a linear voltage gain. Tier 0 is the CP node, with tier numbers increasing to deeper nodes (branch nodes have the highest tier value). This should only be used in cases where the –agc commands do not produce the desired results and manual gain setting is required.

Packetizer Arguments

--spk

Commands bftrackephem to update values in the SonATA packetizer daemons. Sets the sky frequency (when –freq is invoked) and polarization header (when –pol is invoked).

*Example Implementations (*Older)*

Example implementation #1:

```
obs@user1 bbarott/bfu> ./bftrackephem.rb -f 1420 -d 600 -e
“/home/obs/bbarott/20080110/casa.ephem” -n 5 --cal –i 10
--agcdb -6
```

To calibrate the beamformer over five iterations, using the current hookup (set by bfinit) with a skyfrequency of 1420 MHz, 10-second integration time for calibrations, and pre-loading 10 minutes of ephemeris data for the track from the given file for casa. The calibration will not take 10 minutes, but will only take as long as necessary for 5 iterations. The automatic gain controller will adjust the output beam level to be 6 dB below the “safe” level.

Example implementation #2:

```
obs@user1 bbarott/bfu> ./bftrackephem.rb -f 1420 -d 300 -e
“/home/obs/bbarott/20080110/casa.ephem”
```

To cause the beam-former to track casa for five minutes at a sky frequency of 1420 MHz.

Example implementation #3:

```
obs@user1 bbarott/bfu> ./bftrackephem.rb -f 1420 -d 300 -e
“/home/obs/bbarott/20080110/casa.ephem” --write “crosses” -i 5
```

To cause the beam-former to track casa for five minutes and record all cross-correlation results to a series of files.

Example implementation #4:

```
obs@user1 bbarott/bfu> ./bftrackephem.rb -f 1420 -d 3600 -e
“/home/obs/bbarott/20080110/casa.ephem” --timer 5
```

To cause the beam-former to load an hour’s worth of tracking data for casa, and start the auto-update timer. Update control will be released to the BEEs rather than maintained in the user script.

Example implementation #5:

```
obs@user1 bbarott/bfu> ./bftrackephem.rb -f 1420 --azel 40.1,20.2 --
timer 5 –d 300
```

To cause the beam-former to load five minute’s worth of data for a fixed azimuth and elevation, and start the timer for automatic updates.

Example implementation #6:

```
obs@user1 bbarott/bfu> ./bftrackephem.rb -f 1420 --beamephem
x1,../ephem/casa.ephem,x2,../ephem/3c84.ephem –d 600
```

To cause the beam-former to load ten minute’s worth of data. All enabled antennas belonging to the “x1” beam (e.g., 1ax1, 1bx1) will be pointed to the ephemeris given by

casa.ephem. All enabled antennas belonging to the “x2” beam (e.g. 1ax2, 1bx2) will be pointed to the ephemeris given by 3c84.ephem.

Return Codes

The user scripts bftrackephem and bfinit use a uniform set of return codes. A return code of zero indicates successful operation. Any other return code asserts some alert condition. It is the user's responsibility to interpret these codes. While most indicate failure (such as bad arguments and missing files), a few represent diagnostic conditions. Return codes below are given with their numeric and descriptive names. BeamOverflow and LinkError codes may be asserted *with* any other code or alone. Other codes cannot be asserted together.

```
BFERR_ExecOK                        0
BFERR_GeneralError                  1
BFERR_InternalError                 2
BFERR_DRbConnectionFailed           11
BFERR_ExternalSystemCallFailed      12
BFERR_InputError                    21
BFERR_MalformedArgument             22
BFERR_BadAntennaList                23
BFERR_BadInputFile                  24
BFERR_CalibrationOutOfScope         41
BFERR_CalibrationFailed             46
BFERR_BeamOverflow                  64
BFERR_LinkError                     128
```

Software Daemons

The beamformer requires certain daemon processes to be active to function properly. A short description of these is below.

