Interpolation of ASCII-Table Information via Calibration Transfer

F. Masci, 6/18/2003

I. Overview and Aim

One of the options in calibration transfer (*caltrans*) is to create new calibration products by interpolating two pre-existing calibration products with independent records in the *fallback* or *metadata* database tables at the acquisition time of a DCE. The simplest example is interpolation of DN-to-flux conversion factors and their uncertainties. It is envisaged there will be other scalar variables where interpolation may be needed in future. Flexibility in *caltrans* is therefore required to accommodate this. This document describes a method for standardizing the storage and usage of ASCII table calibration-data under the *caltrans* interpolation scheme.

II. Method

i. We assume that all possible ASCII calibration data can be stored in IPAC-table format, such as the hypothetical example below. To make it as generic as possible, the column lengths need not be the same. *Caltrans* need not be concerned with the number of columns or rows – it is just a collection of data whose entries are understood by downstream software (e.g. science pipeline wrappers). The only requirement is that it is in IPAC-table format.

```
\char Comment This table contains parameters for calibrating science data.
\char INSTRUME    = 'MIPS'
\int  CHNLNUM     = 2
\char Index       = 'counter for row entry'
\char fluxconv    = 'conversion factors from DN/sec to microJy/arcsec^(2)'
\char errfluxconv = 'uncertainties for fluxconv'
\char gain        = 'gain values'
\char latcoeff    = 'latent coupling coefficients'
\char inconst     = 'incomprehensible parity constants'
|Index |fluxconv  |errfluxconv |gain    |latcoeff |inconst |
|int   |double    |double      |double  |double   |real    |
 1      1.34       0.05         5.34     3.14      1.21
 2      1.54       0.04         4.23     3.21      1.71
 3      1.41       0.03         4.67     3.54      1.31
 4      1.78       0.08         5.21     3.45      1.54
 5                                       3.23      1.25
 6                                       3.11      1.44
 7                                       3.87      1.45
```

ii. For *caltrans* to carry out the interpolation rule, at least two instances of a calibration-data table must exist in the database. If there are more than two instances, *caltrans* should pick the pair of entries which are closest-in-time as it currently does. If there is only one entry, it should "fall-back" to using the closest-in-time entry.

iii. Below is the new proposed input namelist specification to *caltrans* when interpolation is desired (in this case the range of rule ID's is 400 and above; rule ID = 0 signifies "fallback"). Command-line equivalents will also exist. There is a new namelist parameter: "CalColName#" which is described below. For simplicity, not all required variable specifications are shown.

```
CalType =     'darkcal;    lincal;     flatcal;   paramtable',
CalOutFname = 'dark.fits; lincal.fits; flat.fits; mipsparam.tbl',
CalRule =     '0;          0;         0;        400',
CalColName4 = 'fluxconv;  errfluxconv; inconst';
```

iv. The parameter CalColName# specifies a list of column names where interpolation is desired in CalType product number "#" of the CalType list (in this case # = 4). This adds flexibility in specifying more than

one ASCII parameter file in the CalType list where interpolation may or may not be desired. Below is a schematic of the proposed processing flow when CalRule ≥ 400 is specified:

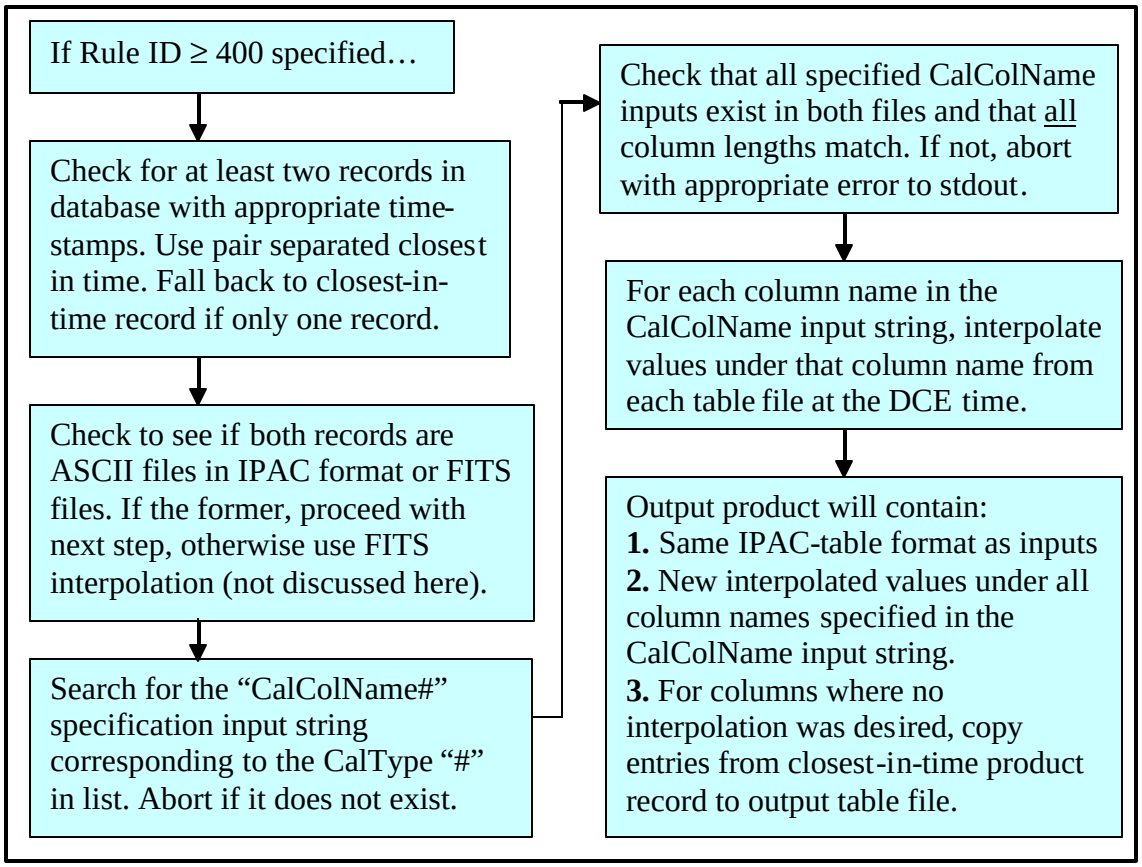

Fig 1. Caltrans processing flow under the interpolation scheme

v. The *tableio* library for reading/writing tables in IPAC format and error checking routines provided therein should be used.

III. Downstream S/W Requirements

Once the above functionality is implemented, it is envisaged that all relevant pipeline modules and/or wrappers which make use of the new cal-product format will need updating. In the case of code written in native C or C++, the *tableio* library can be used. For perl wrappers, it would be advantageous to call a standard perl library for reading IPAC-tables which all systems can use. F. Masci will embark on writing a perl IPAC-table reader for this purpose (which is to be called *perltbli*).