

FNEX 3.0: The Winter Upgrade

FNEX 3.0 Upgrade

Title

1/6/16 10:30 AM

Created



FNEX is a full analysis package for the NOvA experiment. It stands for FNEX. With humble beginnings as a simple Near→Far spectrum extrapolator, it has since evolved to generate Gaussian and Feldman-Cousins confidence level contours, as well as being NOvA's only analysis framework with native support for Near Detector studies.

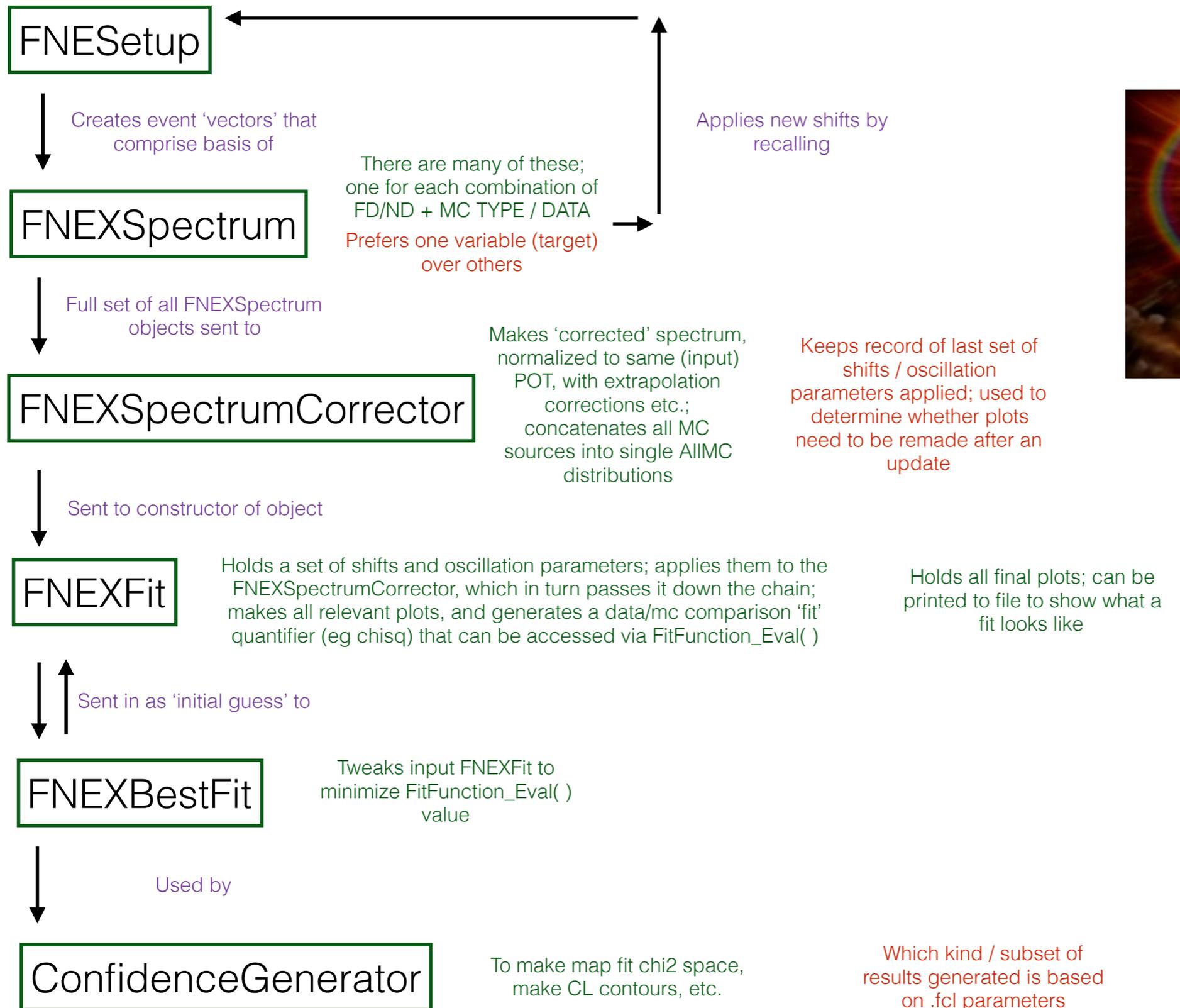
Well met, traveler! If you are reading this, it means that you have joined the crusade to bring FNEX once more roaring back from its molt.