On Each BEE2: b01.bfa, b02.bfa… b05.bfa

A root-owned tree under /opt/bf/ contains subdirectories including:

/bin/ All ruby scripts go here. The daemon script exists here also. To stop daemons, run "./bfdaemon stop", while to start them "./bfdaemon start".

/bof/ All firmware files go here

/log/ Log files go here. Because the BEE2 file system is common, all bee2 log files can be accessed from any Bee2 machine.

On Boot2

A obs-owned tree under /opt/bfu/ contains subdirectories including:

/bin/ All ruby scripts go here. The daemon script exists here also. To stop daemons, run "./bfudaemon.rb stop", while to start them "./bfudaemon.rb start". This will control daemon processes for the Master, B01, B02… B05 servers. All daemons on the BEE2s must be running before these daemons can be successfully started. Daemons are titled "bxx_Beamformer_User_Server_bfa", where xx is the BEE2 number (01 through 15), and "Master_Beamformer_User_Server_BFx" where x is the beamformer number 1..3.

/data/ All --write data is saved here, under a date/time stamped directory

/etc/ Configuration files go here, including hookups, antenna positions, and TDR estimates for fiber lengths.

/log/ Log files go here.

For the SonATA Packetizer

The SonATA packetizer also lives in /opt/bfu/bin. The daemon script "./spkdaemon start" and "./spkdaemon stop" controls the packetizer daemons. Daemons are titled "SonATA_Packetizer_Beamformer_Server_BFx" where x is the beamformer number 1..3.

Indicator Lights

… Summary of indicator lights on BEE2s, iDACs, iADCs [in progress]

A User's Checklist

This guide may not be complete, but should serve as a good start-up for beamformer observations. See ATA documentation for more details about specific commands.

Make ready your ephemeris and ensure the telescopes are prepared for observing:

- `ataephem [delay calibrator]`
- `ataephem [phase calibrator]`
- `ataephem [obs targets]`
- `atatrackephem [ant list] [delay calibrator]`
- `atasetpams [ant list]`
- `atasetfocus [ant list] [observing frequency] --cal`
- `atalnaon [ant list]`

Initialize the beamfomer while waiting for the telescopes to track and focus to finish.

- `bfinit –b [beamformer] –a [antpol list]`
- `bfibob sky`
- `bfibob walsh`
- `bfibob rearm`

Check the focus, confirm settings for all antennas

- `atagetfocus [antlist]`

Repeat focus if any antennas report errors. Otherwise proceed setting for delay cal:

- `atasetskyfreq c 1420`
- `bfibob autoatten`
- `bftrackephem –b [beamformer] –f 1420 –d [300] –i [15] –n [4] –e [delay calibrator] –caldelay –agcdb -3`

If delay calibration was good, proceed to obs. Frequency and calibrate

- `atasetskyfreq c [obsfreq]`
- `bfibob autoatten`
- `atatrackephem [ant list] [phase calibrator]`
- `bftrackephem –b [beamformer] –f [obs freq] –d [i+200] –i[long time] –n[3] –e [phase calibrator] --calphase`

Optionally but not required: Iterate the following for different offsets… 1 MHz, 10 MHz, 50 MHz, 100 MHz to refine your slope estimate. But whenever you do another phase cal, make sure it's at the same obs freq (no offset).

- `bftrackephem –b [beamformer] –f [obs freq + offset] –d [I+200] –I[long] –n[3] –e [phase calibrator] --calfreq`

Just before observing, balance your inputs (Optional and potentially unstable)

- `bftrackephem –b [beamformer] –f [obs freq] –d [30] –I[5] –n[3] –e [phase calibrator] –calamp`

Output file format

During observing, data is saved to time-stamped directories in a BFS file format. This is a tab-delimited format with one line per record. Each record is formatted as below:

