TELCAL Design

5 July 2004

Version 0.1

Contents

1 TELCAL design explanations

- Several comments on the design of the fitting part:
  - *Derived typed are defined to stay general, in particular no array dimension is explicitly specified.* This implied the use of POINTERS.
  - *Everything can not be passed in the subroutine calling sequences.* Indeed, ALL minimization (and thus fitting) algorithms ask for a function to be minimized which have a pre-defined calling sequence. This calling sequence varies with the used minimization methods and libraries. However, those calling sequence inevitably miss some important parameters (as an example, they never pass function parameters that the user want to stay fixed). Properties of the fitted function (and also sometimes grid of measured points) must be passed by another mechanism. In some F77 parts of GILDAS (*e.g.* UV_FIT), this problem is solved by INCLUDE files and COMMON. Here, the F90 "use association" mechanism is used (*i.e.* the variable to be shared is define in a module that is called in all the needed subroutines).
  - *There are some duplication between the fitted function calling sequence and the "association module".* This avoids particular cases for every different minimization methods and libraries.
- The plotting part is done through SIC procedures. This is very flexible but this implies the definition of SIC structures associated to the high levels TELCAL derived types (*e.g.* POINT_CROSS).
- The TELCAL demonstration could be the basis of a TELCAL test suit.

2 TELCAL library interfaces

The user may interact with the TELCAL library at different levels:

- Direct call of very general fitting routines (*e.g.* see fit_1d).
- Direct call of high level functionalites (*e.g.* solve of a cross pointing scan).
- Use of the TELCAL language to access the same high level functionalities (as the GILDAS GTVL library is mainly accessed through the GTVL language).

2.1 Interfaces

The interfaces of the TELCAL F90 subroutines available to the user are located in the `telcal-interfaces.f90` file. Its contents are reproduced here:

```
module telcal_interfaces
  !
  ! General TELCAL
  !
  interface ! Load TELCAL environment (Language, global variables, etc...)
     subroutine load_telcal
     end subroutine load_telcal
  end interface
```

```
!
interface ! Interpret and execute TELCAL commands
   subroutine run_telcal(line,comm,error)
     character*(*) line,comm
     logical error
   end subroutine run_telcal
end interface
!
! FIT related
!
interface ! General unidimensional fitting routine
   subroutine fit_1d(dat,fun)
     use fit_definitions
     type (simple_1d) :: dat
     type (fit_fun)   :: fun
   end subroutine fit_1d
end interface
!
interface ! Initialize the "dat" instance of the simple_1d derived type
   subroutine null_simple_1d(dat)
     use fit_definitions
     type (simple_1d) :: dat
   end subroutine null_simple_1d
end interface
!
interface ! Initialize the "par" instance of the fit_par derived type
   subroutine null_parameter(par)
     use fit_definitions
     type (fit_par) :: par
   end subroutine null_parameter
end interface
!
interface ! Initialize the "fun" instance of the fit_fun derived type
   subroutine null_function(fun)
     use fit_definitions
     type (fit_fun) :: fun
   end subroutine null_function
end interface
!
interface ! Copy the "in" simple_1d derived type into "out"
   subroutine copy_simple_1d(in,out)
     use fit_definitions
     type (simple_1d) :: in,out
   end subroutine copy_simple_1d
end interface
!
interface ! Copy the "in" fit_fun derived type into "out"
```

```
     subroutine copy_function(in,out)
       use fit_definitions
       type (fit_fun) :: in,out
     end subroutine copy_function
  end interface
  !
  ! POINTING related
  !
  interface ! Initialize the point cross structure
     subroutine init_point_cross(ndat)
       use point_definitions
       integer :: ndat ! Size of the data arrays
     end subroutine init_point_cross
  end interface
  !
  interface ! Solve the telescope pointing parameters through a cross
            ! Then plot results
     subroutine solve_point_cross
       use fit_definitions
       use point_definitions
     end subroutine solve_point_cross
  end interface
  !
end module telcal_interfaces
```