Introduction

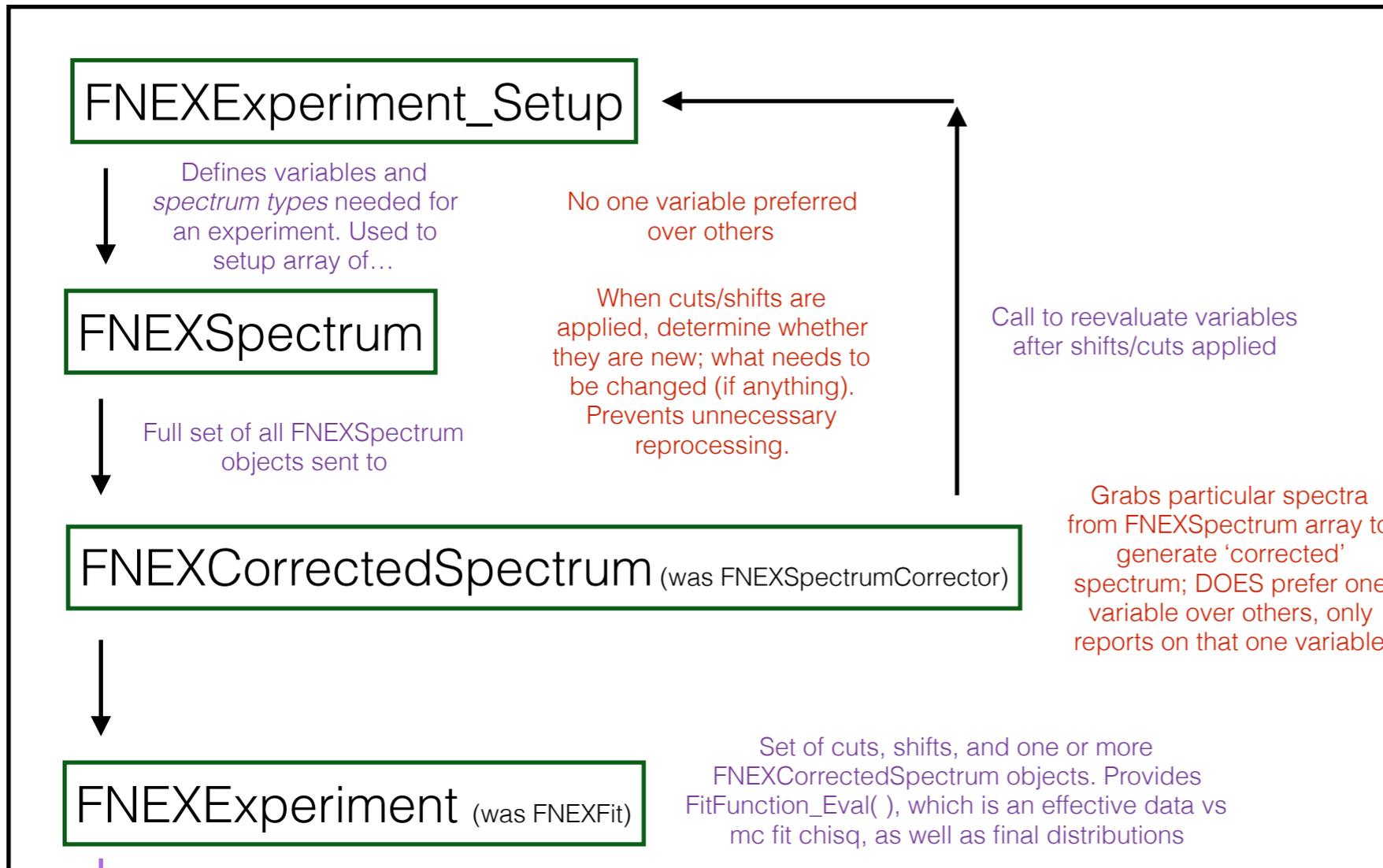
2.0 structure

FNEXexperiment_Setup

FNEX 2.0



FNEX 3.0



FNEXMultiExperiment :: FNEXExperiment

Contains a vector of FNEXExperiment objects; calling `FitFunction_Eval` adds together contributions from all elements in vector; applying Shift A applies that shift to all experiments which are influenced by that shift — potentially with correlations.