% [0] Beamformer:	Indicates the beamformer ID (1 or 2 currently)
% [1] Data Type:	Category of write. See above.
% [2] Time (String):	String of current date/time (local time)
% [3] Time (Numeric):	Numeric (seconds since 1970), *not* atomic time (TAI)
% [4] Sky Frequency:	Floating point MHz
% [5] User Pointing:	The user's commanded ephemeris file, if present
% [6] User Offset Az:	The user's offset azimuth command
% [7] User Offset El:	The user's offset elevation command
% [8] Current Az:	The current azimuth (read from child ant?)
% [9] Current El:	The current elevation (read from child ant?)
%	Note az/el only work if pointing has been loaded
%	in this mode, and not as raw geopoint!
% [10] Hookup Type:	Category of hookup. (1) Antenna, (2) FPGA
% [11] Hookup Name:	String name of the hookup
% [12] CalDelay:	The bulk delay from the hookup cal
% [13] CalPhasorReal:	The phasor from the hookup cal, real part
% [14] CalPhasorImag:	The phasor from the hookup cal, imag part
% [15] CalDpDf:	The dp/df from the hookup cal
% [16] Obs time:	The observation time, in seconds
% [17] Data Length:	Reserved currently, but might be length of following data
% [18] idx:	The index number of the snap (optional)
% [19] pol:	The polarziation of the snap (optional)
% [20]..[24]	Reserved meta data
% [25…EOL]	Data pairs: R0, I0, R1, I1, R2, I2… etc.

Programming Beam Calculators

The “Beam Calculator” is the fundamental object used to process pointing ephemeris (az/el versus time) into antenna-based geometric pointing tables (amplitude, phase, and delay). A versatile interface exists for creating and using custom beam calculators, as well as for providing arbitrary data to these calculators through bftrackephem. Additionally, it is possible to “debug” beam calculators without reinitializing the beamformer, greatly saving development time during testing.

By its nature, geoPoint data is purely geometric, and does not include calibration offsets. The programmer and user should not be concerned with amplitude and phase offsets, as they occur after the beam calculator stage and are hidden from view. However, the programmer should consider amplitude offsets, as it is probable that a calibration will result in unequal signal level among the antennas (eg, maximum SNR weighting). More details on this and the associated concerns will emerge when amplitude balancing has been fully implemented.

Before proceeding, there are a few ground rules:

1. The beamformer class BEAMCALCULATOR is defined at initialization, and contains information for basic pointing and also code to support user scripts.
2. User extensions should be self-contained files, and should extend the BEAMCALCULATOR class. These files are not in the beamformer home directories, but are pathed relative to bftrackephem.
3. User libraries may contain multiple pointing types. The file name determines the library name; the function name determines the pointing type.
4. When debugged, new beam calculators can be deployed to the main library. See Billy Barott to get these approved for the official release code.
5. The phase-center of the beam is defined with the --ephem command. Other ephemeris (such as null positions) should be loaded with –bcephem.
6. Beam groups, as will be discussed, represent a single beam processor and are designated by the polarization and unique beam id (e.g. “x1” or “y1”). However, there is little need for the developer to be aware of different beam groups, as each one receives a copy of the beam calculator and is effectively independent.

Objects accessible from the Beam Calculator

The beam calculator functions may access the following objects:

The following class variables are accessible by the user-defined beam-calculator. However, extreme caution is recommended if you decide to use these directly, as the support code has been designed to eliminate the need for such direct access.

`@vars`

A Hash list containing variable data. Preferred method of accessing is with getClassVariable(), although direct access is allowed.

`@beamtype`

The type of beam specified with –bctype (string). This is automatically interpreted, so the user should not need to access this.

`@antenna`

A complete Hash list of all antennas in the beam former. Again, the user should not access this unless there is an overwhelming reason. Use the list provided by enabledAntennasInGroup() instead.

`@bgname`

A string containing the descriptive name of the current beamgroup (e.g., "x1" or "y1"

`@bglist`

An array containing string hash names of all antennas within the current beam group (even disabled ones). This should not need to be accessed by the user.

The following methods are accessible by the user-defined beam-calculator, and are recommended in common use:

`enabledAntennasInGroup()`

Returns an array whose elements are each ANTENNAHOOKUP objects. Each element of the array is guaranteed to be a member of the current beam group, and also to be enabled in the current hookup. The preferred method of calculating coefficients is to iterate through this group.