2.2 Derived types

The TELCAL derived types are defined in the `telcal-types.f90` files:

```
!
! FIT derived type definitions
!
module fit_definitions
  !
  type :: simple_1d ! Data vector (simple precision, 1d)
     sequence
     integer :: n              ! Number of data
     real(8), pointer :: x(:)     ! Abcissae (in)
     real(8), pointer :: y(:)     ! Values   (in)
     real(8), pointer :: w(:)     ! Weights  (in)
     real(8), pointer :: d(:)     ! derivative of x with time, Not always allocated
  end type simple_1d
  !
  type :: fit_par ! Description of a fitted parameter
     sequence
     character(len=16) :: name ! Parameter name          (in)
     real(8) :: guess          ! Guess                   (in)
     real(8) :: value          ! Fit value               (out)
     real(8) :: error          ! Fit uncertainty         (out)
     real(8) :: mini           ! Minimum possible value  (in)
     real(8) :: maxi           ! Maximum possible value  (in)
     logical :: fixed          ! Is the parameter fixed? (in)
     logical :: padding        ! Padding for structure alignment
  end type fit_par
  !
  type :: fit_fun ! Description of a fitted function
     sequence
     character(len=16) :: name         ! Fitted function name (eg GAUSSIAN)
     character(len=16) :: method       ! Fitting library (eg SLATEC)
     real    :: rms                    ! Residual RMS
     integer :: flag                   ! Error/Quality code
     integer :: ncall                  ! Number of iterations
     integer :: npar                   ! Number of fitted parameters
     type (fit_par), pointer :: par(:) ! Array of parameter descriptions
  end type fit_fun
  !
  type :: fit_var ! Variable fitted parameters only
     sequence
     integer :: n               ! Number
     real(8), pointer :: x(:)   ! Values
     real(8), pointer :: d(:)   ! Errors
     integer, pointer :: idx(:) ! Position index inside full parameter vector
  end type fit_var
  !
end module fit_definitions
!
! POINTING derived type definitions
!
module point_definitions
  use fit_definitions
  !
  type :: point_cross
     sequence
     integer :: sys             ! Scanning coordinate system
     character(16)    :: dir(4) ! Scanning directions
     type (simple_1d) :: dat(4) ! Data vectors
     type (simple_1d) :: sol(4) ! Solution vectors
     type (fit_fun)   :: fun(4) ! Fit functions
  end type
  !
  type (point_cross), save :: pcross
  !
end module point_definitions
!
```

3 TELCAL Language Internal Help

3.1 GAUSS

```
    TELCAL\GAUSS  Xvar  Yvar [Wvar] [/GUESS Area Position Width] [/FIXED
Afix Pfix Wfix]

Fit a one dimensional Gaussian through the (Xvar,Yvar)  points.  An  op-
tional  weight array (Wvar) may be given. Area, position and width first
guess are given through the /GUESS option. Default first guess is 1. Us-
er  may  impose a parameter through the logical variables Afix, Pfix and
Wfix. All parameters are considered variable by default.
```

3.2 POINT

```
    TELCAL\POINT [/CROSS]

Solve for telescope pointing parameters. The default (and currently only
solution) is to solve through a LAMBDA/BETA cross. Parameters are solved
and the fit is plotted as soon as each subscan is finished. User can ac-
cess  to  all related information through the global SIC structure named
PCROSS.
```