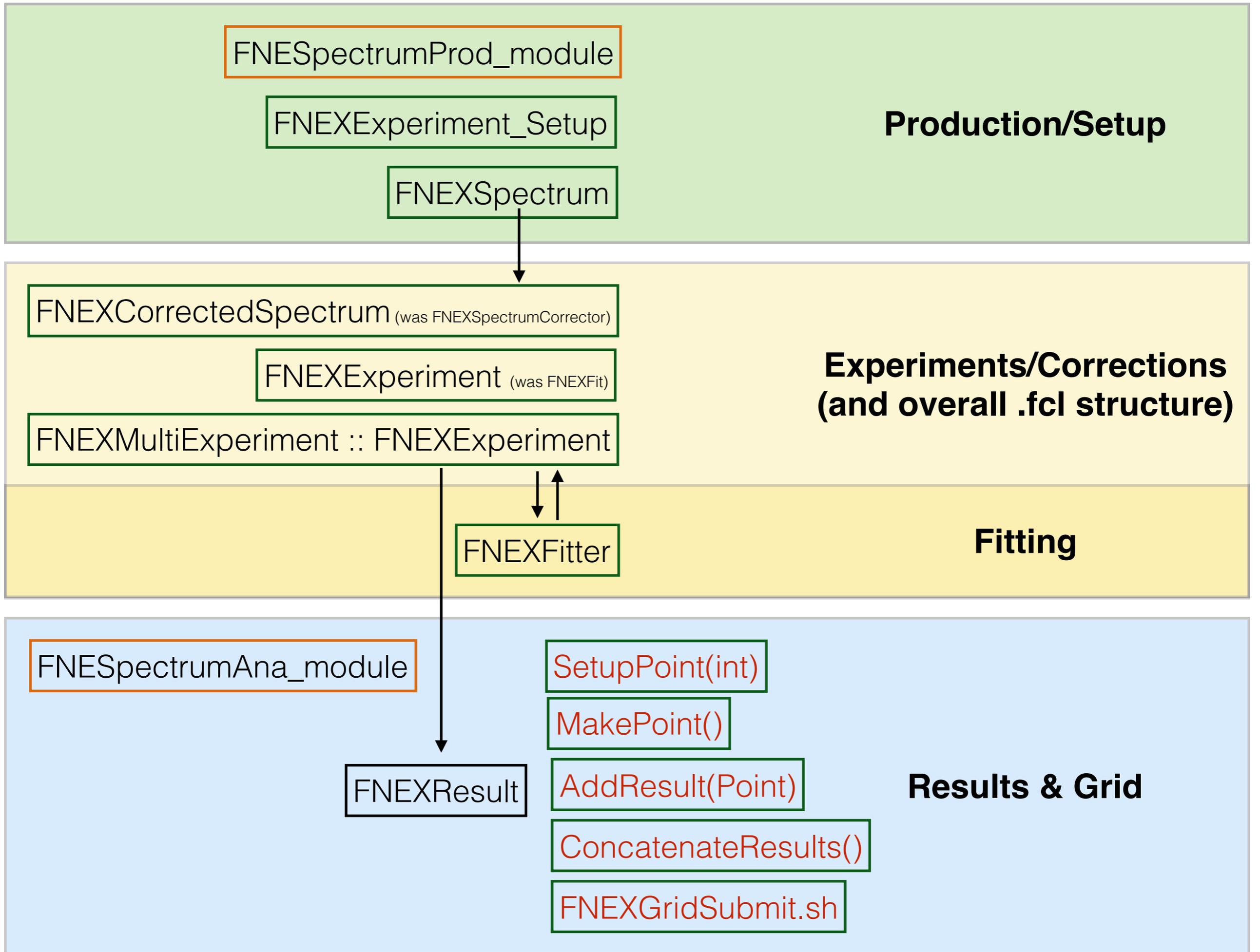
FNEXFitter

Takes in FNEXMultiExperiment; tweaks free parameters in FNEXMultiExperiment to minimize joint `FitFunction_Eval()` value

Needs to be abstracted so particular fitters can be easily swapped in / out

FNEXResult (was ConfidenceGenerator)

Must be generally gridified: a few existing children (templates), but with opportunity for more to be added



FNESpectrumProd_module

FNEXExperiment_Setup

FNEXSpectrum

Production/Setup

Summary:

- 1) Introduce new FNEXVar object that states which other FNEXVars it depends on; generate list of FNEXVar dependencies; only those 'compound' vars that need to be updated are updated after a change in state.
- 2) Take responsibility for "should I remake a plot" away from FNEXExperiment (cur: FNEXFit); give it to individual FNEXSpectrum objects
- 3) Generalize FNEXSpectrum to treat all variables equally (currently has a 'target' variable that it automatically remakes histograms for); generate plots for any variable on demand, unless that plot has already been made and the state has not changed (See 2)
- 4) Move oscillation calculator to FNEXExperiment_Setup object; changes to osc. params are rerouted to the _Setup() object to act on the calculator there; this lets a user implement whatever calculator she wants (e.g., one with sterile generations)