`getClassVariable(varname, erronempty = true)`

The preferred method of access for local variables. Varname is a string containing the name of the variable as the user should have loaded it with –bcvar. The erronempty flag determines whether the function will raise on error to the operating system if a nil value is encountered (indicating that no value was loaded by the user). This is recommended true for required data that can take on no default value in your code (or, when so important that you wish to alert the user to a possible mis-spelling of the variable name). This is recommended false for optional variables, or variable lists (for example, when iterating through null0, null1, null2, null3… until a nil value is encountered).

`calculateTimeLimits(t0in, tmaxin, ephem)`

Determines the temporal bounds to use in calculating data tables. This is best to invoke in the standard way seen in the demo code.

Sample Code

The source code on the following pages represents some demonstrations of beam calculator classes. The first is the code that is included for the default beam, beamDefaultPhased. The second is for a simple projection nulling algorithm.

Note that the variable name “ephemeris” is reserved.

```
def beamDefaultPhased(interpolationInterval, t0in = nil, tmaxin = nil)
      #=============================================================
      # Generates a simple phased on-axis beam
      # First, generate a list of antennas that we'll be working with.
      # Note that ephemeris should be loaded as VALUEARRAYs: A data holder
      # with azimuth and elevation coded vs time entries.
      # Even for fixed pointing, we should use the VALUEARRAY: It supports
      # the .interpolate() method for fixed pointings and is a more general
      # application than having fixed arrays.
      #=============================================================

      ants = enabledAntennasInGroup()         # An object list
      ephem = @vars["ephemeris"]              # The current ephemeris

      # Get the first and last points in the ephemeris, as loaded by the user
      # and filter as requested by the just-in-time limits.
      # This will establish the range of times for interpolation.
      t0, tmax = calculateTimeLimits(t0in, tmaxin, ephem)

      # Clear all currently existing geometric pointing data from
      # all enabled antennas.  This is required before new data can be
      # added to the data tables.
      ants.each{|ant|
            ant.clearGeoPoint()
      }

      #=====================================================================
      # The preceeding lines are setup for calculating the beam.  The
      # following lines do the actual calculation.  Note that we must
      # iterate forward through time, because geoPoint data must be loaded
      # forward-chronologically.
      #=====================================================================

      # Now, step-interpolate through all times that were valid in the ephem:
      t0.step(tmax, interpolationInterval) { |tai|
            # The following lines extract ephemeris data for the current time.
            # From a value array, data is in the format: [tai, [az, el]]
            currentEphemItem = (ephem.interpolate(tai))
            currentTAI = currentEphemItem[0]
            currentAzelPair = currentEphemItem[1]
            currentAz = currentAzelPair[0]
            currentEl = currentAzelPair[1]

            # For every antenna that's currently enabled in this group:
            ants.each{|ant|
                  # Return an ordered triple of the geometric
                  # pointing correction required for this az/el.
                  # The antenna object computes these corrections based on
                  # its individual antenna position.

                  rc = ant.geometricPointing(currentAz, currentEl)

                  # Below is delay correction to be applied, in samples
                  currentSampleDelay = rc[0]

                  # Below is the amplitude weighting (pre-cal), and should
                  # “always” be returned as 1.

                  currentFringeAmplitude = rc[1]

                  # Below is the fringe phase at the current pointing,
                  # in unwrapped radians (range is NOT zero to 2 PI).  The
```

```
                    # user should never wrap this phase to zero/2PI, because it
                    # is used to calculate fringe rate.

                    currentFringePhase = rc[2]

                    # Now that we've calculated the required correction
                    # offsets for this antenna at this time, we write this
                    # data back to the current antenna.  Alternatively, we
                    # could manipulate this data further (e.g. dither the phase
                    # or process the array manifold through a projection null).
                    # Note that this data does NOT include calibration offsets
                    # of any sort: Those are added automatically by the
                    # beamformer at a later stage.

                    currentGeoPoint = [currentTAI, [currentSampleDelay,
                           currentFringeAmplitude, currentFringePhase]]
                    ant.addGeoPoint(currentGeoPoint)
            }
      }
end
```