A Minutes of the TELCAL kickoff meeting (May 2004)

```
Introduction:
-------------

Goal:
  - NCS delegates the telescope calibration to reduction packages
  - need a telescope calibration library, perhaps inside GILDAS

"Desirable" Features
  - Re-use what is available
  - avoid duplication of efforts
  - Standardization
  - Easy maintenance

Contraints
  - many actors
  - short timescales
  - Portability
  - backward compatibility (must be able to process old data)
  - new ATM version

Alternative(s)
  - per package solution ? leads to maintenance issues, and serious loss of time...
  - make a common library
        - requires to define needs and interfaces
        - requires clients to adapt to the library constraints

Beware:
  - RED lifetime was 20 years...
  - Wide distribution of the library through GILDAS
  - IRAM has a record of success, which leads to high expectations
  - Failure is not an option (success is easily forgotten)

For a library, we need to define:
  - Functionalities
  - Interfaces with outside world (ATM / Reduction packages / NCS / other libraries)
  - Internal data format
  - Internal design (backend handling, fitting, ...)

***************************************************************************

Conclusions:
------------

- A somewhat cryptic table of the current situation is attached.
- List of desirable library functionalities:
   * Focus
   * Pointing
   * Chopper
   * Skydips
   * Sideband rejection
   * Array geometry
   * Array gains
   * Baseline
   * Regridding
   (* Fitting maybe for specialized algorithms)

- J.Penalver is now in charge of Pointing Models for the 30m. Pointing
  models thus are outside the scope of the library at least in its first
  iteration.

- Language to be used: F90 with derived types, modules for at least derived
  types and definition of function interfaces.

- Goals:
   * 1st version of library should be available by end of year
     (November 2004 if possible).
   * 1st version meaning same functionalities as providing now by red:
     focus, pointing, chopper (for spectra), skydips (for spectra),
     sideband rejection
   * Priorities in library building:
       1 Focus
       2 Pointing
       3 Chopper
       4 Skydips
       5 Sideband rejection

- Work distribution:
   o Related work without timescale:
      * A.Sievers to distribute an example of 30M-FITS file with corresponding
        documentation for each back-end as soon as they become available
        starting with bolometer and then 4MHz.
      * H.Wiesemeyer to provide translation from 30M-FITS to CLASS data
        format.
      * A.Bacmann and S.Guilloteau to make a memo on bandpass calibration
        at the 30m and PdBI to take into account bandwith increase and new
        ATM possibilities.
   o Library work:
      * Small description of sideband rejection measurement at the 30m.
         => A.Sievers (ASAP)
      * Iteration on interfaces and practical derived types:
         => J.Pety and S.Guilloteau 31.05.2004
     * Test on minimization routines (eg can we realiably use a 2-D
       minimization routine to fit a 1-D problem?):
        => S.Guilloteau 30.06.2004
     * One fully working example (focus):
        => J.Pety and H.Wiesemeyer 30.06.2004
     * A point will then be made enabling new work distribution round.

***************************************************************************

Pointing Methods

   - least square fit
        * Input
            - lambda offsets
            - beta   offsets
            - lambda derivatives
            - beta   derivatives
            - intensities
            - weights
            - fitting shape (e.g. Gaussian,n-Gaussians)
              Parameters, Guesses, Fixed/Fitted, Boundaries???

        * Output
            - parameters
            - errors
            - quality flags

   - evalute fitted function
        * Input
            - fitting shape, parameters
            - input coord (lambda,beta)

        * Output
            - intensities

   - plotting: Fit overplotted over data for two crosses
        * Input:
            - System, Units

        * Output:
            - Screen plot

   - results for NCS
        * Input:
            - Output of fit
            - Logging info
        * Output:
            - XML file
            - sic procedure (OBS)

Comments:
   - dealing with spillover on multi-beam receivers ?
   - start with "single-pixel" version
   - can deal with multi-pixel in two different ways
   - changing the lambda(i), beta(i) for each pixel
   - or putting the relative positions of the beams into the fitting function
   - is there a gain to be obtained by fitting a broadened beam for
     planets ?  need to evaluate that issue separately.

----------------------------------------------------------------------------

Focus Methods: Very similar to pointing...

----------------------------------------------------------------------------

Sideband Rejection:

 - currently being done using the Martin-Pupplet settings, by coupling one