FNESpectrumProd_module

FNEXExperiment_Setup

FNEXSpectrum

Production/Setup

- 1) **FNEXVar** object; holds label and what other FNEXVar it depends on (if isCompound), as well as standard range / binning for histograms.
- 2) **FNEXExperiment_Setup** should create an array of FNEXVar objects with well-defined order at startup (existing framework uses enumerators for this)
- 3) **GenerateDependencies()**: For each FNEXVar object, generates list of pointers / enumerators describing which other variables will change if its value will change; store in some `std::vector< std::vector< >(# dep) >(#vars) fnexvar_dependencies`.
- 4) **FNEXSpectrum** holds list of all shifts applied and current osc. param. values
- 5) **FNEXSpectrum**: When a shift is applied to a FNEXSpectrum object, modifies list **`std::vector< * or enum > vars_to_update`** by checking `FNEXExperiment_Setup::Dependencies()` for that shift (if applicable, see 4)
- 6) When an osc. param. change is applied to a **FNEXSpectrum** object, modifies bool **`update_osc_weights`** (if applicable, see 4)

FNESpectrumProd_module

FNEXExperiment_Setup

FNEXSpectrum

Production/Setup

- 7) **FNEXSpectrum::UpdateValues()** Based on **vars_to_update** and **update_osc_weights**, modify vars and oscillation weights that need updating.
- 8) **FNEXSpectrum::FetchPlot(std::string var_name)** If osc weights are updated and this var does not need to be updated, return existing plots; else generate new plot of that var's distribution and return it.
- 9) **FNEXSpectrum::ApplyWeights()** : instead of finding weights and filling histograms as `—>Fill(entry, weight)`, change to only fill a new **std::vector< > weights**, one for each event; `FetchPlot()` will then create plots according to the **weights** entries.
- 10) **FNESpectrumProd_module** : Based on `FNEXExperiment_Setup`, generate vector of `FNEXSpectrum` objects; fill them appropriately. Save to file. `FNESpectrumAna_module` shouldn't need any major changes in terms of `FNEXSpectrum` loading.

FNESpectrumProd_module

FNEXExperiment_Setup

FNEXSpectrum

Production/Setup

- 1) **FNEXExperiment_Setup::OscCalc()** Returns an osc calculator created within the Setup instance; allows custom templates using any calculator with any associated set of parameters
- 2) **FNEXExperiment_Setup::ChangeOscParam(std::string label, float val)** How to modify the calculator based on osc param change. Allows user to associate their own custom calculators / osc. params with the experiment and describe them in the .fcl file setups as normal.
- 3) **FNEXSpectrum::ApplyShift()** Takes on responsibility for redirecting oscillation parameter change requests to FNEXExperiment_Setup(), instead of trying to modify a variable. Then the single FNEXExperiment_Setup::OscCalc() is used to generate new osc. weights.

FNESpectrumProd_module

FNEXExperiment_Setup

FNEXSpectrum

Production/Setup

Wish List Summary:

We currently have a predefined set of FNEXSpectrum objects; one for each major type of MC and data, for each detector. When we run over a production-level event, FNESpectrumProd_module determines what type of event it is (numu MC in ND?) and then assigns it to the appropriate FNEXSpectrum object. This requires defining a new object whenever we want to start working with different spectra, and running a new set of 'prod' jobs.

- 1) Abstract **FNEXSpectrum** object to include a **IncludesThisEvent()** method (e.g., might say to include only MC numu events, or only MC nue on e events).
- 2) Create **set of standard inherited FNEXSpectrum** objects; in FNEXExperiment_Setup, define which FNEXSpectrum objects ought to be used for this analysis.
- 3) **Prod_module** now runs over all events without deciding into what category they ought to fall; saves them all in **one 'neutral' FNEXSpectrum object**.
- 4) When an **Ana_module** job starts up, **queries FNEXExperiment_Setup for list of all FNEXSpectrum to create**; makes empty FNEXSpectrum objects, then runs through ALL saved events to direct them into the appropriate FNEXSpectrum.

FNESpectrumProd_module

FNEXExperiment_Setup

FNEXSpectrum

Production/Setup

- 1) Wish list: **FNEXSpectrum::IsInSpectrum()** Determines whether an event belongs in this FNEXSpectrum object.
- 2) Wish list: Several inherited **FNEXSpectrum** objects with different definitions of **IsInSpectrum** (e.g., MC numu, or MC nue on e)
- 3) Wish list: **FNEXExperiment_Setup::SelectSpectra()** Assigns a subset of all defined FNEXSpectrum objects to this experiment (e.g., MC numu, or MC nue on e)
- 4) Wish list: **FNESpectrumProd_module**: Makes vector of FNEXSpectrum objects via FNEXExperiment_Setup::SelectSpectra(); replaces hard-coded selection criteria in Event loop with calls to IsInSpectrum() for each associated FNEXSpectrum object.

