Data Flow and Project Class Structure

There are five streams of information that feed into the final angular power spectrum:

Data – Values or counts whose angular power spectra is to be determined.

Mask/Weights – Mask to limit sky coverage to area of interest. Can also be used to put more weight on one part of the sky over another.

These two streams are required. The three additional stream are associated with instrumentation effects. They are from the original method, which was designed to analyze cosmic microwave background data measured by a specific experiment.

Noise – Map of the sky showing the associated instrumentation noise that has to be removed from the basic data stream. Usually inputted as a pixelized or transformed data set.

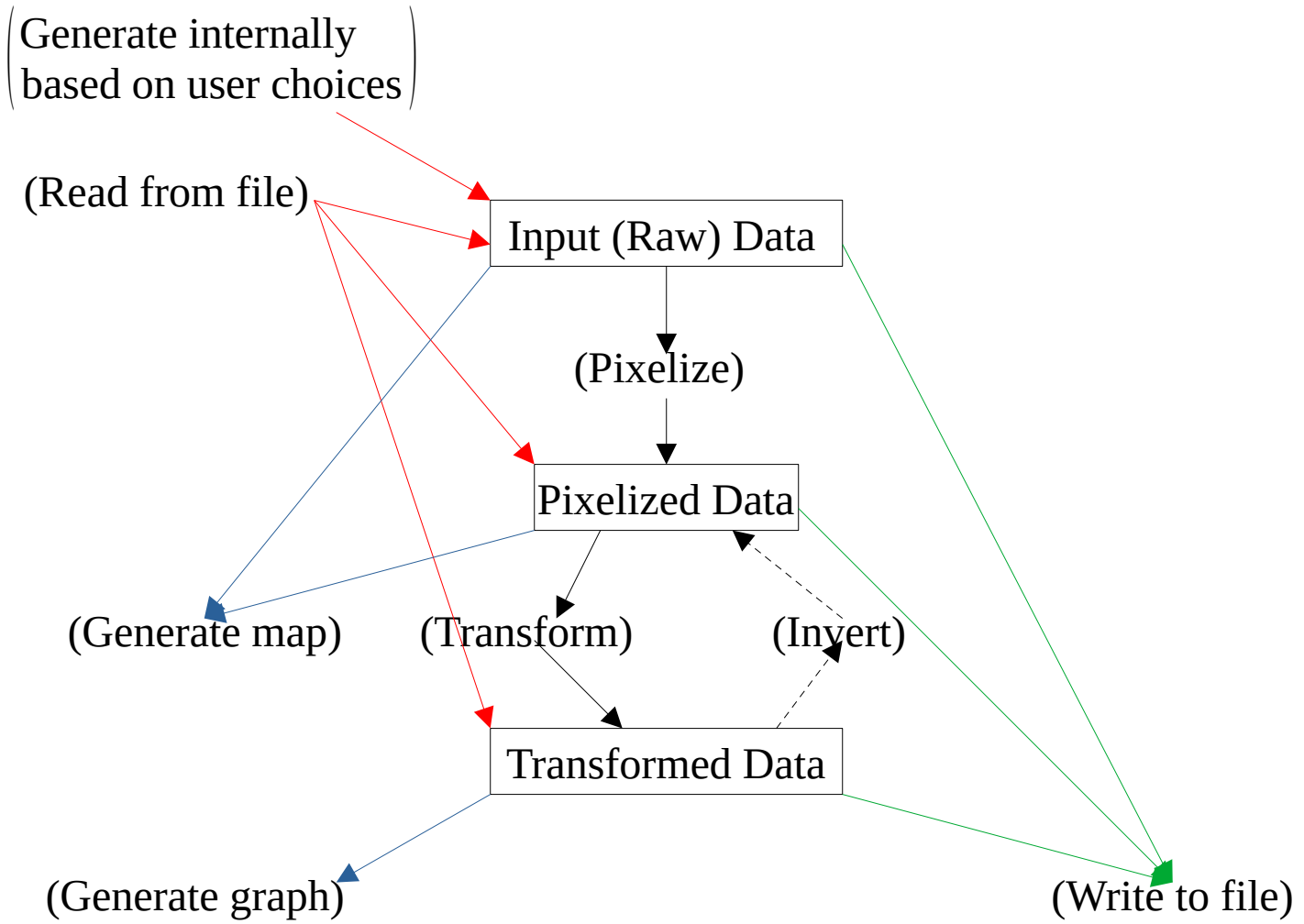
Beam – Map of the sky showing the effect of the finite beam, or scanning window, size. This generates a false angular spike. Usually inputted as a transformed data set.