   of the sidebands with the cold load, the others with the hot load.
 - is the method accurate enough ?
 - can we rely on the  engineer tables ?
 - measurement seems to be the most accurate method, but the precision is
   unknown ?
 - should it be frequency dependent ?

First step: transfert what is currently done in RED into the library ..

----------------------------------------------------------------------------

Chopper:

   * Input:
       - El
       - sideband rejection(i)
       - RF Frequency nu(i) and Bandwidth dnu(i)
       - Signed IF Frequency
       - Hot,Cold,Sky,Offset(dark count) temperature T(i)
       - Hot and Cold Temperatures T(i) and coupling coefficients f(i)
       - Telescope specific: Pamb, Water scale height, altitude, latitude
       - Date

       Pin = fhot B(Thot) + (1-fhot) B(Tamb) = B(Teff)
   * Output:
       - Trec(i)
       - Water
       - Water and Others Zenith Opacities(i)

Tcal:

   * Input:
       - El
       - RF Frequency nu(i)
       - Signed IF Frequency
       - Sideband rejection
       - Set of Tamb (cabine, ground) and coupling coefficients f(i)
       - Water
       - (Interpolated) Trec(i) for dual load

   * Output
       - Tcal(i)
       - Water and Others Zenith Opacities(i) for DSB analysis

---------------------------------------------------------------------------

Skydip

   * Input:
       - El(j)
       - sideband rejection(i)
       - RF Frequency nu(i) and Bandwidth dnu(i)
       - Signed IF Frequency
       - Hot,Cold,Offset(dark count) temperature T(i) TSky(i,j)
       - Hot and Cold Temperatures T(i) and coupling coefficients f(i)
       - Telescope specific: Pamb, Water scale height, altitude, latitude
       - Date
       - Bolometer bandpass(i)

   * Output:
       - Value and errors of Water vapour(i)
       - Value and errors of Feff(i)
       - Quality flag

---------------------------------------------------------------------------

Baseline 1-D

   - Fitting:
       * Input
           - Abscissa(i)
```

```
          - Ordinate(i)
          - Weights(i) (may be 0 to exclude a window)
          - Fitting shape (eg polynomial,spline)

      * Output
          - Fitted parameters
          - RMS

  - Baseline computation:
      * Input
          - Abscissa(i)
          - Fitting shape
          - Fitted parameters

      * Output
          - Baseline(i)

  - Baseline application
      * Input
          - Baseline(i)
          - Ordinate(i)

      * Output
          - Baseline(i)-Ordinate(i)

---------------------------------------------------------------------------

Baseline 1-D: On-Off => To be discussed later.

---------------------------------------------------------------------------

Baseline 2-D: spectral x time

  - Fitting:
      * Input
          - Frequency nu(i)
          - Space      x(j)
          - Ordinate   y(i,j)
          - Weights(i,j) (may be 0 to exclude a window)
          - Fitting shape:
                * Factorized polynomials
                * Box averaging of independant polynomials
                * Cubic spline interpolation of independent polynomials
                * Cubic spline interpolation of independent cubic spline
                * sinusoidal(nu) with coefficients being polynomials(t)

      * Output
```

```
            - Fitted parameters
            - RMS

   - Baseline computation:
       * Input
           - Abscissa(i)
           - Fitting shape
           - Fitted parameters

       * Output
           - Baseline(i)

   - Baseline application
       * Input
           - Baseline(i)
           - Ordinate(i)

       * Output
           - Baseline(i)-Ordinate(i)

On-off: simultaneous fitting of background and baseline

---------------------------------------------------------------------------

Baseline 2-D: Spatial data => to be discussed later

---------------------------------------------------------------------------

Regridding 1-D

       * Input
           - Abscissa(i)
           - Ordinate(i)
           - Weights(i)
           - New reference pixel, value, increment, dimension
           - Algorithm

       * Output
           - New abscissa(i)
           - New ordinate(i)
           - New weights(i)
           - Transformation matrix for signal and weight

Comment: Special case for already regularly sampled data

---------------------------------------------------------------------------
```

```
Regridding 2-D

    - If it can be factorized (eg space x spectral), then use 1-D
      regridding

    - Else
       * Input
           - Abscissa(i)
           - Ordinate(i)
           - Weights(i)
           - New reference pixel, value, increment, dimension
           - Algorithm

       * Output
           - New abscissa(i)
           - New ordinate(i)
           - New weights(i)
           - Transformation matrices for signal and weight

---------------------------------------------------------------------------

Array geometry: No consensus yet. In the first iteration of the library,
this functionality will be delivered by MOPSIC.
```

Index