FNEXCorrectedSpectrum (was FNEXSpectrumCorrector)

FNEXExperiment (was FNEXFit)

FNEXMultiExperiment :: FNEXExperiment

**Experiments/Corrections
(and overall .fcl structure)**

FNEXFitter

Fitting

Summary:

- 1) **FNEXExperiment** (migrate to this from FNEXFit): All responsibility for whether a plot ought to be remade will be moved to the FNEXSpectrum object; can be removed from FNEXExperiment. Rewrite accesses to FNEXSpectrum in terms of FetchPlots()
- 2) **FNEXCorrectedSpectrum**: Holds POT information and correctly normalizes all MC contributions. Does not remake corrected spectra if a state change has not occurred (no shifts affecting this spectrum have been applied).
- 3) **FNEXMultiExperiment**: Holds highest-level vector of shifts and FitFunction_Eval. Responsible for determining how to combined FitFunction_Eval for all linked experiments (e.g., numu and nue), and assigning correlated shifts (e.g., a 1 sigma HadE shift to the FNEXMultiExperiment might be a 1 sigma HadE shift for the numu FNEXExperiment, and a 1.3 sigma HadE shift for the nue FNEXExperiment).
FNEXFitter now acts on a FNEXMultiExperiment object, even if there's only one experiment in it.

FNEXCorrectedSpectrum (was FNEXSpectrumCorrector)

FNEXExperiment (was FNEXFit)

FNEXMultiExperiment :: FNEXExperiment

**Experiments/Corrections
(and overall .fcl structure)**

FNEXFitter

Fitting

- 1) **FNEXExperiment**: Currently is responsible for determining what the last set of applied shifts was, and whether FNEXSpectrum plots need to be remade; should instead still keep a record (redundancy could be useful here), but otherwise should simply pass shift/osc. param. information on to the FNEXSpectrum objects, leave the decision-making on how to apply these shifts up to them.
- 2) **FNEXExperiment**: Should also keep a record of all applied shifts / osc. params; pass that information on to all linked FNEXCorrectedSpectrum objects.
- 3) **FNEXMultiExperiment**: Takes in list of FNEXExperiments; hosts list of shifts to be applied, but applies them to each Experiment according to some correlation matrix (1 sigma shift in HadE on Numu = 1.5 sigma shift in HadE on Nue, as a fictitious example)

FNEXCorrectedSpectrum (was FNEXSpectrumCorrector)

FNEXExperiment (was FNEXFit)

FNEXMultiExperiment :: FNEXExperiment

**Experiments/Corrections
(and overall .fcl structure)**

FNEXFitter

Fitting

- 1) **Feed setup through .fcl file:** Need some consistent method for defining multiple FNEXCorrectedSpectrum objects for any given experiment; defining multiple experiments (each calls a different _Setup file with different cuts, etc.); and defining the MultiExperiment (with systematic uncertainties) that contains many experiments.
- 2) Likely takes for of defining a list of CorrectedSpectrum objects; a list of experiments, each of which references a _Setup file, a set of cuts that it wants to employ, and a list of CorrectedSpectrum objects to generate; and then a multi-experiment, which references the 'attached' experiments

FNEXCorrectedSpectrum (was FNEXSpectrumCorrector)

FNEXExperiment (was FNEXFit)

FNEXMultiExperiment :: FNEXExperiment

**Experiments/Corrections
(and overall .fcl structure)**

FNEXFitter

Fitting

Summary:

- 1) **Fits currently done via FNEspectrumBestFit:** Want instead an abstract class that takes in a FNEXExperiment object, and applies a minimization technique using its list of available shifts, and its FitFunction_Eval() return values.
- 2) Then need to work on generating the best possible **FNEXFitter :: child**; perhaps the best technique is one that uses one minimization technique to select new test points for osc. param space (which is more featureful and smoothly-changing), and another that selects new test points for the syst. shifts (which are likely less featureful, but can cause discontinuous changes in bin content that would make Migrad unhappy)
- 3) As always, make sure FNEXFitter::child used per FNEXMultiExperiment is defined in the .fcl file.