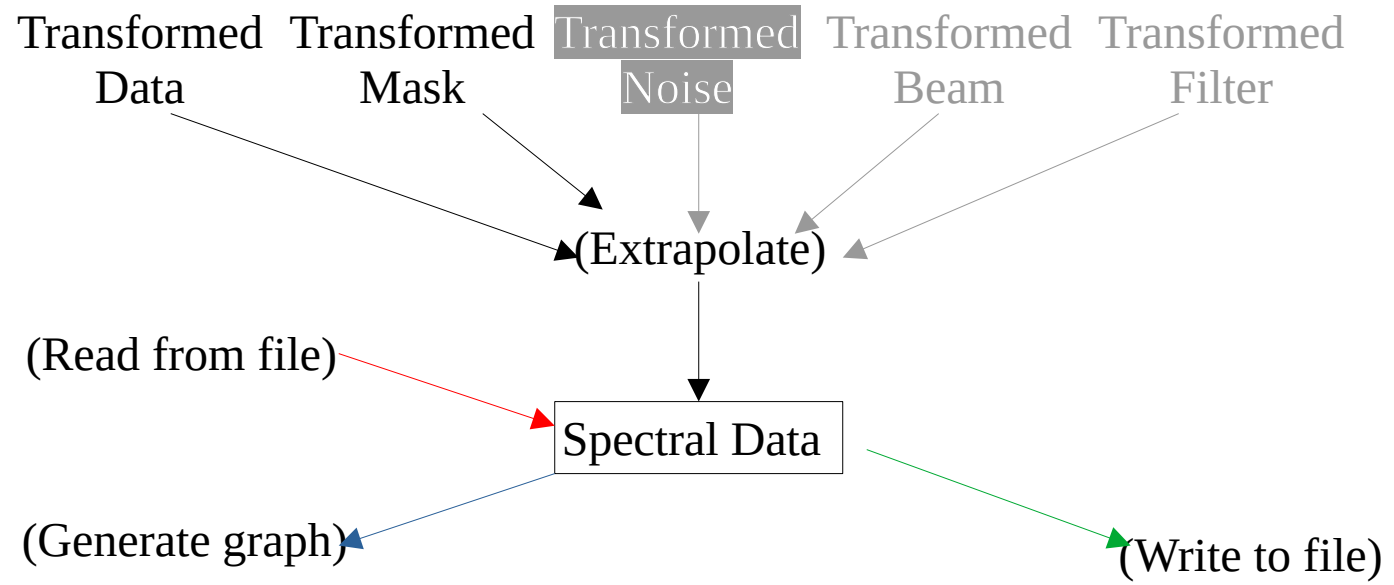
Filter – Map of the sky showing the effect of any filtering function applied to the basic data. The filtering function is used to smooth the data and potentially do preliminary analysis. Usually inputted as a transformed data set.

All five streams could start as raw data maps and so would follow the same analysis path. They are all combined in the final step to generate the extrapolated spectra. If the three optional streams are not specified then the noise transformed data set is set to all zeros while the beam and filter transformed data sets are set to all ones. This is so they don't actually affect the final result.

The basic data flow is shown below. Note that while the path is shown for the data stream, it is the same of all five streams. The only exception is the last step, where all five streams are used to create the final full-sky spectrum. Boxed items are the data sets, parenthetical items are actions. Gray items are optional. Black lines show the data flow through the spectral process. Dashed lines are optional. Red lines correspond to input options while green lines show output options. Blue lines indicate the graphical output.