Notice that there is really very little code in the above example. This code should make an excellent starting point for additional algorithms, and one more is included on the next page. Note that the following example is shown as an entire file, as the user might create.

```
#=================================================================
# This code has not been revised for just-in-time beamforming
# See previous example for t0, tmax assignments and proper
# arguments into beam calculator functions.
#=================================================================
#################################################################
#
#  engbeamcalc1.rb
#
#  Contains some demonstration code for beam nulling.
#  Specifically, this extends the beam calculator with:
#        beamAxialNull()
#          --> VAR = "nullorder"
#          --> Ephem = "ephemeris" (default ephemeris only)
#
#  Author: William C. Barott
#  Original Date: July 26, 2008
#  Last Revised Date:
#
#################################################################


class BEAMCALCULATOR
      #=================================================================
      # This is an extension to the BEAMCALCULATOR.  When ruby loads
      # this file, it will reopen the BEAMCALCULATOR class and add
      # this additional functionality.  Comments from the standard beam
      # calculator have been omitted to save space.
      #=================================================================

      def beamAxialNull(interpolationInterval)
            # Generates a simple null beam.

            ants = enabledAntennasInGroup()         # An object list
            ephem = @vars["ephemeris"]       # The current ephemeris

            nullorder =  (getClassVariable("nullorder", true)).to_i
            nants = ants.length              # Total number of ants
            # The null order determines how many "cycles" we will distribute
            # our antennas around.  EG, null order 1, we will dither phases
            # around the unit circle "once." (so for 4 antennas, we get phases
            # of 0 90 180 270, or null order 2 at each of 0, 180.

            t0 = ((ephem.getData(0)).getData)[0]
            tmax = ((ephem.getData(ephem.length - 1)).getData)[0]

            # Clear current data.
            ants.each{|ant|
                  ant.clearGeoPoint()
            }

            t0.step(tmax, interpolationInterval) { |tai|
                  currentEphemItem = (ephem.interpolate(tai))
                  currentTAI = currentEphemItem[0]
                  currentAzelPair = currentEphemItem[1]
                  currentAz = currentAzelPair[0]
                  currentEl = currentAzelPair[1]

                  ants.each_index{|antidx|
                        ant = ants[antidx]
                        rc = ant.geometricPointing(currentAz, currentEl)
                        currentSampleDelay = rc[0]
```

```
                        currentFringeAmplitude = rc[1]
                        currentFringePhase = rc[2]

                        # Now alter the fringe phase to create a null:
                        alteredFringePhase = currentFringePhase +
(antidx.to_f / nants) * 2 * PI * nullorder

                        # We're going to write the altered fringe phase
                        # to create the null we just made.
                        currentGeoPoint = [currentTAI, [currentSampleDelay,
currentFringeAmplitude, alteredFringePhase]]
                        ant.addGeoPoint(currentGeoPoint)
                    }
                }
        end


end
```

Selected Documentation for Beamformer Objects

The following pages contain selected documentation on the VALUEARRAY and ANTENNAHOOKUP classes. These have been limited to the methods that users might find to be of interest when programming their own beam calculators.

Class VALUEARRAY (selected methods)

new(parent, mode, norder, dsize) object parent, string mode, integer norder, integer dsize

Creates a new VALUEARRAY object and returns it to the environment. Parent and mode are reserved for other functions of the VALUEARRAY not exposed here. It is permissible to set them both to nil, although they must be included arguments.