FNESpectrumAna_module

SetupPoint(int)

MakePoint()

FNEXResult

AddResult(Point)

ConcatenateResults()

FNEXGridSubmit.sh

Results & Grid

Summary:

- 1) **“Abstractify” FNEXResult** (currently ConfidenceGenerator), and create child classes for all existing result objects (e.g. Gaussian uncertainty CL contours; FC uncertainty contours; simple best fits; speed tests)
- 2) **“Gridify” FNEXResult** : require all jobs to be described in terms of data_point objects with a finite number of data_points per job (may depend on .fcl inputs); ‘interpreter’ to fill in a data_point based on current point_index (e.g. for 50x50 grid of points in osc. param. space, point_index ranges from 0-2499, interpreter tells you which osc. param. vals. to look at for each point in this range.
- 3) **FNEXGridSubmit.sh/FNEXDown.sh**: Former is slight modification to existing fnex_submit_fc_job.sh; latter appraises output of grid jobs to check for failed jobs, resubmit missing pieces.

FNESpectrumAna_module

SetupPoint(int)

MakePoint()

FNEXResult

AddResult(Point)

ConcatenateResults()

FNEXGridSubmit.sh

Results & Grid

- 1) **FNEXResult()** : ConfidenceGenerator currently contains separate methods for all result types; abstract to FNEXResult class, with child methods for each type of result.
- 2) FNEXResult takes in a FNEXMultiExperiment object and a 'starting point' (more on that later).
- 3) Like with any other ART object, it can be Configured based on .fcl file parameters; there will be a .fcl 'block' describing each FNEXResult, designed by the inherited class' writer

FNESpectrumAna_module

FNEXResult

SetupPoint(int)

MakePoint()

AddResult(Point)

ConcatenateResults()

FNEXGridSubmit.sh

Results & Grid

- 1) **Gridification:** basic idea: to make results inherently grid-friendly (a necessity for Feldman-Cousins jobs and jobs with many systematics), need to make certain that every job can be broken into a discrete set of 'points':
 - 1) **struct data_point:** contains all info needed for a single data point; input and output (e.g. some initial guess of osc params, the resulting best fit point, and related chi2)
 - 2) **GeneratePoints():** based on input parameters (from .fcl file) determines how many distinct points will be needed for the job.
 - 3) **PreparePoint(int point_index, data_point & this_point):** point_index is a number from 0 to NumPoints-1. Based on current index, assigns appropriate values to data_point

Example: 50 x 50 grid in DelMSq and Sin2Theta space = 2500 data points.

PreparePoint() sets $\text{data_point.DelMSq} = (\text{point_index} \% 50) / 50 * (\text{max_val} - \text{min_val}) + \text{min_val}$, $\text{data_point.Sin2Theta} = (\text{floor}(\text{point_index} / 50) / 50 * (\text{max_val} - \text{min_val}) + \text{min_val}$

FNESpectrumAna_module

SetupPoint(int)

MakePoint()

FNEXResult

AddResult(Point)

ConcatenateResults()

FNEXGridSubmit.sh

Results & Grid

1) **Gridification** (continued)

- 1) **Run() / SetPointsToRun()**: **Run()** runs over the subset of points sent to **SetPointsToRun()**; the latter is set via `.fcl` parameter.
- 2) **RunPoint(data_point & this_point)** Does whatever must be done to the `FNEXMultiExperiment` to fill the 'output' variables in `data_point`
- 3) **Save/LoadDataPoints(art::TFileDirectory * tfd)**
- 4) **Save/LoadDataPoints(std::string filename)**
- 5) **CombinePoints()**: this stage combines all `data_points` loaded / generated by this instance to create more complicated objects. E.g., if each `data_point` represents a chi squared evaluation at some grid location in parameter space, this function might compare chi squared values to generate CL contours
- 6) **Display(art::TFileDirectory * tfd)** Put all final results from `CombinePoints()` into a visually pleasing format, save for later consideration.

FNESpectrumAna_module

SetupPoint(int)

MakePoint()

