The Cal Library (CalLib)
(and a page on Virtual Corrected Data)

George Moellenbrock (NRAO)

CASA Developer's Meeting

2013 June 19

CASA Calibration Model

- A sequence of *formal* corrupting effects:

$$V_{obs} = K\ B\ G\ D\ P\ T\ V_{mod}$$

$$V_{corr} = T^{-1}\ P^{-1}\ D^{-1}\ G^{-1}\ B^{-1}\ K^{-1}\ V_{obs}$$

- *In practice*, a subset of these, sometimes multiples:

$$V_{obs} = B\ G_{fd}\ G\ V_{mod} \qquad \textit{(spectral line)}$$

$$V_{obs} = B\ G_{fd}\ G\ D\ P\ V_{mod} \qquad \textit{(polarimetry)}$$

- Each *practical* term is a caltable, which is expressed as a "VisCal" object, with apply parameters
  - Tool: cb.setapply(table,field,interp,spwmap,calwt)
  - Task: aggregation of many setapplys (along with selection, action methods)

- But VisCal parameters may be different for different parts of the MS (calibrators of different types, science targets)…..

Examples – Current Practice

- Ugly 'parallel' parameters:

```
applycal(vis='data.ms',
        field='3c273', spw='1,3,5,7',
        gaintable=['cal.wvr', 'cal.tsys', 'cal.G', 'cal.antpos'],
        gainfield=['3c273', '3c273', '3c273', ''],
        interp=['nearest', 'linear', 'nearest', 'nearest'],
        spwmap=[[], [0, 9, 0, 11, 0, 13, 0, 15], [],[]])
```

- Need additional applycal runs:

```
applycal(vis='data.ms',
        field='3c279', spw='1,3,5,7',
        gaintable=['cal.wvr', 'cal.tsys', 'cal.G', 'cal.antpos'],
        gainfield=['3c279', '3c279', '3c273', ''],
        interp=['nearest', 'linear', 'linear', 'nearest'],
        spwmap=[[], [0, 9, 0, 11, 0, 13, 0, 15], [],[]])
```

CalLib Essential Requirements

- Encapsulate standard cal apply parameters (gaintable, gainfield, interp, spwmap, calwt) into a single object
  - simplify top-line interface (applycal, pre-apply parameters in gaincal, bandpass, polcal, etc.)
  - a portable container (a text file or python dictionary)
- Increase capability/flexibility
  - Extend features: obsmap, fldmap, antmap, range, flagstyle, etc.
  - Single-pass cal apply variety for different MS selections

CalLib Definition and Syntax

- A set of "cal expressions”
  - Each a set of minimally sufficient KEY=VALUE pairs to express a distinct calibration setup
  - Each "cal expression" has 3 logical sections: TO, FROM, WITH
- Each "cal expression" a row in a file
- TO keywords are large-scale MSSelection parameters
  - obs,field,intent,spw
  - (finer granularity quickly becomes too complex)
- FROM keyword (a la gaintable)
  - caltable (can be multiple)
- WITH keywords are mappings and other cal apply parameters (same multiplicity as caltable)
  - interp, range (interpolation time- and freq-scale info), flagstyle
  - obsmap, fldmap, spwmap, antmap
  - (calwt)
- Only real constraint: Each TO selection must be distinct per caltable
  - Same caltable cannot be applied more than once to the same data

CalLib Example

- In a file called ‘cal.txt’:

```
field=‘’ caltable=[‘cal.wvr’, ‘cal.tsys’, ‘cal.antpos’] \
        fieldmap=[’self’, ’self’, ‘’] \
        interp=[‘nearest’, ‘linear’, ‘nearest’] \
        spwmap=[[], [0, 9, 0, 11, 0, 13, 0, 15], []]
field=‘3c273’ caltable=‘cal.G’ fieldmap=‘3c273’ interp=‘nearest’
field=‘3c279’ caltable=‘cal.G’ fieldmap=‘3c273’ interp=‘linear’
```

- applycal(vis=‘data.ms’,spw=‘1,3,5,7’,callib=‘cal.txt’)
  - Internally: cb.setcallib(callib=‘cal.txt’)

CalLib Implementation Details

- CalLib file imported into python as a dictionary
  - TO/WITH expressions grouped by caltable (traditional setapply per caltable)
- Sorted CalLib dictionary converted to a Record in C++
  - CalExpression class parses each TO/WITH expression for each caltable
- 'Smart' CalMap template
  - Supports *map algorithms ('nearest' obs, fld; 'self' ant, spw; etc.), selection
- Maps hierarchically parsed
  - Precedence: obsmap, fldmap, spwmap, antmap
  - Generates a std::map of sufficient cal interpolation engines
  - Also a std::map of MS selections that point (many-to-one) to interpolation engines
  - Uniqueness ~implicitly policed
  - The implied 4D generalized *maps ('patch panel') never actually exhaustively generated explicitly

CalLib: Some Subtleties

- Distinct MS Selection not necessarily plainly evident in TO expressions
  - Intent and field can overlap mysteriously
  - could be applycal-selection dependent!
- 'Per-FROM' vs. 'per-TO' WITH specifications?
  - e.g., especially calwt; also others
  - on/off switches?
  - ****Encourage primacy of FROM (for each caltable, a list of TO/WITH expressions)*
- Mapping defaults?
  - spwmap, antmap refer to definite hardware elements for which nominal calibration is 'self'
  - obsmap, fldmap refer to transient (time-dep) observational sequencing phenomena for which 'nearest'---or even 'ignore'---is a reasonable default
- Are all selected data "equivalently" calibrated by a CalLib?
  - could be applycal selection dependent!
  - "could botch it currently with multiple applycal runs" is not an argument
  - ****Encourage primacy of FROM*
  - ****Maintain coarse granularity of TO*
  - ****Prefer anomaly correction support via caltable 'munging' or gencal, not via CalLib (which puts general users at greater risk of error)*

CalLib Example (alternate)

```
caltable='cal.wvr'
    fieldmap='self' interp='nearest'
caltable='cal.tsys'
    fieldmap='self' interp='nearest' spwmap=[0,9,0,11,0,13,0,15]
caltable='cal.antpos'
    interp='nearest'
caltable='cal.G'
    field='3c273' fieldmap='3c273' interp='nearest'
    field='3c279' fieldmap='3c273' interp='linear'
```

- (This is my preference, and matches internal representation)

Virtual Corrected Data

- CalLib makes possible OTF calibrated data, deployable anywhere (split, plotms, clean, etc.)
  - User need only have a CalLib file (or dictionary)
  - Applications need only supply a single parameter (callib)
  - Transforming VI/VB specialization handles it (serially with other transformers; sort TBD)
- Relevant high-level classes (ownership TBD, may be context-dependent):
  - Calibrater: ingests CalLib and generates constituent objects (VisCals and a VisEquation)
  - VisEquation: sorts VisCals for ordered apply, provides correct(vb) method
- Solve context support
  - depends on VisEquation 'pivot'
  - partially _correct_ data by downstream calibrations (correctedVisCube)
  - partially _corrupt_ model by upstream calibrations (corruptedModelVisCube)
  - corrupted model should also work outside solve context (e.g., in plotms)