Norder and dsize determine the dimensions of the underlying data of which the VALUEARRAY is comprised. Norder is the differential order (number of Taylor expansion terms), and at present should always be set to 1. dsize is the number of discrete values that the array will handle. So, for example, azimuth and elevation coordinates are handled by a valuearray with norder = 1 and dsize = 2.

getData(idx) integer idx → VALUEHOLDER

Returns the VALUEHOLDER at the given index.

getDataArray() → array of VALUEHOLDER

Returns a pointer to the entire data array of VALUEHOLDERS.

length() → integer

Returns the length of the data array held by the VALUEARRAY. Functionally the same as myobject.getDataArray().length()

addData(indata) array indata

Adds a row of data to the VALUEHOLDER array. This data should be entered in the required format, generically [T, [X0, X1, X2], [X0', X1', X2']]. For example, azel pointing is added [tai, [az, el]]. Note that the new data must occur at a later time than data already in the tables, otherwise interpolation will break.

clearData()

Removes all data from the VALUEHOLDER array, resetting it to an empty array.

trimOldData(tai) float tai

Removes all data from the data array where the time index is before the time included in the argument. Saves "one-point-before" the threshold for maintaining valid interpolation.

interpolate(tai) float tai → array

Returns an array of data in the standard format, e.g. [T, [X0, X1, X2], [X0', X1', X2]]. The "T" should be the same as the TAI argument. Each data point results from a linear interpolation of the temporally nearest entries in the data table.

Class ANTENNAHOOKUP (selected methods)

getHashName()

Returns the antenna's unique hash name, for example 3gx1 is the hash name for antenna 3g, x-polarization, first copy. Note that x1 is also the beam group name.

getTSys(optional freq)

float freq → float

Returns the noise temperature for the antenna. If freq is included, calculates at the given frequency. If freq is omitted, calculates at the current sky frequency. Note that TSys values are not guaranteed to be absolute, but ratios between antennas should be correct.

getATAID()

→ string

Returns the ATA ID node and antenna, for example 1a.

getATAPol()

→ string

Returns the polarization, x or y, for the current antenna.

addAzEl(indata)

array indata

Adds an entry to the azimuth / elevation data table for this antenna. Input data must be of the standard format as an array: [tai, [az, el]]. Data must be entered sequentially, such that every entry is "after" the last entry chronologically. An error will occur during interpolation if this condition is not satisfied.

Note that this data table is used strictly for meta-pointing information. It is preloaded with ephemeris as specified by the user's –ephem command. However, the savvy programmer might wish to alter this meta table if the new beam calculator changes the primary pointing substantially from the "ephemeris."

setAzEl(indata)

array indata

Clears the az/el data table for this antenna and writes one entry, the indata. The same rules as addAzEl apply.

clearAzEl()

Clears the az/el data table.

getAzEl(tai)

float tai → array

Commands the azel data table to interpolate, returning the pointing at a given time. The pointing data is returned in the format [az,el]. The method should return [0,0] if an error occurs accessing the table.

addGeoPoint(indata)

array indata

Adds an entry to the geometric pointing database. This data indicates the applied correction to the antenna and conform to [tai, [delay, amp, phase]]. The same rules apply as for other data tables. The delay is in units of full-samples (but can have a fractional part). Amplitudes must not exceed 1 (clipping). Phase is in unwrapped radians (e.g. not constrained to –pi/pi) to facilitate fringe rotation.

setGeoPoint(indata)

array indata

Replaces the geometric pointing database with the given data.

clearGeoPoint()

Clears the geometric pointing table, leaving it empty.

geometricPointing(az,el)

float az, float el → array

Solves the current antenna's geometric pointing data for the given azimuth and elevation. Returns corrections of delay (in samples), phase (in unwrapped radians), and amplitude that, when applied to this antenna, would cause the data to re-reference to the origin of the coordinate system. Calibration information is not included in this data, as calibration is applied after the beam-calculator. Data is returned in the format Please note the following:

1. Data represents corrections required to phase the given antenna with the coordinate origin, not the response of an antenna to the stimulus. The currently assigned sky frequency is taken into account.

2. Phases are unwrapped, meaning that they are continuous and not constrained to the –pi/pi range. It is important that the user maintain continuous phase when reassigning phases to the antennas. This is because the fringe rotator differentiates these phases, and will encounter problems if phases are discontinuous.

Other Matters

Sometimes things take a minute. If they seem to take an abnormally long time, you might try resetting the daemons. If you do this on boot2 *before* stopping your script, you'll sometimes get more diagnostic information when the daemons stop than you would have gotten otherwise.

If something appears broken, please contact me and we'll get it resolved.