FNEXResult

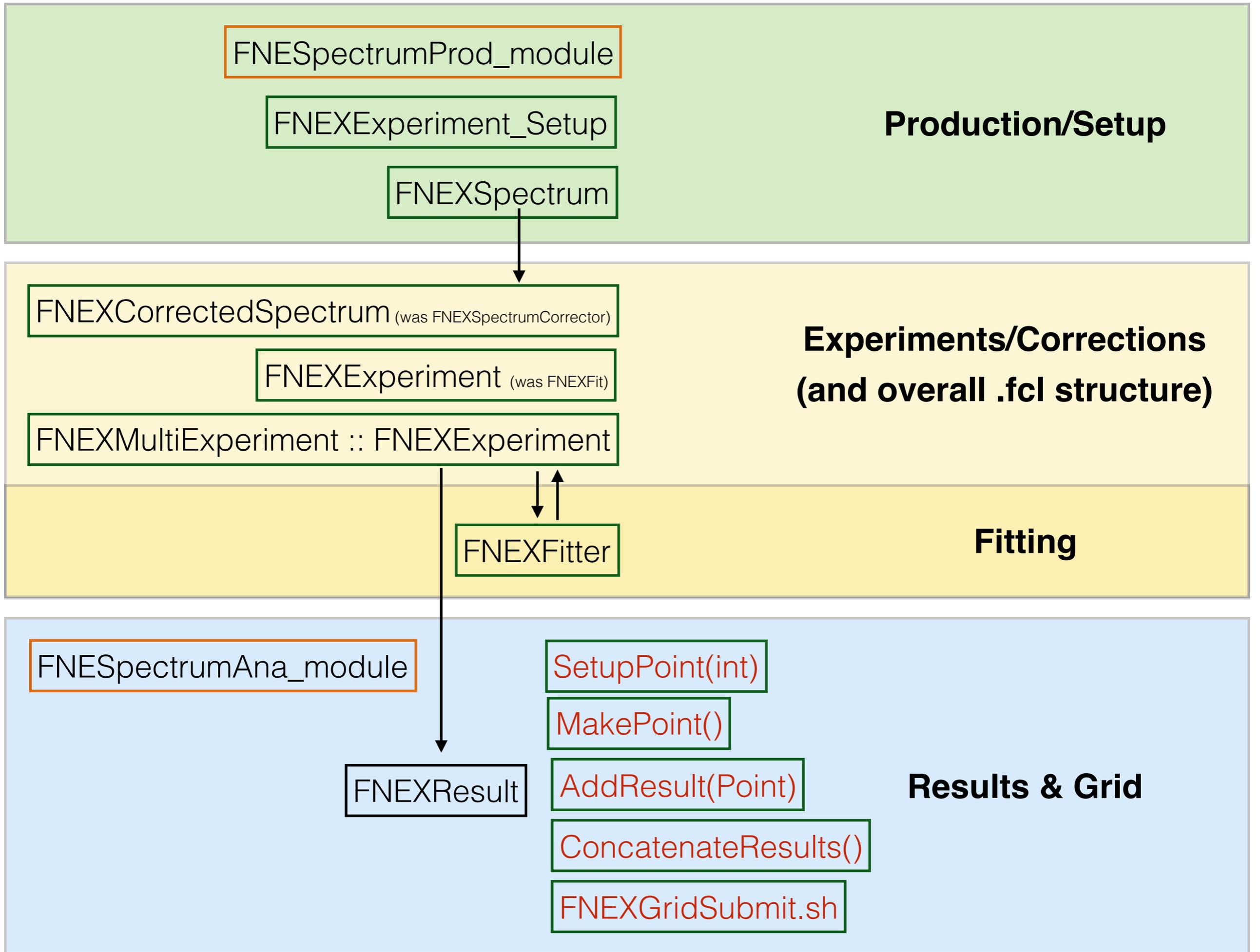
AddResult(Point)

ConcatenateResults()

FNEXGridSubmit.sh

Results & Grid

- 1) **FNESpectrumAna_module** : Must be modified to correctly parse .fcl file; identify what FNEXResult objects are needed; construct these objects; send configuration for each FNEXResult object to the appropriate object; run the object.
- 2) **FNEXGridSubmit.sh** : generalization of submit_c_to_grid.sh , which already does the following:
 - 1) **User provides number of data_points** that will be needed for a certain job (may be .fcl file dependent — should try running GeneratePoints() and looking at output before submitting), **and number of nodes** over which to split the job. (this will be based on user discretion after running a few test points locally). Also provides name of top directory for output.
 - 2) Script then generates a separate job.fcl file for each job, by appending lines that limit the PointsToRun() range for the FNEXResult being run, and then submitting each job with the appropriate job.fcl file
- 3) **FNEXDown.sh** : takes same input as FNEXGridSubmit.sh ; checks that all jobs have completed correctly, and if not, resubmits failed jobs.



Consultants



Production/Setup



**Experiments/Corrections
(and overall .fcl structure)**

Fitting



Results & Grid

Development / Debugging Timeline

Assumes half-time (quarter time) [some time]
for post-docs (scientists) [consultants]

Production/Setup

4-8 weeks

**Experiments/Corrections
(and overall .fcl structure)**

4-6 weeks

Fitting

2 weeks (mostly tests)

Results & Grid

4-6 weeks

**Recreate Numu FA
(no systs)**

1 week (allow debugging as
first major test of FNEX 3.0)

**Recreate Numu/Nue FA
(ALL SYSTS)**

2 weeks (allow debugging / proper
choice of FNEXFitter technique)

Joint Analysis Timeline

TBD

Thrills!