I/O can happen at any stage in the process. The user can start with raw information, or can start further down the chain. Similarly, the results at each step can be saved for later use. This allows the user to break up the process in real time as needed. The user can also input an already generated extrapolated spectrum for further study. This spectrum is view only and cannot be modified. Modifications would have to come in at earlier steps and a new extrapolated spectrum would need to be generated.


Generate internally based on user choices
(Read from file)
Input (Raw) Data
(Pixelize)
Pixelized Data
(Generate map)
(Transform)
(Invert)
Transformed Data
(Generate graph)
(Write to file)


Transformed Data
Transformed Mask
Transformed Noise
Transformed Beam
Transformed Filter
(Extrapolate)
(Read from file)
Spectral Data
(Generate graph)
(Write to file)

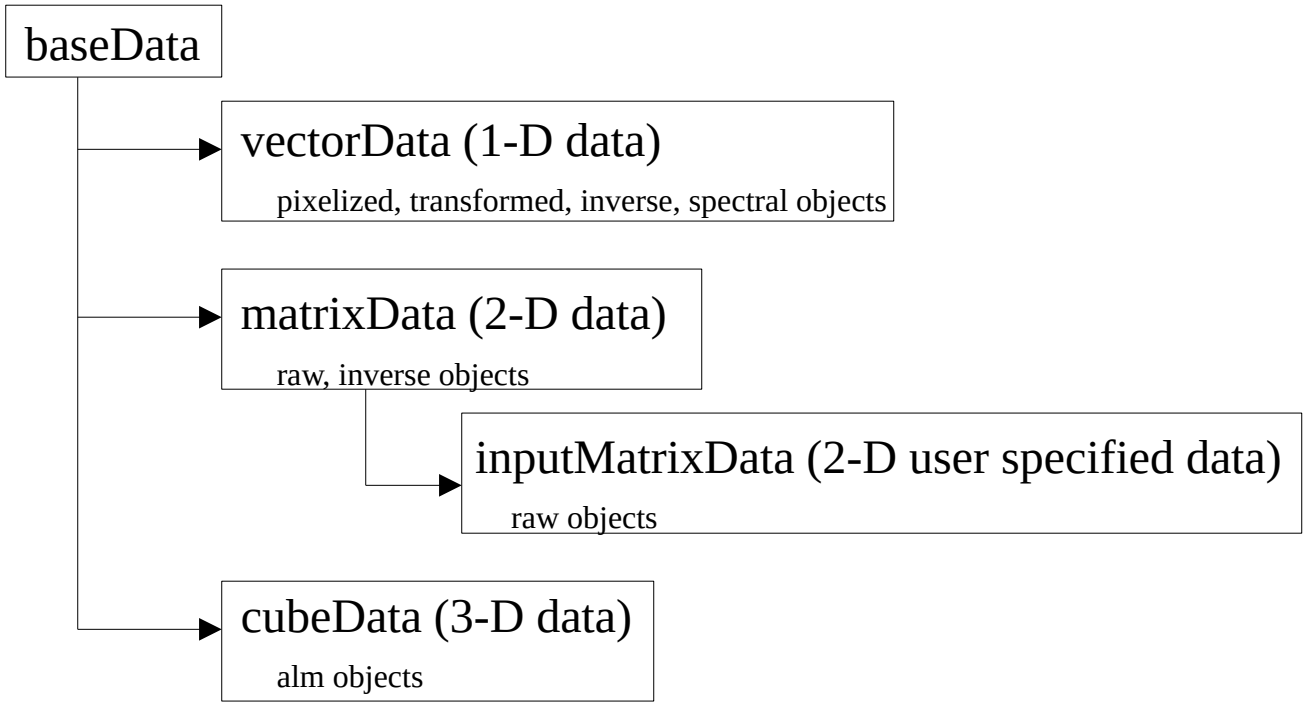
Since there are up to sixteen different data sets, ten different maps, and six different graphs, insuring that all of the various pieces are correctly identified and go with the current analysis chain is critical. This is the function of the association class, which stores pointers to the following objects:

- **Raw data objects** – inputData, inputWeights, inputFilter, inputNoise, inputBeam
- **Pixelized data objects** – pixelizedData, pixelizedWeights, pixelizedFilter, pixelizedNoise, pixelizedBeam, pixelOccupancy (number of points per pixel)
- **Alm data objects (basic transformed objects)** – almData, almWeights, almFilter, almNoise, almBeam
- **Transformed data objects (summed over azmuthal (*m*) index)** – transformedData, transformedWeights, transformedFilter, transformedNoise, transformedBeam
- **Spectral data object** – spectrumData
- **Inverted data objects** – inverseData, inverseWeights, inverseFilter, inverseNoise, inverseBeam
- **Additional data objects needed to calculate spectral object** – couplingMatrix, inverseMatrix
- **Generic map objects** – inputDataMap, inputWeightsMap, inputMap (combined data and weights), inputNoiseMap, inputFilterMap, inputBeamMap, pixelDataMap, pixelWeightsMap, pixelMap (combined data and weights), pixelOccupancyMap, pixelNoiseMap, pixelFilterMap, pixelBeamMap, inverseDataMap, inverseWeightsMap, inverseMap (combined data and weights), inverseNoiseMap, inverseFilterMap, inverseBeamMap
- **Generic graph objects** – transformedDataGraph, transformedWeightsGraph, transformedGraph (combined data and weights), transformedNoiseGraph, transformedFilterGraph, transformedBeamGraph, spectralGraph

The class structure for the program (not including the UI) is

association

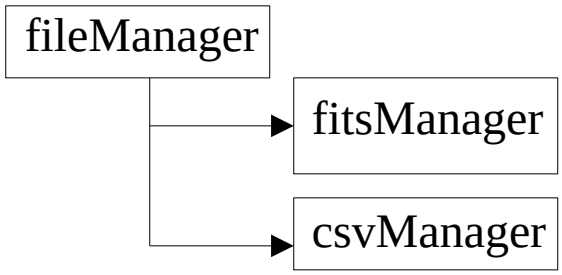
DATA CLASSES



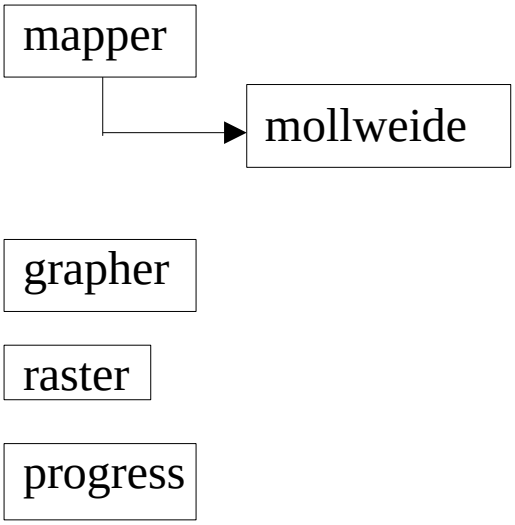
dataMap

dataSpectrum

FILE I/O CLASSES

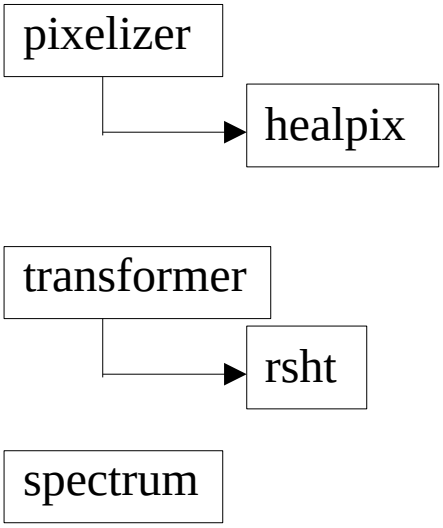


GRAPHICS CLASSES



Note: this will eventually be replaced by gdalmapper

ACTION CLASSES



There is currently redundancy in the graphics classes, as they appear in both libanalyzer and libgraphics. That will need to be cleaned up.