Chills!

SUNDAY!

SUNDAY!

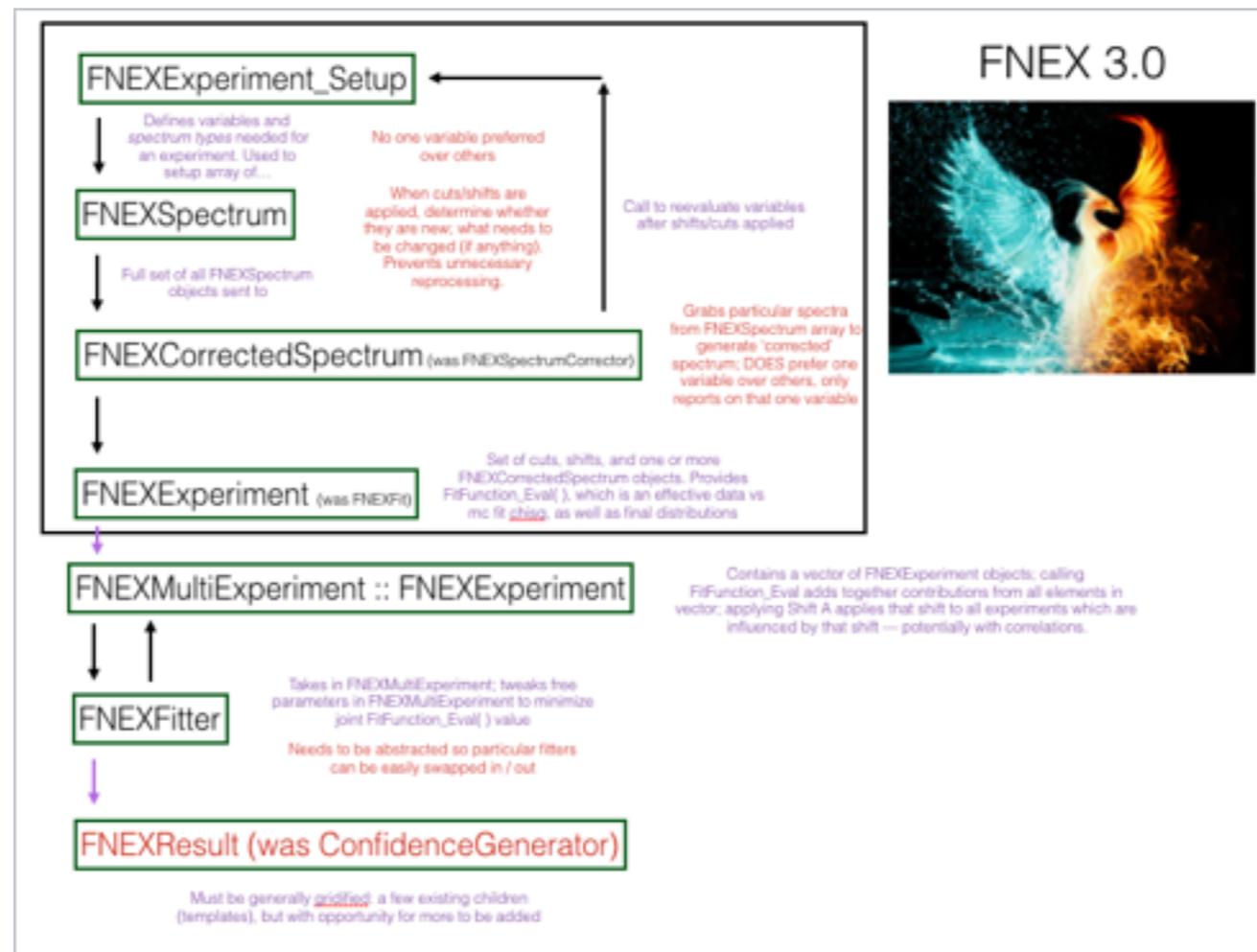
SUNDAY!

ADDITIONAL INFO

FNESpectrumProd_module

Uses **_Setup** to make
FNEXSpectrum objects

Saves them to file



FNESpectrumAna_module

Uses **_Setup** to make
FNEXSpectrum objects

Generates **FNEXExperiment**
objects based on .fcl file

Generates **FNEXResult** objects
based on .fcl file; runs them

For grid jobs, lines
appended to each job's
.fcl file saying what
subset of points in
FNEXResult object
should be processed